Problem Set #4

Saharnaz Babaei Balderlou
saharnaz.babaei@grad.moore.sc.edu

Topics in Microeconomics; Computational Methods; Econ 815 — October 14, 2019

1 MSE

1.1 Interpretation

In this problem, the relationship between an array of explanatory variables and the value of radio station mergers is assessed. For this purpose, we use maximum score estimation method (MSE) to obtain the impact of number of stations owned by the parent company of the buyer, population in the range of target in market m, an indicator of corporate ownership, distance, and HHI measure of market concentration on the payoff to the merger between radio station buyer b and target t in market m. To do so, we use two indicator functions as:

$$\hat{\beta} = \arg\max Q(\beta) = \sum_{y=1}^{Y} \sum_{b=1}^{M_y - 1} \sum_{b'=b+1}^{M_y} \mathbb{1}\left[f(b,t|\beta) + f(b',t'|\beta) \geq f(b',t|\beta) + f(b,t'|\beta)\right] \tag{1}$$

which does not consider transfers and is calculated using Gale-Shapely (GS) Algorithm and

$$\hat{\beta} = \arg\max Q(\beta) = \sum_{y=1}^{Y} \sum_{b=1}^{M_y - 1} \sum_{b'=b+1}^{M_v} \mathbb{1}\left[f(b,t|\beta) - f(b,t'|\beta) \geq p_{bt} - p_{b't'} \wedge f(b',t'|\beta) - f(b',t|\beta) \geq p_{b'}t' - p_{bt}\right] \tag{2}$$

which considers transfers using Becker-Shapely-Subik (BSS) algorithm. We also define two different payoff functions for our desired relationship between variables of interest.

$$f_m(b,t) = x_{1bm}y_{1tm} + \alpha x_{2bm}y_{1tm} + \beta distance_{btm} + \varepsilon_{btm} \tag{3}$$

in which x_1bm is the number of stations owned by parent company of the buyer and y_1tm is the population in range of the target in market m (here two markets in two different years: 2007 anf 2008), x_2bm is an indicator for corporate ownership, and $distance_btm$ is the distance (in miles) between the buyer and target. The second form of payoff function is

$$f_m(b,t) = \delta x_{1bm}y_{1tm} + \alpha x_{2bm}y_{1tm} + \gamma HHI_{tm} + \beta distance_{btm} + \varepsilon_{btm} \tag{4}$$

where HHi_tm is the Hindahl-Hirschman Index measuring market concentration. Estimation results for these four models and algorithms are provided for two markets (for 2007 and 2008 as two independent markets) in section 1.2.

The results of estimation are provided for each market (2007 and 2008) separately along with the results of the whole data set. we can see that using GS algorithm, the interaction between corporate ownership and population in the first model of payoff is positive with the magnitude of 2.6. This amount changes to 0.28 with a perverse sign when we consider the transfers in calculating score function. But the coefficient of distance has positive and relatively large amount in both algorithms. One mile farther target leads the companies to value them as 7.02 unit of payoff higher in GS algorithm and 9.50$ higher in BSS algorithm. This positive coefficient for distance shows that the companies value more on farther away targets to expand their locations.

Using the second model of payoff, the interaction between the number of stations owned and the population has a positive impact on the companies' values with larger magnitude when we consider the prices (transfers) in our estimations. Additionally, the effect of interaction between corporate ownership and population is inconsistent in two algorithms; they have the opposite sing but larger magnitude when we consider transfers. Market concentration has a negative and less than one impact on the mergers value in both algorithms but its magnitude decreases negligibly in the BSS algorithm. Being positive shows that the higher concentrated markets attract the mergers more so they value these markets higher. Consistent with the first model, distance has a positive effect on the mergers' value using either GS or BSS methods.

1.2 Results

```
'''
Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm
    for Market = 2007:
 [0.92210561 9.28028429]
Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm
    for Market = 2008:
 [ 2.14538109 -8.18146729]
Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm
    for Market = 2007:
 [ 0.46232736 -2.28984215 -1.55386686  5.92561435]
Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm
    for Market = 2008:
 [-0.36335895  3.40351716  0.97678885 -2.4677087 ]
Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik
    Algorithm for Market = 2007:
 [1.10908911 9.53467009]
Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik
    Algorithm for Market = 2008:
 [4.34038389 3.06072991]
Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik
    Algorithm for Market = 2007:
 [ 1.72961086  1.01277899 -7.33622937  8.73538285]
Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik
    Algorithm for Market = 2007:
 [ 4.86419225  1.90809676 -3.20188475  5.18417893]
Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm:
 [2.64957342 7.02016782]
Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm:
 [ 0.85215225  0.21654464 -0.68652768  5.83534364]
Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik
    Algorithm:
 [-0.27622885  9.20030751]
Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik
    Algorithm:
 [ 2.47341091 -9.98165126 -0.45013507  9.49689741]
 '''
```

