In [5]:

```
import pickle
db_class=open('C:\\Users\\HP\\OneDrive\\Desktop\\sem4\\NS\\project\\data4\\carts','rb')              #loa
ding carts
carts=pickle.load(db_class)
```

In [6]:

```
db_class1=open('C:\\Users\\HP\\OneDrive\\Desktop\\sem4\\NS\\project\\data4\\f','rb')              #loadin
g f values
f=pickle.load(db_class1)
```

In [7]:

```
db_class2=open('C:\\Users\\HP\\OneDrive\\Desktop\\sem4\\NS\\project\\data4\\weights','rb')              #
loading weights
weights=pickle.load(db_class2)
```

In [8]:

```
db_class3=open('C:\\Users\\HP\\OneDrive\\Desktop\\sem4\\NS\\project\\data4\\g','rb')              #loadin
g g
g=pickle.load(db_class3)
# print(g)
```

In [9]:

```
db_class4=open('C:\\Users\\HP\\OneDrive\\Desktop\\sem4\\NS\\project\\data4\\prices','rb')              #l
oading prices
prices=pickle.load(db_class4)
# print(prices)
```

In [10]:

```
# assigning delta price change to each node randomly
import random
price_change_pos={}
price_change_neg={}
for i in range(500):
    p=random.uniform(0, 0.07)
    price_change_pos[i]=p
    price_change_neg[i]=(p*-1)
```

In [11]:

```
max_rev=0
max_rev_u=0
max_rev_v=0
node_list=[]
total_nodes=[]

for i in range(500):
    total_nodes.append(i)
total_nodes=set(total_nodes)
```

In [12]:

```
# function to create edge list corresponding to a cart
def make_edge_list(cart):
    edge_list={}
    for i in range(500):
        edge_list[i]=[]

    for i in cart:
        for j in range(len(i)-1):
            for k in (j+1,len(i)):
                if k not in edge_list[j]:
                    edge_list[j].append(k)
                if j not in edge_list[k]:
                    edge_list[k].append(j)
    return edge_list
```

In [14]:

```
#function to calculate the updated revenue after changing price of a node
def calculate_max_revenue(price_new_u,price_new_v,f_new_u,f_new_v,g_new,edge_list1,prices1,f1,g1,u1,v1,
revenue_dict1):
    calc=0
    prices1[u1]=price_new_u
    prices1[v1]=price_new_v
    f1[u1]=f_new_u
    f1[v1]=f_new_v
    if u1<v1:
        g1[(u1,v1)]=g_new
    else:
        g1[(v1,u1)]=g_new

    revenue_dict1[u1]=0
    for i in edge_list1[u1]:
        if i <u1:
            calc= calc+(prices1[u1]*(f1[u1]+g1[(i,u1)]))
        else:
            calc= calc+(prices1[u1]*(f1[u1]+g1[(u1,i)]))
    revenue_dict1[u1]=calc

    total_c=sum(revenue_dict1.values())
    return total_c
```

In [15]:

```
#function to count the numebr of triangles to break the tie
def triangle_count(edge_list1,u1,v1):
    lst1=edge_list1[u1]
    lst2=edge_list1[v1]
    s=list(set(lst1) & set(lst2))
    return(len(s))
```

In [16]:

```
#strategy 3 implementation
# this hurestic finds maximum revenue when price of u and v is changed (when u-v have an edge in commo
n)
def strategy3(cart,edge_list,f,g,revenue_dict):

    visited_edges=[]   # to keep track of traversed edges of a graph
    node_list=[]       # to keep track of nodes with price change
    node_revenue={}    # to keep track of revenue
    while(len(node_list)<=100):
        max_rev=0

        for u in edge_list:
            price_new_u= prices[u]*(1+price_change_pos[u])   #calculating new price
            f_new_u= f[u]*(1-price_change_pos[u])            # calculating new f
            for v in edge_list[u]:
                if (u,v) not in visited_edges:
                    price_new_v=prices[v]*(1+price_change_neg[v])      #calculating new price
                    f_new_v= f[v]*(1-price_change_neg[v])               #calculating new f
                    if u<v:
                        g_new= g[u,v]*(1-weights[u]*price_change_pos[u]+weights[v]*price_change_neg[v])
 #calculating new g
                    else:
                        g_new= g[v,u]*(1-weights[u]*price_change_pos[u]+weights[v]*price_change_neg[v])
  #calculating new g
                    calc= calculate_max_revenue(price_new_u,price_new_v,f_new_u,f_new_v,g_new,edge_list
,prices,f,g,u,v,revenue_dict)
                    if calc==max_rev:
                        if max_rev_u != max_rev_v:
                            t1=triangle_count(edge_list,max_rev_u,max_rev_v)    #calculating number of
 triangles
                            t2=triangle_count(edge_list,u,v)
                            if t2>=t1:
                                max_rev= calc
                                max_rev_u= u
                                max_rev_v= v
                    elif calc > max_rev:            # finding max revenue
                        max_rev= calc
                        max_rev_u= u
                        max_rev_v= v

        if max_rev_u not in node_list:            #updating dictionaries based on new values
            node_list.append(max_rev_u)
        if max_rev_v not in node_list:
            node_list.append(max_rev_v)
        prices[max_rev_u]= price_new_u
        prices[max_rev_v]= price_new_v
        f[max_rev_u]= f_new_u
        f[max_rev_v]= f_new_v
        visited_edges.append((max_rev_u,max_rev_v))
        visited_edges.append((max_rev_v,max_rev_u))
        node_revenue[len(node_list)]= max_rev

    return(node_revenue)                          #returning max revenues
```

In [18]:

```
node_revenue_dict={}         #contains all the revenue dictionaries
revenue_dict={}
for i in carts:
    edge_list= make_edge_list(carts[i])
    for product in range(500):
        revenue_dict[product]=0
        revenue_dict[product]+=prices[product]*f[i][product]
        for neighbour in edge_list[product]:
            if(product<neighbour):
                revenue_dict[product]+=prices[product]*g[i][(product,neighbour)]
            else:
                revenue_dict[product]+=prices[product]*g[i][(neighbour,product)]    #generating revenue
 dictionary
    d= strategy3(carts[i],edge_list,f[i],g[i],revenue_dict)
    node_revenue_dict[i]=d
```

In [19]:

```
import pickle
db_class=open('result','wb')         #creating output revenue pickle
pickle.dump(node_revenue_dict,db_class)
db_class.close()
```

In [ ]: