UNIVERSIDAD
COMPLUTENSE
MADRID

FACULTAD DE CIENCIAS ECONÓMICAS Y EMPRESARIALES

DOBLE GRADO EN ECONOMÍA – MATEMÁTICAS Y ESTADÍSTICA

TRABAJO DE FIN DE GRADO

TÍTULO: Tratamientos de Inmunosupresión Sanguínea: Aplicación al Programa Español de Trasplante Renal Cruzado.

AUTOR: Sara Boya Fernández

TUTOR: Carmelo Rodríguez Álvarez

CURSO ACADÉMICO: 2020/2021

CONVOCATORIA: FEBRERO

Índice

RESUMEN

La enfermedad renal crónica avanzada es una afección terminal que, aunque temporalmente se pueda recurrir a la dialización de un paciente, requiere de un trasplante para su supervivencia. El trasplante de riñón cruzado es un problema de diseño de mercados que surge como respuesta para intentar aliviar en parte este problema, ampliando las posibilidades de obtención de un trasplante.

En este trabajo nos vamos a centrar en los efectos de los tratamientos de inmunosupresión, permitiendo la posibilidad de realizar trasplantes renales entre pacientes y donantes con incompatibilidad sanguínea. Empezaremos describiendo brevemente las cuestiones médicas relativas al problema, analizaremos varios modelos de emparejamiento del problema renal cruzado, mostraremos la situación actual del trasplante renal en España y, finalmente, realizaremos simulaciones que nos permitirán evaluar las potenciales ganancias de diferentes estrategias de introducción de estos tratamientos en el programa español de trasplante renal cruzado.

1. INTRODUCCIÓN

Cuando a un paciente se le diagnostica un problema renal grave implica que su vida, si no consigue un trasplante en un tiempo razonable, se va a acortar de forma significativa. En muchas ocasiones existe ese familiar, amigo o pareja que está dispuesto a donar un riñón (el ser humano sólo necesita uno de los dos que tenemos para vivir), pero no basta con eso, si no que ese donante tiene que ser compatible con el paciente para que se pueda realizar el trasplante. Habitualmente, el paciente es incompatible con su donante y pasa a formar parte de la lista de espera, donde permanecerá hasta recibir un riñón proveniente de un fallecido.

Con un programa de intercambio de riñón cruzado se pretende que las parejas de paciente-donante que, por cuestiones médicas no es posible realizar un trasplante entre ellas, intercambien donantes, es decir, los donantes de cada pareja pueden donar al paciente de lo otra pareja, de manera que un paciente va a conseguir un trasplante gracias a otro donante, que también forma parte del programa, mientras su donante done su riñón a otro paciente.

Para mostrar la magnitud del problema considerado, si observamos los datos de EEUU en 2019, se produjeron un total de 23.401 trasplantes renales, siendo 6.867 trasplantes de vivo, de entre los cuáles 1.114 se realizaron gracias al programa de intercambios existente.[1] Si nos fijamos en España para ese mismo año, del total de 3.427 trasplantes, se realizaron 335 con riñones obtenidos a través de un donante vivo fueron y únicamente, 33 trasplantes se realizaron mediante intercambios.[2] Esta última cifra muestra el posible campo de mejora que podría proporcionar un cambio en el diseño o una generalización de los programas de intercambio, como analizaremos en este TFG.

El trasplante renal cruzado es de importante relevancia a nivel sanitario pues se busca el objetivo de aumentar el número de trasplantes, lo que supone aumentar la supervivencia de los pacientes y su calidad de vida. Cabe señalar que los pacientes sin donante también se benefician del programa al disminuir la presión sobre la lista de espera. Asimismo, a

[1] Ver Organ Procurement and Transplantation Network (OPTN).
https://optn.transplant.hrsa.gov/data/view-data-reports/national-data/
[2] Ver Organización Nacional de Trasplantes (ONT)
http://www.ont.es/infesp/Memorias/Actividad_de_Donaci%C3%B3n_y_Trasplante_Renal_2019.pdf

nivel económico, el coste de un trasplante es inferior que el del tratamiento alternativo de diálisis (Sánchez-Escuredo et al., 2015).

El programa de intercambio de donantes de riñón de vivo constituye un problema de asignación económica dentro del contexto de diseño de mercados. Este mercado tiene como particularidad la imposibilidad de fijar precios ya que, a excepción de Irán (Roth, 2015), es ilegal su compra y venta. Por tanto, no podemos analizar el problema desde un punto de vista tradicional de oferta y demanda, si no que se habrá que buscar un mecanismo adecuado de emparejamiento.

El problema de trasplantes de riñón cruzado se empezó a estudiar en los artículos pioneros en este campo Roth et al. (2004, 2005), que propusieron una aproximación sistemática basada en la Teoría de emparejamientos.[3]

Cabe destacar que en este trabajo sólo se va a considerar intercambios entre dos parejas paciente-donante. Esta limitación se puede explicar por las dificultades técnicas (número de quirófanos y profesionales necesarios) que entrañarían intercambios superiores debido a que las operaciones se deben hacer simultáneamente, para evitar el riesgo de que un donante cambie de opinión una vez que el paciente con el que se ha inscrito al programa haya sido trasplantado.

El objetivo de este TFG es adaptar las propuestas del programa escandinavo estudiadas por Andersson y Kratz (2020), que exploran la posibilidad de realizar trasplantes con grupos sanguíneos incompatibles dentro del programa.

Empezaremos analizando en la Sección 2 los detalles médicos relativos a la compatibilidad entre un paciente y un posible donante. A continuación, en la Sección 3, mostraremos los datos sobre trasplantes renales y su evolución en España. Posteriormente, en la Sección 4, estudiaremos el modelo de Andersson y Kratz (2020) de trasplantes cruzados con inmunosupresores. En la Sección 5, realizaremos simulaciones adaptadas al caso español y expondremos una serie de propuestas. Finalmente, en la Sección 6 presentamos las conclusiones.

[3] El trasplante renal cruzado no es el único problema que utiliza mecanismos de asignación y emparejamiento, si no que también es de utilidad para otros problemas como la asignación de alumnos a centros escolares (Abdulkadiroğlu, Sönmez, 2003) o el mercado de médicos residentes (Roth, Alvin E., 1984).

2. CUESTIONES MÉDICAS

La incompatibilidad de un donante con un paciente se puede deber principalmente a dos hechos: la incompatibilidad del grupo sanguíneo ABO o la incompatibilidad de tejidos.

Con respecto a la incompatibilidad ABO, existen dos posibles antígenos (A y B) y a partir de estos, se definen cuatro posibles tipos de células rojas (o grupos sanguíneos): 0, A, B y AB.

Un paciente del grupo A (B) tiene antígenos A (B), por lo que no producirá anticuerpos anti-A (anti-B) pero sí anticuerpos anti-B (anti-A) lo que hará que no sea compatible con un donante que tenga antígenos B (A). Mientras, un paciente del grupo AB tiene antígenos A y B, por lo que no produce anticuerpos contra ninguno de los antígenos y será compatible con un donante de cualquier grupo. Finalmente, en la sangre de un paciente del grupo 0 hay ausencia del antígeno A y del antígeno B lo que implicará incompatibilidad sanguínea con un donante del grupo A, B o AB.[4]

Además de tener en cuenta los grupos sanguíneos, también hay que considerar el factor Rhesus, que es una proteína de los glóbulos rojos. Si un paciente tiene esta proteína se dice que es Rh positivo y en caso contrario, Rh negativo.[5]

Un paciente con Rh negativo tiene incompatibilidad sanguínea con un donante que sea Rh positivo mientras que un paciente con Rh positivo puede ser compatible tanto con un donante Rh positivo como con un donante Rh negativo.[6]

Por otra parte, está la incompatibilidad de tejidos. Los antígenos leucocitarios humanos (HLA) predominan en los glóbulos blancos y son proteínas del sistema inmune que ayudan a diferenciar entre las propias células del paciente y otras sustancias.[7] Cuanto más parecido sea el HLA de un donante y un paciente, más compatibles serán en este aspecto. Para averiguar esta compatibilidad, se realiza una prueba cruzada, que da como resultado el nivel de anticuerpos que tiene el paciente al HLA del donante.

[4] Ver https://www.donarsangre.org/grupos-sanguineos/ (Acceso 11 de diciembre de 2020).
[5] Ver https://www.mayoclinic.org/es-es/tests-procedures/rh-factor/about/pac-20394960 (Acceso 18 de diciembre de 2020).
[6] Ver https://www.donarsangre.org/rh-negativo/ (Acceso 18 de diciembre de 2020).
[7] Ver https://medlineplus.gov/spanish/ency/article/003551.htm (Acceso 18 de diciembre de 2020).

Se dice que esta prueba cruzada es positiva cuando el receptor tiene una alta sensibilización al HLA del donante, lo que significaría un posible rechazo inmediato del trasplante.

En este trabajo vamos a eliminar la incompatibilidad sanguínea, ya que debido a los avances médicos existen técnicas para superar esta incompatibilidad que se basan en la desensibilización del paciente consiguiendo eliminar y/o diluir los anticuerpos contra los antígenos A y B a unos niveles imperceptibles (Rydberg et al., 2005).
Así, con esta técnica, la realización de trasplantes entre dos individuos de grupos sanguíneos incompatibles es factible y posible. Además, hay que tener en cuenta que, aunque no es inocuo, el filtrado no es tóxico para los pacientes y los resultados son idénticos en términos de supervivencia del trasplante en 5 años (Tyden et al., 2007). A pesar de que en los próximos apartados se vaya a permitir la existencia de emparejamientos donde haya incompatibilidad sanguínea, se priorizará que la compatibilidad sea completa con el objetivo de que el paciente se someta al menor número de tratamientos posible.
Para ello, se definen dos tipos de compatibilidad.

- Se dice que un paciente es compatible con un donante cuando haya compatibilidad del grupo ABO y la prueba cruzada entre ellos no es positiva.
  Cuando se realice un trasplante de un paciente compatible con el donante, se denominará trasplante compatible.
- Se dice que un paciente es medio compatible con un donante cuando exista incompatibilidad del grupo ABO, pero la prueba cruzada entre ellos no sea positiva. En este caso, se hablará de trasplante medio compatible.

El objetivo de las autoridades sanitarias será maximizar el número de trasplantes. Como el tratamiento inmunosupresor es más costoso y tiene consecuencias negativas para los pacientes, subsidiariamente se debería minimizar el número de trasplantes medio compatibles.

3. SITUACIÓN ACTUAL EN ESPAÑA

En esta sección se utilizarán los datos que facilita la Organización Nacional de Trasplantes (ONT) en su informe anual sobre el trasplante renal en España.

El trasplante renal más común tiene como origen un riñón perteneciente a un fallecido. En 2019, el 90,21% de los trasplantes fueron efectivamente de este tipo, suponiendo por tanto los trasplantes de vivo solamente un 9,78% (335 trasplantes de un total de 3427). Mientras, la lista de espera para recibir trasplante finalizó el año con 3933 pacientes inscritos.

En este trabajo vamos a estudiar modelos en los cuáles se realizan trasplantes medio-compatibles (con grupo ABO incompatible). En España este tipo de trasplantes, que implican la desensibilización del paciente, suponen un porcentaje escaso dentro de los trasplantes renales de vivo. De 2011 a 2019 este porcentaje fue en media de un 7,2%.

Gráfico 1: Evolución de trasplantes de vivo en España *compatibles y medio-compatibles.*

Fuente: Organización Nacional de Trasplantes.

Se puede observar una tendencia ascendente a partir de 2014, suponiendo en 2019 un 11,3% dentro de los trasplantes de vivo.

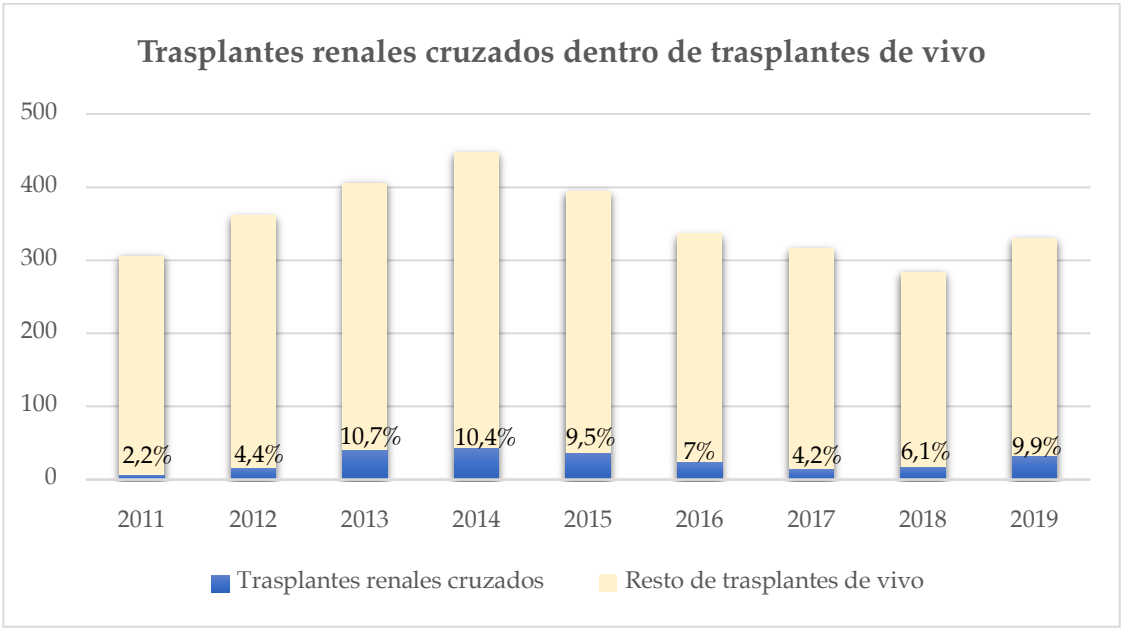
En 2009 se llevó a cabo por primera vez en España un intercambio de donantes de riñón de vivo. Desde entonces hasta 2019 se ha realizado un total de 236 trasplantes renales cruzados.

Gráfico 2: Evolución de trasplantes *cruzados* en España.

Fuente: Organización Nacional de Trasplantes.

Con datos de 2019, un 9,9% de los trasplantes renales de vivo se deben a este programa.

Gráfico 3: Evolución de trasplantes *cruzados* dentro de los trasplantes de vivo en España.

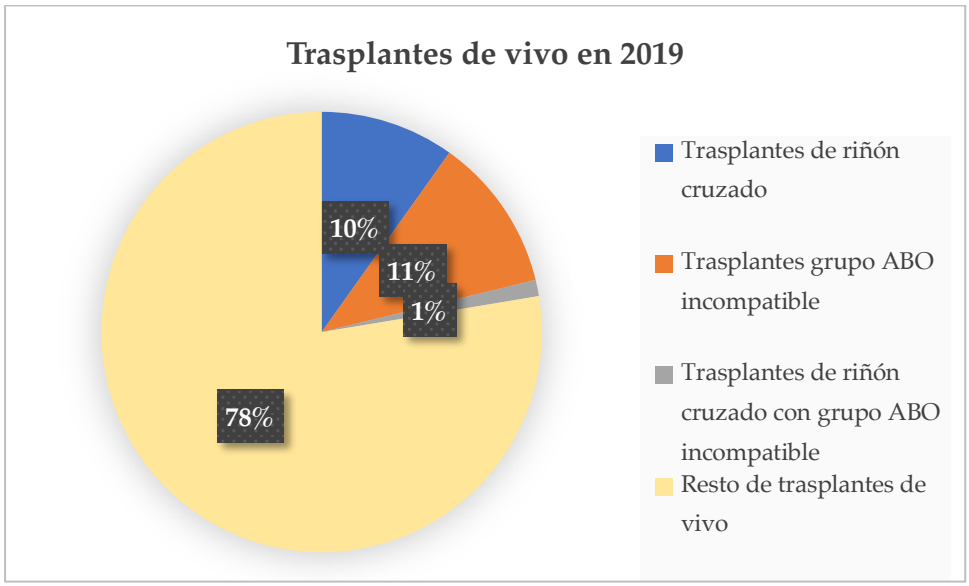


Fuente: Organización Nacional de Trasplantes.

