Management Science Group Assignment

Management Science Group 2

Naramol Pipoppinyo, Risa Yamazaki, Abdukarim Omar, Thanh Dat Nyugen

November 30, 2022

1. Unconstrained Chain Length

1.Mathematical Modeling

$$\textbf{Max.} \quad \sum X_{ij} \quad \forall\, ij \in C \quad (a)$$

$$\textbf{s.t} \quad \sum_{j \in \{R, P\}} X_{ij} \le 1 \quad \forall\, i \in \{D,\ P\} \quad (b)$$

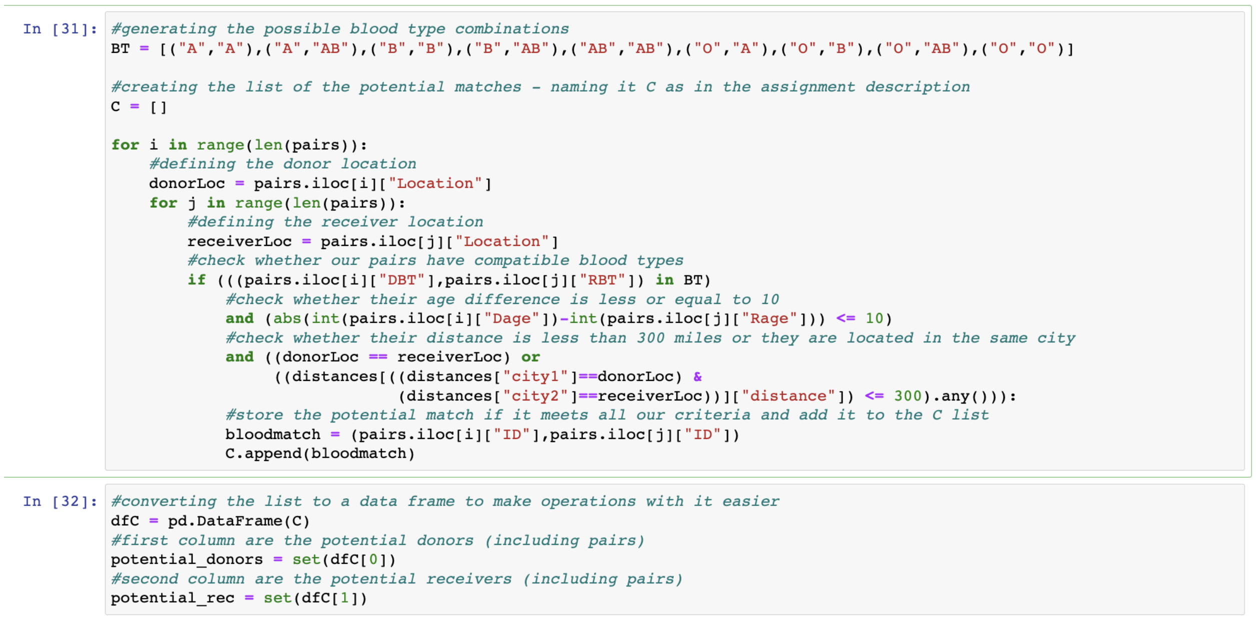
$$\sum_{i \in \{D, P\}} X_{ij} \le 1 \quad \forall\, j \in \{R,\ P\} \quad (c)$$

$$\sum_{j \in \{R, P\}} X_{ij} = \sum_{k \in \{D, P\}} X_{ki} \quad \forall\, i \in P \quad (d)$$

C = all potential matches
D = singular donors in our potential matches
P = pairs in our potential matches
R = singular receivers in our potential matches

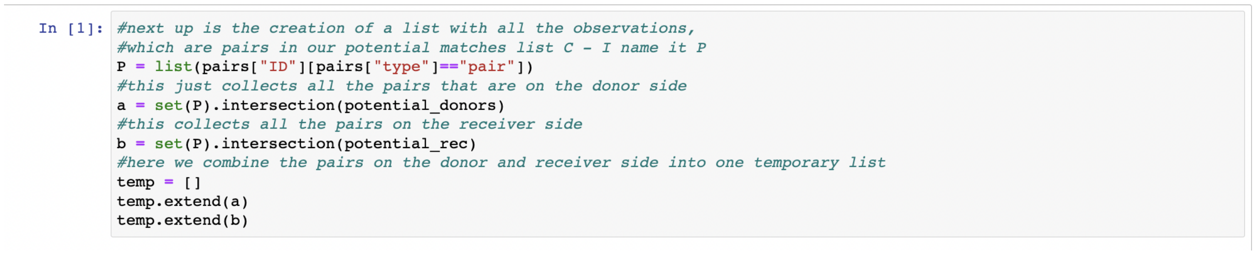
In this model, we are trying to maximize the number of patients that receive a kidney (a). This is for all values in *C* which contain all the potential matches. The potential matches are based on the blood types being compatible, the age difference not being above 10 years, and the distance between the donor and recipient not being larger than 300 miles. This was not coded as a constraint in Gurobi, but rather implemented separately in preparation of the data for the Gurobi optimizer. The first constraint *b* states that for all donors that donate a kidney to a recipient, they can donate no more than one kidney. The second constraint *c* states that for all receivers that receive a kidney from a donor, they can receive no more than one kidney. The last constraint *d* states that donor-receiver pairs only donate a kidney if they receive one, and they can also receive a kidney only if they donate.

