Assignment 3: Improving kidney Exchange Programs [*]

Emanuel Tomé [†], João Ferreira [‡], Ricardo Faria [§]

University of Porto, June 2021

1 Introduction

Renal diseases affect thousands of patients all over the world, who, to survive must be subjected to dialysis, a costly treatment with several negative consequences in their quality of life. As an alternative, patients may receive by transplantation a kidney. Transplantation yields longer survivability and better quality of life, while dialysis is more expensive than transplantation.

The patients can receive a kidney from a deceased donor or from a living donor. However, from deceased donors there is a very large waiting list (5 years or more of waiting). The alternative is receiving from a living donor, usually relatives, spouse, friends or altruistic donors, who volunteers to cede one of his kidneys. However, is some situations transplantation is not possible due to blood or tissue-level incompatibility. In those cases, a donor-patient pair (D_1, P_1) may enter in a pool of pairs in the same situation. If another pair (D_2, P_2) is in the same situation, they may do crossed transplants if D_2 is compatible with P_1 and D_1 with P_2. This exchanges are not limited to two pairs and any cycle in the compatibility graph is a possible exchange. However, the number of possible simultaneous operations in a hospital usually limit the size of the exchanges to three.

In this context, two pools of pairs donor-patient are used. The first, is the historical of previous kidney transplants and the respective survival time. This dataset was used to build a machine learning model to predict the survival time of a patient, given as input to the model health data of the donor and patient. This model was then used to predict the survival time of the possible matches in a pool of pairs donor-patient. Then, an optimisation model was used to assign the donors to patients having as objective maximise the survival time.

2 Exploratory data analysis

In this assignment two different datasets were used. The first dataset, **historical.csv**, has information of past transplants and contains 19 variables and 5000 observations for both donors and patients. Five variables are binary (Donor/ Patient Sex, Donor/ Patient Smoker - 1 if true, Donor Relat - 1 if donor is a relative of patient), six are categorical (Donor/ Patient ABO - ABO blood type, Donor/ Patient hlaB - human leukocyte antigens B, Donor/ Patient hlaDR - human leukocyte antigens DR) and the remaining ones (Donor/ Patient Age, Donor/ Patient Weight, Donor EGFR - Estimated Glomerular Filtration Rate, Donor SBP - Systolic Blood Pressure, Donor BMI - Body Mass Index, Survival Time) are numeric.

For numeric variables is shown in table 1 the mean, standard deviation, min, max and the three quantities for the historical dataset. It can be noticed that none of the variables change sign, being all of them positive. For that reason, the variables were scaled between 0 and 1 to feed the machine learning models. In table 2 is shown the number of unique values, the most common value and the frequency of the most common value for categorical variables of the historical dataset. The second dataset, **pool.csv**, contains the current pool of donor-patient pairs. It has 100 observations and the same variables apart of *Survival Time*, which is the variable to be predicted by the machine learning fitted in the next section. Their descriptive statistics can be found in Appendix A. None of the datasets had missing values. Finally, it should be referred that the categorical variables d_abo and d_hlaB were encoded as is demonstrated in Table 3 for variable d_abo and that the variable Donor Relat was not used in the fitted machine learning models.

[*]This work was submitted on the framework of the course Data-Driven Decision Making

[†]Emanuel Tomé is a student at the Faculty of Sciences of the University of Porto. Currently, he is enrolled in the 1st year of the Master's in Data Science (e-mail: up200702634@edu.fc.up.pt).

[‡]João Ferreira is a student at the Faculty of Sciences of the University of Porto. Currently, he is enrolled in the 1st year of the Master's in Data Science (e-mail: up202004115@edu.fc.up.pt

[§]Ricardo Faria is a student at the Faculty of Sciences of the University of Porto. Currently, he is enrolled in the 1st year of the Master's in Data Science (e-mail: up202004105@edu.fc.up.pt

Table 1: Descriptive statistics of the numerical variables of the Historical Dataset.

	d_age	**d_egfr**	**d_sbp**	**d_bmi**	**d_smoker**	**d_relat**	**d_weight**	**p_age**	**p_smoker**	**p_weight**	**Longevity**
mean	46.3	92.5	139.9	27.5	0.30	0.15	68.1	46.5	0.30	68.3	10.7
std	17.0	19.4	36.3	8.1	0.46	0.35	16.4	17.0	0.46	16.5	7.6
min	13	55	80	10	0	0	30	13	0	30	0
25%	32	76	110	21	0	0	56	32	0	56	4
50%	46	92	140	28	0	0	68	46	0	68	10
75%	61	109	170	34	1	0	80	61	1	81	16
max	80	130	200	45	1	1	110	80	1	110	36