La evolución muestra varios cambios de tendencia. En primer lugar, con la introducción de los trasplantes renales cruzados estos fueron aumentando rápidamente hasta situarse cerca de un 10% en los años 2013, 2014 y 2015. Sin embargo, a partir de 2015 disminuye su porcentaje drásticamente hasta situarse en un 4,2% en 2017. En 2018 vuelve a cambiar la tendencia, esta vez de manera positiva, hasta llegar a casi un 10% en 2019, acercándonos por tanto a las cifras máximas que se llegó en años anteriores.

Una visión del panorama actual, la tenemos viendo la distribución en 2019 de los trasplantes de vivo:

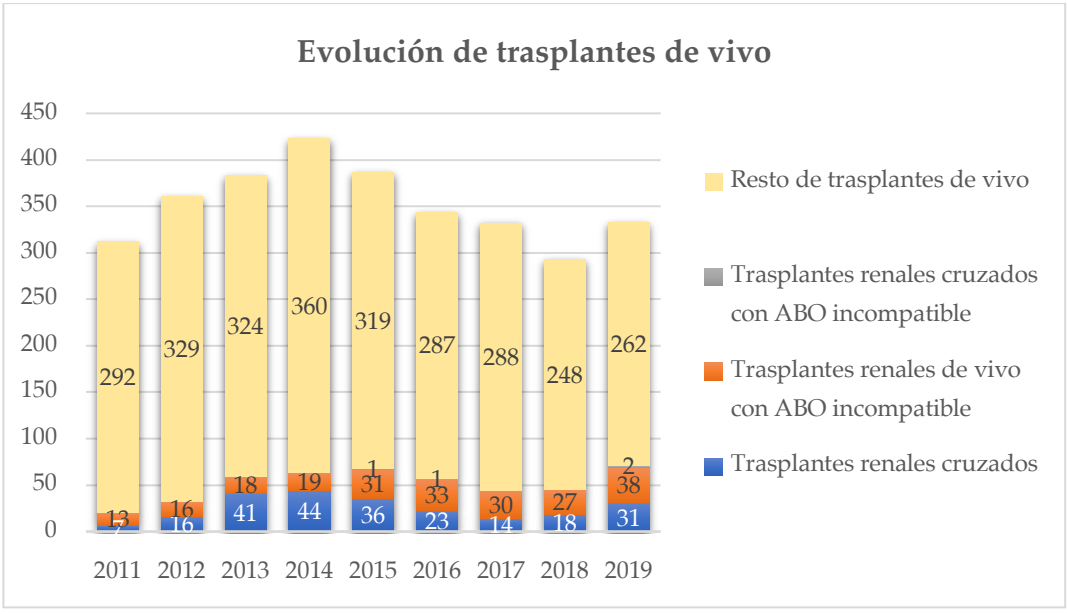
Gráfico 4: Distribución trasplantes de vivo en 2019 en España.



Fuente: Organización Nacional de Trasplantes.

También podemos ver su evolución:

Gráfico 5: Evolución de trasplantes de vivo en España.



Fuente: Organización Nacional de Trasplantes.

Se observa cómo a la vez que descendía el número de trasplantes renales cruzados, aumentaba el número de trasplantes con ABO incompatible, aunque en los últimos años esta tendencia ha cambiado como mencionamos anteriormente. En 2019 ambos porcentajes son muy parecidos.

Programa implantado en España

Uno de los objetivos del programa en España es conseguir mayor disponibilidad de riñones de gente joven (al trasplantar es conveniente que la edad de paciente-donante no sea muy distante) ya que este porcentaje actualmente es bastante bajo debido al descenso de accidentes en seguridad vial.

Sobre el programa implantado en España destacamos los siguientes aspectos (Biró, van de Klundert et al., 2019):

- Se priorizan los intercambios entre únicamente dos parejas de paciente-donante.
- Incorpora donantes altruistas.
- Hace una ronda de emparejamiento cada 4 meses o cada vez que un nuevo donante altruista se incorpora al programa.
- Se pueden inscribir pares compatibles con el incentivo de que nunca se emparejarán con alguien peor.
- Se prefiere intercambio antes que la desensibilización.
- Dentro de los intercambios, se permiten trasplantes medio-compatibles.

Panorama internacional

España tiene la mayor ratio de trasplantes de riñón de fallecido en Europa (European Directorate for the Quality of Medicines, 2017). Esto hace que en el resto de los países del conjunto de Europa haya una mayor implementación de estos programas de intercambio. Hemos visto que en España el porcentaje que suponían los trasplantes de riñón cruzado era de un 10%. Si nos fijamos en otros países este porcentaje es bastante mayor cómo puede ser el 30% de Reino Unido o el 50% de Países Bajos (Biró et al., 2019).

4. PROBLEMA DE ASIGNACIÓN DE RIÑONES

En esta sección exponemos el modelo teórico presentado por Andersson y Kratz (2020), que extiende el modelo básico de Roth et al. (2005) incluyendo la posibilidad de trasplantes medio compatibles a través de tratamientos inmunosupresores.

Un problema de asignación de riñones está formado por un conjunto de pacientes $N=\{1,\ldots,n\}$, donde cada paciente $i\in N$ tiene un donante vivo d_i, que está dispuesto a donar su riñón a otro paciente a cambio de que el paciente con el que forma pareja reciba también un trasplante.
Nótese que el paciente i no es completamente compatible con su propio donante puesto que estos trasplantes se realizan fuera del programa de intercambio de riñones. Por tanto, el conjunto de pacientes N se puede dividir en dos subconjuntos disjuntos N_H y N_I según sea cada paciente medio compatible o incompatible con su propio donante.

La estructura de compatibilidad C describe la compatibilidad entre cada paciente y los distintos donantes inscritos en el programa, es decir, la compatibilidad entre i y d_j para cada $i,j\in N$.

Se definen unas preferencias para el donante $i\in N$: $\succsim_i$ donde $\succ_i$ denota la preferencia estricta y $\sim_i$ la indiferencia. Estas preferencias se deben entender en términos médicos según lo comentado en el punto anterior. Por ejemplo, habría 2 casos en los cuáles $d_i \succ_i d_j$:

- Si el paciente i es compatible con su propio donante d_i pero es medio compatible o incompatible con el donante d_j.
- Si el paciente i es medio compatible con su propio donante d_i pero es incompatible con el donante d_j.

Las preferencias de todos los pacientes en N las agrupamos en la lista $\succcurlyeq := (\succsim_i)_{i\in N}$.
En base a estas preferencias, se dice que un intercambio es factible entre (i, d_i) y (j, d_j) si y sólo si $d_j \succ_i d_i$ y $d_i \succ_j d_j$.

Definimos una función de prioridad que ordena a los pacientes: $\pi: N \to \mathbb{R}_{++}$.

Se asume que la prioridad $\pi(i)$ para cada paciente i∈N es una fracción $\pi(i) = \frac{p(i)}{q}$ para algún p(i) ∈{1, …, p} con p,q∈ $\mathbb{Z}_{++}$ fijos e iguales para todos los pacientes.
Con esta función de prioridad se persigue el objetivo de ayudar, por una parte, a los pacientes que tienen alta sensibilización al HLA y, por otra parte, a los pacientes que debido al grupo sanguíneo son compatibles con menos donantes (por ejemplo, los pacientes del grupo 0- sólo serán compatibles con otros donantes 0- pero no con los donantes de otro grupo sanguíneo), otorgándoles una posición más alta ya que es más complicado encontrar un donante compatible para estos casos.

Así, un problema de intercambio de riñones es una tupla (N, C, π).
Para este un problema (N, C, π) un *matching* M consiste en:

- Un conjunto de pares de intercambio mutuamente factibles. Con esto se consigue que cada pareja de paciente-donante sólo esté emparejada con una pareja de paciente-donante. Al exigir que sea factible, tenemos racionalidad individual ya que ningún paciente estará peor que si se hubiese emparejado con su propio donante, lo cuál incentivará la participación en el programa. Evidentemente si no se garantizase al paciente que el resultado que se obtenga del intercambio será al menos tan bueno cómo si no se hubiese inscrito en él, no se apuntaría al programa.
- Un conjunto N_H que no participan en el intercambio. Los pacientes que no han conseguido emparejarse se quedan en espera. Cuando se inscriban más pacientes se podrá realizar otra ronda de emparejamientos donde volverán a tener la oportunidad de emparejarse.

Vamos a denotar al conjunto de todos los posibles *matching*s como $\mathcal{M}$ y al conjunto de los pacientes emparejados como N*(M). La cardinalidad del último conjunto |N*(M)| nos daría el número de trasplantes realizados con ese *matching* M.

Introducimos ahora varias definiciones:

- Se dice que un *matching* M es **maximal** si N*(M) no está contenido en el conjunto N*(M') para cualquier otro *matching* M'∈ $\mathcal{M}$. Esto significaría que no habría ningún otro *matching* M' que además de emparejar los mismos pacientes que M, habría más pacientes que lograrían match.
- Se dice que un *matching* M es **máximo** si maximiza el número de trasplante sobres todos los *matching*s de $\mathcal{M}$.

- Para dos *matchings* M, M'$\in \mathcal{M}$, se dice que M **domina en el sentido de Pareto** a M' si todos los pacientes en N están al menos tan bien como lo están en M' y al menos uno de esos pacientes está estrictamente mejor (de acuerdo con las preferencias definidas anteriormente $\succcurlyeq$).
- Se dice que un *matching* M$\in \mathcal{M}$ es **eficiente** en el sentido de Pareto si no está dominado en el sentido de Pareto por ningún otro *matching* M'$\in \mathcal{M}$.
  Este sería el mejor *matching* que podríamos encontrar.

3.1. Matchings de prioridad y matchings de prioridad medio compatibles.

Suponemos que hay un diseñador de mercado que distingue entre dos nociones de compatibilidad: trasplantes compatibles y medio compatibles.
Sea B(M) el número de pacientes que están emparejados con un donante compatible en el match M$\in \mathcal{M}$. Nótese que, en general, B(M) no es el mismo conjunto que N*(M) pues también hay una serie de pacientes que están emparejados con un donante medio compatible.

Se dice que unas preferencias $\succsim_B$ pertenecen a una clase de preferencias medio compatible si son completas, transitivas y verifican:

- M $\succ_B$ M' si $\begin{cases} N^*(M') \subset N^*(M), \\ N^*(M) \setminus N^*(M') = \{i\},\ N^*(M') \setminus N^*(M) = \{j\}\ y\ \pi(i) > \pi(j), \\ N^*(M) = N^*(M')\ y\ B(M) > B(M'). \end{cases}$

  Es decir, un *matching* M $\in \mathcal{M}$ es preferido a otro *matching* M'$\in \mathcal{M}$ si:

  1. En M están emparejados todos los pacientes de M' y algunos pacientes que no están emparejados en M', sí lo están en M. Por tanto, en M hay más pacientes emparejados: los que estaban emparejados en M' y alguno más.
  2. El conjunto de matches de M se puede obtener del conjunto de matches de M' reemplazando algún paciente de M' con alguno de mayor prioridad (según la función que hemos definido anteriormente π) que está emparejado en M. Esto quiere decir que M y M' sólo se distinguen en un paciente emparejado y el paciente que empareja M tiene mayor prioridad que el paciente que empareja M'.
  3. En M y M' hay igual número de matches, pero en M el número de matches que son compatibles es mayor que en M'. Así, vamos a priorizar el mayor número de

trasplantes compatibles para evitar tratamientos de desensibilización a pacientes y un mayor uso de inmunosupresores.

- M $\sim_B$ M' si $N^*(M) = N^*(M')$ y $B(M) = B(M')$.

  Esto quiere decir que en M $\in \mathcal{M}$ y M'$\in \mathcal{M}$ hay el mismo número total de matches y de matches compatibles.

Se dice que unas preferencias $\succsim_\pi$ pertenecen a una clase de preferencias compatible si son completas, transitivas y verifican:

- M $\succ_\pi$ M' si $\begin{cases} N^*(M') \subset N^*(M), \\ N^*(M) \setminus N^*(M') = \{i\},\ N^*(M') \setminus N^*(M) = \{j\}\ y\ \pi(i) > \pi(j). \end{cases}$
- M $\sim_\pi$ M' si $N^*(M) = N^*(M')$.

La diferencia entre $\succsim_B$ y $\succsim_\pi$ es que en $\succsim_B$ se prefiere el *matching* que minimiza el número de trasplantes que son medio compatibles mientras que en $\succsim_\pi$ se es indiferente entre uno y otro.

Cómo nuestro objetivo es priorizar el número de trasplantes compatibles para evitar tratamientos de desensibilización a pacientes y un mayor uso de inmunosupresores, vamos a centrarnos en las preferencias medio compatibles $\succsim_B$.

A partir de esto, se define:

- Se dice que M$\in \mathcal{M}$ es un *matching* de **prioridad** si M $\succsim_\pi$ M' para cualquier M'$\in \mathcal{M}$.
  Todos estos *matchings* se agrupan en el conjunto $\mathcal{M}^* \subset \mathcal{M}$.
- Se dice que M$\in \mathcal{M}$ es un *matching* de **prioridad medio-compatible** si M $\succsim_B$ M' para cualquier M'$\in \mathcal{M}$.
  Todos estos *matchings* se agrupan en el conjunto $\mathcal{M}^B \subset \mathcal{M}$.

Los siguientes resultados son probados por Andersson y Kratz (2020, Proposición 1, 2, 3 y 4).

Propiedad 1: Para cualquier problema de intercambio de riñones, todas las relaciones de preferencia $\succsim_B$ inducen el mismo conjunto $\mathcal{M}^B$.

Propiedad 2: Para cualquier problema de intercambio de riñones, todo *matching* de prioridad medio compatibles en un *matching* de prioridad.

Se puede considerar que los *matchings* de prioridad medio compatible son el subconjunto de los *matchings* de prioridad que maximiza el número de trasplantes compatibles.

Propiedad 3: Para cualquier problema de intercambio de riñones, todo *matching* M∈ $\mathcal{M}^B$ es máximo y Pareto eficiente.
Esto tiene como consecuencia que todos los *matchings* de prioridad medio compatibles resultan en el mismo número de trasplantes.

Propiedad 4: Para cualquier problema de intercambio de riñones, el número de trasplantes es el mismo para cualquier *matching* M∈ $\mathcal{M}^*$.

