In [4]:

*# %%*

**import** **sys**

sys.path.append('../')

**from** **typing** **import** Final

*# from helpers.ploting import plotBargraph, plotSubBarGraphs*

**from** **helpers.graphs** **import** intersection\_of\_graphs

*# from helpers.operator\_actions import \_\_getTrueAndNuisanceSources, \_\_getFinalOperatorAlarmRelationGraph, getTrueAndNuisanceSourceNames*

**from** **helpers.alarms** **import** filterAlarmData, getDFWithCommonSourcesInAllMonths, loadAlarmsData, loadOperatorData

**from** **helpers** **import** alarms

**import** **time**

**from** **datetime** **import** datetime, timedelta

**from** **multiprocessing** **import** Pool

**import** **networkx** **as** **nx**

*# from multiprocessing import Pool*

**class** **Action2Alarm**: *# a class which relate operator actions to most suitable corresponding alarms*

**def** \_\_init\_\_(self):

*# print("class Created")*

**pass**

**def** \_\_checkAction\_OnAlarm(self, action, alarm,time\_to\_react): *# Ahmet Can time\_to\_react filter # Done*

**if** action["EventTime"] >= alarm["StartTime"] **and** action["EventTime"] <= alarm["EndTime"] **and** timedelta.total\_seconds(action["EventTime"]-alarm['StartTime']) <=time\_to\_react:

**return** **True**

**else**:

**return** **False**

**def** \_\_addOrUpdateEdge(self,g, e): *# e = (op, alarm) # Done*

**if** g.has\_edge(\*e) == **True**:

g.edges[e]["weight"] += 1

**else**:

g.add\_edge(\*e, weight=1)

**def** \_\_updateNode(self, g, node): *# using partial thats why node as last arg*

*# if operatorf == True:*

*# node = "Operator->" + node*

*# g.nodes[node]["size"] += 0.01*

g.nodes[node]["count"] += 1

**def** constructSingleAlarmsActionsGraph(self,args): *# Done # case 3*

df\_alarms = args[0] *# alarms*

df\_actions = args[1] *# operator actions*

time\_to\_react = args[2] *# time to react*

*# edge\_drop\_factor = args[3]*

g = nx.DiGraph() *# Directed graph*

*# ---------- Adding Nodes----------------*

alarm\_nodes = [alarm **for** alarm **in** df\_alarms['SourceName'].unique()]

action\_nodes = ["Operator->"+action **for** action **in** df\_actions['SourceName'].unique()]

\_ = [g.add\_node(node, size=0, count=0) **for** node **in** alarm\_nodes+action\_nodes]

\_ = [self.\_\_updateNode(g,node) **for** node **in** df\_alarms["SourceName"]]

\_ = [self.\_\_updateNode(g,"Operator->"+node) **for** node **in** df\_actions["SourceName"]]

*# -------- Adding Edges -----------------*

alarms = df\_alarms.to\_dict(orient="records")

actions = df\_actions.to\_dict(orient="records")

edges = [("Operator->"+action["SourceName"], alarm["SourceName"]) **for** action **in** actions **for** alarm **in** alarms **if** self.\_\_checkAction\_OnAlarm(action, alarm,time\_to\_react)]

\_ = [self.\_\_addOrUpdateEdge(g,e) **for** e **in** edges]

*# if edge\_drop\_factor > -1:*

*# g = filterEdges(g, edge\_drop\_factor=edge\_drop\_factor)*

**return** g

**def** \_\_constructMultipleAlarmsActionsGraphs(self,df\_alarms, df\_actions,time\_to\_react,edge\_drop\_factor):

*# graphs = [self.constructSingleAlarmsActionsGraph((df\_alarms[df\_alarms["Year-Month"].isin([month])],df\_actions[df\_actions["Year-Month"].isin([month])],time\_to\_react,edge\_drop\_factor)) for month in df\_alarms["Year-Month"].unique()]*

graphs = []

**for** month **in** df\_alarms["Year-Month"].unique():

inputs = (df\_alarms[df\_alarms["Year-Month"].isin([month])],df\_actions[df\_actions["Year-Month"].isin([month])],time\_to\_react,edge\_drop\_factor)

temp\_g = self.constructSingleAlarmsActionsGraph(inputs)

graphs.append(temp\_g)

