# PSS-DiNAR-refinements

February 25, 2025

Table of Contents



We import all the required Python libraries

The non-default libries are networkX (https://networkx.org/) and py4cytoscape (https://py4cytoscape.readthedocs.io/, only necessary if you wish to view the results in Cytoscape).

You will also need the skm-tools package provided in the same repository as this notebook.

[191]:
```
#pip install networkx
#pip install --upgrade networkx
# pip install "skm-tools[cytoscape] @ git+https://github.com/NIB-SI/skm-tools.
↪git"
```

[192]:
```
import sys
from pathlib import Path
from datetime import datetime
import networkx as nx
import pandas
```

[193]:
```
from collections import defaultdict
```

The following allows us to import functions from the skm-tools package. Note the relative path to the folder containing the "skm_tools" directory.

[194]:
```
# sys.path.append("../")
from skm_tools import load_networks, pss_utils
```

[195]:
```
today = datetime.today().strftime('%Y.%m.%d'); today
```

[195]:
```
'2025.02.25'
```

### 1.2 Path and parameter definitions

[196]:
```
base_dir = Path("./")
data_dir = base_dir / "../input"
output_dir = base_dir / "../output"
print(data_dir)
```

```
..\input
```

[197]:
```
from pathlib import Path

base_dir = Path().resolve()  # Gets the current working directory
data_dir = (base_dir / "../input").resolve()
output_dir = (base_dir / "../output").resolve()

print(data_dir)
print(output_dir)
```

```
\\srvljfs.nib.sql\fito\DEJAVNOSTI\OMIKE\pISA-Projects\_p_Endophytes\_I_Bsubtilis
\_S_06_networkExpress\_A_03_networkExpress-R\input
\\srvljfs.nib.sql\fito\DEJAVNOSTI\OMIKE\pISA-Projects\_p_Endophytes\_I_Bsubtilis
\_S_06_networkExpress\_A_03_networkExpress-R\output
```

### 1.3 Load PSS

To obtain the exact results of the article, download PSS-v1.0.0 from skm.nib.si/downloads, and adjust the below paths accordingly. Otherwise, this code will use the latest live PSS instance.

[198]:
```
#pss_edge_path = data_dir / f"pss-dinar-edges-restricted_2022-10-26.tsv"
#pss_node_path = data_dir / f"pss-dinar-nodes-restricted_2022-10-26.tsv"
import os

pss_edge_path = data_dir / f"rxn-edges-public.tsv"
pss_node_path = data_dir / f"rxn-nodes-public.tsv"

print("Checking files...")
print("Edge file exists:", os.path.exists(pss_edge_path))
print("Node file exists:", os.path.exists(pss_node_path))
```

```
Checking files…
Edge file exists: True
Node file exists: True
```

[199]:
```
#for file in data_dir.glob("*"):
#    print(file)
```

[200]:
```
test = pandas.read_table(pss_edge_path)
test.head()
```

```
test = pandas.read_table(pss_node_path)
test.head()
```

[200]:

```
                name      node_type short_name functional_cluster_id  \
0        LHB1B1|VPg        Complex        NaN                    NaN
1               VPg  ForeignCoding        NaN                    NaN
2  LHB1B1[AT2G34430]    PlantCoding     LHB1B1                fc00367
3         PSB33|VPg        Complex        NaN                    NaN
4   PSB33[AT1G71500]    PlantCoding      PSB33                fc00364

   reaction_id  reaction_type  reaction_effect additional_information  \
0          NaN            NaN              NaN                    NaN
1          NaN            NaN              NaN                    NaN
2          NaN            NaN              NaN                    NaN
3          NaN            NaN              NaN                    NaN
4          NaN            NaN              NaN                    NaN

                             pathway  \
0  Primary metabolism - Photosynthesis
1                      Stress - Biotic
2  Primary metabolism - Photosynthesis
3  Primary metabolism - Photosynthesis
4  Primary metabolism - Photosynthesis

                                     all_pathways  …  \
0  Primary metabolism - Photosynthesis,Stress - B…  …
1                                Stress - Biotic  …
2               Primary metabolism - Photosynthesis  …
3  Primary metabolism - Photosynthesis,Stress - B…  …
4               Primary metabolism - Photosynthesis  …

                    external_links  family  function classification  \
0                              NaN     NaN       NaN            NaN
1  doi:10.1016/j.coviro.2012.09.004     NaN       NaN          virus
2                              NaN     NaN       NaN            NaN
3                              NaN     NaN       NaN            NaN
4                              NaN     NaN       NaN            NaN

             components ath_homologues nta_homologues osa_homologues  \
0  LHB1B1[AT2G34430],VPg            NaN            NaN            NaN
1                    NaN            NaN            NaN            NaN
2                    NaN      AT2G34430            NaN            NaN
3   PSB33[AT1G71500],VPg            NaN            NaN            NaN
4                    NaN      AT1G71500            NaN            NaN

  stu_homologues sly_homologues
0            NaN            NaN
```

```
1            NaN            NaN
2            NaN            NaN
3            NaN            NaN
4            NaN            NaN

[5 rows x 23 columns]
```

[201]:
```
g = load_networks.pss_to_networkx(
    edge_path=pss_edge_path,
    node_path=pss_node_path
)

# print(f"\nNumber of nodes: {g.number_of_nodes()}\nNumber of edges: {g.
 ↪number_of_edges()}")
```

### 1.4 Remove deadend complexes

"deadend" complexes are those created automatically if a binding/oligomerisation is entered into PSS. However, they only make sense to include here if the complex has some downstream interactions it takes part in, since this projection already includes the substrate-substrate interaction.