3.2 Cómo identificar los matchings de prioridad medio-compatibles.

Para identificar los *matchings* vamos a utilizar teoría de grafos.
Para toda estructura de compatibilidad, C, existe un grafo de compatibilidad g=(N,E), siendo N(g) el conjunto de vértices y E(g) el conjunto de enlaces. En nuestro problema todo vértice se va a corresponder con un paciente de N.

- Hay un enlace entre i,j∈N si y sólo si el intercambio entre los pares (i,d_i) y (j,d_j) es factible.
- Hay un bucle en el vértice i∈N si y sólo si i es medio-compatible con su donante d_i. Nótese que si el paciente fuese totalmente compatible con su donante ya los habríamos emparejado antes, por tanto, no se puede dar este caso.

Viéndolo con un ejemplo:

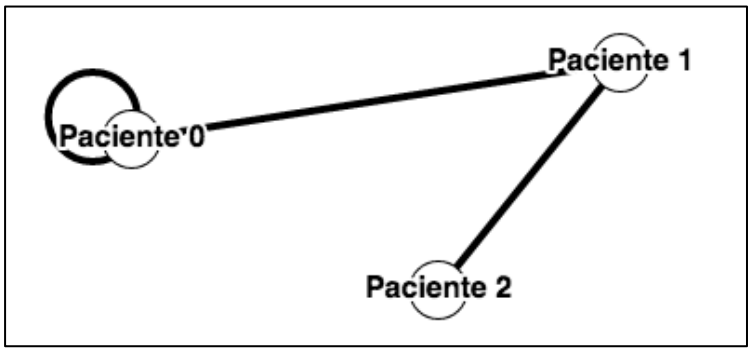


Figura 1: Ejemplo de representación en grafos.

En la Figura 1 se presenta el grafo de compatibilidad para un caso en el que tenemos 3 pacientes (N=3): paciente 0, paciente 1 y paciente 2. El único paciente que es medio

compatible con su propio donante es el paciente 0. Por tanto, el paciente 0 tiene dibujado un bucle (un enlace que sale de él y vuelve a él) en el grafo.
Por otra parte, hay 2 intercambios factibles: el paciente 1 con el donante del paciente 0 (y viceversa) y el paciente 1 con el donante del paciente 2 (y viceversa). Por tanto, están dibujados enlaces que unen a dichos pacientes.

Un *matching* $M \subseteq E$ es un conjunto de enlaces en el grafo disjuntos entre ellos. Exigiendo que los enlaces no sean incidentes se garantiza que cada pareja de donante-paciente está emparejada con sólo una pareja de donante-paciente.

Sobre el mismo ejemplo del caso anterior, vamos a representar un *matching* M:

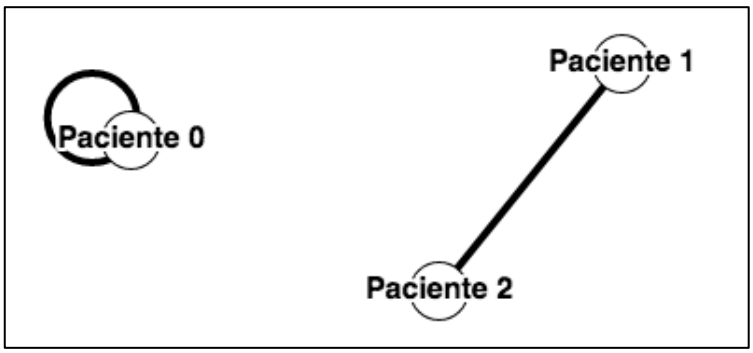


Figura 2: Ejemplo de un *matching* en grafos.

En la Figura 2 se representa el *matching* para el ejemplo anterior que maximiza el número de trasplantes, pues aquí todos los pacientes obtienen trasplante. En este *matching* el paciente 0 está emparejado con su propio donante y el paciente 1 ha hecho match con el paciente 2: el paciente 1 recibirá el riñón del donante del paciente 2 y el paciente 2 recibirá el riñón del donante del paciente 1.
Podemos ver cómo ahora los enlaces no son incidentes (antes se cortaba el enlace que unía Paciente1-Paciente2 con el enlace que unía Paciente 0-Paciente1, y también se cortaba el enlace que unía Paciente0-Paciente1 con el bucle del paciente 0).

Para saber que emparejamientos deben priorizarse sobre otros hay que asignar unos pesos a cada emparejamiento. Aquí entrará en juego la función de prioridad definida anteriormente. Así, siempre que se dude entre dos o más posibles emparejamientos (una misma pareja tiene intercambio factible con otras dos o más parejas), se utilizarán esos pesos para romper ese empate (la pareja se emparejará, de entre las factibles, con la que el intercambio tenga un mayor peso asignado).

Un grafo por pesos (g,w) consiste en un grafo g y un conjunto de pesos $w: = \left(w_{ij}\right)_{ij\in E(g)}$ donde w_{ij} es el peso asignado al vértice $ij \in E(g)$. Definimos S(M,w) como,

$$S(M,w) \coloneqq \sum_{ij\in M} w_{ij}.$$

Un *matching* M es un *matching* **máximo en pesos** en (g,w) si $S(M,w) \geq S(M',w)$ para todo *matching* M'$\in \mathcal{M}$. En dicho caso, el *matching* M asegura que, entre las posibles elecciones, siempre se han seleccionado las de mayor peso, es decir, las que debemos priorizar.

Cómo queremos minimizar el número de trasplantes medio-compatibles, vamos a definir estos pesos no simplemente teniendo en cuenta la prioridad del paciente si no que, además, cuando haya un empate entre dos posibles intercambios en los cuales los pacientes tienen la misma prioridad, el algoritmo seleccione aquella opción en la que haya un trasplante compatible antes que uno medio compatible.

Sea $0 < \varepsilon < \frac{1}{2nq}$, definimos,

$$w_{ij}^{\varepsilon} = \begin{cases} \pi(i) + \pi(j) + v(i,j) + v(j,i) & si\ i \neq j, \\ \pi(i) & si\ i = j. \end{cases}$$

Siendo

$$v(i,j) = \begin{cases} \varepsilon & si\ el\ paciente\ i\ es\ compatible\ con\ el\ donante\ del\ paciente\ j, \\ 0 & en\ caso\ contrario. \end{cases}$$

Ejemplo 1:

Supongamos que $\varepsilon = 0.01, \pi(0) = 0.2,\ \pi(1) = 0.3\ y\ \pi(2) = 0.5$. Más tarde, cuando se realice la simulación se verá una posible forma de determinar esta prioridad.

Además, suponemos:

- El paciente 0 es medio compatible con el donante del paciente 1.
- El paciente 1 es medio compatible con el donante del paciente 0.
- El paciente 1 es medio compatible con el donante del paciente 2.
- El paciente 2 es compatible con el donante del paciente 1.

Con esto, podemos calcula los distintos pesos asignados a cada enlace:

- $w_{00}^{\varepsilon} = \pi(0) = 0.2$
- $w_{01}^{\varepsilon} = \pi(0) + \pi(1) = 0.5$
- $w_{12}^{\varepsilon} = \pi(1) + \pi(2) + v(2,1) = 0.8 + \varepsilon = 0.8 + 0.01 = 0.81$

En la Figura 3 se representa el grafo de compatibilidad con los pesos asignados a cada enlace para el ejemplo anterior.

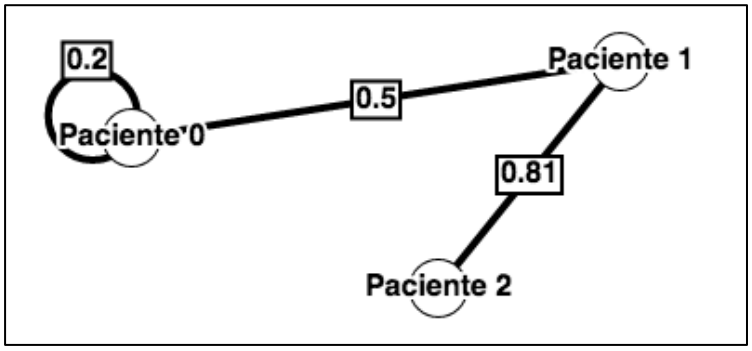


Figura 3: Grafo con pesos. Ejemplo 1.

Además, podemos ahora calcular los distintos *matchings* y ver cuál es el máximo. Tenemos dos posibles *matchings*:

En primer lugar, se considera el *matching* M_1 que empareja al paciente 0 con el donante del paciente 1 (y viceversa) mientras que el paciente 2 no está emparejado. Este *matching* se muestra en la Figura 4.

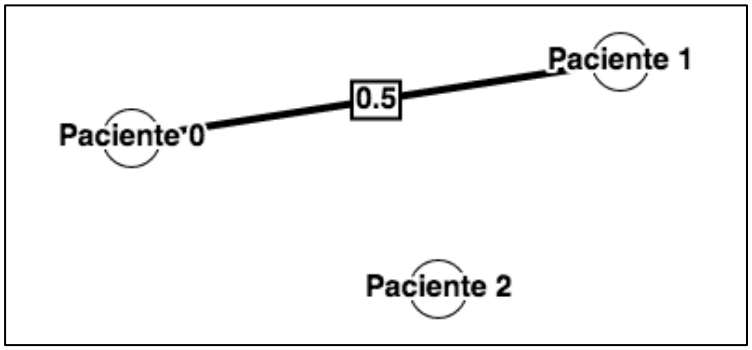


Figura 4: *Matching* M_1. Ejemplo 1.

En este caso:

$$S(M_1, w) \coloneqq \sum_{ij \in M} w_{ij} = w_{01} = 0{,}5.$$

Por último, tenemos el *matching* que empareja al paciente 0 con su propio donante y al paciente 1 con el donante del paciente 2 (y viceversa), que se representa en la Figura 5.

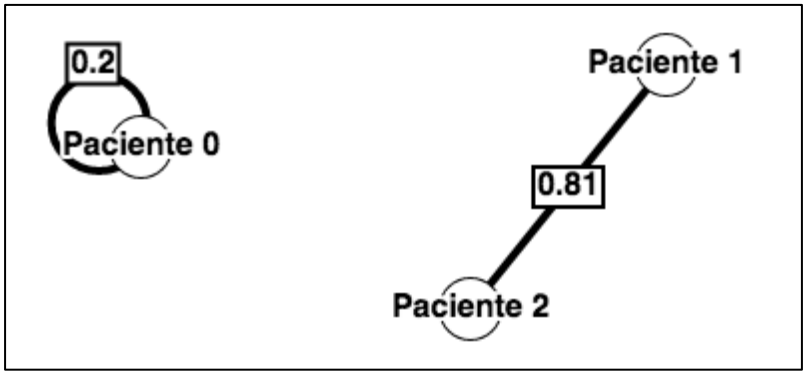


Figura 5: *Matching* M_2. Ejemplo 1.

En este caso:

$$S(M_2, w) \coloneqq \sum_{ij \in M} w_{ij} = w_{00} + w_{12} = 0{,}2 + 0{,}81 = 1{,}01.$$

Se observa que $S(M_2, w) > S(M_1, w)$.

Cómo $S(M_2, w) > S(M', w)$ para todo *matching* M'$\in \mathcal{M}$, M_2 es un *matching* máximo en pesos.

Para cualquier problema de intercambio de riñones con grafo de compatibilidad g, un *matching* es de prioridad medio-compatible si y sólo si es un *matching* de peso máximo en (g, w^{ε}) (Andersson, Kratz, 2020).

Por tanto, nuestro problema se basa en buscar el *matching* de peso máximo en (g, w^{ε}).

3.3 Estrategias de implementación Trasplante renal cruzado.

Existen diferentes alternativas en el diseño de programas de trasplante cruzado:

- Modelo base: En este modelo no se permiten los trasplantes con grupo ABO incompatible.

  Fuera del programa, se emparejará a los pacientes con sus propios donantes si son compatibles. El resto formará el conjunto de parejas de intercambios. Posteriormente, se buscarán intercambios dentro de este conjunto con la condición necesaria de que sean trasplantes compatibles.
- Modelo altruista: En este modelo no se permiten los trasplantes con grupo ABO incompatible. Aquí, todas las parejas de pacientes forman el conjunto de parejas de intercambio, independientemente de que sean compatibles con su paciente, es decir, todas las parejas van a formar parte del programa de intercambios. Un paciente compatible con su propio donante puede ser emparejado con otro donante, aunque se le garantiza que nunca se quedará sin trasplante.
- Modelo a): Se trasplanta a los pacientes que sean medio compatibles o compatibles con su propio donante fuera del programa de intercambios. El resto de los pacientes forman el conjunto de parejas de intercambio, donde se van a permitir los trasplantes

medio compatibles, aunque siempre priorizando que los trasplantes sean compatibles.

- Modelo b): En este modelo se trasplanta a los pacientes que sean compatibles con su propio donante fuera del programa. Aquellos pacientes que son medio compatibles o incompatibles con su donante forman el conjunto de parejas de intercambios. Con esto, se abre la posibilidad de que los pacientes que iban a ser emparejados con su donante medio compatible puedan mejorar y obtener un obtener un trasplante compatible.

  La diferencia con el modelo a) es que en el modelo a) los pacientes que son medio compatibles con su propio donante se emparejan con él desde el principio, mientras que en el modelo b) se les otorga la oportunidad de participar en el programa y por tanto, encontrar una mejor posibilidad (un trasplante compatible).

Nótese que todos los modelos son individualmente racionales, puesto que el paciente tiene incentivo de participación ya que nunca empeorará con respecto a su situación inicial.

El modelo a) es el que actualmente está en funcionamiento en España y el resto de Europa, mientras que en EEUU funciona un modelo descentralizado. No existe ningún país en el cuál esté implantado el modelo b).

3.4 Manipulabilidad.

Una propiedad deseable de los programas de intercambio cruzado es la no manipulabilidad.

La propiedad de no manipulabilidad aplicada a este problema de diseño de mercados significaría que los pacientes y sus médicos deberían tener incentivos a revelar los datos médicos a la Autoridad del Programa. Así, las decisiones sociales estarían basadas en la información correcta. Para garantizar esta propiedad, bastaría con tener preferencias dicotómicas, es decir, diferenciar únicamente entre trasplantes compatibles o incompatibles. Esto es efectivamente así cuando estamos en el modelo Roth et al. (2005) o en el modelo altruista Sönmez y Ünver (2014). Sin embargo, en el modelo a) y b), al considerar también la posibilidad de trasplantes medio compatibles, estamos aumentado

en una dimensión el dominio del problema por lo que las preferencias ya no son dicotómicas.

Sönmez (1999) demuestra que en dominios no restringidos, con individualidad racional y ciclos no limitados sólo hay una regla eficiente no manipulable. Por otra parte, Nicoló y R.A. (2017) prueban que existen reglas eficientes no manipulables con intercambios únicamente entre dos parejas paciente-donante y un dominio basado en la edad del donante.

Vamos a suponer que toda la información médica se puede verificar y que ningún paciente puede afectar a la función de prioridad declarando donantes que realmente son medio-compatibles como incompatibles. Sólo podríamos considerar entonces las manipulaciones en las que un paciente declare que un trasplante medio compatible es inaceptable para él debido al uso de tratamientos inmunosupresores y que serían posibles en este entorno, Nicoló y R.A. (2012). Esto significa que sólo un paciente que recibe un trasplante medio compatible (ya sea de su propio donante o mediante intercambio) puede beneficiarse, pero aún así no es condición suficiente, es decir, no todos mejorarían, sólo un porcentaje de ellos.

