Matching mechanisms for kidney transplantations

Tristan François

January 31, 2021

1   Two Simple Approaches

Question 1

Algorithm 1: Direct Donation

Input: A number n of patient and, for all patients $i \in \mathbb{N}$, their set of compatible donors $K_i \subseteq [\![1, n]\!]$.

Output: A set of pairs donor-patient pairs c and a waiting list w.

1 **initialization**
2 | $c = \emptyset$;
3 | $w = [\,]$;
4 **begin**
5 | **for** $i = 1$ **to** n **do**
6 | | **if** $i \in K_i$ **then**
7 | | | $c = c \cup \{(i, i)\}$;
8 | | **else**
9 | | | $w.\text{add}(i)$
10 | **return** c, w

The implementation in python is pretty straightforward. The python function takes as parameter an integer n and an integer set array of size n. The use of python set allows to test the if condition in constant times. The only difference with pseudo-code is the use of a list rather than a set for the variable c. This choice was made to avoid the unnecessary heaviness of the set data structure, but does not change the complexity.

Question 2

Our matching algorithm must prioritize the patients with the highest priority. It must also ensure that all patients will leave with a donor that suited them better than their paired donor and the waiting list, otherwise some patients may not participate in the exchange program.

Our greedy algorithm therefore consists of browsing the patients in descending order of priority and matching them to their preferred donor whose paired patient does not prefer the waiting list or his donor to the donor of the priority patient.

Algorithm 2: Greedy Matching

Input: A number n of patient, a strict priority list of the patients U and, for all patients $i \in \mathbb{N}$, their set of compatible donors $K_i \subseteq [\![1, n]\!]$ and their strict preference relation P_i over $K_i \cup \{k_i, w\}$. The priority list is an integer list that starts with the index of the highest priority patient and goes to the index of the lowest priority patient.

Output: A matching M.

1 initialization
2 $M = \emptyset$;
3 $is\_matched = Boolean[1..n]$;
4 $sorted\_K = IntegerList[1..n]$;
5   **for** $i = 1$ **to** n **do**
6   is_matched[i] = False;
7   sorted_K[i] = sort(K_i, P_i) ;   // $\mathcal{O}(n \log(n))$

8 begin
   /* Iterate over all patient by decreasing priority   */
9   **for not** $U.is\_empty()$ **do**
10   u = U.pop();
11   **if not** $is\_matched[u]$ **then**
12   **while not** $sorted\_K[u].is\_empty()$ **do**
13   v = sorted_K[u].pop();
14   **if not** $is\_matched[v]$ **then**
           /* If t_v prefer k_u over k_v and w */
15   **if** $t_v\,P_v\,t_u$ **and** $t_v\,P_v\,w$ **then**
16 $M = M \cup \{((k_u, t_u), (k_v, t_v))\}$;
17   is_matched[u] = True;
18   is_matched[v] = True;

19   **return** M

We start by initializing an *is_matched* array to keep track of which pair has already been matched and an integer list array *sorted_K* that contains, for each patient of index i, the indexes of the preferred donors in descending order of preference. The sort function returns a list of the elements of K_i sorted in descending order according to the preference relation P_i.

2 Efficient stategy-proof exchange mechanism

Question 3

Let (k_1, t_1) be a pair of donor-patient and assume there is no cycle. Consider the chain starting from (k_1, t_1). Since there is a finite number n of donor-patient pairs and there is no cycle, the chain cannot be infinite. Finally, since the only way for a chain to stop is that w belong to the chain, (k_1, t_1) is indeed a tail of a w-chain.

Thus either there exists a cycle, or each pair is the tail of some w-chain.

Question 4

Question 5

Let's run the Cycles and Chains Matching Algorithm with rule A on this example. Graphs associated with every round are shown in figure 1.

The initial graph contains a cycle $C_1 = (k_3, t_3, k_2, t_2, k_{11}, t_{11})$. The first round of the algorithm therefore consists in detecting this cycle, assigning the kidneys to the patients and then eliminating the cycle.

After removing the first circle, a new circle $C_2 = (k_5, t_5, k_7, t_7, k_6, t_6)$ is formed. In the second round, this cycle is therefore deleted.

At the start of the third round, there is no more circle. The graph contains longest w-chains : $W_1 = (k_8, t_8, k_4, t_4, k_9, t_9)$ and $W_2 = (k_{10}, t_{10}, k_1, t_1, k_9, t_9)$. Since the highest priority patient t_1 is only present in W_2, W_2 is chosen. The kidneys are assigned and the chain remains in the graph for the next round but patients pointing to other kidneys than k_10 in the chain are redirected.

After the redirect, a new circle $C_3 = (k_4, t_4, k_8, t_8)$ has formed. The fourth round therefore consists in the elimination of this cycle.

