Double-click (or enter) to edit

```
1 from google.colab import drive
2 drive.mount('/content/drive/MyDrive/VC')
```

```
1 !unzip alib-master.zip
2 !pip install gurobipy
3 !pip install unidecode
```

```
Archive:  alib-master.zip
cac9ab53e74ca981d324cc6a622dc29ac04feaf0
replace alib-master/.gitignore? [y]es, [n]o, [A]ll, [N]one, [r]ename: Y
  inflating: alib-master/.gitignore
replace alib-master/LICENSE? [y]es, [n]o, [A]ll, [N]one, [r]ename: A
  inflating: alib-master/LICENSE
  inflating: alib-master/README.md
 extracting: alib-master/alib/__init__.py
  inflating: alib-master/alib/cli.md
  inflating: alib-master/alib/cli.py
  inflating: alib-master/alib/data/conversion.log
  inflating: alib-master/alib/data/convert_topology_zoo_gml_to_yml.sh
  inflating: alib-master/alib/data/topologyZoo/Aarnet.yml
  inflating: alib-master/alib/data/topologyZoo/Abilene.yml
  inflating: alib-master/alib/data/topologyZoo/Abvt.yml
  inflating: alib-master/alib/data/topologyZoo/Agis.yml
  inflating: alib-master/alib/data/topologyZoo/Airtel.yml
  inflating: alib-master/alib/data/topologyZoo/Amres.yml
  inflating: alib-master/alib/data/topologyZoo/Ans.yml
  inflating: alib-master/alib/data/topologyZoo/Arn.yml
  inflating: alib-master/alib/data/topologyZoo/Arnes.yml
  inflating: alib-master/alib/data/topologyZoo/Arpanet196912.yml
  inflating: alib-master/alib/data/topologyZoo/Arpanet19706.yml
  inflating: alib-master/alib/data/topologyZoo/Arpanet19719.yml
  inflating: alib-master/alib/data/topologyZoo/Arpanet19723.yml
  inflating: alib-master/alib/data/topologyZoo/Arpanet19728.yml
  inflating: alib-master/alib/data/topologyZoo/Atmnet.yml
  inflating: alib-master/alib/data/topologyZoo/AttMpls.yml
  inflating: alib-master/alib/data/topologyZoo/Bandcon.yml
  inflating: alib-master/alib/data/topologyZoo/Basnet.yml
  inflating: alib-master/alib/data/topologyZoo/Bbnplanet.yml
  inflating: alib-master/alib/data/topologyZoo/Bellcanada.yml
  inflating: alib-master/alib/data/topologyZoo/Bellsouth.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2003.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2004.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2005.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2006.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2007.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2008.yml
  inflating: alib-master/alib/data/topologyZoo/Belnet2009.yml
  inflating: alib-master/alib/data/topologyZoo/BeyondTheNetwork.yml
  inflating: alib-master/alib/data/topologyZoo/Bics.yml
```

✓ 0s completed at 9:34 PM

```
inflating: alib-master/alib/data/topologyZoo/BtAsiaPac.yml
inflating: alib-master/alib/data/topologyZoo/BtEurope.yml
inflating: alib-master/alib/data/topologyZoo/BtNorthAmerica.yml
inflating: alib-master/alib/data/topologyZoo/Canerie.yml
inflating: alib-master/alib/data/topologyZoo/Carnet.yml
inflating: alib-master/alib/data/topologyZoo/Cernet.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet1993.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet1999.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet2001.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet200304.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet200511.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet200603.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet200706.yml
inflating: alib-master/alib/data/topologyZoo/Cesnet201006.yml
```

!ls

```
#readpickle.py
import pickle
with open('input.pickle', 'rb') as handle:
    b = pickle.load(handle)

sn_graph=b.get("substrate")
nodes_sn_graph=sn_graph.nodes
print(nodes_sn_graph)

SN_node_CRB=sn_graph.node_weights
SN_edge_BW=sn_graph.edge_weights
print(SN_node_CRB)
print(SN_edge_BW)
```

```
84
{0: 20384, 1: 127021, 2: 297374, 3: 140702, 4: 142586, 5: 359681, 6: 497295, 7: 23392
{('22', '18'): 355087, ('18', '22'): 355087, ('3', '2'): 20834, ('2', '3'): 20834, ('
```

```
1
```

```
Mounted at /content/drive
```

```
#create vne
import networkx as nx
import random
import graph
from graph import Parameters
import numpy as np


def create_vne(min_nodes=2, max_nodes=10, no_requests=1, probability=0.4):
```

```
    random_node_list_arr = np.random.uniform(min_nodes, max_nodes, no_requests)
    random_node_list = [int(i) for i in random_node_list_arr]
    new_vne_req = []
    for req in random_node_list:
        G = nx.erdos_renyi_graph(req, probability, directed=False)
        ng = nx.to_dict_of_lists(G)
        g = {}
        for i in ng:
            g[i + 1] = []
            for j in ng[i]:
                g[i + 1].append(j + 1)

        if not nx.is_connected(G):
            null_node_list = [key for key, val in g.items() if not val]
            graph_node_count = {_key: len(_val) for _key, _val in g.items()}
            sorted_dict_list = sorted(
                graph_node_count.items(), key=lambda x: x[1], reverse=True
            )
            if len(null_node_list) != len(g):
                for index, empty_node in enumerate(null_node_list):
                    g[sorted_dict_list[index][0]].append(empty_node)
                    g[empty_node].append(sorted_dict_list[index][0])
            else:
                for i in range(len(g)):
                    for j in range(len(g) - i - 1):
                        if null_node_list[j + 1] not in g[null_node_list[j]]:
                            g[null_node_list[j]].append(null_node_list[j + 1])
                        if null_node_list[j] not in g[null_node_list[j + 1]]:
                            g[null_node_list[j + 1]].append(null_node_list[j])
        new_vne_req.append(g)

    # print("new VNE REQ is",new_vne_req)
    vne = []
    for i in range(len(new_vne_req)):
        edges = set()
        nodes = len(new_vne_req[i])
        for j in range(nodes):
            for k in new_vne_req[i][j + 1]:
                edges.add((str(j), str(k - 1)))
        vne.append(graph.Graph(nodes, edges, Parameters(1, 10, 1, 10, 0, 100, 0, 100, 1,
    #print (vne)
    return vne


if __name__ == "__main__":
    my_vne=create_vne(3,3,1,0.5)
    print("new VNE REQ is",my_vne[0].neighbours)