1.3 Executable Script

```
# Saharnaz Babaei
# Problem Set 4

# Note for later: A package for solving matching games: https://pypi.org/project/
    matching/
import numpy as np
import pandas as pd
from pandas import DataFrame as df
from pandas import ExcelWriter
from pandas import ExcelFile
import itertools
import scipy.optimize as opt
from scipy.optimize import minimize
from scipy.optimize import differential_evolution
```

```python
# easy_install pip
# In command prompt: <pip install geopy> — if failed: https://stackoverflow.com/
    questions/36064495/importerror—no—module—named—geopy—ipython—notebook
import geopy
from geopy import distance
#geopy.distance.vincenty: Deprecated since version 1.13: Vincenty will be removed in
     geopy 2.0. ref.: https://geopy.readthedocs.io/en/stable/#module—geopy.distance


# '''
# My sketch for solving the problem:
# — Read data
# — Prepare data
#     — Convert scale of price and population to 1000 dollars and 1000 people
    respectively;
#     — Two different datasets for each year;
#     — Create counterfactual and factual mergers' dataset;
#     — Create distance for mergers.
#     — a note for myself: https://datacarpentry.org/python—ecology—lesson/03—index—
    slice—subset/
# — def score function (for 2 different models):
#     — without transfer (GS algorithm);
#     — with transfers (BSS algorithm)
# — Optimize score functions and obtain the desired parameters
# '''

df = pd.read_excel("radio_merger_data.xlsx")
df.head()

df['price'] = df['price']/1000
df['population_target'] = df['population_target']/1000

'''
#The number of all possible combinations:
num_comb = len(list(itertools.product(df2007.buyer_id.tolist(), df2007.target_id.
    tolist()))) + len(list(itertools.product(df2008.buyer_id.tolist(), df2008.
    target_id.tolist())))
# Any possible combination (match) between the buyer and target (including real/
    factual and counterfactual data):
B2007 = df2007.loc[:, ['year', 'buyer_id', 'buyer_lat', 'buyer_long', '
    num_stations_buyer', 'corp_owner_buyer']]
B2008 = df2008.loc[:, ['year', 'buyer_id', 'buyer_lat', 'buyer_long', '
    num_stations_buyer', 'corp_owner_buyer']]
T2007 = df2007.loc[:, ['target_id', 'target_lat', 'target_long', 'price', '
    hhi_target', 'population_target']]
T2008 = df2008.loc[:, ['target_id', 'target_lat', 'target_long', 'price', '
    hhi_target', 'population_target']]
def cartesian_prod7(B2007, T2007):
    return(B2007.assign(key=1).merge(T2007.assign(key=1), on='key').drop('key', 1))
combinations_2007 = cartesian_prod7(B2007, T2007)
def cartesian_prod8(B2008, T2008):
    return(B2008.assign(key=1).merge(T2008.assign(key=1), on='key').drop('key', 1))
combinations_2008 = cartesian_prod8(B2008, T2008)
'''
'''
Due to lack of time, could not get rid of factual data; so used another method to
    get counterfactual dataset.
I will work on that later to remove common data in this cartesian data with my
    dataframe which is actually the factual dataset.
'''
dfy = dict(iter(df.groupby('year', as_index = False)))
dfy[2007].describe().to_csv("describe2007.csv")
dfy[2008].describe().to_csv("describe2008.csv")
B = ['year', 'buyer_id', 'buyer_lat', 'buyer_long', 'num_stations_buyer', '
    corp_owner_buyer']
T = ['target_id', 'target_lat', 'target_long', 'price', 'hhi_target', '
    population_target']
data = [pd.DataFrame(dfy[2007]), pd.DataFrame(dfy[2008])]
counterfact = pd.DataFrame()
counterfact = pd.DataFrame([x[B].iloc[i].values.tolist() + x[T].iloc[j].values.
    tolist() for x in data for i in range(len(x)) for j in range(len(x)) if i!= j],
    columns = B + T)
```

```
# http://jonathansoma.com/lede/foundations/classes/pandas%20columns%20and%20
    functions/apply—a—function—to—every—row—in—a—pandas—dataframe/
def dist(d):
    '''
    This function calculates distance between two points.
    b_loc = The coordinates for buyer's location
    t_loc = The coordinates for target's location
    The result is in miles.
    '''
    b_loc = (d['buyer_lat'], d['buyer_long'])
    t_loc = (d['target_lat'], d['target_long'])
    return distance.distance(b_loc, t_loc).miles

df['distance'] = df.apply(dist, axis=1)
counterfact['distance'] = counterfact.apply(dist, axis=1)

df2007 = df[df['year'] == 2007]
df2008 = df[df['year'] == 2008]
cf2007 = counterfact[counterfact['year']==2007]
cf2008 = counterfact[counterfact['year']==2008]

def score1_GS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = x_1bm * y_1tm + alpha * x_2bm * y_1tm + beta * distance_btm +
        epsilon_btm
    payoff to merger = (number of stations owned) * (population in range of target)
        + alpha * (indicator for corporate ownership) * (population in range of
        target) + beta * (distance bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t