The DiNAR projection does not include complexes if the binding/oligomerisation reaction if formulated as inhibition of the substrate(s) (and not activation of the complex). This removes the majority of the problematic complexes. But since (1) the inhibition or activation annotation of the reaction is not always included in the reaction formulation correctlyand (2) somebody maybe hasn't yet added the downstream interactions, let's check for deadend complexes and remove them anyway.

[202]:
```
complexes = [n for n, data in g.nodes(data=True) if data["node_type"] ==␣
 ↪"Complex"]; complexes
```

[202]:
```
['CML|HC-Pro',
 'EDS1|PAD4',
 'EDS1|MPK3|PAD4',
 'CRT|ETR',
 'HSP90|RAR1|SGT1',
 'EDS1|PAD4|SAG',
 'TORC1',
 'NDR1|RIN4',
 'NPR1|TGA2,5,6',
 'RANGAP|Rx',
 'GPAphid2|RANGAP',
 'OBE1|WRKY17',
 'OBE1|WRKY11',
 'CO|OBE1',
 'BAK1|FLS2|flg22',
 'MKS1|WRKY33',
 'GA|GID1',
```

```
'GA|GID1|SLR1',
'WRKY30|WRKY53',
'SCL14|TGA2,5,6',
'SCF',
'GID|SCF',
'D14|MAX2|SCF',
'R-gene|potyvirus',
'EIN3(like)|JAZ',
'COI1|JA-Ile|SCF',
'NPR1|NPR1',
'EBF|SCF',
'DELLA|GA|GID1',
'RISC - vsiRNA',
'CTR|ETR',
'ETP|SCF',
'CML|Ca2+',
'WD/bHLH/MYB',
'BAK1|BRI1',
'SARD1|TCP8',
'TCP8|WRKY28',
'NAC019|TCP8',
'NPR1|WRKY18',
'CDK|NPR1',
'CDK|WRKY6,18',
'CDK|TGA2,5,6',
'PEPR1|PEP1',
'BIK1|PEPR1|PEP1',
'ET|ETR',
'RISC - miR390',
'GAI|GI',
'GI|PIF4',
'EIN3|NPR1',
'CDK|RAP2-6',
'RAP2-6|SNRK2',
'HSP70|HSP90|SGT1',
'CML12|PID',
'PBP1|PID',
'CMI1|ICR1',
'ROP9|SlRBOHB',
'FKBP42|HSP90',
'FKBP42|CAM',
'CBL1|CIPK1',
'CBL9|CIPK1',
'FKBP62|HSP90.1',
'CBL1|CIPK23',
'CBL9|CIPK23',
'ABF1|IDD14',
```

```
 'ABF2|IDD14',
 'ABF3|IDD14',
 'ABF4|IDD14',
 'HSFA1d|PIF4',
 'ABI4|RGL2',
 'ABI5|HY5',
 'CBL2|Ca2+',
 'CBL3|Ca2+',
 'CBL2|CIPK3|Ca2+',
 'CBL2|CIPK9|Ca2+',
 'CBL2|CIPK26|Ca2+',
 'CBL3|CIPK3|Ca2+',
 'CBL3|CIPK9|Ca2+',
 'CBL3|CIPK26|Ca2+',
 'PR1|cholesterol',
 'CBL2|CIPK3',
 'CBL2|CIPK9',
 'CBL2|CIPK23',
 'CBL2|CIPK26',
 'CBL3|CIPK3',
 'CBL3|CIPK9',
 'CBL3|CIPK23',
 'CBL3|CIPK26',
 'PIF3-TOC1-SP6A complex']
```

[203]:
```
# g is a directed graph, so we can just use "neighbors" to find complexes␣
 ↪without out/downstream edges
to_remove = []
for c in complexes:
    if len(list(g.neighbors(c))) == 0:
        print(c)
        to_remove.append(c)
```

```
CML|HC-Pro
EDS1|MPK3|PAD4
CRT|ETR
HSP90|RAR1|SGT1
NDR1|RIN4
RANGAP|Rx
GPAphid2|RANGAP
OBE1|WRKY17
OBE1|WRKY11
CO|OBE1
MKS1|WRKY33
GA|GID1|SLR1
WRKY30|WRKY53
R-gene|potyvirus
```

EIN3(like)|JAZ
DELLA|GA|GID1
SARD1|TCP8
TCP8|WRKY28
NAC019|TCP8
NPR1|WRKY18
CDK|NPR1
CDK|WRKY6,18
CDK|TGA2,5,6
BIK1|PEPR1|PEP1
ET|ETR
RISC - miR390
GAI|GI
GI|PIF4
CDK|RAP2-6
RAP2-6|SNRK2
CML12|PID
PBP1|PID
CMI1|ICR1
ROP9|S1RBOHB
FKBP42|HSP90
FKBP42|CAM
CBL1|CIPK1
CBL9|CIPK1
FKBP62|HSP90.1
CBL1|CIPK23
CBL9|CIPK23
ABF1|IDD14
ABF2|IDD14
ABF3|IDD14
ABF4|IDD14
HSFA1d|PIF4
CBL2|CIPK3|Ca2+
CBL2|CIPK9|Ca2+
CBL2|CIPK26|Ca2+
CBL3|CIPK3|Ca2+
CBL3|CIPK9|Ca2+
CBL3|CIPK26|Ca2+
PR1|cholesterol
CBL2|CIPK3
CBL2|CIPK9
CBL2|CIPK23
CBL2|CIPK26
CBL3|CIPK3
CBL3|CIPK9
CBL3|CIPK23
CBL3|CIPK26
PIF3-TOC1-SP6A complex

[204]:
```
g.remove_nodes_from(to_remove)
```

### 1.5 filter PSS by types

[205]:
```
set([d["node_type"] for n, d in g.nodes(data=True)])
```

[205]:
```
{'Complex',
 'Condition',
 'ForeignAbiotic',
 'ForeignAbstract',
 'ForeignCoding',
 'ForeignEntity',
 'ForeignNonCoding',
 'Metabolite',
 'PlantAbstract',
 'PlantCoding',
 'PlantNonCoding',
 'Process'}
```