Veamos un ejemplo en el que un paciente, declarando unas preferencias falsas sobre su disposición a obtener un trasplante medio compatible, obtiene un resultado mejor que si las declarase verídicamente.

Ejemplo 2:
Supongamos que tenemos 4 pacientes:

- El paciente 0 es medio compatible con el donante el paciente 1 y el paciente 1 es medio compatible con el donante del paciente 0.
- El paciente 0 es compatible con el donante del paciente 3 y el paciente 3 es compatible con el donante del paciente 0.
- El paciente 2 es medio compatible con el donante del paciente 3 y el paciente 3 es medio compatible con el donante del paciente 2.

En la Figura 6 se representa en un grafo de compatibilidad los enlaces factibles entre los pares paciente-donante y en la Figura 7 se representa en un grafo el *matching* que elegiría el algoritmo.

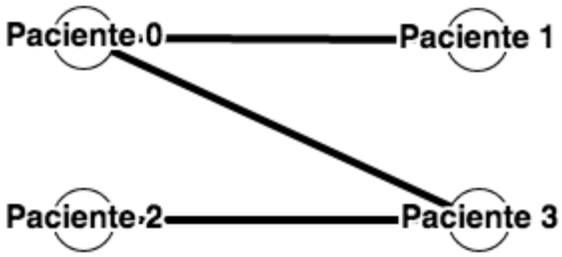


Figura 6: Enlaces factibles declarando preferencias verdaderas. Ejemplo 2.

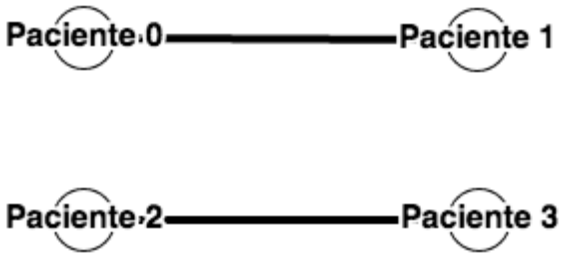


Figura 7: *Matching* que seleccionaría el algoritmo con preferencias verdaderas. Ejemplo 2.

Cómo se busca maximizar el número de trasplantes, el *matching* resultante sería el de la Figura 7 ya que se realizan un total de 2 emparejamientos: el paciente 0 con el paciente 1 y el paciente 2 con el paciente 3, siendo ambos trasplantes medio compatibles.

Ahora supongamos que el paciente 0 declara otras preferencias que no son las verdaderas y comunica que no puede aceptar un trasplante medio compatible por el elevado uso de inmunosupresores. En este caso, estaríamos eliminando un intercambio factible entre el paciente 0 y el paciente 1.

En la Figura 8 se representa el grafo de compatibilidad con los enlaces factibles para el problema manipulado y en la Figura 9 se representa el *matching* que seleccionaría el algoritmo.

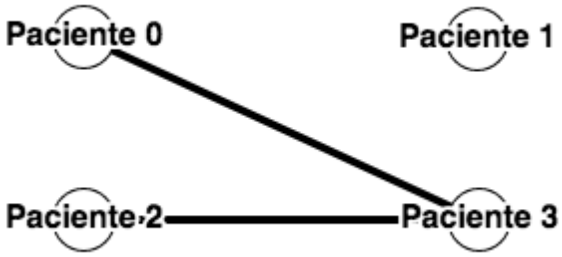


Figura 8: Enlaces factibles manipulando preferencias. Ejemplo 2.

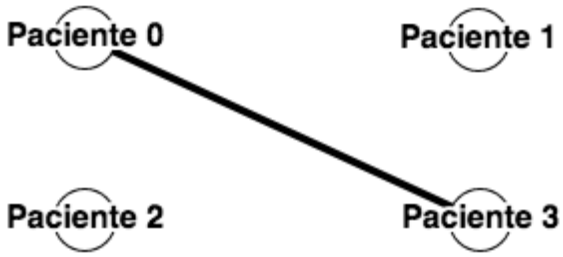


Figura 9: *Matching* que seleccionaría el algoritmo con preferencias manipuladas. Ejemplo 2.

Como sólo se puede hacer un emparejamiento el algoritmo seleccionará el emparejamiento entre el paciente 0 y el paciente 3 puesto que los trasplantes son compatibles. Hemos visto que el paciente 0 ha mejorado con respecto a la situación anterior pues ahora va a recibir un trasplante compatible y antes recibía un trasplante medio-compatible. Sin embargo, a consecuencia de esta manipulación, ha disminuido el número de trasplantes: de 4 a 2.
Nótese que en este ejemplo no hemos hablado de la función de prioridad, por tanto, hemos podido ver que el mecanismo es manipulable sin depender de ella.

Andersson y Kratz (2020) estimaron que el porcentaje de pacientes que pueden beneficiarse mediante este tipo de manipulación aumenta con el tamaño de parejas inscritas, siendo de un 24,8% para el modelo a) y un 8,3% para el modelo b) si consideramos una población de 25 pacientes hasta llegar a unos porcentajes de 56% y 39,9% respectivamente para una población de 500 pacientes.
Evidentemente, este tipo de intentos de manipulación implican riesgo. Si un paciente puede recibir un trasplante medio compatible y declara que no, podría no salir emparejado en los intercambios y quedarse sin ningún tipo de trasplante. Por tanto, el coste podría llegar incluso a acabar con su vida lo que hace que pueda disminuir el incentivo a intentar manipularlo.

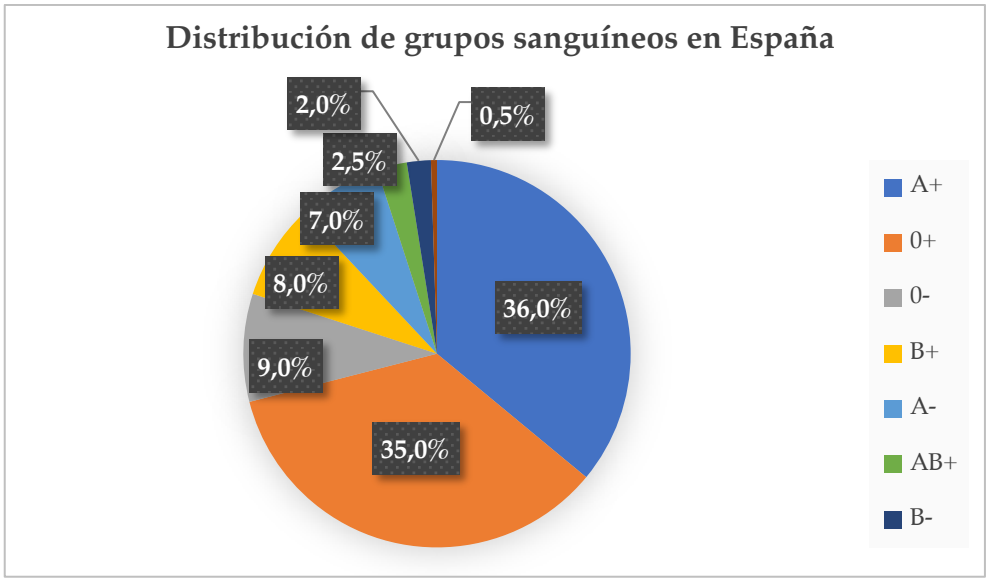
En conclusión, tanto el modelo a) como el modelo b) son manipulables, aunque el posible coste en caso de fallo es muy elevado, lo que podría eliminar incentivos para manipularlo. Además, el modelo a) es, en general, más manipulable que el modelo b).

5. SIMULACIÓN

5.1. Datos iniciales.

Para la realización de las simulaciones se ha utilizado la distribución de los grupos sanguíneos en España:

Gráfico 6: Distribución de los grupos sanguíneos en España.



Fuente: Cruz Roja[8]

Para simular la probabilidad de tener una prueba cruzada positiva se ha realizado lo indicado por Roth at al. (2007). Suponemos que la distribución de PRA en la población es la siguiente: un 70,19% tiene PRA baja, un 20% PRA media y un 9,81% PRA alta. Así, según un paciente tenga la PRA baja, media o alta se calcula la probabilidad de tener una prueba cruzada positiva:

- Si un paciente tiene PRA baja, la probabilidad de tener una prueba cruzada positiva con un donante aleatorio se estima en un 5%.
- Si un paciente tiene PRA media, la probabilidad de tener una prueba cruzada positiva con un donante aleatorio se estima en un 45%.
- Si un paciente tiene PRA alta, la probabilidad de tener una prueba cruzada positiva con un donante aleatorio se estima en un 90%.

[8] Ver https://www.donarsangre.org/grupos-sanguineos/ (Acceso 11 de diciembre de 2019).

Además, hemos supuesto que al 25% de los pacientes no les es posible la realización de un trasplante medio-compatible.

La función de prioridad utilizada es la misma que proponían Andersson y Kratz (2020):

$$\pi(i) = PRA(i) * (\%\ de\ los\ donantes\ con\ los\ que\ son\ incompatibles).$$

Nótese que con esta función por una parte priorizamos a aquellos pacientes que tienen más complicado encontrar un donante debido a su alta sensibilización al HLA y a su vez, también se prioriza a los pacientes que debido al grupo sanguíneo tienen menos probabilidad de encontrar un donante compatible.

Un aspecto que no se ha tenido en cuenta en el trabajo es que cuando el paciente es una mujer y su donante es su marido, es más probable que la prueba cruzada entre ellos sea negativa si ha habido algún embarazo (Zenios, Woodle & Ross, 2001).

En las simulaciones calculamos los matchings máximos utilizando la implementación Gurobi del algoritmo de Edmonds (1965), propuesta por E. Dupont.[9]
Estas simulaciones se han obtenido a través de una programación en Python, que se presenta en el Anexo del TFG.

Se realizaron 1.000 simulaciones de los modelos básico, a), b) y altruista para poblaciones de tamaño 25, 50, 100 y 200 respectivamente. Con respecto a la convergencia de los resultados obtenidos en la simulación, se ha calculado la desviación típica, que se sitúa en valores entre 0,004 y 0,0003 para los distintos tamaños y medidas calculadas, por lo que se puede afirmar que se produce convergencia. Destacamos que los modelos que convergen más rápidamente, es decir que presentan menores desviaciones típicas, son los modelos a) y b).

[9] https://github.com/EmilienDupont/kidneyExchange/blob/master/kidneyexchange.py (Acceso 20 de noviembre de 2020).

5.2. Resultados de la simulación.

En la Tabla 1 se presentan los resultados medios de los trasplantes obtenidos en los diferentes modelos, replicando el análisis de Andersson y Kratz (2020) para la distribución de características sanguíneas de la población española.

Tabla 1: Resultados de la simulación.

n	Modelo	Sin trasplante	Trasplante compatible con su propio donante	Trasplante medio compatible con su propio donante	Intercambio compatible	Intercambio medio compatible
25	**Base**	51,68%	44,51%	0%	3,81%	0%
	Modelo a)	20,79%	43,78%	25,84%	5,63%	3,95%
	Modelo b)	11,54%	43,68%	23,96%	10,4%	5,96%
	Altruista	29,68%	34,60%	0%	35,72%	0%
50	**Base**	50,02 %	43,75%	0%	12,31%	0%
	Modelo a)	16,77%	44,11%	25,83%	7,9%	5,39%
	Modelo b)	11,54%	43,84%	23,85%	13,75%	7,02%
	Altruista	28,79%	32,87%	0%	38,34%	0%
100	**Base**	47,51%	43,97%	0%	8,52%	0%
	Modelo a)	13,48%	43,93%	26,31%	9,96%	6,35%
	Modelo b)	7,49%	43,78%	24,09%	16,48%	8,17%
	Altruista	24,1%	32,12%	0%	43,78%	0%
200	**Base**	45,29%	43,87%	0%	10,84%	0%
	Modelo a)	10,81%	43,8%	25,99%	12,74%	6,65%
	Modelo b)	2,3%	43,75%	22,95%	20,07%	10,9%
	Altruista	26,97%	38,42%	0%	34,6%	0%

Si consideramos el modelo base con una población de tamaño 50, en media la mitad de los pacientes no recibirían trasplante. En cambio, si consideramos el modelo a) y el modelo b) esta cifra se rebaja a un 16,77% y un 11,54% respectivamente.

Realizar una transición del modelo a) al modelo b) supondría un aumento en el porcentaje de trasplantados de un 5,23% (16,77 - 11,54). Eso sí para lograr este objetivo un 30,87% (23,85 + 7,02) de los 50 pacientes va a recibir un trasplante cruzando la compatibilidad sanguínea por lo que será necesario la desensibilización de dichos pacientes.
El porcentaje de pacientes que obtiene un riñón compatible es de un 57,59% (43,84 + 13,75), mayor que en el modelo a) que representa un 52,01% (44,11 + 7,9).
Esta conclusión es válida para todos los tamaños de población considerados.
Con una población de n= 25: 54,08 (43,68 + 10,4) > 49,41 (43,78 + 5,63).
Con una población de n= 100: 60,26 (43,78 + 16,48) > 53,89 (43,93 + 9,96).
Con una población de n= 200: 63,82 (43,75 + 20,07) > 56,54 (43,8 + 12,74).

Además, en los resultados podemos ver cómo el impacto de introducir trasplantes cruzando la compatibilidad sanguínea frente al modelo base es mayor que la mejora que supone considerar donantes altruistas. Esta conclusión es válida tanto para el modelo a) cómo para el modelo b).

5.3. Análisis comparativo con la situación real en España.

En primer lugar, vamos a comparar los resultados con la simulación del modelo a) ya que como habíamos comentado anteriormente, es semejante al mecanismo existente.
En la simulación para un tamaño de 200 pacientes el porcentaje de trasplantes de vivo compatibles que no se realizaron por intercambio ni cruzando la compatibilidad de grupo ABO es de un 43,8% mientras que en los datos vimos que en España este porcentaje es de un 78%. Esto refleja que efectivamente se podría lograr un mayor número de trasplantes tanto gracias a realizar intercambios compatibles como a llevar a cabo trasplantes medio-compatibles.
También se observa en la simulación que, de los 200 pacientes, un 12,74% (25 pacientes) obtendrían trasplante compatible gracias a intercambios y un 6,65% (13 pacientes) obtendrían un trasplante medio compatible mediante estos emparejamientos. Por tanto, tendríamos un total de 38 pacientes emparejados por intercambios gracias a este mecanismo para un tamaño de 200 pacientes. En España, en 2019 se realizaron un total de 33 trasplantes de este tipo (31 compatibles y 2 medio compatibles). Viendo que en la lista de espera están un total de 3933 pacientes podemos concluir que no sería complicado

mejorar esos resultados promoviendo la inscripción de un mayor número de pacientes a este programa.