**return** graphs

**def** getOperatorAlarmRelationGraph(self, df\_alarms, df\_actions, min\_graphs\_intersection\_filter,time\_to\_react,edge\_drop\_factor):

*# print(">> Finding relation between operator action and alarm")*

graphs = self.\_\_constructMultipleAlarmsActionsGraphs(

df\_alarms, df\_actions,time\_to\_react,edge\_drop\_factor)

**assert** len(graphs) == len(list(df\_alarms["Year-Month"].unique()))

*# print(f"\n\n\n\n>> Taking intersection of {len(graphs)} sub-graphs")*

G = intersection\_of\_graphs(graphs, min\_graphs\_intersection\_filter) *# min\_graphs\_intersection\_filter is the gamma filter from the paper*

**return** G

*# def printGraph(self,g):*

*# print(" ========================== start printing ================================")*

*# main\_g = g*

*# operator\_nodes = [action for action in g.nodes if action.find("Operator") != -1]*

*# for op in operator\_nodes:*

*# action\_count = main\_g.nodes[op]['count']*

*# num\_neg = len(list(main\_g.neighbors(op)))*

*# print(f"{op} Count:{action\_count}|| Correspodning Alarms = {num\_neg} ", end = " " )*

*# # s = ""*

*# for n in main\_g.neighbors(op):*

*# alarm\_count = main\_g.nodes[n]['count']*

*# edge\_weight = main\_g.edges[(op,n)]['weight']*

*# action\_alarms\_ratio = alarm\_count/action\_count*

*# print(f"{n}(count:{alarm\_count})(weight:{edge\_weight}), (ratio: {action\_alarms\_ratio})", end = ", ")*

*# print("\n ========================== end ================================")*

*# def \_\_getTrueAndNuisanceSources(self,df,g):*

*# source\_names\_requried\_action = [*

*# source for source in g.nodes if source.find("Operator") == -1]*

*# df\_temp = df[~df["SourceName"].isin(source\_names\_requried\_action)]*

*# sources\_not\_require\_action = df\_temp["SourceName"].unique()*

*# return source\_names\_requried\_action, sources\_not\_require\_action*

*# def getTrueAndNuisanceSourceNames(self, df\_alarms,df\_operator, num\_sub\_graphs, min\_intersection\_f,edge\_filter):*

*# sources\_need\_action, sources\_dont\_need\_action = self.\_\_getTrueAndNuisanceSources(df\_alarms, g=filterEdges(self.\_\_getFinalOperatorAlarmRelationGraph(df\_alarms=df\_alarms, df\_actions=df\_operator,*

*# num\_graphs=num\_sub\_graphs, min\_graphs\_intersection\_filter=min\_intersection\_f), edge\_drop\_factor=edge\_filter))*

*# return sources\_need\_action, sources\_dont\_need\_action*

*# def \_\_addOrUpdateEdge(self,g, e):*

*# if g.has\_edge(\*e) == True:*

*# g.edges[e]["weight"] += 1*

*# else:*

*# g.add\_edge(\*e, weight=1)*

*# return False*

In [5]:

*# /home/waris/Github/research/tupras-analysis/data/processed/alarms/final/final-all-months-alarms.csv*

*""" Lodading the Data and Preprocessing """*

PATH = "../../data/"

*# path = PATH + "/processed/alarms\_with\_feed/"*

*# PATH = "/home/waris/Github/research/tupras-analysis/data/"*

alarm\_file\_path = PATH + "processed/alarms\_with\_feed/final/final-all-months-alarms-feed.csv"

op\_action\_file\_path = PATH + "processed/operator-actions/final/final-all-month-actions.csv"

start = time.time()

df\_temp\_alarms\_main =loadAlarmsData(file\_path=alarm\_file\_path)

df\_temp\_actions = loadOperatorData(file\_path=op\_action\_file\_path)

*""" Common Sources in all months. Try it but can be skipped. """*

*# df\_main\_alarms = getDFWithCommonSourcesInAllMonths(df\_main\_alarms)*

*""" Chaning name 2 alias for alarm data but skipping it """*

*# source2Alias, alias2source = convertSourceNamesToAlias(df\_main\_alarms)*

print("Total Time to load the data ", time.time()-start)

df\_temp\_alarms\_main

df\_temp\_alarms\_main.info()