[206]:
```
keep = ['Complex',
        'Metabolite',
        'PlantAbstract',
        'PlantCoding',
        'PlantNonCoding',
        'Process'
        ]
pss_utils.filter_pss_nodes(g, node_types = keep, remove_isolates=True)
```

```
Removed 83 nodes from network.
```

[206]:
```
{'VPg': 'wrong node type',
 'HC-Pro': 'wrong node type',
 'Drought': 'wrong node type',
 'P3': 'wrong node type',
 'CI': 'wrong node type',
 'CP': 'wrong node type',
 'NPR1 high & JAZ high': 'wrong node type',
 'NPR1 high & JAZ low': 'wrong node type',
 'flg22': 'wrong node type',
 'potyvirus': 'wrong node type',
 'vsiRNA34327': 'wrong node type',
 'vsiRNA12986': 'wrong node type',
 'Waterlogging': 'wrong node type',
 'SA low': 'wrong node type',
 'Heat': 'wrong node type',
 'NPR1 low & JAZ low': 'wrong node type',
 'trichous-bacteria': 'wrong node type',
```

```
'SA high': 'wrong node type',
'viral dsRNA': 'wrong node type',
'vsiRNA': 'wrong node type',
'me-vsiRNA': 'wrong node type',
'6K2': 'wrong node type',
'P1': 'wrong node type',
'NIa-Pro': 'wrong node type',
'NIb': 'wrong node type',
'red light': 'wrong node type',
'Phytophthora': 'wrong node type',
'TIC56': 'wrong node type',
'BAK1|FLS2|flg22': 'complex component removed',
'LHB1B1[AT2G34430]': 'isolate',
'PSB33[AT1G71500]': 'isolate',
'EDF2[AT1G68840]': 'isolate',
'SERPIN1[AT1G47710]': 'isolate',
'clpP1[ATCG00670]': 'isolate',
'FQR1[AT5G54500]': 'isolate',
'PR-3 like[AT2G43590]': 'isolate',
'TDX[AT3G17880]': 'isolate',
'RH8[AT4G00660]': 'isolate',
'RPS12C[AT2G32060]': 'isolate',
'MTI20.11[AT5G57860]': 'isolate',
'CPIP[AT1G10350,AT3G08910]': 'isolate',
'ATPB[ATCG00480]': 'isolate',
'MIND1[AT5G24020]': 'isolate',
'PAA2[AT2G05840]': 'isolate',
'PBB2[AT5G40580]': 'isolate',
'PBE1[AT1G13060]': 'isolate',
'GAPC2[AT1G13440]': 'isolate',
'AT3G17020[AT3G17020]': 'isolate',
'PUB15[AT5G42340]': 'isolate',
'PSAK[AT1G30380]': 'isolate',
'TIP1-2[AT3G26520]': 'isolate',
'OBE2[AT5G48160]': 'isolate',
'O2-deficiency': 'isolate',
'TUBA2[AT1G50010]': 'isolate',
'REM12[AT2G24680]': 'isolate',
'EIF4E1[AT4G18040]': 'isolate',
'GER3[AT5G20630]': 'isolate',
'PAB2[AT4G34110]': 'isolate',
'DCL2,4[AT3G03300,AT5G20320]': 'isolate',
'PAB4[AT2G23350]': 'isolate',
'PAB8[AT1G49760]': 'isolate',
'PORB[AT4G27440]': 'isolate',
'PORA[AT5G54190]': 'isolate',
'FBA8[AT3G52930]': 'isolate',
```

```
 'GSTF9[AT2G30860]': 'isolate',
 'HUP54[AT4G27450]': 'isolate',
 'VDAC2[AT5G67500]': 'isolate',
 'PSBR[AT1G79040]': 'isolate',
 'MSL[AT1G53470,AT1G58200,AT1G78610,AT2G17000,AT2G17010,AT3G14810,AT4G00290,AT5G
10490,AT5G12080,AT5G19520]': 'isolate',
 'BAS1[AT3G11630]': 'isolate',
 'PSBP1[AT1G06680]': 'isolate',
 'RCA[AT2G39730]': 'isolate',
 'MED37E[AT5G02500]': 'isolate',
 'IPP2[AT3G02780]': 'isolate',
 'ABCF1[AT5G60790]': 'isolate',
 'CLPR3[AT1G09130]': 'isolate',
 'psbB[ATCG00680]': 'isolate',
 'BCCP1[AT5G16390]': 'isolate',
 'GRF6[AT5G10450]': 'isolate',
 'CAC2[AT5G35360]': 'isolate',
 'MPPBETA[AT3G02090]': 'isolate',
 'ATPC1[AT4G04640]': 'isolate',
 'bHLH93[SOLTU.DM.02G007350]': 'isolate'}
```

### 1.6 Remove "uninteresting" metabolites, and connect all their neighbours

[207]:
```
metabolites = [n for n, data in g.nodes(data=True) if data["node_type"] ==␣
 ↪"Metabolite"]; metabolites
```