Table 2: Descriptive statistics of the categorical variables of the Historical Dataset.

	d_sex	d_abo	d_hlaB	d_hlaDR	p_sex	p_abo	p_hlaB	p_hlaDR
unique	2	4	4	4	2	4	4	4
top	F	O	o	ab	F	O	b	o
freq	2736	3703	1270	1263	2742	1996	1262	1278

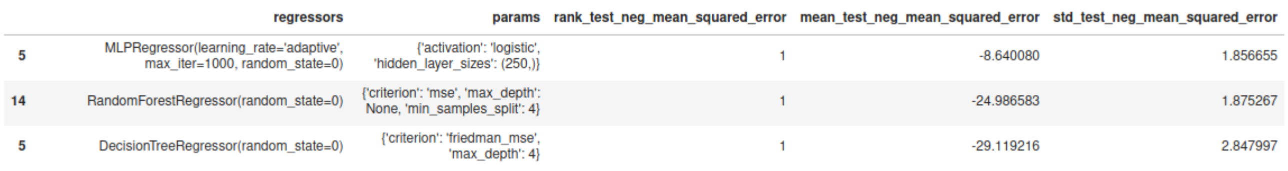
Table 3: Example of One Hot Encoding for Donor Blood Type

d_ABO	d_abo_A	d_abo_B
A	1	0
B	0	1
AB	1	1
O	0	0

3 Machine Learning model to estimate the survival time

In order to fit a machine learning model to predict the survival time, we start by separating the historical dataset in 70% of the data points for training test and the remaining 30% for the testing dataset. We then performed a a grid search 10-fold cross-validation using three machine learning models: Multi-layer Perceptron (MLP), a Random Forest (RF) and a Decision Tree (DT). It should be referred that all variables were rescalled to be in the range 0-1 before they feed the machine learning algorithms. It should be said also that the variable Donor Relat was not used to train the model. The used loss function was the *Mean Squared Error* in order to penalise more models with a higher amount of larger errors. The hyperparamenters used in the Grid search can be found in Appendix B.

In Figure 1 are presented the results of the performed grid search, where the chosen hyperparamenters as well as their mean squared error are presented for each family of algorithms considered. We can notice that the best model is the MLP with the logistic function as activation function and an hidden layer with 250 nodes. The computation of this grid search took about 828 seconds in a Virtual Machine with an Intel Core i7-8565U CPU @1.80GHz, 14.6GB of RAM running Ubuntu 20.04.2 LTS. This includes all the training steps and all as well as all the sequential programming executed in the making of the grid. It should also be said that we also tested a ensemble model with the 3 models presented in Figure 1. However, lower mean square error was obtained both in cross-validation and in the test set.



	regressors	params	rank_test_neg_mean_squared_error	mean_test_neg_mean_squared_error	std_test_neg_mean_squared_error
5	MLPRegressor(learning_rate='adaptive', max_iter=1000, random_state=0)	{'activation': 'logistic', 'hidden_layer_sizes': (250,)}	1	-8.640080	1.856655
14	RandomForestRegressor(random_state=0)	{'criterion': 'mse', 'max_depth': None, 'min_samples_split': 4}	1	-24.986583	1.875267
5	DecisionTreeRegressor(random_state=0)	{'criterion': 'friedman_mse', 'max_depth': 4}	1	-29.119216	2.847997

Figure 1: Grid Search CV results.

We then fitted the MLP model with the obtained best hyperparameters to the whole train set, which took about 148.3 seconds in a Virtual Machine with an Intel Core i7-8565U CPU @1.80GHz, 14.6GB of RAM running Ubuntu 20.04.2 LTS. We then obtained a mean square error of 8.60 and mean absolute error of 1.90 in the test set. We also performed a visual validation of the model predictions in the test set, where the model seemed to have a good behaviour. Since the model seems to be appropriate and give good results, we fitted the MLP model with the chosen hyperparameters using the all historical dataset. This is the model that will be used to predict the survival time of each patient in the next section. The training of this model took about 192.5 seconds in a Virtual Machine with an Intel Core i7-8565U CPU @1.80GHz, 14.6GB of RAM running Ubuntu 20.04.2 LTS.