    '''
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10)]
GS1_dif_2007 = differential_evolution(score1_GS, bounds, args=(df2007, cf2007),
    strategy='best1bin', maxiter=10000)
GS1_dif_2008 = differential_evolution(score1_GS, bounds, args=(df2008, cf2008),
    strategy='best1bin', maxiter=10000)

def scoreT1_GS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = x_1bm * y_1tm + alpha * x_2bm * y_1tm + beta * distance_btm +
        epsilon_btm
    payoff to merger = (number of stations owned) * (population in range of target)
        + alpha * (indicator for corporate ownership) * (population in range of
        target) + beta * (distance bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t
    '''
```

```
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10)]
GST1_dif = differential_evolution(scoreT1_GS, bounds, args=(df, counterfact),
    strategy='best1bin', maxiter=10000)

def score2_GS(params, m, n):

    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = delta * x_1bm * y_1tm + alpha * x_2bm * y_1tm + gamma* HHI_tm + beta
        * distance_btm + epsilon_btm
    payoff to merger = delta * (number of stations owned) * (population in range of
        target) + alpha * (indicator for corporate ownership) * (population in range
         of target) + gamma * (market concentration index) + beta * (distance
        bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t

    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1]
         * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['
        hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params
        [1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']

    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
GS2_dif_2007 = differential_evolution(score2_GS, bounds, args=(df2007, cf2007),
    strategy='best1bin', maxiter=10000)
GS2_dif_2008 = differential_evolution(score2_GS, bounds, args=(df2008, cf2008),
    strategy='best1bin', maxiter=10000)

def scoreT2_GS(params, m, n):

    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = delta * x_1bm * y_1tm + alpha * x_2bm * y_1tm + gamma* HHI_tm + beta
        * distance_btm + epsilon_btm
    payoff to merger = delta * (number of stations owned) * (population in range of
        target) + alpha * (indicator for corporate ownership) * (population in range
```

```
         of target) + gamma * (market concentration index) + beta * (distance
        bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t

    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1]
         * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['
        hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params
        [1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']

    L = f_b_t + f_cb_ct
    R = f_cb_t + f_b_ct
    indicator=(L>R)
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
GST2_dif = differential_evolution(scoreT2_GS, bounds, args=(df, counterfact),
    strategy='best1bin', maxiter=10000)

def score1_BSS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = x_1bm * y_1tm + alpha * x_2bm * y_1tm + beta * distance_btm +
        epsilon_btm
    payoff to merger = (number of stations owned) * (population in range of target)
        + alpha * (indicator for corporate ownership) * (population in range of
        target) + beta * (distance bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t
    '''
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10)]
BSS1_dif_2007 = differential_evolution(score1_BSS, bounds, args=(df2007, cf2007),
    strategy='best1bin', maxiter=10000)
BSS1_dif_2008 = differential_evolution(score1_BSS, bounds, args=(df2008, cf2008),
    strategy='best1bin', maxiter=10000)

def scoreT1_BSS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
```

```
    Indicator == 0 otherwise.
    f(b,t,m) = x_1bm * y_1tm + alpha * x_2bm * y_1tm + beta * distance_btm +
        epsilon_btm
    payoff to merger = (number of stations owned) * (population in range of target)
        + alpha * (indicator for corporate ownership) * (population in range of
        target) + beta * (distance bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t
    '''
    f_b_t = m['num_stations_buyer'] * m['population_target'] + params[0] * m['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_cb_ct = n['num_stations_buyer'] * n['population_target'] + params[0] * n['
        corp_owner_buyer'] * n['population_target'] + params[1] * n['distance']
    f_cb_t = n['num_stations_buyer'] * m['population_target'] + params[0] * n['
        corp_owner_buyer'] * m['population_target'] + params[1] * m['distance']
    f_b_ct = m['num_stations_buyer'] * n['population_target'] + params[0] * m['