[207]:
```
['CA',
 'CA-CoA',
 'Ca2+',
 'MeCLA',
 'IsoChor-9-Glu',
 'N-pyruvoyl-L-Glu',
 'SA',
 'GA',
 'DMAPP',
 'prenyl-tRNA',
 'tRNA-adenine',
 'cZ-riboside',
 'cZ-ribotide',
 'cZ',
 'cZ-glucosides',
 'UDP-Glc',
 'DZ-riboside',
 'DZ-ribotide',
 'tZ',
 'DZ',
 'iP',
```

```
'L-Met',
'SAMe',
'ACC',
'ET',
'ALA',
'MDA',
'ROS',
'Thr',
'Ile',
'13-HPOT',
'12,13-EOT',
'OPDA',
'OPC8',
'OPC8-CoA',
'OPC6-CoA',
'OPC4-CoA',
'JA-CoA',
'JA',
'MeJA',
'JA-Ile',
'12-OH-JA-Ile',
'MeSA',
'Chor',
'IsoChor',
'Prep',
'Phe',
'BA',
'SAG',
'O2',
'e-',
'Geranylgeranyl-PP',
'ent-Copalyl-PP',
'ent-Kaurene',
'ent-Kaurenoic acid',
'GA12',
'GA9',
'GA20',
'GA8',
'GA34',
'GA-methylester',
'&beta;-Carotene',
'9-cis-&beta;-carotene',
'CL',
'CLA',
'HMBDP',
'iP-ribotide',
'phospholipids',
```

```
'DZ-glucosides',
'iP-riboside',
'tZ-riboside',
'tZ-ribotide',
'tZ-glucoside',
'p-Coumaric acid',
'iP-glucosides',
'Cu2+',
'Violaxanthin',
'9-cis-Violaxanthin',
'Phytoene',
'SL',
'Abscisic aldehyde',
'ABA',
'Xanthoxin',
'Antheraxanthin',
'OG',
'L-arogenate',
'Tyr',
'Zeaxanthin',
'&beta;-Cryptoxanthin',
'&gamma;-Carotene',
'Lycopene',
'all-trans-Neurosporene',
'all-trans-zeta-Carotene',
'all-trans-Phytofluene',
'3H3PP-CoA',
'3O3PP-CoA',
'9-cis-10&prime;-apo-&beta;-carotenal',
'cis-prenyl-tRNA',
'BA-CoA',
'BD',
'SA-Asp',
'Trp',
'IAM',
'S-alkyl-thiohydroximate',
'Thiohydroximate',
'MVA',
'5-phosphomevalonate',
'IAA-Ala',
'IAA',
'IPA',
'Brassinolide',
'IAOx',
'IAOx N-oxide',
'IAN',
'IAA-Leu',
```

```
 'MeIAA',
 'eATP',
 'IAA-Glu',
 'IAA-Asp',
 'L-Dopa',
 'DOPAL',
 'PAA',
 'SAH',
 'L-homo-cys',
 'PA',
 'Glu',
 'P5C',
 'InsP4',
 'InsP5',
 'InsP6',
 'InsP7',
 'InsP8',
 'ABA-GE',
 '8&prime; OH-ABA',
 'Phaseic acid',
 'DPA',
 'oxIAA',
 '12-OH-JA',
 'cholesterol']
```

[208]:
```
# List here the metabolites to keep as is
#metabolites_to_keep = [
#    'Ca2+',
#    'SA',
#    'GA',
#    'cZ',
#    'ET',
#    'JA',
#    'JA-Ile',
#    'ROS',
#    'O2',
#    'Cu2+',
#    'ABA',
#    'SL',
#    'IAA',
#]
metabolites_to_keep = ['ROS', 'Ca2+']
```

[209]:
```
reaction_types_to_collapse = [
    "catalysis",
    "translocation"
]
```

[210]:
```
for metabolite in metabolites:
    if not metabolite in metabolites_to_keep:

        # find all neighbours that are: (not Metabolite) or (in␣
↪metabolites_to_keep)
        targets = []
        edges = {}
        sources = []

        for target in g.successors(metabolite):
            e = g[metabolite][target][0]

            # right type of reaction
            if e['reaction_type'] in reaction_types_to_collapse:

                # right type of target node
                target_type = g.nodes()[target]['node_type']
                if target_type == "Metabolite":
                    if not (target in metabolites_to_keep):
                        continue

                edges[target] = {'interaction_type': 'activation',
                                 'directed': 'True',
                                 'reaction_type': 'catalysis',
                                 'reaction_effect': 'activation',
                                 'reaction_id': e['reaction_id'],
                                 'source_edge_type': 'PRODUCT/SUBSTRATE',
                                 'target_edge_type': 'PRODUCT/SUBSTRATE',
                                 'note': f'metabolite collapse {metabolite}'
                                 }

                targets.append(g.nodes()[target])

        for source in g.predecessors(metabolite):
            e = g[source][metabolite][0]

            # right type of reaction
            if e['reaction_type'] in reaction_types_to_collapse:

                # right type of source node
                source_type = g.nodes()[source]['node_type']
                if source_type == "Metabolite":
                    if not (source in metabolites_to_keep):
                        continue

                sources.append(g.nodes()[source])
```

```
        for source in sources:
            for target in targets:
                if source != target:
                    g.add_edge(source['name'], target['name'],␣
 ↪**edges[target['name']])

        g.remove_node(metabolite)
```

[211]:
```
print(f"\nNumber of nodes: {g.number_of_nodes()}\nNumber of edges: {g.
 ↪number_of_edges()}")
```

```
Number of nodes: 399
Number of edges: 732
```

[212]:
```
# Exercise - remove self loops and duplicated edges?
```

### 1.7 main component

[213]:
```
g = g.subgraph(max(nx.weakly_connected_components(g), key=len)).copy()
```

## 2 Cytoscape

First open the Cytoscape application. Then the following cell will load the required library and and make sure you can connect to the Cytoscape application.