2.Data Preparation



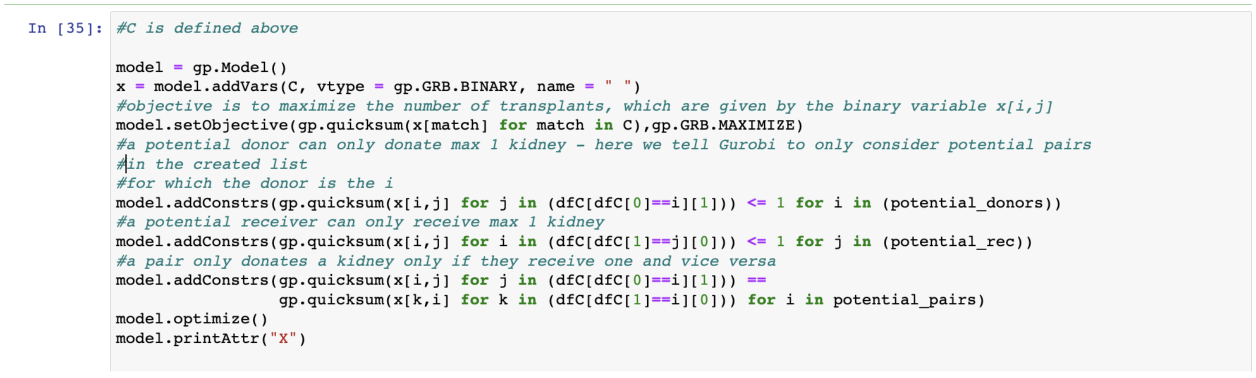
In [31]:
```
#generating the possible blood type combinations
BT = [("A","A"),("A","AB"),("B","B"),("B","AB"),("AB","AB"),("O","A"),("O","B"),("O","AB"),("O","O")]

#creating the list of the potential matches - naming it C as in the assignment description
C = []

for i in range(len(pairs)):
    #defining the donor location
    donorLoc = pairs.iloc[i]["Location"]
    for j in range(len(pairs)):
        #defining the receiver location
        receiverLoc = pairs.iloc[j]["Location"]
        #check whether our pairs have compatible blood types
        if (((pairs.iloc[i]["DBT"],pairs.iloc[j]["RBT"]) in BT)
            #check whether their age difference is less or equal to 10
            and (abs(int(pairs.iloc[i]["Dage"])-int(pairs.iloc[j]["Rage"])) <= 10)
            #check whether their distance is less than 300 miles or they are located in the same city
            and ((donorLoc == receiverLoc) or
                 ((distances[((distances["city1"]==donorLoc) &
                              (distances["city2"]==receiverLoc))]["distance"]) <= 300).any())):
            #store the potential match if it meets all our criteria and add it to the C list
            bloodmatch = (pairs.iloc[i]["ID"],pairs.iloc[j]["ID"])
            C.append(bloodmatch)
```

In [32]:
```
#converting the list to a data frame to make operations with it easier
dfC = pd.DataFrame(C)
#first column are the potential donors (including pairs)
potential_donors = set(dfC[0])
#second column are the potential receivers (including pairs)
potential_rec = set(dfC[1])
```



In [1]:
```
#next up is the creation of a list with all the observations,
#which are pairs in our potential matches list C - I name it P
P = list(pairs["ID"][pairs["type"]=="pair"])
#this just collects all the pairs that are on the donor side
a = set(P).intersection(potential_donors)
#this collects all the pairs on the receiver side
b = set(P).intersection(potential_rec)
#here we combine the pairs on the donor and receiver side into one temporary list
temp = []
temp.extend(a)
temp.extend(b)
```

3. Implement and Solve in Gurobi

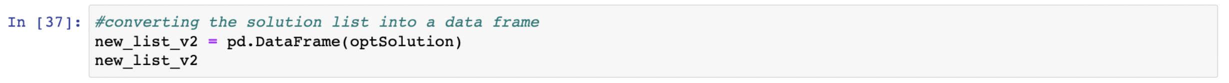


In [35]:
```
#C is defined above

model = gp.Model()
x = model.addVars(C, vtype = gp.GRB.BINARY, name = " ")
#objective is to maximize the number of transplants, which are given by the binary variable x[i,j]
model.setObjective(gp.quicksum(x[match] for match in C),gp.GRB.MAXIMIZE)
#a potential donor can only donate max 1 kidney - here we tell Gurobi to only consider potential pairs
#in the created list
#for which the donor is the i
model.addConstrs(gp.quicksum(x[i,j] for j in (dfC[dfC[0]==i][1])) <= 1 for i in (potential_donors))
#a potential receiver can only receive max 1 kidney
model.addConstrs(gp.quicksum(x[i,j] for i in (dfC[dfC[1]==j][0])) <= 1 for j in (potential_rec))
#a pair only donates a kidney only if they receive one and vice versa
model.addConstrs(gp.quicksum(x[i,j] for j in (dfC[dfC[0]==i][1])) ==
                 gp.quicksum(x[k,i] for k in (dfC[dfC[1]==i][0])) for i in potential_pairs)
model.optimize()
model.printAttr("X")
```