In [7]:

df\_temp\_alarms\_main.drop\_duplicates(keep=**False**,inplace=**True**)

df\_temp\_alarms\_main.info()

df\_temp\_actions.info()

In [8]:

df\_temp\_actions.drop\_duplicates(keep=**False**, inplace=**True**)

df\_temp\_actions

In [9]:

common\_df = df\_temp\_alarms\_main[df\_temp\_alarms\_main['SourceName'].isin(df\_temp\_actions['SourceName'].unique())]

common\_df\_sources = common\_df['SourceName'].unique()

common\_df\_sources

In [10]:

df\_temp\_alarms\_main[df\_temp\_alarms\_main['SourceName'].isin(['47PIC2038\_Y'])]

In [11]:

df\_temp\_actions[df\_temp\_actions['SourceName'].isin(['47PIC2038\_Y'])]

In [12]:

*"""*

*alarms -> operator*

*47TI1521 -> 47HIC1504, 47HIC1503*

*47TI1512C -> (47FIC017, wrong) (47TIC1511, possible correct)*

*47LIC1509 -> 47FIC1523, 47LIC1510, 47LIC1509*

*47LIC3408 -> 47LIC3404*

*47LIC1503 -> 47FIC1507*

*47TIC1511 -> 47FIC1502/3/4/5*

*47TI2058B Count:79|| Corresponding Actions = 13, => (\*\*47TI2058B, count:114,weight:43, alarm/action: 0.69, weight/count: 0.54)*

*47LIC1505 Count:216|| Corresponding Actions = 3, => (\*\*47LIC1505, count:1357,weight:266, alarm/action: 0.16, weight/count: 1.23)*

*47TI2057A Count:100|| Corresponding Actions = 3, => (\*\*47TI2057A, count:113,weight:45, alarm/action: 0.88, weight/count: 0.45)*

*47FIC3410 Count:288|| Corresponding Actions = 6, => (\*\*47FIC3410, count:528,weight:100, alarm/action: 0.55, weight/count: 0.35)*

*47TI2058A Count:49|| Corresponding Actions = 19, => (\*\*47TI2058A, count:60,weight:30, alarm/action: 0.82, weight/count: 0.61)*

*47FIC020 Count:104|| Corresponding Actions = 17, => (\*\*47FIC020, count:369,weight:162, alarm/action: 0.28, weight/count: 1.56)*

*47PIC027 Count:96|| Corresponding Actions = 1, => (\*\*47PIC027, count:4496,weight:342, alarm/action: 0.02, weight/count: 3.56)*

*47PIC1504 Count:59|| Corresponding Actions = 1, => (\*\*47PIC1504, count:476,weight:180, alarm/action: 0.12, weight/count: 3.05)*

*19LIC104 Count:157|| Corresponding Actions = 9, => (\*\*19LIC104, count:308,weight:102, alarm/action: 0.51, weight/count: 0.65)*

*48FFIC019 Count:85|| Corresponding Actions = 1, => (\*\*48FFIC019, count:591,weight:51, alarm/action: 0.14, weight/count: 0.6)*

*48LI015 Count:442|| Corresponding Actions = 8, => (\*\*48LI015, count:1457,weight:456, alarm/action: 0.3, weight/count: 1.03)*

*48LI011 Count:759|| Corresponding Actions = 26, => (\*\*48LI011, count:91,weight:56, alarm/action: 8.34, weight/count: 0.07)*

*47LIC002 Count:328|| Corresponding Actions = 5, => (\*\*47LIC002, count:565,weight:150, alarm/action: 0.58, weight/count: 0.46)*

*47LIC015 Count:132|| Corresponding Actions = 1, => (\*\*47LIC015, count:191,weight:73, alarm/action: 0.69, weight/count: 0.55)*

*47LIC1509 Count:335|| Corresponding Actions = 9, => (47FIC1523, count:781,weight:361, alarm/action: 0.43, weight/count: 1.08) (47LIC1510, count:448,weight:169, alarm/action: 0.75, weight/count: 0.5) (\*\*47LIC1509, count:750,weight:123, alarm/action: 0.45, weight/count: 0.37)*