Pasamos ahora a comparar los datos de España con el modelo b) que como hemos visto en los resultados de la simulación, es el modelo que mejores resultados arroja.
Hemos visto en la simulación para un tamaño de 200 pacientes que el modelo b) obtiene como resultado que un 20,07% de pacientes obtendría un trasplante compatible mediante intercambios y un 10,09% un trasplante medio-compatible. Esto en cifras serían 40 pacientes que reciben un trasplante compatible y 20 pacientes un trasplante medio-compatible.
Si nos fijamos en los datos de España, 60 pacientes (40 + 20) son más de los que obtienen actualmente (en 2019) un riñón gracias al programa de intercambio pues esta cifra es de 33 trasplantes (31 compatibles y 2 medio-compatibles). Esto refleja el potencial de mejora existente pues esta simulación era para un total de 200 pacientes y hay que tener en cuenta que la lista de espera en España a finales de 2019 se encontraba en 3933 pacientes. Si bien es cierto, no todos los pacientes tendrán un donante disponible (no hay datos de cuantos pacientes están inscritos en el programa de intercambio), solamente se necesitaría que un 5,09% de estos pacientes se inscribiesen al programa de intercambio para tener a los 200 pacientes del tamaño. Sin embargo, como se mencionaba anteriormente, hay que tener en cuenta que en los pacientes la mitad se empareja con su propio donante (son compatibles) y no formarían parte de la lista de espera, por lo que dicho porcentaje mencionado anteriormente del 5,09% debería ser mayor.

Un aspecto que podría llamar la atención es cómo el porcentaje de trasplantes medio-compatibles es significativamente mayor en estos modelos que en los datos reales. Por ejemplo, para un tamaño de 200 pacientes, con el modelo b) se tiene que un 33,85% (22,95% + 10,9%) de los pacientes recibe un trasplante cruzando la compatibilidad sanguínea mientras que en 2019 esa cifra fue de un 11%. Para explicar este hecho hay que tener en cuenta que con este modelo apenas un 2,3% de los pacientes no obtendrían un trasplante y un paciente preferirá recibir un trasplante medio-compatible antes que no recibir ninguno.

Sobre la importancia del tamaño del programa de parejas de pacientes-donantes.

Vamos a comparar el tamaño del programa de intercambios cruzados de España con otros europeos. Para el caso de España y Reino Unido se utilizarán las cifras que se recogen en (Biró, Haase-Kromwijk et al., 2019) mientras que para el caso escandinavo se utilizará lo expuesto en (Andersson, T., Wennberg, L., Lindnér, P., Duus Weinreich, I., Skov, K., Bistrup, C., 2020).

Para empezar, hay que tener en cuenta que en España se registran aproximadamente 110 parejas de paciente-donante en cada ronda de emparejamiento. Por su parte, en el programa de Reino Unido, que incluye a Inglaterra, Gales, Escocia y Irlanda del norte, esta cifra se eleva a 250 parejas. Mientras, Suecia cuenta con 35-40 parejas en cada ronda, aunque este país forma parte del programa escandinavo de trasplantes, Scandinaviatrasplant, en el cuál se incluye también a Estonia, Islandia, Finlandia, Noruega y Dinamarca, siendo el tamaño conjunto de aproximadamente 100 parejas.

Cabe destacar que hay pequeñas diferencias en el diseño de los programas de emparejamiento analizados. Por ejemplo, en el caso del programa escandinavo no se permiten donantes altruistas, aunque está previsto que una vez que el programa esté completamente implementado sí se permita (Scandinaviantransplant, 2018), a diferencia de Reino Unido y España que en la actualidad ya está disponible esa posibilidad.

Nótese que en este análisis cabe esperar que no haya mucha diferencia entre la sensibilización de los distintos pacientes de unos programas a otros, puesto que en todos los casos se trata de pacientes en la lista de espera.

Conociendo las cifras relativas al tamaño de los programas, podemos comparar los resultados sobre los trasplantes realizados.

En el programa escandinavo de trasplantes, para 100 parejas inscritas se realizan una media de 14 trasplantes,[10] por lo que son necesarias 7,1 parejas para realizar un intercambio. Sin embargo, cabe destacar que este programa es relativamente nuevo mientras que los otros dos programas ya llevan una década de funcionamiento, lo que puede hacer que ese ritmo sea menor ya que no está completamente implementado, además de que todavía no incorpora donantes altruistas, como habíamos comentado anteriormente.

[10] Ver http://www.scandiatransplant.org/data/scandiatransplant-figures (Acceso 31 de enero de 2021).

Por otra parte, en España, en media en los últimos años se realizan 30 trasplantes renales cruzados[11] y recordando que su tamaño es de aproximadamente 110 parejas, se tiene que son necesarias en media 3,7 parejas para realizar un intercambio.

Finalmente, en Reino Unido se realizan 85 trasplantes renales cruzados en media[12], cuando se registran alrededor de 250 parejas en cada ronda de emparejamiento, por lo que son necesarias 2,9 parejas para llevar a cabo un trasplante cruzado.

En conclusión, podemos ver como al aumentar el número de parejas inscritas en los programas, cada vez son necesarias menos parejas para conseguir un emparejamiento exitoso, lo que pone de relevancia la importancia de aumentar el tamaño de los programas de intercambio.

Otra posibilidad para lograr un mayor número de trasplantes que sale a la luz en el análisis es la cooperación con países del entorno mediante programas internacionales, aunque hay que tener en cuenta que siempre existirá una limitación de distancia debido al tiempo que un riñón puede estar fuera del cuerpo humano para ser trasladado, puesto que en estos programas se desplazan los órganos y no los pacientes (Biró et al., 2019).

Estimación del límite superior del tamaño del programa.

Sabiendo que la lista de espera finalizó 2019 con 3.933 pacientes inscritos, vamos a hacer una serie de supuestos a modo de ejemplo para entender mejor el potencial de mejora existente gracias a este tipo de programas de intercambio.

- Supuesto 1: El 25% de los pacientes tienen disponible un donante (1.000). Una justificación a este porcentaje es que la mayoría de los pacientes de la lista de espera son mayores, por lo que el coste-beneficio de recibir un riñón de donante vivo no compensa, ya que son pocos años de vida ganados para compensar la pérdida del riñón del donante.
- Supuesto 2: El programa de intercambio renal cruzado está generalizado y la mitad de ellos están dispuestos a apuntarse al junto con su donante, por lo que se inscriben un total de 500 parejas.

Para estudiar el funcionamiento del programa con este número de parejas inscritas vamos a utilizar los resultados vistos en la simulación para un tamaño de 200 pacientes. Nótese

[11] Ver http://www.ont.es/infesp/Paginas/Memorias.aspx (Acceso 30 de enero de 2021).
[12] Ver https://www.odt.nhs.uk/statistics-and-reports/organ-specific-reports/ (Acceso 31 de enero de 2021).

que estos porcentajes están infravalorados pues realmente al tener un número más amplio de pacientes inscritos en el programa, serían mayores.
Hemos visto que en todos los modelos alrededor de un 43-44% de los pacientes son compatibles con su propio donante por lo que vamos a excluir a un 44% de los pacientes ya que esos trasplantes no entrarían dentro del programa.
Utilizando el modelo a), que como se ha hecho hincapié anteriormente es el más similar al existente en España, se tendría que un 16,48% se trasplantaría con un paciente medio-compatible, es decir, 82 pacientes. Además, un 8,17% se trasplantaría con un paciente medio-compatible, es decir, 41 pacientes. En total, gracias al programa se beneficiarían 123 pacientes. Esta cifra se puede comparar con el número de trasplantes cruzados actual en España, que cómo se ha mencionado anteriormente en 2019 se situó en 33. Como hemos observado, una generalización del programa existente en España supondría un aumento de 33 a 123 trasplantes, contando únicamente los que se han producido por intercambio (además de estos habría trasplantes compatibles con sus propios donantes y medio compatibles).

Por otra parte, podríamos pensar que pasaría si se realizase una transición a un modelo más ambicioso como puede ser el modelo b). Para los supuestos descritos anteriormente, se tendría que un 20,07% de los pacientes se trasplantaría con un donante compatible mediante el intercambio, lo que serían 100 pacientes. Además, otro 10,9% también conseguiría un trasplante gracias al programa con un donante medio compatible, siendo en cifras 55 pacientes. Por tanto, tendríamos un total de 155 trasplantes en comparación con los 123 que vimos para el modelo a). Nótese que aumentan tanto los trasplantes compatibles como los trasplantes medio compatibles al pasar al modelo b). Aunque este incremento en el número de trasplantes es significativo, la cuestión más relevante es lograr que el programa tenga una mayor difusión, divulgarlo y convencer a los pacientes a participar, independientemente de cuál sea el modelo que se utilice. Evidentemente, se conseguirían resultados mejores con el modelo b (tanto en número de trasplantes como en trasplantes compatibles) pero el incremento en el número de trasplantes de pasar de un modelo a otro no es tan notorio cómo la mejora que supondría aumentar el número de pacientes inscritos.

5.4. Propuestas.

Una vez vistos los resultados de la simulación, considero principalmente tres propuestas para incrementar el número de trasplantes en España, priorizando los trasplantes compatibles.

En primer lugar, las autoridades sanitarias podrían realizar una mayor divulgación del programa actual de intercambios, recomendando este programa a aquellos pacientes que tengan la posibilidad de tener un donante y éste no sea compatible con ellos.
En la simulación y en la comparativa entre los tamaños de los programas de los países se ha podido observar la importancia del número de parejas inscritas. Cuantos más pacientes haya inscritos en el programa, más fácil será emparejar a los pacientes y donantes.
Hemos estimado un posible límite superior, viendo que los números de España están lejos de los que se obtendría con el modelo a) que podría asemejarse al programa implantado en nuestro país, por lo que existe un amplio margen de mejora que se basaría simplemente en que el programa esté más generalizado.

Por otra parte, la Autoridad del Programa podría implementar otro diseño más ambicioso del mismo como el expuesto en el modelo b).
Una vez que los pacientes han sido informados del programa, es posible que alguno de ellos sea medio compatible con su donante y prefiera trasplantarse inmediatamente sin tener que involucrarse en el programa, lo cuál llevaría a un proceso de desensibilización del paciente y a un mayor uso de inmunosupresores. Esto podría evitarse si se implementa el modelo b) pues así este paciente también entraría a formar parte del programa con el incentivo de que nunca empeorará, sólo podría mejorar recibiendo un trasplante compatible y evitando por tanto tener que someterse a más tratamientos para su desensibilización.
Hemos visto en la simulación cómo este modelo ha sido el que mejores resultados ha arrojado en todos los aspectos: tanto en número de trasplantes totales cómo en número de trasplantes compatibles.
Al realizar una transición del modelo a) al modelo b) todos los pacientes que estaban emparejados en el modelo a) lo seguirán estando en el modelo b). Sin embargo, hay que tener en cuenta que puede haber diferencias en si recibirán un riñón compatible o medio-compatible. La mayoría de ellos se mantendría igual, un porcentaje mejoraría (con el

modelo a obtenía un trasplante medio compatible y ahora en el modelo b obtiene uno compatible) y el resto empeoraría, pero este último porcentaje es menor que el de mejora por lo que es recomendable transicionar, Andersson y Kratz (2020).

Finalmente, la Autoridad del Programa podría ampliar el mismo internacionalmente. Como hemos visto en el ejemplo del programa escandinavo, el tamaño del programa se vería incrementado gracias a esta cooperación entre países cercanos, aunque habría que tener en cuenta las limitaciones que nos encontraríamos al tener que desplazar órganos vivos a mayor distancia.

6. CONCLUSIONES

Los avances logrados en los últimos años tanto a nivel económico, mediante el diseño de los primeros programas de intercambio renal cruzado, como a nivel médico, desarrollando técnicas que permiten realizar trasplantes cruzando la compatibilidad sanguínea, han abierto un abanico de nuevas posibilidades de trasplante para pacientes con enfermedad renal crónica avanzada.

El análisis de los distintos mecanismos de asignación lo hemos centrado en el modelo b debido que es el que mejores resultados obtiene en términos de trasplantes realizados y en priorización de trasplantes compatibles sobre los trasplantes medio compatibles, con el objetivo así de someter al paciente al menor número de tratamientos posible. Bajo el modelo b), se consigue identificar emparejamientos que son eficientes en el sentido de Pareto y máximos (se obtiene el mayor número de trasplantes posibles) pero manipulables por una parte de los pacientes. Aunque estos mecanismos sean manipulables, hay poco incentivo para que un paciente intente manipularlo debido a que el coste puede llegar a ser su vida.

En las simulaciones y las comparativas con otros países hemos visto la existencia de un potencial de mejora para la situación actual de España. En el caso de que España decidiese seguir apostando por su modelo actual, en los resultados vimos reflejada la necesidad de lograr una mayor difusión de su programa de intercambios para conseguir un mayor

número de parejas paciente-donante inscritas. Otra forma de aumentar el tamaño del programa sería expandirlo internacionalmente, aunque nos encontramos con limitaciones de tiempo y distancia. Finalmente, también podríamos considerar realizar una transición a un diseño de programa de intercambios renales cruzados más similar al modelo b, que vimos que era el que mejores resultados arrojaba en las simulaciones. Cualquiera de las alternativas podría lograr aumentar de forma considerable el número de trasplantes realizados en España, siempre que el sistema nacional de salud tenga los recursos necesarios (quirófanos disponibles, profesionales sanitarios, etc.) para hacer frente a una mayor demanda.

7. REFERENCIAS

Abdulkadiroğlu, A. & Sönmez, T. 2003, "School Choice: A Mechanism Design Approach", *American Economic Review,* vol. 93, no. 3, pp. 729-747.

Andersson, T., Wennberg, L., Lindnér, P., Duus Weinreich, I., Skov, K., Bistrup, C. 2020, *De första njurbytena mellan två skandinaviska länder är gjorda*, Läkartidningen 2019:117.

Andersson, T. & Kratz, J. 2020, "Pairwise Kidney Exchange over the Blood Group Barrier", *Review of Economic Studies,* vol. 87, no. 3, pp. 1091-1133.

Biró, P., Haase-Kromwijk, B., Andersson, T., Ásgeirsson, E.I., Baltesová, T., Boletis, I., Bolotinha, C., Bond, G., Böhmig, G., Burnapp, L., Cechlárová, K., Ciaccio, P.D., Ma, Fronek, J., Hadaya, K., Hemke, A., Jacquelinet, C., Johnson, R., Kieszek, R., Kuypers, D.R., Leishman, R., Macher, M., Manlove, D., Menoudakou, G., Salonen, M., Smeulders, B., Sparacino, V., Spieksma, F.C.R., Valentín, M.O., Wilson, N. & Van Der Klundert, J. 2019, "Building Kidney Exchange Programmes in Europe – An Overview of Exchange Practice and Activities", *Transplantation,* vol. 103, no. 7.

Biró, P., van de Klundert, J., Manlove, D., Pettersson, W., Andersson, T., Burnapp, L., Chromy, P., Delgado, P., Dworczak, P., Haase, B., Hemke, A., Johnson, R., Klimentova, X., Kuypers, D., Nanni Costa, A., Smeulders, B., Spieksma, F., Valentín, M.O. & Viana, A. 2019, "Modelling and optimisation in European Kidney Exchange Programmes", *European Journal of Operational Research,* vol. 13, no. 4.

European Directorate for the Quality of Medicines 2017, "Newsletter transplant: International figures on donation and transplantation 2016.", http://www.ont.es/publicaciones/Documents/NEWSLETTER%202017_baja%20(2).pdf.

Nicoló, A. & Rodríguez-Álvarez, C. 2012, "Transplant quality and patients' preferences in paired kidney exchange", *Games and Economic Behavior,* vol. 74, no. 1, pp. 299-310.

Nicoló, A. & Rodríguez-Álvarez, C. 2017, "Age-based preferences in paired kidney exchange", *Games and Economic Behavior,* vol. 102, no. 1, pp. 508-524.

Roth, A.E. 2015, *Who Gets What - and Why,* Eamon Dolan Books.