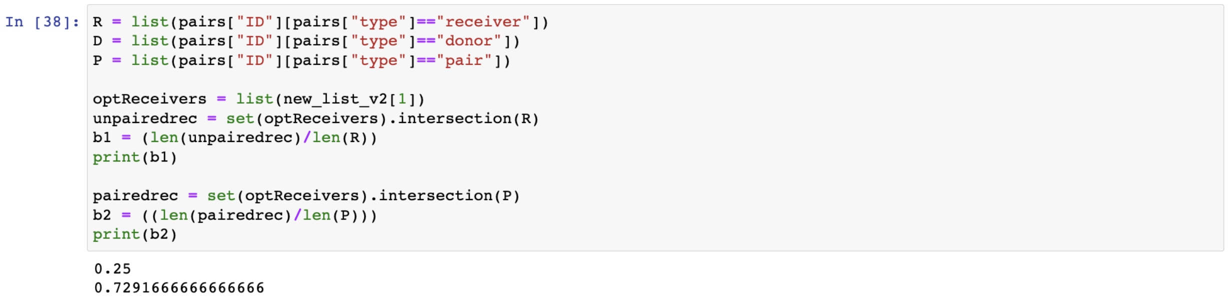
4. Analyze Solution

a) According to the maximized optimal solution, there should be 90 transplantations that take place.

b) 25% of paired patients will receive a kidney paired patient. And 72.9% of unpaired patients will receive a kidney unpaired patients

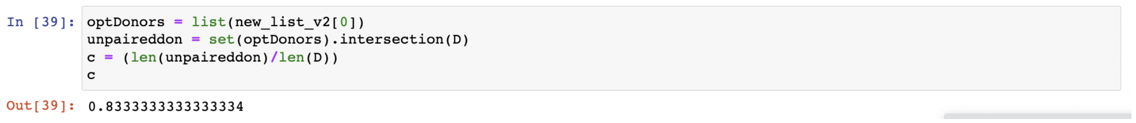


```
In [37]: #converting the solution list into a data frame
         new_list_v2 = pd.DataFrame(optSolution)
         new_list_v2
```



```
In [38]: R = list(pairs["ID"][pairs["type"]=="receiver"])
         D = list(pairs["ID"][pairs["type"]=="donor"])
         P = list(pairs["ID"][pairs["type"]=="pair"])

         optReceivers = list(new_list_v2[1])
         unpairedrec = set(optReceivers).intersection(R)
         b1 = (len(unpairedrec)/len(R))
         print(b1)

         pairedrec = set(optReceivers).intersection(P)
         b2 = ((len(pairedrec)/len(P)))
         print(b2)

         0.25
         0.7291666666666666
```

c) 83.3% of unpaired donors donated a kidney.



```
In [39]: optDonors = list(new_list_v2[0])
         unpaireddon = set(optDonors).intersection(D)
         c = (len(unpaireddon)/len(D))
         c

Out[39]: 0.8333333333333334
```

d)

DBT	RBT	Percentage (%)
A	A	100
	B	50
	O	71.4
	AB	100
B	A	33.3
	O	12.5
O	A	94.1
	B	100
	O	87
	AB	100
AB	A	50
	O	0

	AB	50

Table 1: Percentage of kidney receivers (300 miles)

The percentage of the pairs who received a transplant for different pair blood type combinations was calculated. This was done by counting the number of pairs that receive a kidney in the optimal solution by its blood combination over the total number of pairs for each blood type combination. A table was created on Python which counted the number of pairs in each 13 different unique blood combinations. Another table was created to count the number of pairs in the optimal solution subsetted into the 13 different blood combinations. The two tables were extracted into Excel to calculate the percentage.

100% of AA and AAB pairs received kidneys (see table 1). Half of the AB pairs received the kidneys (50%), and 71.4% of AO pairs received kidneys. 33.3% of BA pairs received kidneys. 12.5% of BO pairs received the kidney. 94.1% of OA, 100% of OB and OAB received kidneys. For OO pairs, 87% of pairs received the kidneys. 50% of ABA pairs and ABAB pairs received kidneys. However, none of the ABO pairs received a kidney.

The bar chart visualizes the distribution of the pairs that received kidneys for different blood type combinations. The x-axis represents each pairs blood type combinations, and the y axis represents the percentage of pairs that receive kidneys for different blood type combination.