More py4cytoscape documentation is here: https://py4cytoscape.readthedocs.io/

[214]:
```
from skm_tools import cytoscape_utils
```

### 2.1 Start Cytoscape and then proceed

[215]:
```
import py4cytoscape as p4c
p4c.cytoscape_ping()
p4c.cytoscape_version_info()
```

```
In cyrest_get: HTTPConnectionPool(host='127.0.0.1', port=1234): Max retries
exceeded with url: /v1/version (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at
0x0000022D7AAE6890>: Failed to establish a new connection: [WinError 10061] No
connection could be made because the target machine actively refused it'))
```

```
---------------------------------------------------------------------------
ConnectionRefusedError                    Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nection.
 ↪py:198, in HTTPConnection._new_conn(self)
```

```
    197 try:
--> 198     sock = connection.create_connection(
    199         (self._dns_host, self.port),
    200         self.timeout,
    201         source_address=self.source_address,
    202         socket_options=self.socket_options,
    203     )
    204 except socket.gaierror as e:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\ut .l\connection
 ↪py:85, in create_connection(address, timeout, source_address, socket_options)
     84 try:
---> 85     raise err
     86 finally:
     87     # Break explicitly a reference cycle

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\ut .l\connection
 ↪py:73, in create_connection(address, timeout, source_address, socket_options)
     72     sock.bind(source_address)
---> 73 sock.connect(sa)
     74 # Break explicitly a reference cycle

ConnectionRefusedError: [WinError 10061] No connection could be made because th␣
 ↪target machine actively refused it

The above exception was the direct cause of the following exception:

NewConnectionError                        Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nectionpool.
 ↪py:787, in HTTPConnectionPool.urlopen(self, method, url, body, headers,␣
 ↪retries, redirect, assert_same_host, timeout, pool_timeout, release_conn,␣
 ↪chunked, body_pos, preload_content, decode_content, **response_kw)
    786 # Make the request on the HTTPConnection object
--> 787 response = self._make_request(
    788     conn,
    789     method,
    790     url,
    791     timeout=timeout_obj,
    792     body=body,
    793     headers=headers,
    794     chunked=chunked,
    795     retries=retries,
    796     response_conn=response_conn,
    797     preload_content=preload_content,
    798     decode_content=decode_content,
    799     **response_kw,
```

```
    800 )
    802 # Everything went great!

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nnectionpool.
 ↪py:493, in HTTPConnectionPool._make_request(self, conn, method, url, body,␣
 ↪headers, retries, timeout, chunked, response_conn, preload_content,␣
 ↪decode_content, enforce_content_length)
    492 try:
--> 493     conn.request(
    494         method,
    495         url,
    496         body=body,
    497         headers=headers,
    498         chunked=chunked,
    499         preload_content=preload_content,
    500         decode_content=decode_content,
    501         enforce_content_length=enforce_content_length,
    502     )
    504 # We are swallowing BrokenPipeError (errno.EPIPE) since the server is
    505 # legitimately able to close the connection after sending a valid␣
 ↪response.
    506 # With this behaviour, the received response is still readable.

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nnection.
 ↪py:445, in HTTPConnection.request(self, method, url, body, headers, chunked,␣
 ↪preload_content, decode_content, enforce_content_length)
    444     self.putheader(header, value)
--> 445 self.endheaders()
    447 # If we're given a body we start sending that in chunks.

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.305 .
 ↪0_x64__qbz5n2kfra8p0\lib\http\client.py:1278, in HTTPConnection.
 ↪endheaders(self, message_body, encode_chunked)
   1277     raise CannotSendHeader()
-> 1278 self._send_output(message_body, encode_chunked=encode_chunked)

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.305 .
 ↪0_x64__qbz5n2kfra8p0\lib\http\client.py:1038, in HTTPConnection.
 ↪_send_output(self, message_body, encode_chunked)
   1037 del self._buffer[:]
-> 1038 self.send(msg)
   1040 if message_body is not None:
   1041
   1042     # create a consistent interface to message_body

File C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.305 .
 ↪0_x64__qbz5n2kfra8p0\lib\http\client.py:976, in HTTPConnection.send(self, dat )
```

```
    975 if self.auto_open:
--> 976     self.connect()
    977 else:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nection.
 ↪py:276, in HTTPConnection.connect(self)
    275 def connect(self) -> None:
--> 276     self.sock = self._new_conn()
    277     if self._tunnel_host:
    278         # If we're tunneling it means we're connected to our proxy.

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nection.
 ↪py:213, in HTTPConnection._new_conn(self)
    212 except OSError as e:
--> 213     raise NewConnectionError(
    214         self, f"Failed to establish a new connection: {e}"
    215     ) from e
    217 sys.audit("http.client.connect", self, self.host, self.port)

NewConnectionError: <urllib3.connection.HTTPConnection object at␣
 ↪0x0000022D7AAE6890>: Failed to establish a new connection: [WinError 10061] N␣
 ↪connection could be made because the target machine actively refused it

The above exception was the direct cause of the following exception:

MaxRetryError                             Traceback (most recent call last)
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\requests\a apters.
 ↪py:667, in HTTPAdapter.send(self, request, stream, timeout, verify, cert,␣
 ↪proxies)
    666 try:
--> 667     resp = conn.urlopen(
    668         method=request.method,
    669         url=url,
    670         body=request.body,
    671         headers=request.headers,
    672         redirect=False,
    673         assert_same_host=False,
    674         preload_content=False,
    675         decode_content=False,
    676         retries=self.max_retries,
    677         timeout=timeout,
    678         chunked=chunked,
    679     )
    681 except (ProtocolError, OSError) as err:
```

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\co nectionpool.
 ↪py:841, in HTTPConnectionPool.urlopen(self, method, url, body, headers,␣
 ↪retries, redirect, assert_same_host, timeout, pool_timeout, release_conn,␣
 ↪chunked, body_pos, preload_content, decode_content, **response_kw)
    839     new_e = ProtocolError("Connection aborted.", new_e)
--> 841 retries = retries.increment(
    842     method, url, error=new_e, _pool=self, _stacktrace=sys.exc_info()[2]
    843 )
    844 retries.sleep()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\urllib3\ut l\retry.
 ↪py:519, in Retry.increment(self, method, url, response, error, _pool,␣
 ↪_stacktrace)
    518     reason = error or ResponseError(cause)
--> 519     raise MaxRetryError(_pool, url, reason) from reason  # type:␣
 ↪ignore[arg-type]
    521 log.debug("Incremented Retry for (url='%s'): %r", url, new_retry)