        corp_owner_buyer'] * n['population_target'] + params[1] * m['distance']
    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10)]
BSST1_dif = differential_evolution(scoreT1_BSS, bounds, args=(df, counterfact),
    strategy='best1bin', maxiter=10000)

def score2_BSS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the
        indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = delta * x_1bm * y_1tm + alpha * x_2bm * y_1tm + gamma* HHI_tm + beta
        * distance_btm + epsilon_btm
    payoff to merger = delta * (number of stations owned) * (population in range of
        target) + alpha * (indicator for corporate ownership) * (population in range
         of target) + gamma * (market concentration index) + beta * (distance
        bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t

    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1]
         * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['
        hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params
        [1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params
        [1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['
        hhi_target'] + params[3] * m['distance']

    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
BSS2_dif_2007 = differential_evolution(score2_BSS, bounds, args=(df2007, cf2007),
    strategy='best1bin', maxiter=10000)
```

```python
BSS2_dif_2008 = differential_evolution(score2_BSS, bounds, args=(df2008, cf2008), strategy='best1bin', maxiter=10000)

def scoreT2_BSS(params, m, n):
    '''
    This function calculates the payoff function for mergers to be used in the indicator function.
    Indicator == 1 if f(b,t) + f(b',t') > f(b',t) + f(b,t')
    Indicator == 0 otherwise.
    f(b,t,m) = delta * x_1bm * y_1tm + alpha * x_2bm * y_1tm + gamma* HHI_tm + beta * distance_btm + epsilon_btm
    payoff to merger = delta * (number of stations owned) * (population in range of target) + alpha * (indicator for corporate ownership) * (population in range of target) + gamma * (market concentration index) + beta * (distance bwbuyer and target) + (error term)
    market m == 2007 & 2008
    buyer b
    target t

    '''
    f_b_t = params[0] * m['num_stations_buyer'] * m['population_target'] + params[1] * m['corp_owner_buyer'] * m['population_target'] + params[2] * m['hhi_target'] + params[3] * m['distance']
    f_cb_ct = params[0] * n['num_stations_buyer'] * n['population_target'] + params[1] * n['corp_owner_buyer'] * n['population_target'] + params[2]*n['hhi_target'] + params[3] * n['distance']
    f_cb_t = params[0] * n['num_stations_buyer'] * m['population_target'] + params[1] * n['corp_owner_buyer'] * m['population_target'] + params[2] * m['hhi_target'] + params[3] * m['distance']
    f_b_ct = params[0] * m['num_stations_buyer'] * n['population_target'] + params[1] * n['corp_owner_buyer'] * n['population_target'] + params[2] * m['hhi_target'] + params[3] * m['distance']

    L1 = f_b_t - f_b_ct
    R1 = m['price'] - n['price']
    L2 = f_cb_ct - f_cb_t
    R2 = n['price'] - m['price']
    indicator= ((L1 >= R1) & (L2 >= R2))
    total_payoff = indicator.sum()
    return -total_payoff

bounds = [(-5,5), (-10,10), (-10, 10), (-10,10)]
BSST2_dif = differential_evolution(scoreT2_BSS, bounds, args=(df, counterfact), strategy='best1bin', maxiter=10000)

print('Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm for Market = 2007: \n', GS1_dif_2007.x)
print('Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm for Market = 2008: \n', GS1_dif_2008.x)
print('Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm for Market = 2007: \n', GS2_dif_2007.x)
print('Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm for Market = 2008: \n', GS2_dif_2008.x)
print('Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik Algorithm for Market = 2007: \n', BSS1_dif_2007.x)
print('Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik Algorithm for Market = 2008: \n', BSS1_dif_2008.x)
print('Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik Algorithm for Market = 2007: \n', BSS2_dif_2007.x)
print('Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik Algorithm for Market = 2007: \n', BSS2_dif_2008.x)
# ##################################################################################################

print('Estimated Parameters for Model 1, without transfers, using Gale-Shapely Algorithm: \n', GST1_dif.x)
print('Estimated Parameters for Model 2, without transfers, using Gale-Shapely Algorithm: \n', GST2_dif.x)
print('Estimated Parameters for Model 1, with transfers, using Becker-Shapely-Subik Algorithm: \n', BSST1_dif.x)
print('Estimated Parameters for Model 2, with transfers, using Becker-Shapely-Subik Algorithm: \n', BSST2_dif.x)
```