```

```
new VNE REQ is {0: {'1', '2'}, 1: {'0'}, 2: {'0'}}
```

```
'''
Placing a VNE initially using greedy approach but on best fit. Then we shall increase th
of these node and then run genetic algorithm on it
'''
from networkx.algorithms.summarization import snap_aggregation
t=list(SN_node_CRB.keys())
t.sort(key=lambda x:SN_node_CRB[x],reverse=True)
req=my_vne[0].node_weights
print(req)
vne=list(req.keys())
vne.sort(key=lambda x:req[1],reverse=True)
print(vne)
assign={}
for i in range(len(vne)):
  assign[vne[i]]=t[i+20]
print(assign)


```

```
{0: 2, 1: 2, 2: 5}
[0, 1, 2]
{0: 5, 1: 39, 2: 52}
```

```
def revenue(vnr):
  # sum of bw requirements and compute requirements
  bwSum, crSum = 0, 0
  vnr_edge_bw = vnr.edge_weights
  vnr_crb = vnr.node_weights

  for edge in vnr_edge_bw:
    bwSum += vnr_edge_bw[edge]

  for node in vnr_crb:
    crSum += vnr_crb[node]

  return bwSum


```

```
1
```

```
1
```

```
def vnr_ga(D,G,R):
  if fitness(D,G,R):return
  t=generate_solution_components(D)
  best=best_reconfig(G,R)
  for i in range(len(D)):
```

```
    assign[i]=best[i]

def best_reconfig(R,G):
  P=initial_population_selection(G,R,N)
  for i in range(NMAX):
    for j in range(N):
      for k in range(N):
        if j==k:continue
        P=P.append(cross_over(S[j],S[k]))
    if i%f:
      for j in range(N):
        P=P.append(mutate(S[j]))
      P=population_selection(P,N)
  return P[0]
def fitness(D,G,P):
  cnt=0
  ans=[]
  for key in assign:
    assign[i]=t[cnt]
    ans.append(t[cnt])
    cnt+=1
  print("After migration current mapped ones using vnr-ga")
  print(ans)
  return True


def hamming_distance(x,y):
  '''
  Hamming_distance func takes two parameters x,y
  Param x:Tuple of length =n
  Param y:Tuple of length =n
  returns hammind distance
  '''
  for i,j in zip(x,y):
    count=0
    if i!=j:count+=1
  return count


def cxOnePoint(ind1, ind2,ind):
    """Executes a one point crossover on the input :term:`sequence` individuals.
    The two individuals are modified in place. The resulting individuals will
    respectively have the length of the other.
    :param ind1: The first individual participating in the crossover.
    :param ind2: The second individual participating in the crossover.
    :param ind: index at we have to perform crossover between two sequences
    :returns: A tuple of two individuals.
    """
    size = min(len(ind1), len(ind2))
    cxpoint =ind
    ind1[cxpoint:], ind2[cxpoint:] = ind2[cxpoint:], ind1[cxpoint:]
```

```
    ind1[cxpoint:], ind2[cxpoint:] = ind2[cxpoint:], ind1[cxpoint:]

    return ind1, ind2
def effective_distance(A,B):
  f=A.index(1)
  l=B[::-1].index(1)
  l=len(B)-l-1
  return abs(f-l)

def cross_over(A,B):
  ind=effective_distance(A,B)
  dis=hamming_distance(A,B,ind)
  if dis>=2:
    return cxOnePoint(A,B)
  else: return None
def mutate(x):
  for i in len(x):
    x[i]=1-x[i]
'''
new_requirements is a list which has increasing demands than the assigned server weight
Now this should be reocnfigured using vnr_ga algorithm
'''
new_requirements=[t[assign[i]]+1 for i in assign.keys()]
#perform vnr_ga algorithm
vnr_ga(sn_graph,req,new_requirements)

```

```
After migration current mapped ones using vnr-ga
[6, 73, 22]
```

```
1
```