Roth, A.E. 1984, "The Evolution of the Labor Market for Medical Interns and Residents: A Case Study in Game Theory", *Journal of Political Economy,* vol. 92, no. 6, pp. 991-1016.

Roth, A.E., Sönmez, T. & Utku Ünver, M. 2004, "Kidney Exchange", *Quarterly Journal of Economics,* vol. 119, no. 2, pp. 457-488.

Roth, A.E., Sönmez, T. & Ünver, M.U. 2005, "Pairwise kidney exchange", *Journal of Economic Theory,* vol. 125, no. 2, pp. 151-188.

Roth, A.E., Sönmez, T. & Ünver, M.U. 2007, "Efficient Kidney Exchange: Coincidence of Wants in Markets with Compatibility-Based Preferences", *American Economic Review,* vol. 97, no. 3, pp. 828-851.

Rydberg, L., Rydberg, L., Bengtsson, A., Bengtsson, A., Samuelsson, O., Samuelsson, O., Nilsson, K., Nilsson, K., Breimer, M. & Breimer, M. 2005, "In vitro assessment of a new ABO immunosorbent with synthetic carbohydrates attached to sepharose", *Transplant International,* vol. 17, no. 11, pp. 666-672.

Sánchez-Escuredo, A., Alsina, A., Diekmann, F., Revuelta, I., Esforzado, N., Ricart, M.J., Cofán, F., Torregrosa, J.V., Campistol, J.M., Oppenheimer, F. & Fernandez, E. 2015, "Economic Analysis of the Treatment of End-stage Renal Disease Treatment: Living-donor Kidney Transplantation Versus Hemodialysis", *Transplantation Proceedings,* vol. 47, no. 1, pp. 30-33.

Scandinaviantransplant 2018, *STEP-programme v1.9*, http://www.scandiatransplant.org/organ-allocation/ScandiaKPDProgram1.9.pdf.

Sönmez, T. 1999, "Strategy-proofness and Essentially Single-valued Cores", *Econometrica,* vol. 67, pp. 677-689.

Sönmez, T. & Ünver, M.U. 2014, "Altruistically Unbalanced Kidney Exchange", *Journal of Economic Theory,* vol. 152, pp. 105-129.

Tyden, G., Donauer, J., Wadström, J., Kumlien, G., Wilpert, J., Nilsson, T., Genberg, H., Pisarski, P. & Tufveson, G. 2007, "Implementation of a Protocol for ABO-Incompatible Kidney Transplantation – A Three-Center Experience With 60 Consecutive Transplantations", *Transplantation,* vol. 83, no. 9, pp. 1153-1155.

Zenios, S.A., Woodle, E.S. & Ross, L.F. 2001, "Primum Non Nocere: Avoiding Harm to Vulnerable Wait List Candidates in an Indirect Kidney Exchange", *Transplantation,* vol. 72, no. 4, pp. 648-654.

8. ANEXO

Simulación_modelo_B

[ ]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import random
from gurobipy import *
GRB_LICENSE_FILE = '/Users/sara/Documents/AAA/gurobi.lic'
```

[ ]:
```
num_pacientes = 25
epsilon = 0.01
n = num_pacientes
num_simulaciones = 1000
q = 1
```

[ ]:
```
epsilon < 1/(2*n*q)
```

[ ]:
```
blood_info = pd.DataFrame(data=np.zeros((8, 8)),␣
 ↪columns=["AB+","AB-","A+","A-","B+","B-","0+","0-"])
blood_info.iloc[:,0] = 1.0
blood_info.iloc[7,:] = 1
for k in range(8):
    blood_info.iloc[k,k] = 1
blood_info.iloc[3,1] = 1
blood_info.iloc[3,2] = 1
blood_info.iloc[5,1] = 1
blood_info.iloc[5,4] = 1
blood_info.iloc[6,2] = 1
blood_info.iloc[6,4] = 1
```

[ ]:
```
def f_comp_matrix_b(num_pacientes):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    blood_probability_list = [0.025, 0.005, 0.36, 0.07, 0.08, 0.02, 0.35, 0.09]
    blood_matrix = np.zeros((num_pacientes, 2))
    blood_df = pd.DataFrame(data=blood_matrix, columns=["blood_pac",␣
 ↪"blood_don"])
    PRA_list = ["low", "medium", "high"]
    PRA_probability_list = [0.7019, 0.2, 0.0981]
```

```
    crossmatch_list = [0,1]
    comp_matrix = np.zeros((num_pacientes, num_pacientes))
    self_matched = []
    pi = []
    for i in range(num_pacientes):
        blood_df.iloc[i,0] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        blood_df.iloc[i,1] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        list_compat = []
        paciente_PRA = random.choices(PRA_list, PRA_probability_list)[0]

        for j in range(num_pacientes):
            if paciente_PRA == 'low':
                crossmatch_probability_list = [0.05, 0.95]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            elif paciente_PRA == 'medium':
                crossmatch_probability_list = [0.45, 0.55]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            else:
                crossmatch_probability_list = [0.9, 0.1]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])

        if paciente_PRA == 'low':
            pi.append(0.25)
        elif paciente_PRA == 'medium':
            pi.append(0.5)
        else:
            pi.append(0.75)


        if list_compat[i] == 1:
            x = blood_list.index(blood_df.iloc[i,1])
            y = blood_list.index(blood_df.iloc[i,0])
            if blood_info.iloc[x,y] == 1:
                self_matched.append(i)
            else:
                comp_matrix[i] = list_compat
        else:
            comp_matrix[i] = list_compat
    return [comp_matrix, self_matched, blood_df, pi]
```

[ ]:
```
def f_pi(num_pacientes, pi, blood_df):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        num_incompat_sang = 0
        sang_pac = blood_list.index(blood_df.iloc[i,0])
        for j in range(num_pacientes):
            sang_donante = blood_list.index(blood_df.iloc[j,1])
            if blood_info.iloc[sang_donante, sang_pac] != 1:
                num_incompat_sang = num_incompat_sang + 1
        if num_incompat_sang == 0:
            perc_incompat_sang = 0.5/num_pacientes
        else:
            perc_incompat_sang = num_incompat_sang/num_pacientes
        pi[i] = pi[i]*perc_incompat_sang
    return pi
```

[ ]:
```
def f_compatibility_matrix_b(num_pacientes, comp_matrix, blood_df):
    compatibility_matrix = np.zeros((num_pacientes, num_pacientes))
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        num_aleat = random.random()
        if num_aleat < 0.75:
            permite_half = True
        else:
            permite_half = False
        for j in range(num_pacientes):
            if comp_matrix[i,j] == 1:
                x = blood_list.index(blood_df.iloc[j,1])
                y = blood_list.index(blood_df.iloc[i,0])
                if blood_info.iloc[x,y] == 1:
                    compatibility_matrix[i,j] = 2
                else:
                    if permite_half is True:
                        if i == j:
                            compatibility_matrix[i,j] = 1
                        else:
                            if comp_matrix[i,i] != 1:
                                compatibility_matrix[i,j] = 1
    return compatibility_matrix
```

[ ]:
```
def f_exchanges(num_pacientes, compatibility_matrix, self_matched):
    fuera = np.copy(compatibility_matrix)
    for i in range(num_pacientes):
        j = 0
        marcador = 1
        for j in range(marcador, num_pacientes):
            if i != j:
```

```
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador = marcador + 1
        marcador2 = num_pacientes
        for j in range(0, i):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador2 = marcador2 - 1
    exchanges = pd.DataFrame(data=fuera)
    return exchanges
```

[ ]:
```
def f_matriz_pesos(epsilon, n, exchanges, pi, q, compatibility_matrix):
    if epsilon < 1/(2*n*q):
        matriz_pesos = np.zeros((exchanges.shape[1], exchanges.shape[1]))
        for i in range(exchanges.shape[1]):
            for j in range(exchanges.shape[1]):
                if compatibility_matrix[i,j] != 0:
                    if exchanges.iloc[i,j] == 2:
                        vij = epsilon/(2*n*q)
                    else:
                        vij = 0
                    if exchanges.iloc[j,i] == 2:
                        vji = epsilon/(2*n*q)
                    else:
                        vji = 0
                    if exchanges.iloc[i,j] != 0:
                        matriz_pesos[i,j] = pi[i] + pi[j] + vij + vji
        return matriz_pesos
    else:
        return print('Error en los epsilon')
```

[ ]:
```
def gurobi_opt(vertices, aristas):
    m = Model()
    m.params.OutputFlag = 0
    m.setParam('TimeLimit', 60)
    ciclos = {}
    for arista in aristas:
        comp1 = arista[0]
        comp2 = arista[1]
        if (comp1 <= comp2 and (comp2, comp1) in aristas):
            ciclos[(comp1, comp2)] = aristas[(comp1, comp2)]
    variables = {}
    for ciclo in ciclos:
        variables[ciclo] = m.addVar(vtype=GRB.BINARY)
    m.update()
    for vertice in vertices:
```

```
        restricciones = []
        for ciclo in variables:
            if vertice in ciclo:
                restricciones.append(variables[ciclo])
        if restricciones:
            m.addConstr(quicksum(restricciones[i] for i in␣
↪range(len(restricciones))) <= 1)
    m.setObjective(quicksum(variables[ciclo]*ciclos[ciclo] for ciclo in␣
↪ciclos), GRB.MAXIMIZE)
    m.optimize()
    if m.status != 2:
        return print("Error en Gurobi")
    sol = []
    for ciclo in variables:
        if (variables[ciclo].X == 1):
            sol.append(ciclo)
    return sol
```

[ ]:
```
def f_mejor_matching(matriz_pesos):
    df_prueba = pd.DataFrame(data = matriz_pesos)
    dicc = {}
    for i in range(len(df_prueba)):
        for j in range(len(df_prueba)):
            if df_prueba.iloc[i,j] != 0:
                dicc[(i,j)] = df_prueba.iloc[i,j]
    aristas = dicc
    vertices = [i for i in range(df_prueba.shape[0])]
    sol = gurobi_opt(vertices, aristas)
    return sol
```

[ ]:
```
def func_aux(result):
    lista = []
    for i in range(len(result)):
        if result[i][0] != result[i][1]:
            lista.append(result[i][0])
            lista.append(result[i][1])
    return lista
```

[ ]:
```
def func_aux2(larga, corta):
    for i in range(len(corta)):
        elemento = corta[i]
        if elemento in larga:
            larga.remove(elemento)
    return larga
```

```
[ ]: def f_calcula_medidas_b(num_pacientes, epsilon, n, q):
         funcion1 = f_comp_matrix_b(num_pacientes)
         comp_matrix = funcion1[0]
         self_matched = funcion1[1]
         blood_df = funcion1[2]
         pi2 = funcion1[3]
         pi = f_pi(num_pacientes, pi2, blood_df)
         compatibility_matrix = f_compatibility_matrix_b(num_pacientes, comp_matrix,␣
     ↪blood_df)
         exchanges = f_exchanges(num_pacientes, compatibility_matrix, self_matched)
         matriz_pesos = f_matriz_pesos(epsilon, n, exchanges, pi, q,␣
     ↪compatibility_matrix)
         result = f_mejor_matching(matriz_pesos)
         other_matched = []
         abo_exchange = 0
         abo_self_matched = []
         for i in range(len(result)):
             if result[i][0] == result[i][1]:
                 if compatibility_matrix[result[i][0], result[i][1]] == 1:
                     abo_self_matched.append(result[i][0])
             else:
                 if compatibility_matrix[result[i][0], result[i][1]] != 1:
                     other_matched.append(result[i][0])
                 if compatibility_matrix[result[i][1], result[i][0]] != 1:
                     other_matched.append(result[i][1])
         lista = func_aux(result)
         abo_exchange = func_aux2(lista, other_matched)
         perc_pacientes_self_matched = len(self_matched) / num_pacientes
         num_pacientes_transplant = len(self_matched) + len(other_matched) +␣
     ↪len(abo_exchange) + len(abo_self_matched)
         perc_pacientes_transplant = num_pacientes_transplant / num_pacientes
         perc_pacientes_no_transplant = 1 - perc_pacientes_transplant
         perc_transplant_exchange = len(other_matched) / num_pacientes
         perc_transpl_abo_self_matched = len(abo_self_matched) / num_pacientes
         perc_transpl_abo_exchange = len(abo_exchange) / num_pacientes
         return [perc_pacientes_no_transplant, perc_pacientes_self_matched,␣
     ↪perc_transpl_abo_self_matched, perc_transplant_exchange,␣
     ↪perc_transpl_abo_exchange]
```

```
[ ]: def simulaciones_b(num_simulaciones, num_pacientes, epsilon, n, q):
         perc_pacientes_no_transplant = []
         perc_pacientes_self_matched = []
         perc_transpl_abo_self_matched = []
         perc_transplant_exchange = []
         perc_transpl_abo_exchange = []
         for i in range(num_simulaciones):
             aux = f_calcula_medidas_b(num_pacientes, epsilon, n, q)
```

```
        perc_pacientes_no_transplant.append(aux[0])
        perc_pacientes_self_matched.append(aux[1])
        perc_transpl_abo_self_matched.append(aux[2])
        perc_transplant_exchange.append(aux[3])
        perc_transpl_abo_exchange.append(aux[4])
    media_perc_pacientes_no_transplant = np.mean(perc_pacientes_no_transplant)
    media_perc_pacientes_self_matched = np.mean(perc_pacientes_self_matched)
    media_perc_transpl_abo_self_matched = np.mean(perc_transpl_abo_self_matched)
    media_perc_transplant_exchange = np.mean(perc_transplant_exchange)
    media_perc_transpl_abo_exchange = np.mean(perc_transpl_abo_exchange)
    return [media_perc_pacientes_no_transplant,␣
↪media_perc_pacientes_self_matched
            , media_perc_transpl_abo_self_matched,␣
↪media_perc_transplant_exchange, media_perc_transpl_abo_exchange]
```

[ ]:
```
simulaciones_b(num_simulaciones, 25, epsilon, n, q)
```

[ ]:
```
simulaciones_b(num_simulaciones, 50, epsilon, n, q)
```

[ ]:
```
simulaciones_b(num_simulaciones, 100, epsilon, n, q)
```

[ ]:
```
simulaciones_b(num_simulaciones, 200, epsilon, n, q)
```

Simulación_modelo_A

[ ]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import random
from gurobipy import *
GRB_LICENSE_FILE = '/Users/sara/Documents/AAA/gurobi.lic'
```

[ ]:
```
num_pacientes = 25
epsilon = 0.01
n = num_pacientes
num_simulaciones = 1000
q = 1
```

[ ]:
```
epsilon < 1/(2*n*q)
```

[ ]:
```
blood_info = pd.DataFrame(data=np.zeros((8, 8)),␣
 ↪columns=["AB+","AB-","A+","A-","B+","B-","0+","0-"])
blood_info.iloc[:,0] = 1.0
blood_info.iloc[7,:] = 1
for k in range(8):
    blood_info.iloc[k,k] = 1
blood_info.iloc[3,1] = 1
blood_info.iloc[3,2] = 1
blood_info.iloc[5,1] = 1
blood_info.iloc[5,4] = 1
blood_info.iloc[6,2] = 1
blood_info.iloc[6,4] = 1
```