*48TIC034 Count:1971|| Corresponding Actions = 22, => (47TIC937E, count:3867,weight:605, alarm/action: 0.51, weight/count: 0.31) (47TIC927E, count:2940,weight:531, alarm/action: 0.67, weight/count: 0.27) (47PIC3522, count:3165,weight:854, alarm/action: 0.62, weight/count: 0.43) (47TIC007, count:3026,weight:561, alarm/action: 0.65, weight/count: 0.28) (47TIC259, count:2908,weight:511, alarm/action: 0.68, weight/count: 0.26) (47LIC3103, count:2783,weight:970, alarm/action: 0.71, weight/count: 0.49) (48FIC007, count:877,weight:523, alarm/action: 2.25, weight/count: 0.27) (48TIC008, count:1429,weight:510, alarm/action: 1.38, weight/count: 0.26) (\*\*48TIC034, count:1872,weight:1567, alarm/action: 1.05, weight/count: 0.8) (47FIC3102, count:2280,weight:718, alarm/action: 0.86, weight/count: 0.36)*

*47PIC2033A Count:1374|| Corresponding Actions = 48, => (47FIC017, count:2156,weight:387, alarm/action: 0.64, weight/count: 0.28) (47FIC1520, count:2020,weight:432, alarm/action: 0.68, weight/count: 0.31) (47PIC2033B, count:2315,weight:456, alarm/action: 0.59, weight/count: 0.33) (47TIC3520, count:2745,weight:539, alarm/action: 0.5, weight/count: 0.39) (\*\*47PIC2033A, count:2410,weight:466, alarm/action: 0.57, weight/count: 0.34) (47TIC927E, count:2940,weight:585, alarm/action: 0.47, weight/count: 0.43) (47PIC3522, count:3165,weight:640, alarm/action: 0.43, weight/count: 0.47) (47TIC007, count:3026,weight:585, alarm/action: 0.45, weight/count: 0.43) (47TIC259, count:2908,weight:545, alarm/action: 0.47, weight/count: 0.4) (47HIC1502, count:1809,weight:362, alarm/action: 0.76, weight/count: 0.26) (47HIC1501, count:1773,weight:353, alarm/action: 0.77, weight/count: 0.26) (47FIC2002, count:2351,weight:487, alarm/action: 0.58, weight/count: 0.35) (47HIC2008, count:2991,weight:625, alarm/action: 0.46, weight/count: 0.45) (47LIC3103, count:2783,weight:674, alarm/action: 0.49, weight/count: 0.49) (47LIC3404, count:1674,weight:394, alarm/action: 0.82, weight/count: 0.29) (47FIC3102, count:2280,weight:542, alarm/action: 0.6, weight/count: 0.39)*

*47PIC3522 Count:1164|| Corresponding Actions = 35, => (47FIC017, count:2156,weight:307, alarm/action: 0.54, weight/count: 0.26) (47TIC3520, count:2745,weight:656, alarm/action: 0.42, weight/count: 0.56) (\*\*47PIC3522, count:3165,weight:1524, alarm/action: 0.37, weight/count: 1.31) (47LIC3404, count:1674,weight:534, alarm/action: 0.7, weight/count: 0.46) (47FIC3102, count:2280,weight:422, alarm/action: 0.51, weight/count: 0.36)*

*47FIC1535 Count:1227|| Corresponding Actions = 34, => (\*\*47FIC1535, count:4930,weight:3601, alarm/action: 0.25, weight/count: 2.93) (47FIC1534, count:1528,weight:959, alarm/action: 0.8, weight/count: 0.78) (47FIC013, count:1786,weight:491, alarm/action: 0.69, weight/count: 0.4) (47FIC1542, count:1838,weight:598, alarm/action: 0.67, weight/count: 0.49) (47TIC937E, count:1674,weight:508, alarm/action: 0.73, weight/count: 0.41) (47TIC274, count:2318,weight:779, alarm/action: 0.53, weight/count: 0.63) (47TIC3520, count:1666,weight:521, alarm/action: 0.74, weight/count: 0.42) (47TIC927E, count:1301,weight:409, alarm/action: 0.94, weight/count: 0.33) (47PIC3522, count:1788,weight:446, alarm/action: 0.69, weight/count: 0.36) (47TIC007, count:1278,weight:334, alarm/action: 0.96, weight/count: 0.27) (47TIC259, count:1306,weight:477, alarm/action: 0.94, weight/count: 0.39) (47FIC1563, count:2227,weight:796, alarm/action: 0.55, weight/count: 0.65) (47HIC2008, count:1521,weight:532, alarm/action: 0.81, weight/count: 0.43) (47HIC2004, count:1048,weight:357, alarm/action: 1.17, weight/count: 0.29) (47PIC1507, count:660,weight:481, alarm/action: 1.86, weight/count: 0.39)*