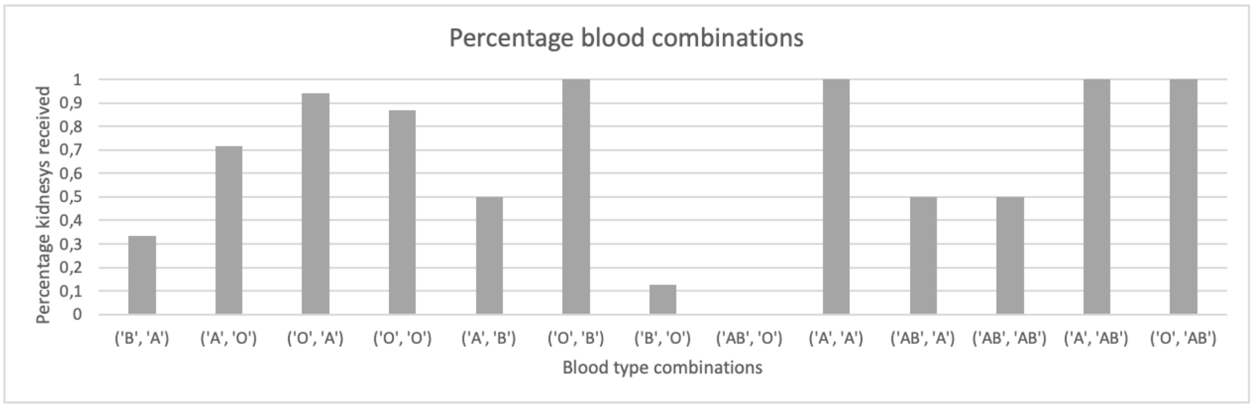
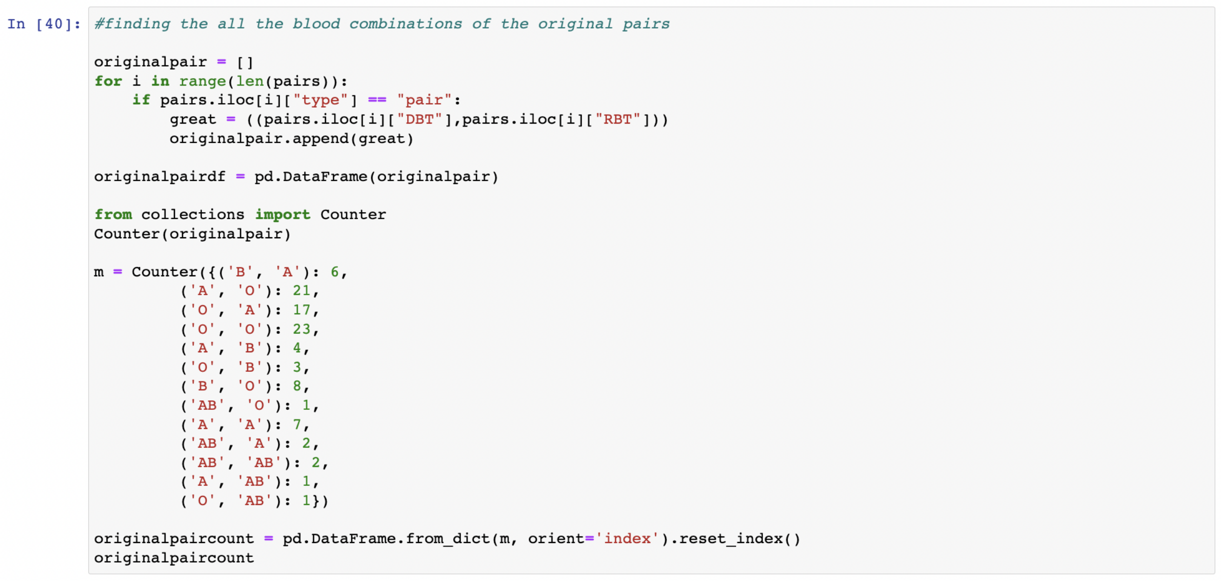
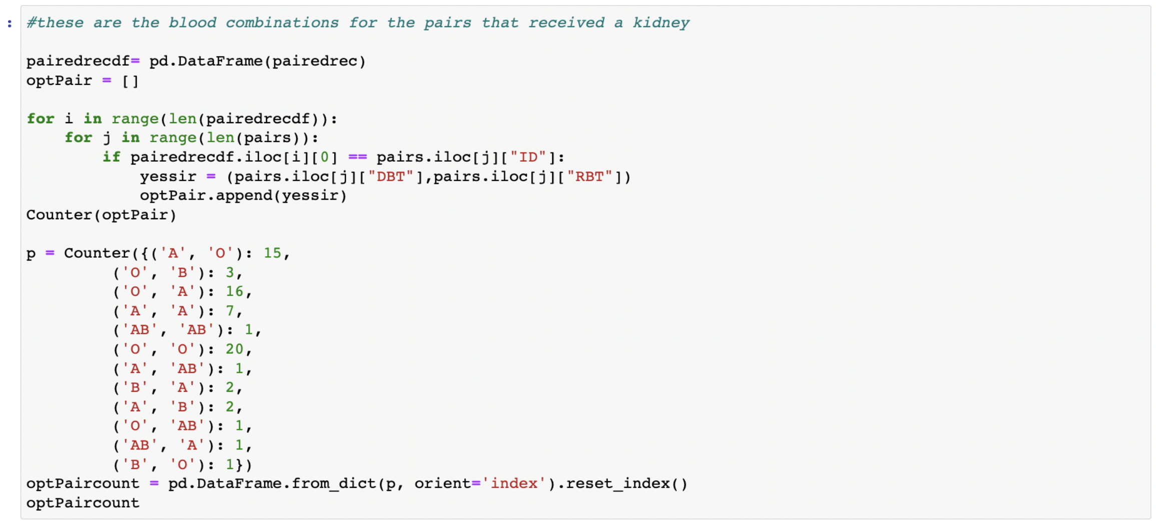


Figure 1: Graph of kidney receivers (%)



```
In [40]: #finding the all the blood combinations of the original pairs

         originalpair = []
         for i in range(len(pairs)):
             if pairs.iloc[i]["type"] == "pair":
                 great = ((pairs.iloc[i]["DBT"],pairs.iloc[i]["RBT"]))
                 originalpair.append(great)

         originalpairdf = pd.DataFrame(originalpair)

         from collections import Counter
         Counter(originalpair)

         m = Counter({('B', 'A'): 6,
                  ('A', 'O'): 21,
                  ('O', 'A'): 17,
                  ('O', 'O'): 23,
                  ('A', 'B'): 4,
                  ('O', 'B'): 3,
                  ('B', 'O'): 8,
                  ('AB', 'O'): 1,
                  ('A', 'A'): 7,
                  ('AB', 'A'): 2,
                  ('AB', 'AB'): 2,
                  ('A', 'AB'): 1,
                  ('O', 'AB'): 1})

         originalpaircount = pd.DataFrame.from_dict(m, orient='index').reset_index()
         originalpaircount
```



```
: #these are the blood combinations for the pairs that received a kidney

  pairedrecdf= pd.DataFrame(pairedrec)
  optPair = []

  for i in range(len(pairedrecdf)):
      for j in range(len(pairs)):
          if pairedrecdf.iloc[i][0] == pairs.iloc[j]["ID"]:
              yessir = (pairs.iloc[j]["DBT"],pairs.iloc[j]["RBT"])
              optPair.append(yessir)
  Counter(optPair)

  p = Counter({('A', 'O'): 15,
           ('O', 'B'): 3,
           ('O', 'A'): 16,
           ('A', 'A'): 7,
           ('AB', 'AB'): 1,
           ('O', 'O'): 20,
           ('A', 'AB'): 1,
           ('B', 'A'): 2,
           ('A', 'B'): 2,
           ('O', 'AB'): 1,
           ('AB', 'A'): 1,
           ('B', 'O'): 1})
  optPaircount = pd.DataFrame.from_dict(p, orient='index').reset_index()
  optPaircount
```

e)