[ ]:
```
def f_comp_matrix(num_pacientes):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    blood_probability_list = [0.025, 0.005, 0.36, 0.07, 0.08, 0.02, 0.35, 0.09]
    blood_matrix = np.zeros((num_pacientes, 2))
    blood_df = pd.DataFrame(data=blood_matrix, columns=["blood_pac",␣
 ↪"blood_don"])
    PRA_list = ["low", "medium", "high"]
    PRA_probability_list = [0.7019, 0.2, 0.0981]
    crossmatch_list = [0,1]
```

```
    comp_matrix = np.zeros((num_pacientes, num_pacientes))
    self_matched = []
    abo_self_matched = []
    pi = []
    no_permite_half_list = []
    for i in range(num_pacientes):
        blood_df.iloc[i,0] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        blood_df.iloc[i,1] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        list_compat = []
        paciente_PRA = random.choices(PRA_list, PRA_probability_list)[0]
        if paciente_PRA == 'low':
            pi.append(0.25)
        elif paciente_PRA == 'medium':
            pi.append(0.5)
        else:
            pi.append(0.75)

        for j in range(num_pacientes):
            if paciente_PRA == 'low':
                crossmatch_probability_list = [0.05, 0.95]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            elif paciente_PRA == 'medium':
                crossmatch_probability_list = [0.45, 0.55]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            else:
                crossmatch_probability_list = [0.9, 0.1]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])

        num_aleat = random.random()
        if num_aleat < 0.75:
            permite_half = True
        else:
            permite_half = False
            no_permite_half_list.append(i)

        if list_compat[i] == 1:
            x = blood_list.index(blood_df.iloc[i,1])
            y = blood_list.index(blood_df.iloc[i,0])
            if blood_info.iloc[x,y] == 1:
                self_matched.append(i)
            else:
                if permite_half is True:
```

```
                    abo_self_matched.append(i)
                else:
                    no_permite_half_list.append(i)
        else:
            comp_matrix[i] = list_compat
    return [comp_matrix, self_matched, blood_df, pi, abo_self_matched,␣
↪no_permite_half_list]
```

[ ]:
```
def f_pi(num_pacientes, pi, blood_df):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        num_incompat_sang = 0
        sang_pac = blood_list.index(blood_df.iloc[i,0])
        for j in range(num_pacientes):
            sang_donante = blood_list.index(blood_df.iloc[j,1])
            if blood_info.iloc[sang_donante, sang_pac] != 1:
                num_incompat_sang = num_incompat_sang + 1
        if num_incompat_sang == 0:
            perc_incompat_sang = 0.5/num_pacientes
        else:
            perc_incompat_sang = num_incompat_sang/num_pacientes
        pi[i] = pi[i]*perc_incompat_sang
    return pi
```

[ ]:
```
def f_compatibility_matrix(num_pacientes, comp_matrix, blood_df,␣
↪no_permite_half_list):
    compatibility_matrix = np.zeros((num_pacientes, num_pacientes))
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        if i in no_permite_half_list:
            permite_half = False
        else:
            permite_half = True
        for j in range(num_pacientes):
            if comp_matrix[i,j] == 1:
                x = blood_list.index(blood_df.iloc[j,1])
                y = blood_list.index(blood_df.iloc[i,0])
                if blood_info.iloc[x,y] == 1:
                    compatibility_matrix[i,j] = 2
                else:
                    if permite_half is True:
                        compatibility_matrix[i,j] = 1
    return compatibility_matrix
```

[ ]:
```
def f_exchanges(num_pacientes, compatibility_matrix, self_matched):
    fuera = np.copy(compatibility_matrix)
    for i in range(num_pacientes):
```

```
        j = 0
        marcador = 1
        for j in range(marcador, num_pacientes):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador = marcador + 1
        marcador2 = num_pacientes
        for j in range(0, i):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador2 = marcador2 - 1
    exchanges = pd.DataFrame(data=fuera)
    return exchanges
```

[ ]:
```
def f_matriz_pesos(epsilon, n, exchanges, pi, q, compatibility_matrix):
    if epsilon < 1/(2*n*q):
        matriz_pesos = np.zeros((exchanges.shape[1], exchanges.shape[1]))
        for i in range(exchanges.shape[1]):
            for j in range(exchanges.shape[1]):
                if compatibility_matrix[i,j] != 0:
                    if exchanges.iloc[i,j] == 2:
                        vij = epsilon/(2*n*q)
                    else:
                        vij = 0
                    if exchanges.iloc[j,i] == 2:
                        vji = epsilon/(2*n*q)
                    else:
                        vji = 0
                    if exchanges.iloc[i,j] != 0:
                        matriz_pesos[i,j] = pi[i] + pi[j] + vij + vji
        return matriz_pesos
    else:
        return print('Error en los epsilon')
```

[ ]:
```
def gurobi_opt(vertices, aristas):
    m = Model()
    m.params.OutputFlag = 0
    m.setParam('TimeLimit', 60)
    ciclos = {}
    for arista in aristas:
        comp1 = arista[0]
        comp2 = arista[1]
        if (comp1 <= comp2 and (comp2, comp1) in aristas):
            ciclos[(comp1, comp2)] = aristas[(comp1, comp2)]
    variables = {}
```

```
    for ciclo in ciclos:
        variables[ciclo] = m.addVar(vtype=GRB.BINARY)
    m.update()
    for vertice in vertices:
        restricciones = []
        for ciclo in variables:
            if vertice in ciclo:
                restricciones.append(variables[ciclo])
        if restricciones:
            m.addConstr(quicksum(restricciones[i] for i in␣
↪range(len(restricciones))) <= 1)
    m.setObjective(quicksum(variables[ciclo]*ciclos[ciclo] for ciclo in␣
↪ciclos), GRB.MAXIMIZE)
    m.optimize()
    if m.status != 2:
        return print("Error en Gurobi")
    sol = []
    for ciclo in variables:
        if (variables[ciclo].X == 1):
            sol.append(ciclo)
    return sol
```

[ ]:
```
def f_mejor_matching(matriz_pesos):
    df_prueba = pd.DataFrame(data = matriz_pesos)
    dicc = {}
    for i in range(len(df_prueba)):
        for j in range(len(df_prueba)):
            if df_prueba.iloc[i,j] != 0:
                dicc[(i,j)] = df_prueba.iloc[i,j]
    aristas = dicc
    vertices = [i for i in range(df_prueba.shape[0])]
    sol = gurobi_opt(vertices, aristas)
    return sol
```

[ ]:
```
def f_calcula_medidas(num_pacientes, epsilon, n, q):
    funcion1 = f_comp_matrix(num_pacientes)
    comp_matrix = funcion1[0]
    self_matched = funcion1[1]
    blood_df = funcion1[2]
    pi2 = funcion1[3]
    pi = f_pi(num_pacientes, pi2, blood_df)
    no_permite_half_list = funcion1[4]
    compatibility_matrix = f_compatibility_matrix(num_pacientes, comp_matrix,␣
↪blood_df, no_permite_half_list)
    exchanges = f_exchanges(num_pacientes, compatibility_matrix, self_matched)
    matriz_pesos = f_matriz_pesos(epsilon, n, exchanges, pi, q,␣
↪compatibility_matrix)
```

```
    result = f_mejor_matching(matriz_pesos)
    other_matched = []
    abo_self_matched = funcion1[4]
    abo_exchange = 0
    for i in range(len(result)):
        if compatibility_matrix[result[i][0], result[i][1]] == 1:
            abo_exchange = abo_exchange + 1
        else:
            other_matched.append(result[i][0])
        if compatibility_matrix[result[i][1], result[i][0]] == 1:
            abo_exchange = abo_exchange + 1
        else:
            other_matched.append(result[i][1])
    perc_pacientes_self_matched = len(self_matched) / num_pacientes
    num_pacientes_transplant = len(self_matched) + len(other_matched) +␣
↪len(abo_self_matched) + abo_exchange
    perc_pacientes_transplant = num_pacientes_transplant / num_pacientes
    perc_pacientes_no_transplant = 1 - perc_pacientes_transplant
    perc_transplant_exchange = len(other_matched) / num_pacientes
    perc_transpl_abo_self_matched = len(abo_self_matched) / num_pacientes
    perc_transpl_abo_exchange = abo_exchange / num_pacientes
    return [perc_pacientes_no_transplant, perc_pacientes_self_matched,␣
↪perc_transpl_abo_self_matched, perc_transplant_exchange,␣
↪perc_transpl_abo_exchange]
```

[ ]:
```
def simulaciones(num_simulaciones, num_pacientes, epsilon, n, q):
    perc_pacientes_no_transplant = []
    perc_pacientes_self_matched = []
    perc_transpl_abo_self_matched = []
    perc_transplant_exchange = []
    perc_transpl_abo_exchange = []
    for i in range(num_simulaciones):
        aux = f_calcula_medidas(num_pacientes, epsilon, n, q)
        perc_pacientes_no_transplant.append(aux[0])
        perc_pacientes_self_matched.append(aux[1])
        perc_transpl_abo_self_matched.append(aux[2])
        perc_transplant_exchange.append(aux[3])
        perc_transpl_abo_exchange.append(aux[4])
    media_perc_pacientes_no_transplant = np.mean(perc_pacientes_no_transplant)
    media_perc_pacientes_self_matched = np.mean(perc_pacientes_self_matched)
    media_perc_transpl_abo_self_matched = np.mean(perc_transpl_abo_self_matched)
    media_perc_transplant_exchange = np.mean(perc_transplant_exchange)
    media_perc_transpl_abo_exchange = np.mean(perc_transpl_abo_exchange)
    return [media_perc_pacientes_no_transplant,␣
↪media_perc_pacientes_self_matched
            , media_perc_transpl_abo_self_matched,␣
↪media_perc_transplant_exchange, media_perc_transpl_abo_exchange]
```

[ ]:
```
simulaciones(num_simulaciones, 25, epsilon, n, q)
```

[ ]:
```
simulaciones(num_simulaciones, 50, epsilon, n, q)
```

[ ]:
```
simulaciones(num_simulaciones, 100, epsilon, n, q)
```

[ ]:
```
simulaciones(num_simulaciones, 200, epsilon, n, q)
```

Simulación_modelo_Base

[ ]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import random
from gurobipy import *
GRB_LICENSE_FILE = '/Users/sara/Documents/AAA/gurobi.lic'
```

[ ]:
```
num_pacientes = 25
epsilon = 0.01
n = num_pacientes
num_simulaciones = 1000
q = 1
```

[ ]:
```
epsilon < 1/(2*n*q)
```

[ ]:
```
blood_info = pd.DataFrame(data=np.zeros((8, 8)),␣
 ↪columns=["AB+","AB-","A+","A-","B+","B-","0+","0-"])
blood_info.iloc[:,0] = 1.0
blood_info.iloc[7,:] = 1
for k in range(8):
    blood_info.iloc[k,k] = 1
blood_info.iloc[3,1] = 1
blood_info.iloc[3,2] = 1
blood_info.iloc[5,1] = 1
blood_info.iloc[5,4] = 1
blood_info.iloc[6,2] = 1
blood_info.iloc[6,4] = 1
```

[ ]:
```
blood_info
```

[ ]:
```
def f_comp_matrix_basic(num_pacientes):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    blood_probability_list = [0.025, 0.005, 0.36, 0.07, 0.08, 0.02, 0.35, 0.09]
    blood_matrix = np.zeros((num_pacientes, 2))
    blood_df = pd.DataFrame(data=blood_matrix, columns=["blood_pac",␣
 ↪"blood_don"])
    PRA_list = ["low", "medium", "high"]
```

```
    PRA_probability_list = [0.7019, 0.2, 0.0981]
    crossmatch_list = [0,1]
    comp_matrix = np.zeros((num_pacientes, num_pacientes))
    self_matched = []
    pi = []
    for i in range(num_pacientes):
        blood_df.iloc[i,0] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        blood_df.iloc[i,1] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        list_compat = []
        list_compat_sang = []
        paciente_PRA = random.choices(PRA_list, PRA_probability_list)[0]

        for j in range(num_pacientes):
            if paciente_PRA == 'low':
                crossmatch_probability_list = [0.05, 0.95]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            elif paciente_PRA == 'medium':
                crossmatch_probability_list = [0.45, 0.55]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            else:
                crossmatch_probability_list = [0.9, 0.1]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])

        if paciente_PRA == 'low':
            pi.append(0.25)
        elif paciente_PRA == 'medium':
            pi.append (0.5)
        else:
            pi.append(0.75)

        if list_compat[i] == 1:
            x = blood_list.index(blood_df.iloc[i,1])
            y = blood_list.index(blood_df.iloc[i,0])
            if blood_info.iloc[x,y] == 1:
                self_matched.append(i)
        else:
            comp_matrix[i] = list_compat
    return [comp_matrix, self_matched, blood_df, pi]
```

[ ]:
```
def f_pi(num_pacientes, pi, blood_df):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
```

```
        num_compat_sang = 0
        sang_pac = blood_list.index(blood_df.iloc[i,0])
        for j in range(num_pacientes):
            sang_donante = blood_list.index(blood_df.iloc[j,1])
            if blood_info.iloc[sang_donante, sang_pac] == 1:
                num_compat_sang = num_compat_sang + 1
        perc_compat_sang = num_compat_sang/100
        pi[i] = pi[i]*perc_compat_sang
    return pi
```

[ ]:
```
def f_compatibility_matrix_basic(num_pacientes, comp_matrix, blood_df):
    compatibility_matrix = np.zeros((num_pacientes, num_pacientes))
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        for j in range(num_pacientes):
            if comp_matrix[i,j] == 1:
                x = blood_list.index(blood_df.iloc[j,1])
                y = blood_list.index(blood_df.iloc[i,0])
                if blood_info.iloc[x,y] == 1:
                    compatibility_matrix[i,j] = 2
    return compatibility_matrix
```

[ ]:
```
def f_exchanges(num_pacientes, compatibility_matrix, self_matched):
    fuera = np.copy(compatibility_matrix)
    for i in range(num_pacientes):
        j = 0
        marcador = 1
        for j in range(marcador, num_pacientes):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador = marcador + 1
        marcador2 = num_pacientes
        for j in range(0, i):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador2 = marcador2 - 1
    exchanges = pd.DataFrame(data=fuera)
    return exchanges
```

[ ]:
```
def f_matriz_pesos(epsilon, n, exchanges, pi, q, compatibility_matrix):
    if epsilon < 1/(2*n*q):
        matriz_pesos = np.zeros((exchanges.shape[1], exchanges.shape[1]))
        for i in range(exchanges.shape[1]):
            for j in range(exchanges.shape[1]):
                if compatibility_matrix[i,j] != 0:
```