4 Optimisation problem to find the best donor-patient matching

4.1 Problem formulation

The problem to be optimised can be stated as follows: given a set of V pairs donor-patient, find a subset of pairs donor-patient in order to maximise the survival time of the patients. The pairs donor-patient can be in cycles with the maximum of k vertices. In this work it will be presented results for k varying from 1 to 4, although our final proposal will be made for $K = 3$ since this is usually the maximum number of possible simultaneous operations in a hospital. The problem data is synthesised in Table 4.

Table 4: Problem data.

V	Set of pairs Donor-Patient (vertices in the original graph)

The variables of the problem are then the maximum size of a cycle (k), the number of cycles (m), the set of cycles (C), the vertices in each cycle c ($Cycle_c$), the loads (survival time) of each cycle (w_c), which is the sum of all the survival times of all patients in the cycle (given by the machine learning model described in previous section), an the binary variable x_c that is 1 if the cycle c is chosen and 0 otherwise. Note that the variables m, C, $Cycle_c$ and w_c depend on the choice of the maximum size of a cycle (k). We then want to maximise the survival time of all patients considered to receive a transplant, that is, we want to maximise $\sum_c w_c x_c$, having the restriction that all vertices i in the graph can only belong at most to one cycle. In other words, the summation over all the selected cycles in solution is limited to one ($\sum_{c:i\in c} x_c \leq 1$ for $\forall i$). To summarise, the variables and formulation are as follows:

Variables:

k		Maximum size of a cycle
m		Number of cycles
C		Set of cycles
$Cycle_c$	$c \in C$	Vertices in each cycle c
w_c	$c \in C$	Loads (survival time) of each cycle c
x_c	$c \in C$	1 if the cycle c is chosen, 0 otherwise

Formulation:

$$\text{maximise: } \sum_c w_c x_c \tag{1}$$

$$\text{subject to: } \sum_{c:i\in c} x_c \leq 1 \qquad \forall i \tag{2}$$

$$x_c \in 0, 1 \qquad \forall c \tag{3}$$

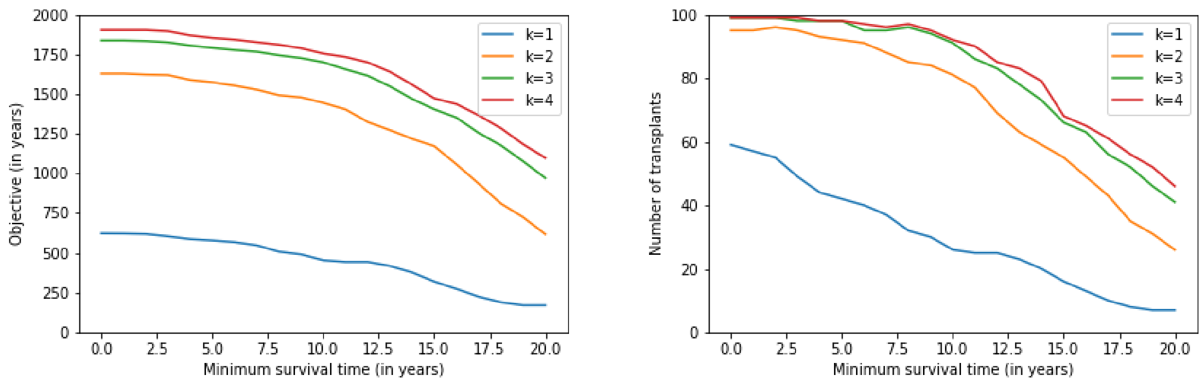
The AMPL source code for the model can be found in section C.1 of the Appendix C.