MaxRetryError: HTTPConnectionPool(host='127.0.0.1', port=1234): Max retries␣
 ↪exceeded with url: /v1/version (Caused by NewConnectionError('<urllib3.
 ↪connection.HTTPConnection object at 0x0000022D7AAE6890>: Failed to establish ␣
 ↪new connection: [WinError 10061] No connection could be made because the␣
 ↪target machine actively refused it'))

During handling of the above exception, another exception occurred:

ConnectionError                           Traceback (most recent call last)
Cell In[215], line 2
      1 import py4cytoscape as p4c
----> 2 p4c.cytoscape_ping()
      3 p4c.cytoscape_version_info()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:133, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    131     return log_return(func, value)
    132 except Exception as e:
--> 133     log_exception(func, e)
    134 finally:
    135     log_finally()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:130, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    128 log_incoming(func, *args, **kwargs)
    129 try:
--> 130     value = func(*args, **kwargs) # Call function being logged
    131     return log_return(func, value)
```

```
    132 except Exception as e:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\cytoscape_s
 ↪py:58, in cytoscape_ping(base_url)
     37 """Ping Cytoscape
     38
     39 Tests the connection to Cytoscape via CyREST and verifies that supporte␣
 ↪versions of Cytoscape and CyREST API are loaded.
   (…)
     55     You are connected to Cytoscape!
     56 """
     57 from .py4cytoscape_utils import verify_supported_versions
---> 58 verify_supported_versions(1, 3.6, base_url=base_url)
     59 return narrate('You are connected to Cytoscape!')

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytoscap
 ↪py:848, in verify_supported_versions(cyrest, cytoscape, base_url, caller)
    823 def verify_supported_versions(cyrest=1, cytoscape=3.6,␣
 ↪base_url=DEFAULT_BASE_URL, caller=None):
    824     """Throws exception if min supported versions of api and cytoscape␣
 ↪are not running.
    825
    826     Extracts numerics from api and major cytoscape versions before␣
 ↪making comparison.
   (…)
    846         >>> verify_supported_versions(1, '3.10.1')
    847     """
--> 848     nogo =␣
 ↪check_supported_versions(cyrest=cyrest, cytoscape=cytoscape, base_url=base_ur )
    850     if nogo:
    851         if caller is None: caller = sys._getframe(1).f_code.co_name

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytoscap
 ↪py:792, in check_supported_versions(cyrest, cytoscape, base_url, caller,␣
 ↪test_cytoscape)
    768 """Checks to see if min supported versions of api and cytoscape are␣
 ↪running.
    769
    770 Extracts numerics from api, major and patch cytoscape versions before␣
 ↪making comparison, per semantic versioning @ https://semver.org/
   (…)
    788     >>> check_supported_versions(1, '3.10.0')
    789 """
    790 if isinstance(cytoscape, float): cytoscape = str(cytoscape)
--> 792 v = cytoscape_system.cytoscape_version_info(base_url=base_url)
    793 v_api_str = v['apiVersion']
```

```
    794 v_cy_str = v['cytoscapeVersion']  if test_cytoscape is None else␣
 ↪test_cytoscape

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:133, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    131     return log_return(func, value)
    132 except Exception as e:
--> 133     log_exception(func, e)
    134 finally:
    135     log_finally()

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:130, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    128 log_incoming(func, *args, **kwargs)
    129 try:
--> 130     value = func(*args, **kwargs) # Call function being logged
    131     return log_return(func, value)
    132 except Exception as e:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\cytoscape_s
 ↪py:81, in cytoscape_version_info(base_url)
     61 @cy_log
     62 def cytoscape_version_info(base_url=DEFAULT_BASE_URL):
     63     """Return the versions of the current Cytoscape and CyREST API.
     64
     65     Args:
   (…)
     79         {'apiVersion': 'v1', 'cytoscapeVersion': '3.8.1',␣
 ↪'automationAPIVersion': '0.0.0', 'py4cytoscapeVersion': '0.0.2'}
     80     """
---> 81     versions = commands.cyrest_get('version', base_url=base_url)
     82     if len(versions) == 0:
     83         raise CyError('CyREST connection problem. py4cytoscape cannot␣
 ↪continue!')