*47TIC3520 Count:2969|| Corresponding Actions = 37, => (47TIC274, count:5269,weight:783, alarm/action: 0.56, weight/count: 0.26) (\*\*47TIC3520, count:2745,weight:1515, alarm/action: 1.08, weight/count: 0.51) (47PIC3522, count:3165,weight:1071, alarm/action: 0.94, weight/count: 0.36) (47LIC3404, count:1674,weight:819, alarm/action: 1.77, weight/count: 0.28) (47PIC027, count:4674,weight:813, alarm/action: 0.64, weight/count: 0.27)*

*"""*

In [39]:

df\_temp\_alarms = df\_temp\_alarms\_main

*# df\_temp\_alarms = df\_temp\_alarms[df\_temp\_alarms["SourceName"].isin(["47TI1512C"])]*

*# df\_temp\_alarms = df\_temp\_alarms[df\_temp\_alarms["SourceName"].isin(["47TI1521","47TI1512C","47LIC3408","47LIC1509","47LIC1503", '47TIC1511'])]*

*# df\_temp\_actions = df\_temp\_actions[df\_temp\_actions["SourceName"].isin(["47FIC1502","47FIC1503","47FIC1504","47FIC1505","47FIC1523","47FIC1507"])]*

*# df\_temp\_actions = df\_temp\_actions[df\_temp\_actions["Year-Month"].isin(df\_temp\_alarms['Year-Month'].unique())]*

*#actions "47FIC1502","47FIC1503","47FIC1504","47FIC1505" #,"47FIC1523","47FIC1507"*

*# "47FIC1502","47FIC1503","47FIC1504","47FIC1505","47FIC1523"*

*#alarms "47TIC1511","47LIC1503",'47LIC1509'*

*# df\_temp\_alarms = df\_temp\_alarms.sort\_values(by='StartTime')*

*# df\_temp\_actions = df\_temp\_actions.sort\_values(by='EventTime')*

*# df\_temp\_actions.to\_csv("temp\_actions.csv", index = False)*

*# df\_temp\_alarms.to\_csv("temp\_alarms.csv", index = False)*

In [40]:

print("Alarms >",df\_temp\_alarms["SourceName"].value\_counts())

print("Actions >", df\_temp\_actions["SourceName"].value\_counts())

In [41]:

*# temp\_obj = Action2Alarm()*

*# test\_g = temp\_obj.constructSingleAlarmsActionsGraph((df\_temp\_alarms,df\_temp\_actions,0.5\*60\*60))*

*# main\_g = test\_g*

*# # %%*

*# operator\_nodes = [action for action in main\_g.nodes if action.find("Operator") != -1]*

*# dict\_to\_print = {}*

*# for op in operator\_nodes:*

*# action\_count = main\_g.nodes[op]['count']*

*# num\_neg = len(list(main\_g.neighbors(op)))*

*# s = ""*

*# for n in main\_g.neighbors(op):*

*# alarm\_count = main\_g.nodes[n]['count']*

*# edge\_weight = main\_g.edges[(op,n)]['weight']*

*# action\_alarms\_ratio = alarm\_count/action\_count*

*# s = s+ f" {n}(count:{alarm\_count})(weight:{edge\_weight}), "*

*# dict\_to\_print[op] = s*

*# for op in dict\_to\_print.keys():*

*# action\_count = main\_g.nodes[op]['count']*

*# num\_neg = len(list(main\_g.neighbors(op)))*

*# print(f"{op} Count:{action\_count}|| Correspodning Alarms = {num\_neg}, => {dict\_to\_print[op]} " )*