4.2 Data preparation

Before running the optimisation problem, we had to do some data preparation to feed the AMPL model. Therefore, we started by computing the compatibilities between all the donors and patients, taking in consideration the blood type compatibilities. The source code used for that can be found in section C.2 of the Appendix C. We obtained 6299 compatible pairs. We then computed the survival time for each compatible pair using the machine learning model described in the previous section, using the code presented in the section C.3, Appendix C. We then computed the cycles using the functions get_all_cycles and cycles_line presented in section C.4. After that, the loads of each cycle were computed using the function determine_loads (Appendix C.5. The function kep_cycle (section C.4, Appendix C) is then used to obtain the optimal solution using AMPL. We also implemented two more functions: remove_cycles (section C.6) and remove_cycles_less_than_thr (section C.7). The former is used to remove cycles where the survival time is bigger if the patient receives the kidney from his own donor. This function was used in all the conducted optimisations since from our point of view it does not make sense a pair to be in the pool if their survival time is lower than they were not. The latter function is used to remove cycles where the survival time for one of the patients in the cycle is lower than a given threshold. This function was used to do a sensitivity analysis presented in the next section and also for our final solution, where a threshold of 8 years was defined.

4.3 Solution

In order to gain sensitivity to the variation of k (maximum number of points in a cycle) and to the threshold related with the minimum number of years that a transplanted patient should survive, we conduced a sensitivity analysis varying k between 1 and 4 and the threshold between 0 and 20 years. In Figure 2 are presented the variation of the objective in the optimisation as well as of the number of transplants with the minimum number of years that a transplanted patient survive for different values of k. For the analysis of those figures, one can notices that, as expected, the number of transplants increase with k. Moreover, the number of transplants as well the objective of the optimisation problem (maximise the total number of years of the transplanted patients) decreases with the augment of the threshold related with the minimal number of survival time. Indeed, both the number of transplants and the objective seems to be more or less constant for a threshold between 0 and 8 years, starting to have a higher decrease for a threshold higher than about 8 years. For that reason, we propose a solution using a threshold of 8 years. The optimum number of survival time was 1743 years and the final solution has 38 cycles (5 with one pair donor-patient, 8 with two pairs donor patient and 25 with three pairs donor-patient) and 95 transplants. It took about 7.9s to compute the optimum in a Virtual Machine with an Intel Core i7-8565U CPU @1.80GHz, 14.6GB of RAM running Ubuntu 20.04.2 LTS. We should notice, however, that some of the taken choices to obtain our solution may be validated both by specialists in the domain and by an ethics committee before it can put in practice.



(a) (b)

Figure 2: Variation of the objective (a) and of the number of transplants (b) with the minimum number of years that a transplanted patient survive for different values of k.

5 Conclusions

The kidney exchange problem is of superior importance in our society. In order to solve this problem in the given pool of donor-patient pairs, we maximised the predicted survival time of each transplanted patient. To predict the survival time of each possible combination between a donor and a patient in the pool, we used a machine learning model trained in a historical dataset of past transplants. Three families of algorithms were tested in the historical dataset, namely Multi-layer Perceptron, (MLP) Random Forest and Decision Tree. The best hyperparameters were determined using 10-fold cross validation and the model with lower mean squared error was an MLP with one hidden layer of 250 nodes with the logistic activation function. An ensemble with the best models of those three families was also tried, however a lower mean square error was obtained using only the MLP model.

The MLP model was then used to predict the survival time of each possible compatible combination in the pool dataset. The optimisation problem formulation was detailed described as well as all the used code to handle the data was presented. It should be noticed that our solution contemplates at the maximum 3 nodes in a cycle and only have pairs where it is predicted by our machine learning model that the transplanted patient will live at least 8 years. This threshold was chosen through a parametric analysis. Moreover, cycles where a patient is predicted to have a survival time lower than receiving from his own donor were not considered in the pool. We should emphasise that those choices and assumptions should be validated both by specialists in the domain and by an ethics committee before it can put in practice.

Appendixes

A Exploratory data analysis

A.1 Descriptive statistics of the pool dataset

Table 5: Descriptive statistics of the numerical variables of the Pool Dataset.

	d_age	d_egfr	d_sbp	d_bmi	d_smoker	d_relat	d_weight	p_age	p_smoker	p_weight
mean	48.1	93.2	141.8	27.7	0.25	0.13	67.1	45.2	0.22	69.4
std	16.4	18.7	36.7	8.6	0.44	0.34	16.7	15.7	0.42	16.0
min	18	60	80	12	0	0	30	18	0	40
25%	34	79	105	21	0	0	56	32	0	57
50%	49	89	145	29	0	0	67	47	0	68
75%	61	110	170	34	1	0	78	59	0	83
max	76	129	200	44	1	1	104	75	1	105

Table 6: Descriptive statistics of the categorical variables of the Pool Dataset.