There is no new circle and the patient t_{12} now points to k_{10} forming a chain of maximum length $W_3 = (k_{12}, t_{12}, k_{10}, t_{10}, k_1, t_1, k_9, t_9)$. Round 5 assigns kidneys k_{10} to patient t_{12} and there are no more patients with no kidneys. The algorithm stops.

The final matching is:

$$\begin{pmatrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} & t_{11} & t_{12} \\ k_9 & k_{11} & k_2 & k_8 & k_7 & k_5 & k_6 & k_4 & w & k_1 & k_3 & k_{10} \end{pmatrix}$$
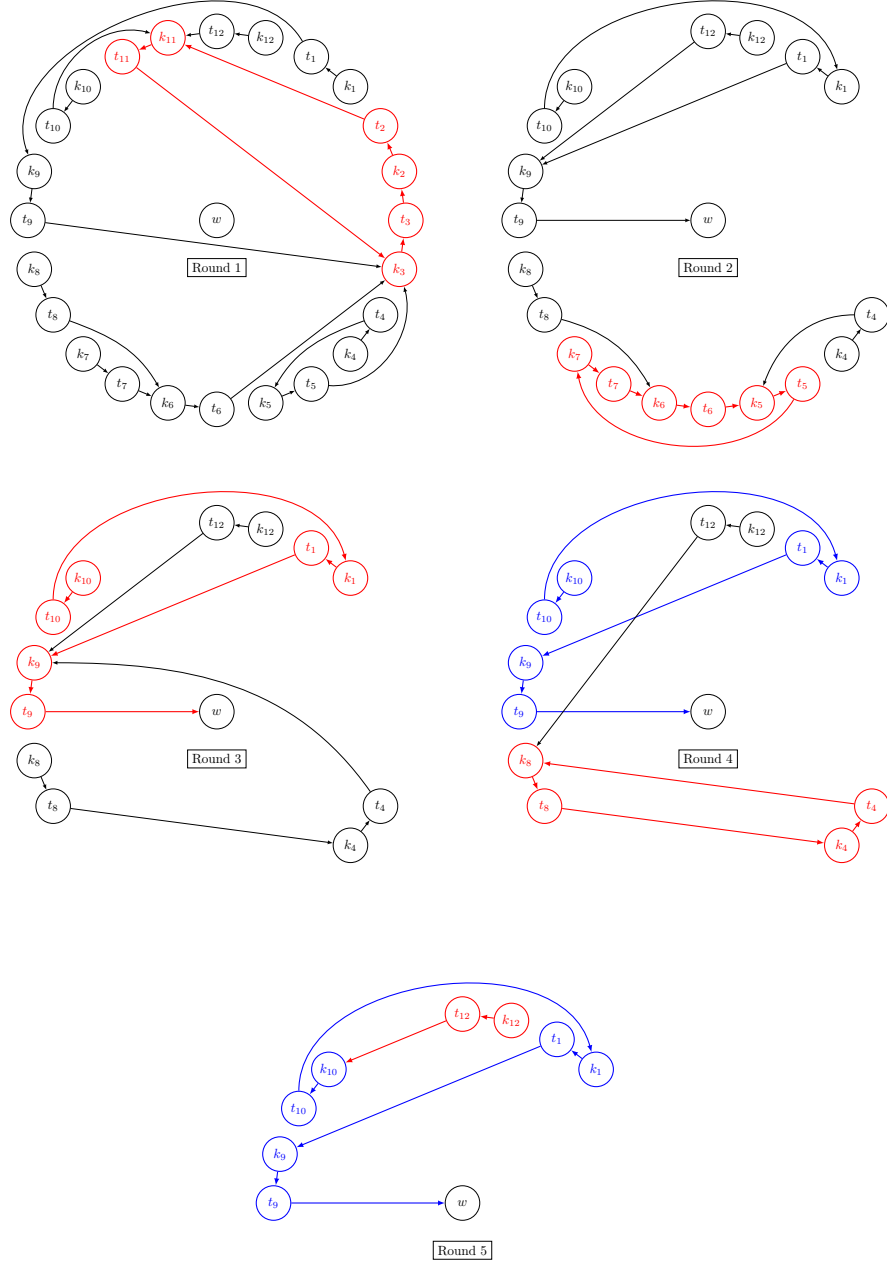
Figure 1: State of the graph at every round.

4

Question 6

If the selection rule keep in the procedure any selected w-chain at a non-terminal round, then every unasigned kidney is available at every round of the procedure. Therefore, at every round of the algorithm, a patient can only be assigned to his best choice among remaining kidneys. The only way to give him a better kidney is to take it back from a patient whose assignement occurs in a previous round. Since this patient had his best choice at the time, he will strictly disprefer the new matching. Thus, the exchange mechanism is efficient.

