Supplementary File S2: A graph-based approach to mapping human exposure-outcome associations for chemical contaminants

Taylor A. M. Wolffe[1,2], Paul Whaley[1,3], Crispin Halsall[1]

[1]Lancaster Environment Centre, Lancaster University, Lancaster, UK
[2]Yordas Group, Lancaster Environment Centre, Lancaster University, Lancaster, UK
[3]Evidence-Based Toxicology Collaboration, Johns Hopkins Bloomberg School of Public Health, Baltimore, MD 21205, USA

Relating controlled vocabulary code to nodes of extracted data within the graph

First, all packages required for the processing of the raw data were imported. A connection with the Neo4j graph database was also established:

In [7]:
```
#Importing all required packages...
import pandas as pd
from py2neo import  Graph, Node, Relationship

#Connecting to the neo4j graph database...
graph = Graph("http://localhost:7474/db/data/", auth=('UserHere', 'YourPassWordHere'))
```

Chemical Exposure Code

The code assigned by Sobus et al. at the publication level was mapped to the individually extracted chemical exposures within each publication. For example, Sobus et al. may have categorised “Author et al. 2010” as “metals/metalloids”. In this mapping exercise, we extracted the individual chemicals for “Author et al. 2010” e.g. “cadmium, blood”, “lead, blood” etc. By matching the publication (e.g. “Author et al. 2010”) we used Sobus et al.’s code to categorize all individually extracted chemical exposures (e.g. “cadmium, blood” and “lead, blood” would both be individually coded as “metals/metalloids”).

In [9]:
```
#reading in the code assigned by Sobus et al. to each publication (Table S3) as a pandas dataframe:
Sobus_Coding = pd.read_excel('Table_S3.xlsx')

#Creating a list of all the unique controlled vocabulary codes used by Sobus et al to categorize included studies:
Sobus_Code = []
for index, row in Sobus_Coding.iterrows():
    if row['Chemical group assigned by Sobus et al'] not in Sobus_Code:
        Sobus_Code.append(row['Chemical group assigned by Sobus et al'])

#Creating nodes for each controlled vocabulary code*:
for node in Sobus_Code:
  a = Node("SobusCode", name="`" + node + "`")
  graph.create(a)

#Using a cypher query to create "CODED_AS" relationships between single chemical exposures and the corresponding code
 assigned by Sobus et al. to the publications in which they are reported:
Create_Code_Rels = []
for index, row in Sobus_Coding.iterrows():
    Create_Code_query = '''match (n:SingleChemicalExposure)<-[r*..3]-(m:Publication) where m.RefID="''' + row['Referen
ce'] + '''"
    match (p:SobusCode) where p.name="`''' + row['Chemical group assigned by Sobus et al'] + '''`"''' + '''
    merge (n)-[t:CODED_AS]->(p)'''
    Create_Code_Rels.append(Create_Code_query)

#Using a cypher query to create "CODED_AS" relationships between mixed chemical exposures and the corresponding code a
ssigned by Sobus et al. to the publications in which they are reported:
for index, row in Sobus_Coding.iterrows():
    Create_Code_query = '''match (n:MixedChemicalExposure)<-[r*..2]-(m:Publication) where m.RefID="''' + row['Referenc
e'] + '''"
    match (p:SobusCode) where p.name="`''' + row['Chemical group assigned by Sobus et al'] + '''`"''' + '''
    merge (n)-[t:CODED_AS]->(p)'''
    Create_Code_Rels.append(Create_Code_query)

#Running the cypher queries which create the "CODED_AS" relationships:
for query in Create_Code_Rels:
    graph.run(query)

#*Only run the above code once, to prevent creating duplicate nodes in the graph.
```

Due to the higher resolution of the graph, there was no need for a "multi-group" code. In fact, retaining a "multi-group" code lead to spurious relationships in the graph. A query was run to delete all "multi-group" nodes from the graph, along with their relationships. Another query was run to find all chemicals without an assigned controlled vocabulary code. These chemicals were manually re-assigned a code (other than "multi-group").