	d_sex	d_abo	d_hlaB	d_hlaDR	p_sex	p_abo	p_hlaB	p_hlaDR
unique	2	4	4	4	2	4	4	4
top	F	O	a	o	F	O	a	a
freq	58	44	30	28	50	36	27	33

B Machine Learning Model

B.1 Grid search parameters

Multi-layer Perceptron:

- Number of hidden layers: 1
- Hidden layer sizes: 150, 200 and 250
- Activation function: identity, logistic, tanh and relu

Random Forest:

- Number os estimators: 100
- Split criterion: mse and friedman_mse
- Max depth: 3, 4, 10 or none

Decision Tree:

- Split criterion: mse
- Max depth: 3, 4, 10 or nonr
- minimum number of samples required to split an internal node: 2, 3, 4 and 5

C Optimisation problem

C.1 AMPL model

```
set V; # vertices in the original graph
param m; # number of cycles
set C := 0..m-1;
set CYCLE{C}; # vertices in each cycle
param loads{C};

var x{C} binary;
```

```

maximize z: sum {c in C} loads[c] * x[c];

Packing {i in V}: sum {c in C: i in CYCLE[c]} x[c] <= 1;
```

C.2 Code snippet for computation of the adjacency list

```python
# Computation of the adjacency list

compatibilities = {} # key is the donnor
for k1 in range(1,pool.shape[0]+1):

    # If donor is from blood type 'O'
    if pool.loc[k1,'d_abo']=='O':
        compatibilities[k1] = list(range(1,pool.shape[0]+1))

    # If donor is from blood type 'A'
    elif pool.loc[k1,'d_abo']=='A':
        compatibilities[k1] = list(pool.loc[np.logical_or(pool.loc[:,'p_abo']=='A', pool.loc
    [:,'p_abo']=='AB')].index)

    # If donor is from blood type 'B'
    elif pool.loc[k1,'d_abo']=='B':
        compatibilities[k1] = list(pool.loc[np.logical_or(pool.loc[:,'p_abo']=='B', pool.loc
    [:,'p_abo']=='AB')].index)

    # If donor is from blood type 'AB'
    else:
        compatibilities[k1] = list(pool.loc[pool.loc[:,'p_abo']=='AB'].index)
```

C.3 Source code for computation of the prediction of the survival time for each possible pair donor-patient

```python
columns_donor=['d_age', 'd_sex', 'd_egfr', 'd_sbp', 'd_bmi', 'd_smoker',
       'd_weight', 'd_abo_A', 'd_abo_B', 'd_hlaB_a', 'd_hlaB_b', 'd_hlaDR_a',
       'd_hlaDR_b']
columns_patient=['p_age', 'p_sex', 'p_smoker', 'p_weight', 'p_abo_A',
       'p_abo_B', 'p_hlaB_a', 'p_hlaB_b', 'p_hlaDR_a', 'p_hlaDR_b']

dict_st = {} #key is the donnor

t0 = time.time()
for k_donor in compatibilities.keys():
    df = pd.DataFrame()
    for k_patient in compatibilities[k_donor]:
        df = pd.concat((df,pool_2.loc[k_donor,columns_donor].append(pool_2.loc[k_patient,
    columns_patient]).to_frame().transpose()))

    dict_st[k_donor]=list(rgressor_final.predict(scaler_minmax_all.transform(df)))

t1 = time.time()
print(f'[INFO] Computation time: {t1-t0}s')
```