Cities	**Percentage (%)**
Amsterdam	66.7
Copenhagen	57.1
Dublin	60
Düsseldorf	85.7
Frankfurt	100
Istanbul	88.9

Lisbon	100
London	83.3
Madrid	33.3
Manchester	100
Millan	100
Moscow	50
Oslo	75
Palma de Mallorca	33.3
Paris	50
Rome	66.7
Vienna	50
Zürich	100

Table 2: Percentage of receivers in different cities (300 miles)

To calculate the percentage, the same as 1.4d was done where two tables were created on python (cities of pairs in the original table and cities of pairs in the optimal solution) and extracted to excel to calculate. For each city the total number of optimal pairs were divided over the total number of pairs from the city (see table 2). For Frankfurt, Millan, Lisbon, Zürich and Manchester, 100 % of pairs receive kidneys. 66.7% of pairs from Amsterdam receive kidneys. 88.3% of paired patients in London receive kidneys. 60% of pairs from Dublin received kidneys. 50% of pairs from Moscow, Vienna and Paris can receive a kidney. 88.9% of paired patients in Istanbul can receive kidneys. For Copenhagen, the percentage of paired patients receiving the kidney is 57.1%. For Düsseldorf, 85.7% of patients can receive kidneys. Lastly, the percentage of patients who can receive kidneys in Madrid is 33.3%

The bar chart below visualizes the percentage of paired patients who receive a kidney in different cities. The x-axis is the list of cities, and the y-axis shows the percentage of the paired patients.

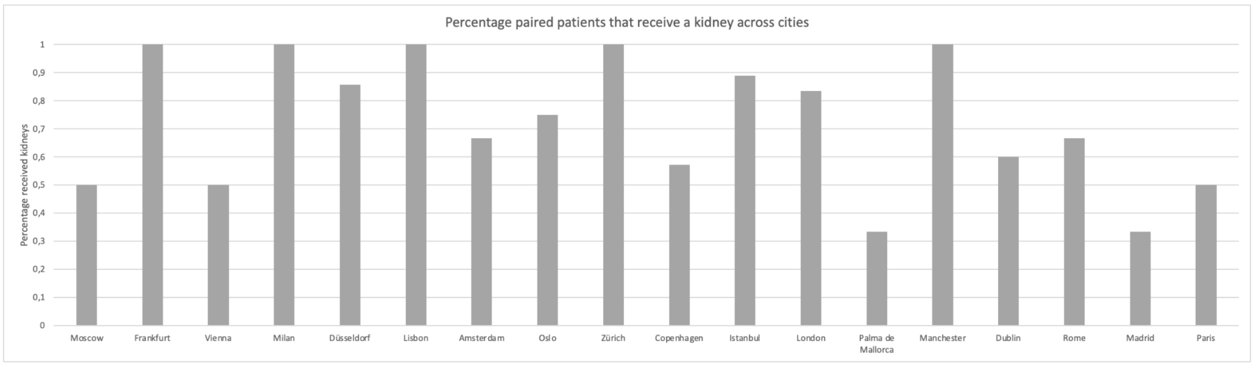


Figure 2: Percentage of receivers in different cities (300 miles)

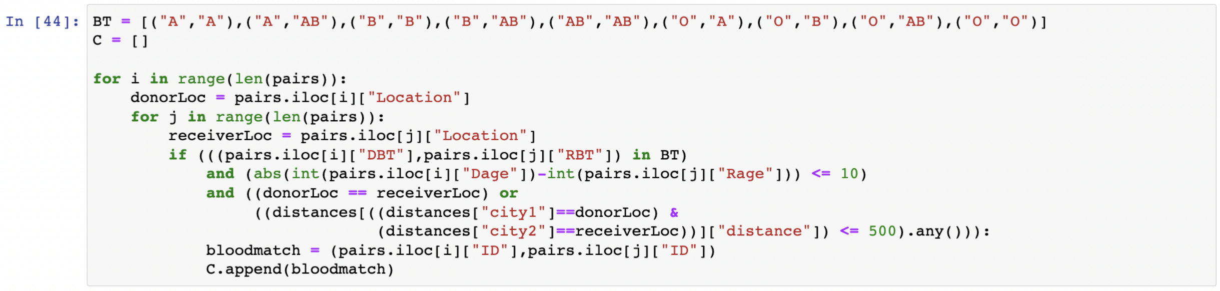
```
In [43]: #These are the cities of the pairs from the optimal solution

         cityopt = []
         for i in range(len(pairedrecdf)):
             for j in range(len(pairs)):
                 if pairedrecdf.iloc[i][0] == pairs.iloc[j]["ID"]:
                     locator = (pairs.iloc[j]["Location"])
                     cityopt.append(locator)
         Counter(cityopt)

         from collections import Counter
         d = Counter({'Frankfurt': 6,
                   'Düsseldorf': 6,
                   'Vienna': 2,
                   'Milan': 3,
                   'Istanbul': 8,
                   'Amsterdam': 4,
                   'Moscow': 5,
                   'Oslo': 3,
                   'Manchester': 6,
                   'Lisbon': 5,
                   'Zürich': 4,
                   'Copenhagen': 4,
                   'Dublin': 3,
                   'Rome': 2,
                   'London': 5,
                   'Palma de Mallorca': 2,
                   'Madrid': 1,
                   'Paris': 1})
         cityoptCount = pd.DataFrame.from_dict(d, orient='index').reset_index()
         cityoptCount
```

```
In [42]: #these are all the cities of the pairs from the original data

         allcityv2 = []
         for i in range(len(pairs)):
             if pairs.iloc[i]["type"] == "pair":
                 allcity = ((pairs.iloc[i]["Location"]))
                 allcityv2.append(allcity)


         Counter(allcityv2)

         o = Counter({'Moscow': 10,
                   'Frankfurt': 6,
                   'Vienna': 4,
                   'Milan': 3,
                   'Madrid': 3,
                   'Düsseldorf': 7,
                   'Lisbon': 5,
                   'Amsterdam': 6,
                   'Oslo': 4,
                   'Zürich': 4,
                   'Copenhagen': 7,
                   'Istanbul': 9,
                   'London': 6,
                   'Palma de Mallorca': 6,
                   'Manchester': 6,
                   'Dublin': 5,
                   'Rome': 3,
                   'Paris': 2})
         allcityCount = pd.DataFrame.from_dict(o, orient='index').reset_index()
         allcityCount
```