*# # temp\_obj.printGraph(main\_g)*

In [42]:

*# temp\_obj = Action2Alarm()*

*# # test\_g = temp\_obj.constructSingleAlarmsActionsGraph((df\_temp\_alarms,df\_temp\_actions,2.5\*60\*60))*

*# test\_g = temp\_obj.getOperatorAlarmRelationGraph(df\_alarms=df\_temp\_alarms, df\_actions=df\_temp\_actions, min\_graphs\_intersection\_filter=4, time\_to\_react=8\*60\*60)*

*# main\_g = test\_g*

*# operator\_nodes = [action for action in main\_g.nodes if action.find("Operator") != -1]*

*# dict\_to\_print = {}*

*# for op in operator\_nodes:*

*# action\_count = main\_g.nodes[op]['count']*

*# num\_neg = len(list(main\_g.neighbors(op)))*

*# # print(f"{op} Count:{action\_count}|| Correspodning Alarms = {num\_neg} ", end = " " )*

*# s = ""*

*# for n in main\_g.neighbors(op):*

*# alarm\_count = main\_g.nodes[n]['count']*

*# edge\_weight = main\_g.edges[(op,n)]['weight']*

*# action\_alarms\_ratio = alarm\_count/action\_count*

*# s = s+ f" {n}(count:{alarm\_count})(weight:{edge\_weight}), "*

*# dict\_to\_print[op] = s*

*# for op in dict\_to\_print.keys():*

*# action\_count = main\_g.nodes[op]['count']*

*# num\_neg = len(list(main\_g.neighbors(op)))*

*# print(f"{op} Count:{action\_count}|| Correspodning Alarms = {num\_neg}, => {dict\_to\_print[op]} " )*

In [43]:

**def** filterEdges(g, edge\_drop\_factor):

*# g = nx.DiGraph(g)*

remove\_edges = []

**for** source, target, weight **in** g.edges.data("weight"):

alarm\_count = g.nodes[target]["count"]

action\_count = g.nodes[source]["count"]

alarms2action\_ratio = alarm\_count/action\_count

weight\_alarm\_ratio = weight/alarm\_count

*# if edge\_drop\_factor \* weight < g.nodes[target]["count"]:*

**if** alarms2action\_ratio < 0.20:

remove\_edges.append((source, target))

**elif** alarms2action\_ratio <0.10 **and** weight\_alarm\_ratio < 0.25:

remove\_edges.append((source, target))

g.remove\_edges\_from(remove\_edges)

g.remove\_nodes\_from(list(nx.isolates(g)))

**return** g

**def** foo1(args):

df\_alarms = args[0]

df\_actions =args[1]

min\_intersect = args[2]

time\_to\_react = args[3]

edge\_drop\_factor = args[4]

actionAndAlarms = []

action2alarm = Action2Alarm()

*# print(f'Months in Alarms = {df\_alarms["Year-Month"].unique()} \n Months in Actions = {df\_actions["Year-Month"].unique()}')*

main\_g = action2alarm.getOperatorAlarmRelationGraph(df\_alarms=df\_alarms, df\_actions=df\_actions, min\_graphs\_intersection\_filter=min\_intersect, time\_to\_react=time\_to\_react,edge\_drop\_factor=edge\_drop\_factor)

*# %%*

*# operator\_nodes = [action for action in main\_g.nodes if action.find("Operator") != -1]*

alarm\_nodes = [alarm **for** alarm **in** main\_g.nodes **if** alarm.find("Operator") == -1]

*# print(f">> # of Edges Main Graph {main\_g.number\_of\_edges()}")*

*# print(f">> Total number of operator Tags in the graph={len(operator\_nodes)}")*

*# print(f">> Total number of Alarm Tags in the graph = {len(main\_g.nodes)- len(operator\_nodes)}")*

dict\_to\_print = {}

**for** node **in** alarm\_nodes:

alarm\_count = main\_g.nodes[node]['count']

num\_neg = len(list(main\_g.predecessors(node)))

*# print(f"{op} Count:{action\_count}|| Correspodning Alarms = {num\_neg} ", end = " " )*

s = ""