Question 7

If we swap k_8 and k_9 in the order of preference of the twelfth patient, then in round 3 the longest w-chain becomes $(k_{12}, t_{12}, k_8, t_8, k_4, t_4, k_9, t_9)$ and t_{12} is assigned to k_8, which he strictly prefers to k_{10}, that he would get by being honest about his preferences. Therefore the exchange mechanism with chain selection rule A is not strategy-proof.
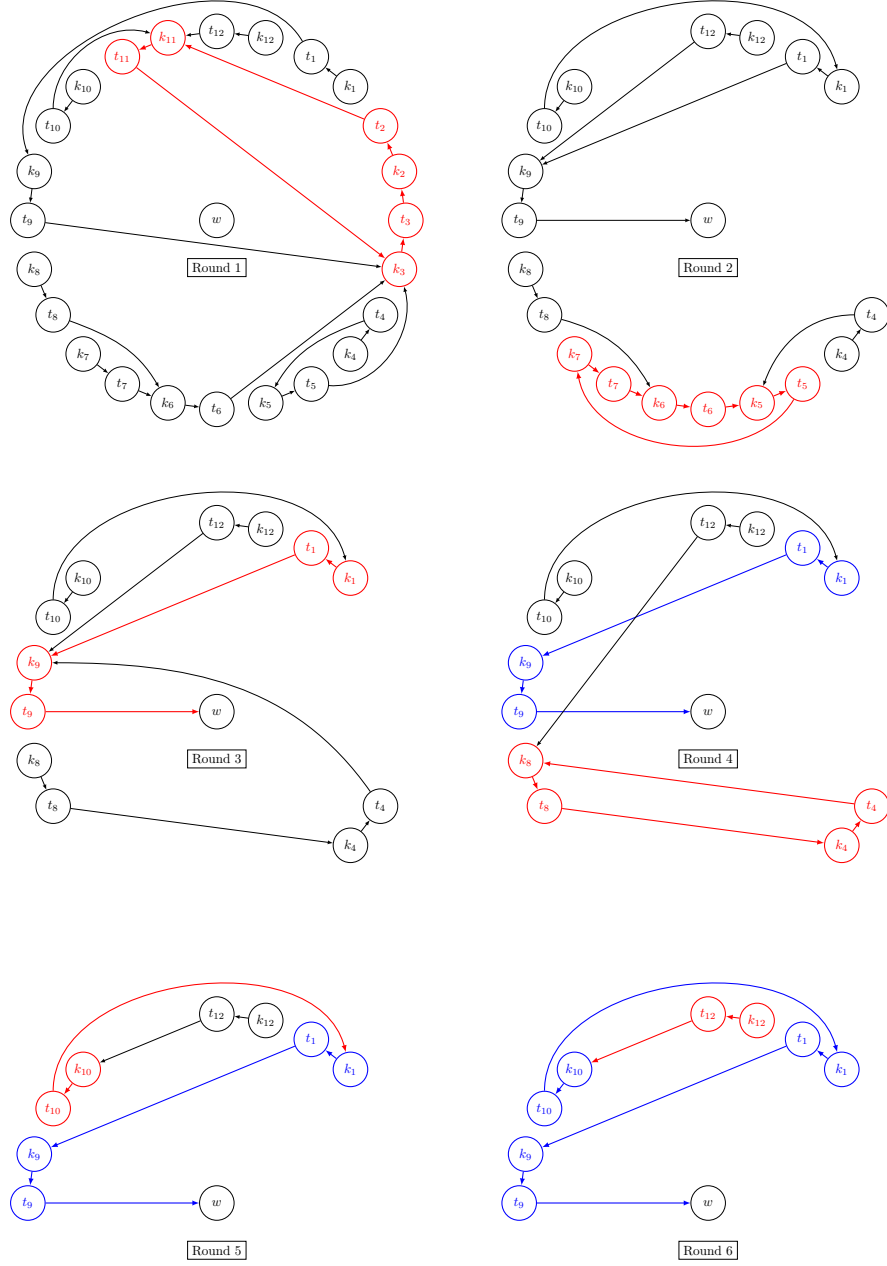
Question 8

Figure 2: State of the graph at every round.

3 Integer programming formulation

Question 9

In order to list all the minimal infeasible paths, we will simply use a recursive function `Aux` which, given a graph G, a vertex v and an integer k, returns all the paths of lengths $k + 1$ starting at v. In order to avoid cycles, we add an *inpath* variable, a set containing the vertices already in the current path. Finally, we called this function on all the vertices of the graph.

Algorithm 3: Minimal infeasible paths

Input: $G = (V, E)$ a graph and k the threshold.
Output: The set of all minimal infeasible paths.