```
                    if exchanges.iloc[i,j] == 2:
                        vij = epsilon/(2*n*q)
                    else:
                        vij = 0
                    if exchanges.iloc[j,i] == 2:
                        vji = epsilon/(2*n*q)
                    else:
                        vji = 0
                    if exchanges.iloc[i,j] != 0:
                        matriz_pesos[i,j] = pi[i] + pi[j] + vij + vji
        return matriz_pesos
    else:
        return print('Error en los epsilon')
```

[ ]:
```
def gurobi_opt(vertices, aristas):
    m = Model()
    m.params.OutputFlag = 0
    m.setParam('TimeLimit', 60)
    ciclos = {}
    for arista in aristas:
        comp1 = arista[0]
        comp2 = arista[1]
        if (comp1 <= comp2 and (comp2, comp1) in aristas):
            ciclos[(comp1, comp2)] = aristas[(comp1, comp2)]
    variables = {}
    for ciclo in ciclos:
        variables[ciclo] = m.addVar(vtype=GRB.BINARY)
    m.update()
    for vertice in vertices:
        restricciones = []
        for ciclo in variables:
            if vertice in ciclo:
                restricciones.append(variables[ciclo])
        if restricciones:
            m.addConstr(quicksum(restricciones[i] for i in␣
↪range(len(restricciones))) <= 1)
    m.setObjective(quicksum(variables[ciclo]*ciclos[ciclo] for ciclo in␣
↪ciclos), GRB.MAXIMIZE)
    m.optimize()
    if m.status != 2:
        return print("Error en Gurobi")
    sol = []
    for ciclo in variables:
        if (variables[ciclo].X == 1):
            sol.append(ciclo)
    return sol
```

[ ]:
```
def f_mejor_matching(matriz_pesos):
    df_prueba = pd.DataFrame(data = matriz_pesos)
    dicc = {}
    for i in range(len(df_prueba)):
        for j in range(len(df_prueba)):
            if df_prueba.iloc[i,j] != 0:
                dicc[(i,j)] = df_prueba.iloc[i,j]
    aristas = dicc
    vertices = [i for i in range(df_prueba.shape[0])]
    sol = gurobi_opt(vertices, aristas)
    return sol
```

[ ]:
```
def f_calcula_medidas_basic(num_pacientes, epsilon, n, q):
    funcion1 = f_comp_matrix_basic(num_pacientes)
    comp_matrix = funcion1[0]
    self_matched = funcion1[1]
    blood_df = funcion1[2]
    pi2 = funcion1[3]
    pi = f_pi(num_pacientes, pi2, blood_df)
    compatibility_matrix = f_compatibility_matrix_basic(num_pacientes,␣
↪comp_matrix, blood_df)
    exchanges = f_exchanges(num_pacientes, compatibility_matrix, self_matched)
    matriz_pesos = f_matriz_pesos(epsilon, n, exchanges, pi, q,␣
↪compatibility_matrix)
    result = f_mejor_matching(matriz_pesos)
    other_matched = []
    abo_self_matched = 0
    abo_exchange = 0
    for i in range(len(result)):
        other_matched.append(result[i][0])
        other_matched.append(result[i][1])
    perc_pacientes_self_matched = len(self_matched) / num_pacientes
    num_pacientes_transplant = len(self_matched) + len(other_matched)
    perc_pacientes_transplant = num_pacientes_transplant / num_pacientes
    perc_pacientes_no_transplant = 1 - perc_pacientes_transplant
    perc_transplant_exchange = len(other_matched) / num_pacientes
    perc_transpl_abo_self_matched = abo_self_matched / num_pacientes
    perc_transpl_abo_exchange = abo_exchange / num_pacientes
    return [perc_pacientes_no_transplant, perc_pacientes_self_matched,␣
↪perc_transpl_abo_self_matched, perc_transplant_exchange,␣
↪perc_transpl_abo_exchange]
```

[ ]:
```
def simulaciones_basic(num_simulaciones, num_pacientes, epsilon, n, q):
    perc_pacientes_no_transplant = []
    perc_pacientes_self_matched = []
    perc_transpl_abo_self_matched = []
    perc_transplant_exchange = []
```

```
    perc_transpl_abo_exchange = []
    for i in range(num_simulaciones):
        aux = f_calcula_medidas_basic(num_pacientes, epsilon, n, q)
        perc_pacientes_no_transplant.append(aux[0])
        perc_pacientes_self_matched.append(aux[1])
        perc_transpl_abo_self_matched.append(aux[2])
        perc_transplant_exchange.append(aux[3])
        perc_transpl_abo_exchange.append(aux[4])
    media_perc_pacientes_no_transplant = np.mean(perc_pacientes_no_transplant)
    media_perc_pacientes_self_matched = np.mean(perc_pacientes_self_matched)
    media_perc_transpl_abo_self_matched = np.mean(perc_transpl_abo_self_matched)
    media_perc_transplant_exchange = np.mean(perc_transplant_exchange)
    media_perc_transpl_abo_exchange = np.mean(perc_transpl_abo_exchange)
    return [media_perc_pacientes_no_transplant,␣
↪media_perc_pacientes_self_matched,
            media_perc_transpl_abo_self_matched,␣
↪media_perc_transplant_exchange, media_perc_transpl_abo_exchange]
```

[ ]:
```
simulaciones_basic(num_simulaciones, 25, epsilon, n, q)
```

[ ]:
```
simulaciones_basic(num_simulaciones, 50, epsilon, n, q)
```

[ ]:
```
simulaciones_basic(num_simulaciones, 100, epsilon, n, q)
```

[ ]:
```
simulaciones_basic(num_simulaciones, 200, epsilon, n, q)
```

Simulación_modelo_Altruista

[ ]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import random
from gurobipy import *
GRB_LICENSE_FILE = '/Users/sara/Documents/AAA/gurobi.lic'
```

[ ]:
```
num_pacientes = 25
epsilon = 0.01
n = num_pacientes
num_simulaciones = 1000
q = 1
```

[ ]:
```
epsilon < 1/(2*n*q)
```

[ ]:
```
blood_info = pd.DataFrame(data=np.zeros((8, 8)),␣
 ↪columns=["AB+","AB-","A+","A-","B+","B-","0+","0-"])
blood_info.iloc[:,0] = 1.0
blood_info.iloc[7,:] = 1
for k in range(8):
    blood_info.iloc[k,k] = 1
blood_info.iloc[3,1] = 1
blood_info.iloc[3,2] = 1
blood_info.iloc[5,1] = 1
blood_info.iloc[5,4] = 1
blood_info.iloc[6,2] = 1
blood_info.iloc[6,4] = 1
```

[ ]:
```
def f_comp_matrix_altruista(num_pacientes):

    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    blood_probability_list = [0.025, 0.005, 0.36, 0.07, 0.08, 0.02, 0.35, 0.09]
    blood_matrix = np.zeros((num_pacientes, 2))
    blood_df = pd.DataFrame(data=blood_matrix, columns=["blood_pac",␣
 ↪"blood_don"])
    PRA_list = ["low", "medium", "high"]
    PRA_probability_list = [0.7019, 0.2, 0.0981]
```

```
    crossmatch_list = [0,1]
    comp_matrix = np.zeros((num_pacientes, num_pacientes))
    pi = []
    self_matched = []
    for i in range(num_pacientes):
        blood_df.iloc[i,0] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        blood_df.iloc[i,1] = random.choices(blood_list,␣
↪blood_probability_list)[0]
        list_compat = []
        paciente_PRA = random.choices(PRA_list, PRA_probability_list)[0]
        if paciente_PRA == 'low':
            pi.append(0.25)
        elif paciente_PRA == 'medium':
            pi.append(0.5)
        else:
            pi.append(0.75)

        for j in range(num_pacientes):
            if paciente_PRA == 'low':
                crossmatch_probability_list = [0.05, 0.95]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            elif paciente_PRA == 'medium':
                crossmatch_probability_list = [0.45, 0.55]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
            else:
                crossmatch_probability_list = [0.9, 0.1]
                list_compat.append(random.choices(crossmatch_list,␣
↪crossmatch_probability_list)[0])
        comp_matrix[i] = list_compat
    return [comp_matrix, self_matched, blood_df, pi]
```

[ ]:
```
def f_pi(num_pacientes, pi, blood_df):
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        num_compat_sang = 0
        sang_pac = blood_list.index(blood_df.iloc[i,0])
        for j in range(num_pacientes):
            sang_donante = blood_list.index(blood_df.iloc[j,1])
            if blood_info.iloc[sang_donante, sang_pac] == 1:
                num_compat_sang = num_compat_sang + 1
        perc_compat_sang = num_compat_sang/100
        pi[i] = pi[i]*perc_compat_sang
    return pi
```

[ ]:
```
def f_compatibility_matrix_altruista(num_pacientes, comp_matrix, blood_df):
    compatibility_matrix = np.zeros((num_pacientes, num_pacientes))
    blood_list = ["AB+","AB-","A+","A-","B+","B-","0+","0-"]
    for i in range(num_pacientes):
        for j in range(num_pacientes):
            if comp_matrix[i,j] == 1:
                x = blood_list.index(blood_df.iloc[j,1])
                y = blood_list.index(blood_df.iloc[i,0])
                if blood_info.iloc[x,y] == 1:
                    compatibility_matrix[i,j] = 2
    return compatibility_matrix
```

[ ]:
```
def f_exchanges(num_pacientes, compatibility_matrix, self_matched):
    fuera = np.copy(compatibility_matrix)
    for i in range(num_pacientes):
        j = 0
        marcador = 1
        for j in range(marcador, num_pacientes):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador = marcador + 1
        marcador2 = num_pacientes
        for j in range(0, i):
            if i != j:
                if fuera[i,j] == 0:
                    fuera[j,i] = 0
            marcador2 = marcador2 - 1
    exchanges = pd.DataFrame(data=fuera)
    return exchanges
```

[ ]:
```
def f_matriz_pesos(epsilon, n, exchanges, pi, q, compatibility_matrix):
    if epsilon < 1/(2*n*q):
        matriz_pesos = np.zeros((exchanges.shape[1], exchanges.shape[1]))
        for i in range(exchanges.shape[1]):
            for j in range(exchanges.shape[1]):
                if compatibility_matrix[i,j] != 0:
                    if exchanges.iloc[i,j] == 2:
                        vij = epsilon/(2*n*q)
                    else:
                        vij = 0
                    if exchanges.iloc[j,i] == 2:
                        vji = epsilon/(2*n*q)
                    else:
                        vji = 0
                    if exchanges.iloc[i,j] != 0:
                        matriz_pesos[i,j] = pi[i] + pi[j] + vij + vji
```

```
        return matriz_pesos
    else:
        return print('Error en los epsilon')
```

[ ]:
```
def gurobi_opt(vertices, aristas):
    m = Model()
    m.params.OutputFlag = 0
    m.setParam('TimeLimit', 60)
    ciclos = {}
    for arista in aristas:
        comp1 = arista[0]
        comp2 = arista[1]
        if (comp1 <= comp2 and (comp2, comp1) in aristas):
            ciclos[(comp1, comp2)] = aristas[(comp1, comp2)]
    variables = {}
    for ciclo in ciclos:
        variables[ciclo] = m.addVar(vtype=GRB.BINARY)
    m.update()
    for vertice in vertices:
        restricciones = []
        for ciclo in variables:
            if vertice in ciclo:
                restricciones.append(variables[ciclo])
        if restricciones:
            m.addConstr(quicksum(restricciones[i] for i in␣
↪range(len(restricciones))) <= 1)
    m.setObjective(quicksum(variables[ciclo]*ciclos[ciclo] for ciclo in␣
↪ciclos), GRB.MAXIMIZE)
    m.optimize()
    if m.status != 2:
        return print("Error en Gurobi")
    sol = []
    for ciclo in variables:
        if (variables[ciclo].X == 1):
            sol.append(ciclo)
    return sol
```

[ ]:
```
def f_mejor_matching(matriz_pesos):
    df_prueba = pd.DataFrame(data = matriz_pesos)
    dicc = {}
    for i in range(len(df_prueba)):
        for j in range(len(df_prueba)):
            if df_prueba.iloc[i,j] != 0:
                dicc[(i,j)] = df_prueba.iloc[i,j]
    aristas = dicc
    vertices = [i for i in range(df_prueba.shape[0])]
    sol = gurobi_opt(vertices, aristas)
```

```
    return sol
```

[ ]:
```
def f_calcula_medidas_altruista(num_pacientes, epsilon, n, q):
    funcion1 = f_comp_matrix_altruista(num_pacientes)
    comp_matrix = funcion1[0]
    self_matched = funcion1[1]
    blood_df = funcion1[2]
    pi2 = funcion1[3]
    pi = f_pi(num_pacientes, pi2, blood_df)
    compatibility_matrix = f_compatibility_matrix_altruista(num_pacientes,␣
↪comp_matrix, blood_df)
    exchanges = f_exchanges(num_pacientes, compatibility_matrix, self_matched)
    matriz_pesos = f_matriz_pesos(epsilon, n, exchanges, pi, q,␣
↪compatibility_matrix)
    result = f_mejor_matching(matriz_pesos)
    other_matched = []
    abo_self_matched = 0
    abo_exchange = 0
    for i in range(len(result)):
        if result[i][0] == result[i][1]:
            self_matched.append(result[i][0])
            if compatibility_matrix[result[i][0], result[i][1]] == 1:
                abo_self_matched = abo_self_matched + 1
        else:
            other_matched.append(result[i][0])
            other_matched.append(result[i][1])
            if compatibility_matrix[result[i][0], result[i][1]] == 1:
                abo_exchange = abo_exchange + 1
            if compatibility_matrix[result[i][1], result[i][0]] == 1:
                abo_exchange = abo_exchange + 1
    perc_pacientes_self_matched = len(self_matched) / num_pacientes
    num_pacientes_transplant = len(self_matched) + len(other_matched)
    perc_pacientes_transplant = num_pacientes_transplant / num_pacientes
    perc_pacientes_no_transplant = 1 - perc_pacientes_transplant
    perc_transplant_exchange = len(other_matched) / num_pacientes
    perc_transpl_abo_self_matched = abo_self_matched / num_pacientes
    perc_transpl_abo_exchange = abo_exchange / num_pacientes
    return [perc_pacientes_no_transplant, perc_pacientes_self_matched,␣
↪perc_transpl_abo_self_matched, perc_transplant_exchange,␣
↪perc_transpl_abo_exchange]
```

[ ]:
```
def simulaciones_altruista(num_simulaciones, num_pacientes, epsilon, n, q):
    perc_pacientes_no_transplant = []
    perc_pacientes_self_matched = []
    perc_transpl_abo_self_matched = []
    perc_transplant_exchange = []
    perc_transpl_abo_exchange = []
```

```
    for i in range(num_simulaciones):
        aux = f_calcula_medidas_altruista(num_pacientes, epsilon, n, q)
        perc_pacientes_no_transplant.append(aux[0])
        perc_pacientes_self_matched.append(aux[1])
        perc_transpl_abo_self_matched.append(aux[2])
        perc_transplant_exchange.append(aux[3])
        perc_transpl_abo_exchange.append(aux[4])
    media_perc_pacientes_no_transplant = np.mean(perc_pacientes_no_transplant)
    media_perc_pacientes_self_matched = np.mean(perc_pacientes_self_matched)
    media_perc_transpl_abo_self_matched = np.mean(perc_transpl_abo_self_matched)
    media_perc_transplant_exchange = np.mean(perc_transplant_exchange)
    media_perc_transpl_abo_exchange = np.mean(perc_transpl_abo_exchange)
    return [media_perc_pacientes_no_transplant,␣
↪media_perc_pacientes_self_matched, media_perc_transpl_abo_self_matched
           , media_perc_transplant_exchange, media_perc_transpl_abo_exchange]
```

[ ]:
```
simulaciones_altruista(num_simulaciones, 25, epsilon, n, q)
```

[ ]:
```
simulaciones_altruista(num_simulaciones, 50, epsilon, n, q)
```

[ ]:
```
simulaciones_altruista(num_simulaciones, 100, epsilon, n, q)
```

[ ]:
```
simulaciones_altruista(num_simulaciones, 200, epsilon, n, q)
```