**Management Science Group Assignment:**

**Kidney Transplant Management**

Lieke de Jong: 632256

Eran Kan Kohen: 523228

Martijn van Heijningen: 480800

Thirza Schuilwerve: 634674

**Question 1.1**

Mathematical notations:

0: otherwise

1: donor i gives kidney to receiver j

1a

Constrains:

1b

1c

1d

1e

1f

Blood type constrains:

1g

1h

1i

1j

The following paragraph explains the objective and constraints described above:

1a states the sum of the donations from donor to receiver, and this needs to be maximized.

1b is the constraint which states that a pair will donate a kidney only if they receive one. Though, a receiver of a pair can also receive a kidney from an unpaired donor, without the donor of the pair needing to give a kidney. So, the outflow of what a donor gives away in a pair can be smaller or equal to what a receiver of a pair gets.

The constraint in 1c says that a donor and recipient can donate or receive at most one kidney.

1d describes that x is a binary variable. 1e describes that the absolute value of the difference in age can at most be 10 years. For constraint 1f, the difference between locations of the donor and receiver can at most be 300 miles.

The blood type constraints are in 1g, h, i and j. Constraint 1g states that if the blood type of the donor is A the blood type of the receiver must be A or AB. In addition, 1h means that if the blood type of the donor is B the blood type of the receiver must be B or AB. Additionally, 1i states that if the blood type of the donor is AB the blood type the receiver must be AB as well. Finally, the constraint 1j means that i the blood type of the donor is O, the blood type of the receiver must be A, B, AB or O.

**Question 1.2**

*Figure 1.2.1,* presented below, shows the python code used to prepare the data. The first step was to create a list of all the possible combinations based on a few constraints discussed in section 1.

First, all the blood type combinations are created. For example, someone with blood type A can donate to someone with blood type A and AB. Subsequently, all the IDs in the pair data set are separated into two lists. The first list is all\_donors which consists of the IDs of the donors and pairs, and the second list is all\_receiver, which consists of the IDs of the receivers and pairs. At last, the blood type, location and age of both donor and receiver are found. Adding the blood type, age and location constraints gives all possible combinations of donors and receivers, which are saved in the compatible list.

*Figure: 1.2.1*

Graphical user interface, text, application, email

Description automatically generated

**Question 1.3**

*Figure: 1.3.1*

*Graphical user interface, text, application

Description automatically generated*

The Gurobi code above maximises the number of possible transplants using the list of *compatible* donor-receivers based on the constraints listed in section 1.1 and explained in 1.2.

The model builds upon the compatible list by also adding the constraint that each pair can only donate a kidney if the pair receives one. However, also taking into account that the pair can also receive from an independent donor. Additionally, a donor can only donate one kidney and the receiver can also only receive one kidney.

After implementing these constraints, the model is run and the optimal (maximum) solution is found.

**Question 1.4**

The code to used to find the correct answers to these questions can be found in Appendix A.

1. According to the optimal solution using Gurobi, 94 transplantations will take place.
2. Of the paired patients, 78.1% will receive a donor. Of the unpaired patients, 23.8% will receive a donor.
3. 91.7% of the un-paired donors donate a kidney.
4. As seen in the figure below, 81 (86.2%) paired patients have the same blood type, and 13 (13.8%) have different blood types. See figure 1.4.1.

*Figure 1.4.1*

Chart, bar chart

Description automatically generated

1. The graph below (figure 1.4.2) shows the percentage of paired patients that receive a kidney for each city. The highest percentage is in Manchester, where 100% of the patients were paired. The city with the lowest percentage of paired patients was Madrid with 10%.

*Figure 1.4.2*

*Chart, bar chart

Description automatically generated*

**Question 1.5**

The code is used to find the correct answers to these questions can be found in Appendix A, however, do not that the following constraint was adjusted:

*Distant constraint changed to max 500 miles*

Dlocation- Rlocation ≤500 miles

1. Using Gurobi, and adjusting the code for compiling the compatible donors and recipients, the new optimal number of transplants is 105.
2. Percentage of paired and un-paired who will receive a kidney:
   1. Paired: 27.5% (0.0.275)
   2. Un-Paired: 86.46% (0.8646)
3. Percentage of un-paired donors that will receive a kidney: 95.83% (0.9583)
4. Percentage of paired patients that receive a kidney from a different blood type: 10.48% (0.1048)

*Figure 1.5.1*

Chart, bar chart

Description automatically generated

1. The graph below shows the percentage of paired patients that receive a kidney for each city. The highest percentage is in Manchester, where 100% of the patients were paired. The city with the lowest percentage of paired patients was Vienna with 36.36%.

*Figure 1.5.2*

Chart, bar chart

Description automatically generated

**Question 2.1**

The second question looks at the longest chain of donations in the optimal solution. For loops were coded to format the output and track chains of donations.

A picture containing text

Description automatically generated

The result revealed that the longest chain had 7 transactions.

Text

Description automatically generated with medium confidence

**Question 2.2 & 2.3**

After getting the longest loop, another for loop was coded to detect all chains which include more than 3 transactions.

Graphical user interface, text, application

Description automatically generated

Later, an iterative function was defined to add constraints to the Gurobi model. The way the function works is quite simple. It optimizes a model, checks if there are chains longer than 3 transactions and if such chains exists, it adds them to the “chains\_bigger\_than\_three list”. Afterwards, it adds the constraint that 4 transactions which happen after each other in a chain can’t all be happening. The function loops over itself until it adds all of the chain constraints and finally finds a solution which does not have a chain longer than 3 transactions.

A picture containing text

Description automatically generated

The result was very interesting. The optimal solution was 94 transactions and there were 4817 constraints added to the model!

**Appendix**

**Graphical user interface, text, application

Description automatically generated**

**Text

Description automatically generated**

**A picture containing text

Description automatically generated**

**Table

Description automatically generated**

**Table

Description automatically generated**

**Text

Description automatically generated with low confidence**