1 begin
2 $paths = \emptyset$;
3   **for** $v \in V$ **do**
4 $paths = paths \cup \{\text{Aux}(G, v, k, \emptyset)\}$;
5   **return** $paths$;

   /* Return the list of all minimal infeasible path starting
     from v */

6 Function `Aux`(G, v, k, *inpath*)**:**
7 $inpath = inpath \cup \{v\}$;
8   **if** $k = 0$ **then**
9   **return** $\{[v]\}$;
10 $paths = \emptyset$;
11   **for** $neighbour \in v.neighbours$ **do**
12   **if** $neighbour \notin inpath$ **then**
13   **for** $path \in \text{Aux}(G, neighbour, k - 1, inpath)$ **do**
14 $paths = paths \cup \{path.add\_first(v)\}$;

15 $inpath = inpath \setminus \{v\}$;
16   **return** $paths$;

The implementation in python is very similar to the pseudo-code, except that the algorithm returns a list of paths rather than a set. The graph is given in the form of adjacency lists, in order to be able to iterate quickly on the neighbors.

file	CPU time	$|V|$	$|E|$	number of paths
test2.txt	$0.097 * 10^{-3}$ seconds	12	18	45
test3.txt	$31.008 * 10^{-3}$ seconds	17	229	28188

Figure 3: Mean execution time over 1000 iterations on an Intel i5-6200U processor.

Question 10

Let n be the number of patients and M the adjacency matrix of the graph. We will denote by $m_{i,j}$ the coefficients of M. We therefore have $m_{i,j} = 1$ if the kidney k_i is compatible with the patient t_j and $m_{i,j} = 0$ otherwise. We also define Π as the set of minimal infeasible paths.

For all $(i,j) \in [\![1,n]\!]^2$, we define a variable $a_{i,j}$ which is equal to 1 if the kidney k_i is assigned to the patient t_j and equal to 0 otherwise. We denote by A the assignment matrix whose coefficients are $a_{i,j}$. Since we are trying to maximize the number of transplants, our objective function will be:

$$\max_{A} \sum_{(i,j) \in [\![1,n]\!]^2} a_{i,j} \tag{1}$$

However, we are subject to a few of constraints. To begin with, by definition of $a_{i,j}$, we have the following first set of constraints.

$$\forall (i,j) \in [\![1,n]\!]^2,\ a_{i,j} \in \{0,1\} \tag{2}$$

In addition a kidney k_i can only be assigned to a patient t_j if t_j is compatible with k_i, in other words $a_{i,j}$ cannot be equal to 1 if $m_{i,j}$ is 0. Therefore $a_{i,j}$ is necessarily less than or equal to $m_{i,j}$. So we have a new set of constraints:

$$\forall (i,j) \in [\![1,n]\!]^2,\ a_{i,j} \leq m_{i,j} \tag{3}$$

In addition, a patient cannot receive more than 1 kidney and a donor will not donate his kidney if his paired patient does not receive a kidney. Thus, we have:

$$\forall j \in [\![1,n]\!],\ \sum_{i=1}^{n} a_{i,j} \leq 1 \tag{4}$$

$$\forall j \in [\![1,n]\!],\ \sum_{i=1}^{n} a_{j,i} \leq \sum_{i=1}^{n} a_{i,j} \tag{5}$$

Finally, we need to verify that there is no exchange cycle of length more than the threshold K. To do this, it suffices to check that none of the minimal infeasible paths is taken. We thus obtain our last set of constraints:

$$\forall \{i_1, i_2, \ldots, i_{K+1}\} \in \Pi,\ \sum_{k=1}^{K} a_{i_k, i_{k+1}} \leq K - 1 \tag{6}$$

Our integer linear program is therefore the maximization of 1 subject to 2, 3, 4, 5 and 6.

Question 11

Algorithm 4: Branch and Bound

Input: $G = (V, E)$ a graph and k the threshold.
Output: The set of all minimal infeasible paths.

1 begin
2 $paths = \emptyset$;
3 **for** $v \in V$ **do**
4 $paths = paths \cup \{\texttt{Aux}(G, v, k, \emptyset)\}$;
5 **return** $paths$;

/* Return the list of all minimal infeasible path starting
   from v */
6 Function $\texttt{Aux}(G, v, k, inpath)$:
7 $inpath = inpath \cup \{v\}$;
8 **if** $k = 0$ **then**
9   **return** $\{[v]\}$;
10 $paths = \emptyset$;
11 **for** $neighbour \in v.neighbours$ **do**
12   **if** $neighbour \notin inpath$ **then**
13   **for** $path \in \texttt{Aux}(G, neighbour, k-1, inpath)$ **do**
14 $paths = paths \cup \{path.add\_first(v)\}$;

15 $inpath = inpath \setminus \{v\}$;
16 **return** $paths$;