5. Analyze Solution 2

The optimal solution was rerun with the new constraint in which the donors could be no more than 500 miles apart. The results are as follows:

a) 104 matches for transplants will take place according to the optimal solution.
b) 28.8% of paired patients will receive a kidney paired patient. And 84.4% of unpaired patients will receive a kidneyunpaired patients.
c) 95.8% of unpaired donors donated a kidney.



```
In [44]: BT = [("A","A"),("A","AB"),("B","B"),("B","AB"),("AB","AB"),("O","A"),("O","B"),("O","AB"),("O","O")]
         C = []

         for i in range(len(pairs)):
             donorLoc = pairs.iloc[i]["Location"]
             for j in range(len(pairs)):
                 receiverLoc = pairs.iloc[j]["Location"]
                 if (((pairs.iloc[i]["DBT"],pairs.iloc[j]["RBT"]) in BT)
                     and (abs(int(pairs.iloc[i]["Dage"])-int(pairs.iloc[j]["Rage"])) <= 10)
                     and ((donorLoc == receiverLoc) or
                          ((distances[((distances["city1"]==donorLoc) &
                                       (distances["city2"]==receiverLoc))]["distance"]) <= 500).any())):
                     bloodmatch = (pairs.iloc[i]["ID"],pairs.iloc[j]["ID"])
                     C.append(bloodmatch)
```

d) Under the constraint of a maximum of 500 miles, the number of patients who can get the transplant of a kidney for the pair of AB, AO, BA, BO, OA, and OO increased (see table 3). For AB it increased from 50% to 75%, AO increased from 71.4% to 85.7%, BA increased from 33.3% to 66.7%, BO increased from 12.5% to 37.5%, OA increased from 94.1% to 100%, and OO increased from 87% to 96%.

DBT	**RBT**	**Percentage (%)**
A	A	100
	B	75
	O	85.7
	AB	100
B	A	66.7
	O	37.5
O	A	100
	B	100
	O	96
	AB	100
AB	A	50
	O	0
	AB	50

Table 3: Percentage of kidney receivers (500 miles)

The visualization of the percentage of receivers in the different blood combination after the adjustment of the distance constraint is shown below.

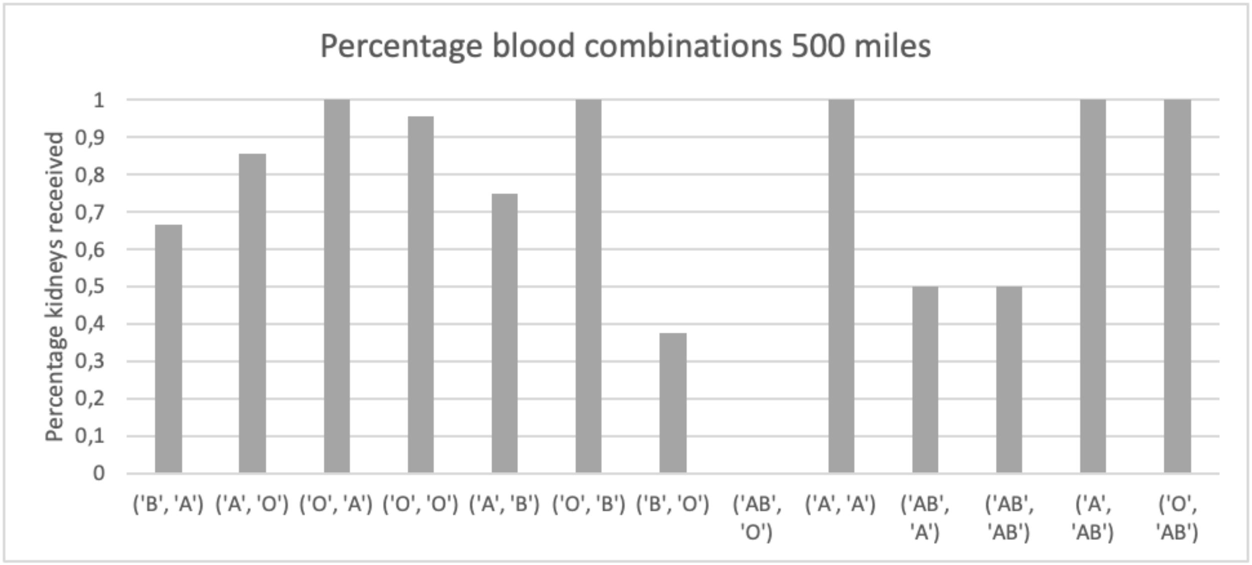


Figure 3: Graph of kidney receivers (%)