In [10]:
```
#detaching all the chemicals related to a "multi-group" node through a cypher query:
delete_multi_group = '''match (n)-[r]->(m:SobusCode)
where m.name = "`multi-group`"
detach delete (r)'''

graph.run(delete_multi_group)

#Finding all chemicals that were subsequently left unrelated to a controlled vocabulary code:
uncoded_singles = '''MATCH (n:SingleChemicalExposure)
WHERE NOT (n)-[:CODED_AS]->()
return n.name as UnCodedChems'''

uncoded_multis = '''MATCH (n:MixedChemicalExposure)
WHERE NOT (n)-[:CODED_AS]->()
return n.name as UnCodedChems'''

#Creating an excel document of the single chemical exposures which were previously assigned as "multi-group" for manua
l re-assignment:
uncoded_singles = uncoded_singles.to_data_frame()
uncoded_singles.to_excel('Uncoded_Singles_For_Manual_Coding.xlsx')

#Creating an excel document of the mixed chemical exposures which were previously assigned as "multi-group" for manual
re-assignment:
uncoded_multis = graph.run(uncoded_multis)
uncoded_multis = uncoded_multis.to_data_frame()
uncoded_multis.to_excel('Uncoded_Multis_For_Manual_Coding.xlsx')
```

The output excel files were used to manually re-assign controlled vocabulary code. This re-assigned code was then matched to nodes in the graph and the corresponding chemicals related to controlled vocabulary code nodes through a "CODED_AS" relationship.

In [11]:
```
#reading in the manually assigned codes (Tables S4 and S5):
singles = pd.read_excel("Table_S4.xlsx")
multis = pd.read_excel("Table_S5.xlsx")

#creating a list of exposure-CODED_AS->code triples:
uncoded_triples = []
for index, row in singles.iterrows():
    uncoded_triples.append([row['UnCodedChems'], "CODED_AS", row['Code']])

for index, row in multis.iterrows():
    uncoded_triples.append([row['UnCodedChems'], "CODED_AS", row['Assigned Code']])

#Manually creating a "Dioxins, furans, PCBs" node, which was previously missing from the included studies*:
a = Node("SobusCode", name="`Dioxins, furans, PCBs`")
graph.create(a)

#Matching and relating chemical exposures to controlled vocabulary code:
for triple in uncoded_triples:
    b = graph.nodes.match(name=triple[0]).first()
    c = graph.nodes.match(name=triple[2]).first()
    d = Relationship(b, triple[1], c)
    graph.create(d)
```

Health Outcome Code

Similarly, the code manually assigned for the health outcomes (Supplementary Table S6) was used to create health code nodes, and these nodes related to relevant health outcome nodes through a "CODED_AS" relationship.

In [12]:
```
#Creating a list of unique health outcomes recorded in Table S2 for manual coding:
df = pd.read_excel('Table_S2.xlsx')

df_health = df[['AssociatedOutcome']]

df_health = df_health.drop_duplicates()

df_health.to_excel('Check_Health_Outcomes_For_Coding.xlsx')


#Reading in the manually assigned health outcome nodes (Table S6)...
health_code = pd.read_excel('Table_S6.xlsx')

#removing all trailing white spaces from the values in the dataframe
health_code = health_code.apply(lambda x: x.str.strip() if x.dtype == "object" else x)

#Creating a list of health outcome-CODED_AS->code triples:
health_code_triples = []

#slicing the last four columns of the data frame which contain the health outcome codes:
code_cols = health_code.iloc[:, -4:]

#creating a list of column names for the columns containing health outcomes:
cols = list(code_cols.columns)

for index, row in health_code.iterrows():
    for item in cols:
         if pd.notna(row[item]) is True:
                health_code_triples.append(['`' + row['AssociatedOutcome'] + '`', 'CODED_AS', '`' + row[item] + '`'])



#Creating health code nodes*:
health_code_nodes = []

for item in health_code_triples:
    if item[2] not in health_code_nodes:
        health_code_nodes.append(item[2])

for node in health_code_nodes:
    a = Node("HealthOutcomeCode", name=node)
```

```
    graph.create(a)

#Matching and relating the nodes within each triple in the graph*:
for triple in health_code_triples:
    b = graph.nodes.match("HealthOutcome", name=triple[0]).first()
    c = graph.nodes.match("HealthOutcomeCode", name=triple[2]).first()
    d = Relationship(b, triple[1], c)
    graph.create(d)

#*Only run the above code once, to prevent creating duplicate nodes in the graph.
```