C.4 Source code with the cycle formulation functions (cycle_formulation.py)

```python
from amplpy import AMPL, Environment, DataFrame

def kep_cycle(adj, cycles, ampl_model="kep.mod", loads=[]):
    ampl = AMPL(Environment())
    ampl.option['solver'] = 'gurobi'
    ampl.read(ampl_model)
    ampl.set['V'] = list(adj) # vertices are keys in 'adj'
    ampl.param['m'] = len(cycles)
    loads_ampl = ampl.getParameter('loads')
    loads_ampl.setValues(loads)

    for i,c in enumerate(cycles):
        ampl.set['CYCLE'][i] = list(c)


    ampl.solve()
    z = ampl.obj['z']
    print("Optimum number of survival time:", z.value())

    print("Selected cycles:")
    x = ampl.var['x']
    sol = []

    for i,c in enumerate(cycles):
        if x[i].value() > 0.5:
            sol.append(c)

    return sol, z.value()

def get_all_cycles(adj,K):
    tovisit = set(adj.keys())
    visited = set()
    cycles = set()
    loads = []
    for i in tovisit:
        tmpvisit = set(tovisit)
        tmpvisit.remove(i)
        first = i
        all_cycles(cycles,[],first,tmpvisit,adj,K)
    return cycles

def normalize(cycle):
    cmin = min(cycle)
    while cycle[0] != cmin:
        v = cycle.pop(0)
        cycle.append(v)

def all_cycles(cycles, path, node, tovisit, adj, K):
    for i in adj[node]:
        if i in path:
            j = path.index(i)
            cycle = path[j:]+[node]
            normalize(cycle)
            cycles.add(tuple(cycle))
        if i in tovisit:
            if K-1 > 0:
                all_cycles(cycles,path+[node],i,tovisit-set([i]),adj,K-1)
    return cycles

def cycles_line(adj):
    donors = set(adj.keys())
    cycles = set()
    for donor in donors:
        if donor in adj[donor]:
            cycles.add((donor,))
    return cycles
```

C.5 Source code of the function to determine the loads (survival time) of each cycle

```
# function to determine the survival time of each cicle. Returns a list with the survival time
def determine_loads(cycles, compatibilities, dict_st):
    loads = []
    for cycle in cycles:
        load = 0
        for i, point in enumerate(cycle):
            if i==(len(cycle)-1):
                load += dict_st[point][compatibilities[point].index(cycle[0])]
            else:
                load += dict_st[point][compatibilities[point].index(cycle[i+1])]

        loads.append(load)
    return loads
```

C.6 Source code of the function remove_cycles

```
# Function to remove cycles that implies a survival time lower than the st if the patient
    recieves from his own donor
def remove_cycles(cycles_pair, cycles, compatibilities, dict_st):
    cycles_to_remove = set()
    for cycle in cycles_pair: # loop over all pairs that donor gives to his patient
        # Survival time of the patient if receives from his donor
        st_cycle = dict_st[cycle[0]][compatibilities[cycle[0]].index(cycle[0])]

        for cycle_all in cycles: # Loop over all cycles
            if cycle[0] in cycle_all:
                index = cycle_all.index(cycle[0])-1 # index of the donor in this cycle to the
    patient in cycle(first loop)
                donor = cycle_all[index]

                index_for_st=compatibilities[donor].index(cycle[0])
                st_in_this_cycle = dict_st[donor][index_for_st]

                if st_cycle>st_in_this_cycle:
                    cycles_to_remove.add(cycle_all)

    return cycles-cycles_to_remove, cycles_to_remove
```

C.7 Source code of the function remove_cycles_less_than_thr

```
# Function to remove cycles that implies a survival time lower than the st if the patient
    recieves from his own donnor
def remove_cycles_less_tan_thr(cycles, thr, compatibilities, dict_st):
    cycles_to_remove = set()
    for cycle in cycles: # loop over all pairs that donor gives to his patient
        for i, point in enumerate(cycle):
            if i==(len(cycle)-1):
                load = dict_st[point][compatibilities[point].index(cycle[0])]
            else:
                load = dict_st[point][compatibilities[point].index(cycle[i+1])]

            if load<thr:
                cycles_to_remove.add(cycle)

    return cycles-cycles_to_remove, cycles_to_remove
```

C.8 Variation of the number o cycles with the minimum number of years that a transplanted patientsurvive

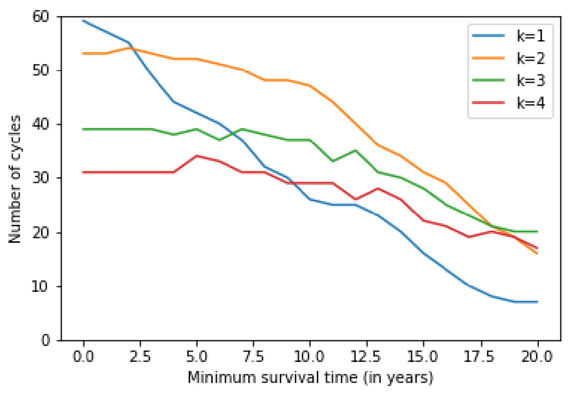


Figure 3: Variation of the number o cycles with the minimum number of years that a transplanted patient survive.