**for** op\_node **in** main\_g.predecessors(node):

*# print(op\_node)*

act\_count = main\_g.nodes[op\_node]['count']

edge\_weight = main\_g.edges[(op\_node,node)]['weight']

action\_alarms\_ratio = round(alarm\_count/act\_count,2)

weight\_alarm\_ratio = round(edge\_weight/alarm\_count,2)

*# s = s+ f" {n}(count:{alarm\_count})(weight:{edge\_weight}), (alarm/action: {action\_alarms\_ratio}), (alarm/weight: {action\_weight\_ratio}) "*

*# dict\_to\_print[op] = s*

op\_node = op\_node.replace("Operator->","")

**if** op\_node == node:

s = s+ f"(\*\***{**op\_node**}**, count:**{**act\_count**}**,weight:**{**edge\_weight**}**, alarm/action: **{**action\_alarms\_ratio**}**, weight/alarm: **{**weight\_alarm\_ratio**}**) "

dict\_to\_print[node] = s

**elif** action\_alarms\_ratio > 0.4 **and** weight\_alarm\_ratio>0.25:

s = s+ f"(2-**{**op\_node**}**, count:**{**act\_count**}**,weight:**{**edge\_weight**}**, alarm/action: **{**action\_alarms\_ratio**}**, weight/alarm: **{**weight\_alarm\_ratio**}**) "

dict\_to\_print[node] = s

**elif** action\_alarms\_ratio>0.3 **and** weight\_alarm\_ratio > 0.8: *# these ratios are not finalized and final ratios will depend on the Ahmet Can feedback*

s = s+ f"(3-**{**op\_node**}**, count:**{**act\_count**}**,weight:**{**edge\_weight**}**, alarm/action: **{**action\_alarms\_ratio**}**, weight/alarm: **{**weight\_alarm\_ratio**}**) "

dict\_to\_print[node] = s

*# elif action\_alarms\_ratio >0.5 and action\_alarms\_ratio <=1:*

*# print(f"{n}(count:{alarm\_count})(weight:{edge\_weight}), (ratio: {action\_alarms\_ratio})", end = ", ")*

**for** alarm\_node **in** dict\_to\_print.keys():

alarm\_count = main\_g.nodes[alarm\_node]['count']

num\_neg = len(list(main\_g.predecessors(alarm\_node)))

s = f"**{**alarm\_node**}** Count:**{**alarm\_count**}**|| Corresponding Actions = **{**num\_neg**}**, => **{**dict\_to\_print[alarm\_node]**}** "

print(s)

actionAndAlarms.append(s)

**return** actionAndAlarms

In [44]:

print(f"Total Alarms: **{**df\_temp\_alarms.shape**}**, Total Operator Action: **{**df\_temp\_actions.shape**}**, Total Sources: **{**len(df\_temp\_alarms['SourceName'].unique())**}** ")

*# %%*

*"""*

*Filter the Alarm Data*

*1. Ignore the communication Alarms*

*2. Ignore the momentary alarms => 20 seconds*

*3. Remove Staling Alarms => 12 hours*

*4. Remove sources which are triggered less 20 in whole dataset*

*5. Include all the months*

*6. DO SKIP ANY SOURCENAME IF IGNORING COMMUNICATION ALARMS*

*"""*

ignore\_comm\_alarms :Final = **True**

momentary\_alarms\_f:Final = 2\*60 *# seconds*

staling\_alarm\_f:Final = (60\*60) \* 4 *# hours*

min\_alarms\_per\_source\_f:Final = 40 *# any source which is not triggered atleast 20 times in whole dataset will be removed*

months\_f:Final = df\_temp\_alarms["Year-Month"].unique()

print(f">> Unique Year-Month **{**months\_f**}**")

snames\_f:Final = [] *# ONLY USE IF NOT IGNORING COMM ALRMS*

df\_alarms\_new = filterAlarmData(df\_temp\_alarms, months=months\_f, sources\_filter=snames\_f,

monmentarly\_filter=momentary\_alarms\_f, staling\_filter=staling\_alarm\_f, ingore\_communication\_alarms=ignore\_comm\_alarms, min\_alarms\_per\_source=min\_alarms\_per\_source\_f)

df\_actions\_new = df\_temp\_actions[df\_temp\_actions["Year-Month"].isin(months\_f)]

sname2months = {}

**for** sname **in** df\_alarms\_new["SourceName"].unique():

sname2months[sname] = df\_alarms\_new[df\_alarms\_new["SourceName"].isin([sname])]["Year-Month"].unique()

monthCount2Snames = {}

**for** key **in** range(len(list(df\_alarms\_new["Year-Month"].unique()))):

monthCount2Snames[key+1] = []