The percentage of paired patients that receive a kidney for each city is visualized below:

Cities	**Percentage (%)**
Amsterdam	83.3
Copenhagen	57.1
Dublin	80
Düsseldorf	100
Frankfurt	100
Istanbul	88.9
Lisbon	100
London	83.3
Madrid	100
Manchester	100

Millan	100
Moscow	50
Oslo	75
Palma de Mallorca	83.3
Paris	100
Rome	100
Vienna	75
Zürich	100

Table 4: Percentage of receivers in different cities (500 miles)

After the adjustment of the distance constraint, the percentage of paired patients who received the kidney transplant increased for several cities. For example, Vienna increased in the percentage from 50% to 75% (see table 4). However there were some cities that did not see an increase in percentage. For instance, London still has a 83.3% percentage after the adjustment in the constraint (see table 4).

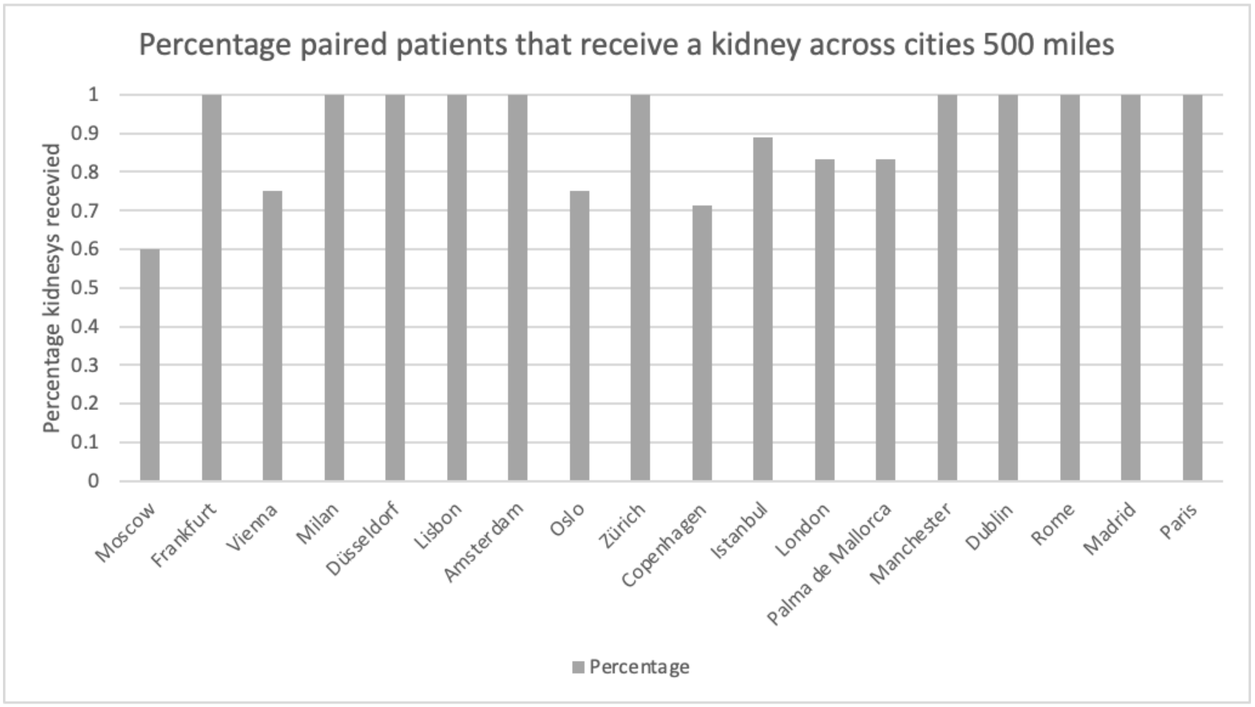
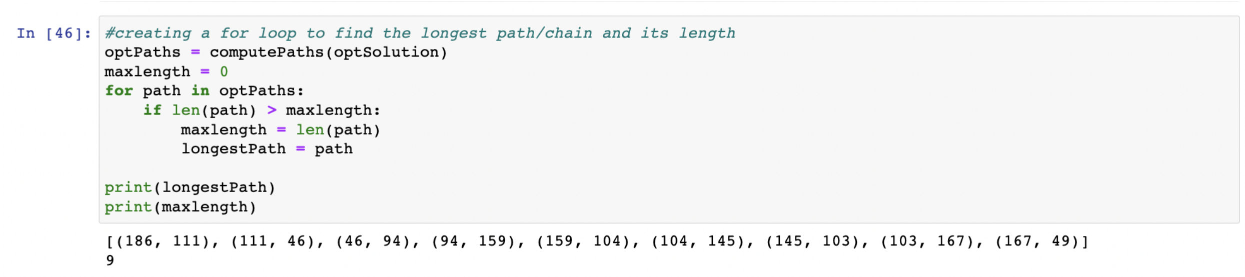


Figure 4: Percentage of receivers in different cities (500 miles)

2. Avoiding Long Chains

1. Calculate longest path for current solution

The longest path in the current solution is 9.