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:133, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    131     return log_return(func, value)
    132 except Exception as e:
--> 133     log_exception(func, e)
    134 finally:
    135     log_finally()
```

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\py4cytosca
 ↪py:130, in cy_log.<locals>.wrapper_log(*args, **kwargs)
    128 log_incoming(func, *args, **kwargs)
    129 try:
--> 130     value = func(*args, **kwargs) # Call function being logged
    131     return log_return(func, value)
    132 except Exception as e:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\commands.
 ↪py:160, in cyrest_get(operation, parameters, base_url, require_json, raw_get)
    158             return r.text
    159 except requests.exceptions.RequestException as e:
--> 160     _handle_error(e)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\commands.
 ↪py:150, in cyrest_get(operation, parameters, base_url, require_json, raw_get)
    148 try:
    149     url = build_url(base_url, operation)
--> 150     r =␣
 ↪_do_request('GET', url, params=parameters, base_url=base_url, raw_request=raw get)
    151     r.raise_for_status()
    152     try:

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\commands.
 ↪py:720, in _do_request(method, url, base_url, raw_request, **kwargs)
    717 if not raw_request:
    718     do_initialize_sandbox(requester, base_url=base_url) # make sure␣
 ↪there's a sandbox before executing a command
--> 720 return requester(method, url, **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\backoff\_s nc.
 ↪py:105, in retry_exception.<locals>.retry(*args, **kwargs)
     96 details = {
     97     "target": target,
     98     "args": args,
   (…)
    101     "elapsed": elapsed,
    102 }
    104 try:
--> 105     ret = target(*args, **kwargs)
    106 except exception as e:
    107     max_tries_exceeded = (tries == max_tries_value)
```

```
File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\py4cytosca e\commands.
 ↪py:709, in _do_request_local(method, url, **kwargs)
    705 @backoff.on_exception(backoff.expo, requests.exceptions.ConnectionError
 ↪max_tries=10)
    706 def _do_request_local(method, url, **kwargs):
    707     # Call CyREST via a local URL
    708     log_http_request(method, url, **kwargs)
--> 709     r = requests.request(method, url, **kwargs)
    710     log_http_result(r)
    711     return r

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\requests\a i.
 ↪py:59, in request(method, url, **kwargs)
     55 # By using the 'with' statement we are sure the session is closed, thus
 ↪we
     56 # avoid leaving sockets open which can trigger a ResourceWarning in som
     57 # cases, and look like a memory leak in others.
     58 with sessions.Session() as session:
---> 59     return session.request(method=method, url=url, **kwargs)

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\requests\s ssions.
 ↪py:589, in Session.request(self, method, url, params, data, headers, cookies,
 ↪files, auth, timeout, allow_redirects, proxies, hooks, stream, verify, cert,
 ↪json)
    584 send_kwargs = {
    585     "timeout": timeout,
    586     "allow_redirects": allow_redirects,
    587 }
    588 send_kwargs.update(settings)
--> 589 resp = self.send(prep, **send_kwargs)
    591 return resp

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\requests\s ssions.
 ↪py:703, in Session.send(self, request, **kwargs)
    700 start = preferred_clock()
    702 # Send the request
--> 703 r = adapter.send(request, **kwargs)
    705 # Total elapsed time of the request (approximately)
    706 elapsed = preferred_clock() - start

File ~\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.
 ↪10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\requests\a lapters.
 ↪py:700, in HTTPAdapter.send(self, request, stream, timeout, verify, cert,
 ↪proxies)
    696     if isinstance(e.reason, _SSLError):
    697         # This branch is for urllib3 v1.22 and later.
```

```
    698         raise SSLError(e, request=request)
--> 700     raise ConnectionError(e, request=request)
    702 except ClosedPoolError as e:
    703     raise ConnectionError(e, request=request)

ConnectionError: HTTPConnectionPool(host='127.0.0.1', port=1234): Max retries␣
↪exceeded with url: /v1/version (Caused by NewConnectionError('<urllib3.
↪connection.HTTPConnection object at 0x0000022D7AAE6890>: Failed to establish ␣
↪new connection: [WinError 10061] No connection could be made because the␣
↪target machine actively refused it'))
```

We set the Cytoscape collection name for this notebook.

[ ]:
```
COLLECTION = f"PSS DiNAR projection ({today})"
COLLECTION
```

### 2.2 Load the PSS network into Cytoscape

We load the network, set a visual style, and apply the CoSE layout.

With skm-tools, we provide a default style for PSS, colouring the nodes by pathway.

Returned is the ID of the network view in Cytoscape.

[ ]:
```
pss_network_suid = p4c.networks.create_network_from_networkx(g, title="Complete␣
↪PSS", collection=COLLECTION)
```

### 2.3 Exporting graph to file

[ ]:
```
df = nx.to_pandas_edgelist(g)
df.head()
df.to_csv(output_dir / f"pss-refined-edgelist-{today}.tsv", sep="\t",␣
↪index=None)
```

### 2.4 coordinates

[ ]:
```
# https://gist.github.com/carissableker/620f10ffdaedcb18faeac52aa1a9acdb
```

## 3 ignore all below

[ ]:
```
# Apply a style
#cytoscape_utils.apply_builtin_style(pss_network_suid, 'pss')
#pss_network_suid
```

# 4 Example path extraction

Before running this section, depending on the problem at hand, you may want to remove nodes such as virus related, of perhaps abiotic stress.

First we identify the nodes of interest.

[ ]:
```
JA = [x for x,y in g.nodes(data=True) if y['name']=="JA"][0]
SA = [x for x,y in g.nodes(data=True) if y['name']=="SA"][0];
ROS = [x for x,y in g.nodes(data=True) if y['name']=="ROS"][0]

print(JA)
print(SA)
print(ROS)
```

## 4.1 JA –> SA + JA –> ROS

[ ]:
```
JA_paths = [p for p in nx.all_shortest_paths(g, source=JA, target=SA)]
JA_paths += [p for p in nx.all_shortest_paths(g, source=JA, target=ROS)]
JA_paths
```

## 4.2 SA –> JA + SA –> ROS

[ ]:
```
SA_paths = [p for p in nx.all_shortest_paths(g, source=SA, target=JA)]
SA_paths += [p for p in nx.all_shortest_paths(g, source=SA, target=ROS)]
SA_paths
```

## 4.3 ROS –> JA + ROS –> SA

[ ]:
```
ROS_paths = [p for p in nx.all_shortest_paths(g, source=ROS, target=JA)]
ROS_paths += [p for p in nx.all_shortest_paths(g, source=ROS, target=SA)]
ROS_paths
```

# 5 Visualise paths in Cytoscape

Now we're going to highlight the paths we identified in the network by applying style bypasses.