**for** sname **in** sname2months.keys():

monthCount2Snames[len(sname2months[sname])].append(sname)

print(f">> Month Count 2 alarms: **{**monthCount2Snames**}**" )

*"""*

*For Opertartor Graphs*

*1. Construct sub-graphs equal to number of months*

*2. Min intersections is sub-graphs/2 + 1*

*3. Remove edges which are not significantaly contributing to nodes count*

*It will vary in op actions and in grouping. In op actions*

*it will be higher but in groupign it will be lower.*

*"""*

*# num\_sub\_graphs:Final = len(months\_f)*

*# min\_intersection\_f:Final = 2*

*# edge\_filter:Final = 1 # BE CAREFUL OVER HERE*

print("Alarms >",df\_alarms\_new["SourceName"].value\_counts())

print("Actions >", df\_actions\_new["SourceName"].value\_counts())

actionAndAlarms =[]

*# action2alarm = Action2Alarm()*

**for** sname **in** df\_alarms\_new['SourceName'].unique():

mc = len(sname2months[sname])

intersect\_threshold = mc-1 *# gamma*

**if** mc <=5:

**continue**

*# print(f">> Source Name {sname}, Alarm Appear in {mc} months, Intersection Threshold ={intersect\_threshold}" )*

df1 = df\_alarms\_new[df\_alarms\_new['SourceName'].isin([sname])]

df2 = df\_actions\_new[df\_actions\_new['Year-Month'].isin(list(df1['Year-Month'].unique()))]

actionAndAlarms += foo1((df1,df2,intersect\_threshold,30\*60\*60, -1))

*# endAlarmTime should be around 1 hour*

*# for aa in actionAndAlarms:*

*# print(f"++{aa}")*

*# foo1((df\_temp\_alarms,df\_temp\_actions,6,4\*60\*60))*

In [45]:

*"""*

*47TI1512C -> 47FIC017, 47FIC013, 47TIC937E, 47TIC274, 47TIC927E, 47TIC007, 47TIC259, 47FIC1563*

*"""*

*"""*

*alarms -> operator*

*47TI1521 -> 47HIC1504, 47HIC1503*

*47TI1512C -> (47FIC017, wrong) (47TIC1511, possible correct)*

*47LIC1509 -> 47FIC1523, 47LIC1510, 47LIC1509*

*47LIC3408 -> 47LIC3404*

*47LIC1503 -> 47FIC1507*

*47TIC1511 -> 47FIC1502/3/4/5*

*47TI1521 Count:397|| Correspodning Actions = 16, => (47HIC1504, count:910,weight:321, alarm/action: 0.44, weight/count: 0.81) (47HIC1503, count:900,weight:331, alarm/action: 0.44, weight/count: 0.83) correct*

*47TI1512C Count:1091|| Correspodning Actions = 8, => (47FIC017, count:2156,weight:56, alarm/action: 0.51, weight/count: 0.05) wrong one, possibly correct: 47TIC1511*

*47LIC1509 Count:335|| Correspodning Actions = 9, => (47FIC1523, count:781,weight:361, alarm/action: 0.43, weight/count: 1.08) (47LIC1510, count:448,weight:169, alarm/action: 0.75, weight/count: 0.5) (47LIC1509, count:750,weight:123, alarm/action: 0.45, weight/count: 0.37) almost correct*

*47LIC3408 Count:880|| Correspodning Actions = 3, => (47LIC3404, count:1674,weight:846, alarm/action: 0.53, weight/count: 0.96) correct*

*"""*

In [46]:

*# for aa in actionAndAlarms:*

*# print(aa)*

**with** open("output.txt",'w') **as** f:

**for** aa **in** actionAndAlarms:

f.write(aa)

f.write("**\n**")

*# f.writelines(actionAndAlarms)*