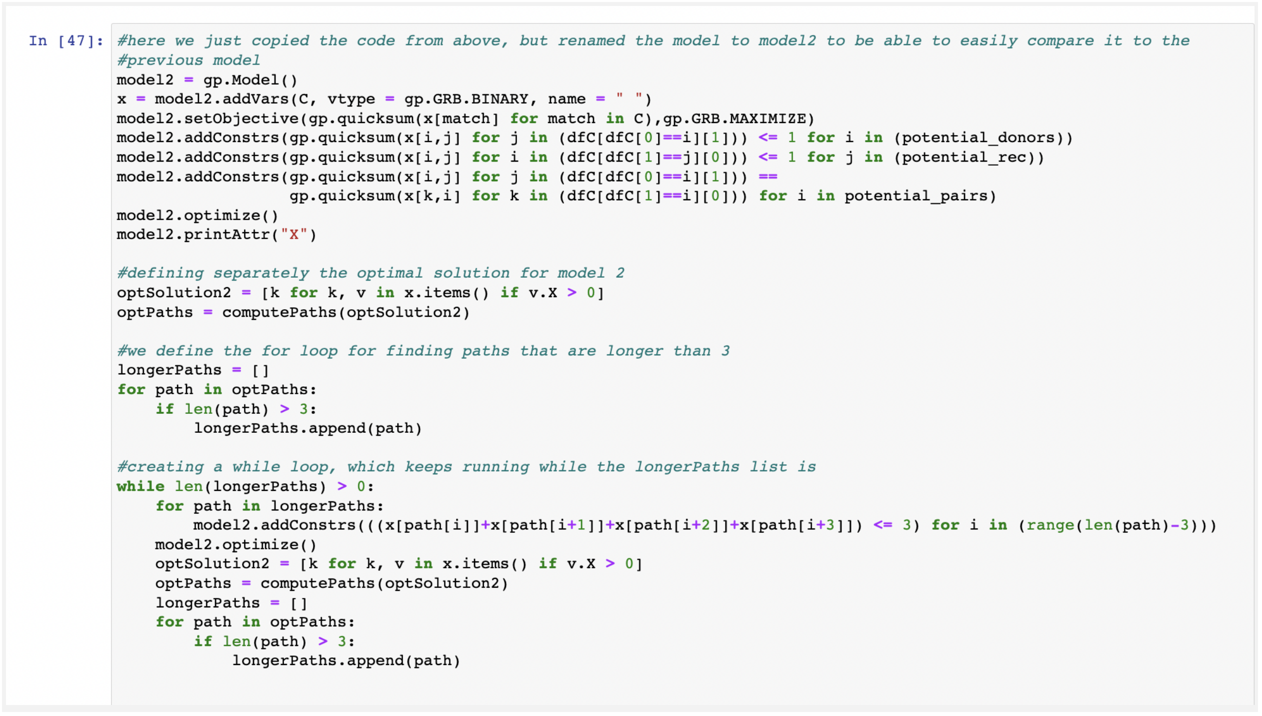


```
In [46]: #creating a for loop to find the longest path/chain and its length
         optPaths = computePaths(optSolution)
         maxlength = 0
         for path in optPaths:
             if len(path) > maxlength:
                 maxlength = len(path)
                 longestPath = path

         print(longestPath)
         print(maxlength)

         [(186, 111), (111, 46), (46, 94), (94, 159), (159, 104), (104, 145), (145, 103), (103, 167), (167, 49)]
         9
```

3.Analyze Solution for the new optimal solution (300 miles)

a) 90 transplantations will take place

b) From the first solution, the number of constraints was at 369 and increased to 3222. Therefore 2853 constraints were added.



```
In [47]: #here we just copied the code from above, but renamed the model to model2 to be able to easily compare it to the
         #previous model
         model2 = gp.Model()
         x = model2.addVars(C, vtype = gp.GRB.BINARY, name = " ")
         model2.setObjective(gp.quicksum(x[match] for match in C),gp.GRB.MAXIMIZE)
         model2.addConstrs(gp.quicksum(x[i,j] for j in (dfC[dfC[0]==i][1])) <= 1 for i in (potential_donors))
         model2.addConstrs(gp.quicksum(x[i,j] for i in (dfC[dfC[1]==j][0])) <= 1 for j in (potential_rec))
         model2.addConstrs(gp.quicksum(x[i,j] for j in (dfC[dfC[0]==i][1])) ==
                           gp.quicksum(x[k,i] for k in (dfC[dfC[1]==i][0])) for i in potential_pairs)
         model2.optimize()
         model2.printAttr("X")

         #defining separately the optimal solution for model 2
         optSolution2 = [k for k, v in x.items() if v.X > 0]
         optPaths = computePaths(optSolution2)

         #we define the for loop for finding paths that are longer than 3
         longerPaths = []
         for path in optPaths:
             if len(path) > 3:
                 longerPaths.append(path)

         #creating a while loop, which keeps running while the longerPaths list is
         while len(longerPaths) > 0:
             for path in longerPaths:
                 model2.addConstrs(((x[path[i]]+x[path[i+1]]+x[path[i+2]]+x[path[i+3]]) <= 3) for i in (range(len(path)-3)))
             model2.optimize()
             optSolution2 = [k for k, v in x.items() if v.X > 0]
             optPaths = computePaths(optSolution2)
             longerPaths = []
             for path in optPaths:
                 if len(path) > 3:
                     longerPaths.append(path)
```