In [1]:

```
#importing all the necessary lib
import os
import numpy as np
import pandas as pd
#import graph as gra # http://www.python-course.eu/graphs_python.php
import call_graph
from csv import writer
from multiprocessing import Pool,Process
import graph as gra
from tqdm import tqdm
from glob import glob
```

In [2]:

```
#defining the opcodes and blocks for call graph features
opcodes = ['call','int','ja','jb','jc','je','jg','jge','jl','jle','jmp','jna','jnb','jnl','
blocks = ['sub_', 'loc_', 'locret_']
```

In [3]:

```
call_opcodes = ['call','int']
call_blocks = ['sub_']
```

In [4]:

```
#taken from https://github.com/dchad/malware-detection
def construct_call_graph(lines):
    vertex = '.program_entry_point' # this is the root node, corresponds to the program ent
    vertex_count = 1
    edge_count = 0
    cfgraph = gra.Graph()
    cfgraph.add_vertex(vertex)

    for row in lines:
        row = row.rstrip('\r\n')  # get rid of newlines they are annoying.
        if ';' in row:
            row = row.split(';')[0] # get rid of comments they are annoying.
        #print(row)

      # Cleaning of data.
        row = row.replace('short','').replace('ds:',' ')
        row = row.replace('dword','').replace('near','')
        row = row.replace('ptr','').replace(':',' ').replace(',',' ') #.replace('??',' ')
        row = row.replace('@','').replace('?','')
        parts = row.split() # tokenize code line

        if (len(parts) < 4): # this is a comment line
            continue

        if (parts[3] == 'endp'): # ignore subroutine end labels
            continue

      # check for subroutines and block labels
      # block and subroutine labels are always after the .text XXXXXXXX relative address
        for block in call_blocks:
            token = parts[2]
            idx = token.find(block)
            if ((idx == 0) or (parts[3] == 'proc')):
              # add new vertex to the graph, we are now in a new subroutine
                vertex = token
                cfgraph.add_vertex(vertex)
              # print("Vertex: " + vertex)
                vertex_count += 1
                break

      # now check for edge opcode
        for opcode in call_opcodes: # check the line for a new edge
            if opcode in parts:
              # Extract desination address/function name/interrupt number as the directed e
                idx = parts.index(opcode)
                edge_count += 1
                if ((idx + 1) < len(parts)): # in a few ASM files there is no operand, disa
                    next_vertex = parts[idx + 1]
                else:
                    next_vertex = "none"
                cfgraph.add_edge(vertex, next_vertex)
              # print("Edge: " + vertex + " " + parts[idx] + " " + edge)
                break

    # print("Vertex Count: {:d}".format(vertex_count))

    return cfgraph
```

In [5]:

```
def extract_call_graphs(tfiles):
    """Function to extract call graph features"""
    asm_files = [i for i in tfiles if '.asm' in i]
    ftot = len(asm_files)

    feature_file = 'data/'+ '-malware-call-graph-features.csv'
    print('Graph Feature file:', feature_file)

    graph_lines = []
    graph_features = []
    graph_file = open('data/' + '-malware-call-graphs.gv', 'w') # write as a graphviz DOT f
    with open(feature_file, 'w') as f:
        # write the column names for the csv file
        fw = writer(f)
        #colnames = ['filename','vertex_count','edge_count','delta_max','density','diameter
        colnames = ['filename','vertex_count','edge_count','delta_max','density']
        fw.writerow(colnames)

        # Now iterate through the file list and extract the call graph from each file.
        for idx, fname in enumerate(tqdm(asm_files)):
            fasm = open(ext_drive + fname, 'r', errors='ignore')

            lines = fasm.readlines()

            call_graph = construct_call_graph(lines)
            cgvc = call_graph.n_vertices()
            cgec = call_graph.n_edges()
            cgdm = call_graph.delta_max()
            cgde = call_graph.density()
            # cdia = call_graph.diameter() this is constantly problematic !!!
            graph_features.append([fname[:fname.find('.asm')]] + [cgvc, cgec, cgdm, cgde])
            call_graph.set_graph_name(fname[:fname.find('.asm')])
            graph_lines.append(call_graph.to_str('singlenoleaf'))

            del(call_graph) # for some reason new graphs get appended to the previous graph

            # Print progress
            if (idx + 1) % 10 == 0:
                fw.writerows(graph_features)
                graph_file.writelines(graph_lines)
                graph_features = []
                graph_lines = []

        # Write remaining files
        if len(graph_lines) > 0:
            fw.writerows(graph_features)
            graph_file.writelines(graph_lines)
            graph_features = []
            graph_lines = []

    graph_file.close()
```

In [6]:

```
def main():
"""Function to perform multiprocessing"""
    ext_drive = 'C:/Users/Anirban/Desktop/MAchine Learning/chap43_microsoft_malware_detecti
# multiprocessing using 11 cores
    tfiles = os.listdir(ext_drive)
    quart = int(len(tfiles)/11)
    train1 = tfiles[:quart]
    train2 = tfiles[quart:(2*quart)]
    train3 = tfiles[(2*quart):(3*quart)]
    train4 = tfiles[(3*quart):(4*quart)]
    train5 = tfiles[(4*quart):(5*quart)]
    train6 = tfiles[(5*quart):(6*quart)]
    train7 = tfiles[(6*quart):(7*quart)]
    train8 = tfiles[(7*quart):(8*quart)]
    train9 = tfiles[(8*quart):(9*quart)]
    train10 = tfiles[(9*quart):(10*quart)]
    train11 = tfiles[(10*quart):]
    print(len(tfiles), quart)
    trains = [train1, train2, train3, train4,train5,train6,train7,train8,train9,train10,tra
    p = Pool(11)
    p.map(call_graph.extract_call_graphs, trains)

if __name__=="__main__":
    main()
```

```
10868 988
```

In [17]:

```
#combining all the call graph features obtained usng multiprocessing into a single csv file
df_call_graph = pd.concat( [ pd.read_csv(csv) for csv in glob('data/'+'*.csv') ] )
df_call_graph.head()
```

Out[17]:

	filename	vertex_count	edge_count	delta_max	density
0	8i6m0aVAwsdSY2FEfU59	115	100	33	0.201613
1	8IbD9QgPs01NR7STfAxH	222	307	126	0.097152
2	8iBGtATMYPI0cqpVC2d5	5285	5455	103	0.006000
3	8iDWJ4yKzNSAQjnxwO70	122	119	3	0.138211
4	8inLjyQfkReMHmNUE4qg	194	132	25	0.053119

In [16]:

```
df_call_graph.to_csv('final_call_graph_features.csv') # saving the dataframe into a csv  fi
```