We set the colours here:

[ ]:
```
JA_COLOUR = "#66a61e"
SA_COLOUR = "#34858d"
ROS_COLOUR = "#dc1c1c"
```

We don't want to recolour already highlighted path elements, so we keep track of them here:

[ ]:
```
done_nodes, done_edges = [], []
```

```
[ ]: for p in ROS_paths:
         done_nodes_now, done_edges_now = cytoscape_utils.highlight_path(p,␣
     ↪ROS_COLOUR, skip_nodes=done_nodes, skip_edges=done_edges)
         done_nodes += done_nodes_now
         done_edges += done_edges_now
     # Note - need to improve edge colouring
```

```
[ ]: for p in JA_paths:
         done_nodes_now, done_edges_now = cytoscape_utils.highlight_path(p,␣
     ↪JA_COLOUR, skip_nodes=done_nodes, skip_edges=done_edges)
         done_nodes += done_nodes_now
         done_edges += done_edges_now
```

```
[ ]: for p in SA_paths:
         print(p)
         done_nodes_now, done_edges_now = cytoscape_utils.highlight_path(p,␣
     ↪SA_COLOUR, skip_nodes=done_nodes, skip_edges=done_edges)
         done_nodes += done_nodes_now
         done_edges += done_edges_now
```

At this point, the Cytoscape session has a network view of the filtered PSS, and highlighting of the paths we extracted from our targeted searches.

## 5.1 Extract subnetworks in Cytoscape

Properly inspecting the identified paths is a bit hard within the complete network, so here we pull out the subnetworks of and surrounding the paths.

### 5.1.1 The edge induced network

The first, and smallest subnetwork, is created by extracting only the edges that are present on the paths.

```
[ ]: network_edge_induced_suid = cytoscape_utils.subnetwork_edge_induced_from_paths(
         paths=JA_paths + SA_paths + ROS_paths,
         g=g,
         parent_suid=pss_network_suid,
         name="identified paths (edge induced)",
     )
```

We apply a new layout to this subnetwork

```
[ ]: _ = p4c.layouts.layout_network('cose', network=network_edge_induced_suid)
```

### 5.1.2 The node induced network

Now we extract the network based on the nodes along the paths, meaning any edges between those nodes that are not on the paths are also extracted.

[ ]:
```
nodes = list(set([y for x in JA_paths + SA_paths + ROS_paths for y in x]))
```

[ ]:
```
network_node_induced_suid = cytoscape_utils.subnetwork_node_induced(
    nodes=nodes,
    parent_suid=pss_network_suid,
    name="identified paths (node induced)",
)
```

Instead of applying a network layout algorithm, we can copy the layout from the previous subnetwork.

[ ]:
```
_ = p4c.layouts.layout_copycat(
    network_edge_induced_suid,
    network_node_induced_suid
)
```

### 5.1.3 Neighbours

For more context around our paths, we can include the first neighbours in the view. We can use the Cytoscape first neighbour selection functionality.

[ ]:
```
network_neighbours_suid = cytoscape_utils.subnetwork_neighbours(
    nodes=nodes,
    parent_suid=pss_network_suid,
    name="identified paths + 1st neighbours",
)
```

[ ]:
```
_ = p4c.layouts.layout_network('cose', network=network_neighbours_suid)
```

### 5.1.4 Additional filtering of the neighbours

There are many neighbours displayed now, and we are perhaps only interested in the ones that are connected to at least two of the original path nodes, so we can make a filter using networkX neighbour functions.

[ ]:
```
filtered_neighbours = []
for n in g.nodes():
    if (len([x for x in nx.MultiGraph(g).neighbors(n) if (x in done_nodes)]) >␣
 ↪1) and (n not in done_nodes):
        filtered_neighbours.append(n)
```

[ ]:
```
network_neighbours_filtered_suid = cytoscape_utils.subnetwork_node_induced(
    nodes=nodes+filtered_neighbours,
    parent_suid=pss_network_suid,
    name="identified paths + 1st neighbours (filtered)",
)
```

[ ]:
```
p4c.layouts.layout_copycat(
    network_neighbours_suid,
    network_neighbours_filtered_suid
)
```

### 5.2 Saving the session

Save the Cytoscape session:

[ ]:
```
p4c.session.save_session(str(output_dir / f"PSS-DiNAR-JA-SA-ROS-{today}.cys"))
```

### 5.3 Exporting graph to file

NetworkX does not have the greatest export formats, so making a Pandas dataframe and saving that seems the best.

[ ]:
```
df = nx.to_pandas_edgelist(g)
df.head()
```

[ ]:
```
df.to_csv(output_dir / f"pss-dinar-refined-edgelist-{today}.tsv", sep="\t",␣
↪index=None)
```

#### 5.3.1 Exporting the subnetwork

[ ]:
```
g_subgraph_node_induced = nx.induced_subgraph(g, nodes)
print(f"\nNumber of nodes: {g_subgraph_node_induced.number_of_nodes()}\nNumber␣
↪of edges: {g_subgraph_node_induced.number_of_edges()}")
```

[ ]:
```
nx.to_pandas_edgelist(g_subgraph_node_induced)\
    .to_csv(output_dir / f"pss-dinar-subgraph-refined-edgelist-{today}.tsv",␣
↪sep="\t", index=None)
```

[ ]:
```

```

### 5.4 Exporting the Cytoscape networks to PDF

[ ]:
```
collection_suid = p4c.get_collection_suid(network_edge_induced_suid)
```

[ ]:
```
from skm_tools import cytoscape_pdf_utils
```

[ ]:
```
cytoscape_pdf_utils.export_collection_to_pdfs(collection_suid, output_dir /␣
↪"figures")
```

[ ]:
```
cytoscape_pdf_utils.export_collection_to_single_pdf(collection_suid, output_dir␣
↪/ "figures" / "single_pdf", caption=True)
```

[ ]:
```
# END
```

[ ]:

[ ]: