UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

IMPLEMENTACIÓN Y OPTIMIZACIÓN DEL PROCESO PARA RESOLVER EL PROBLEMA DE ASIGNACIÓN DE ÓRGANOS USANDO PROGRAMACIÓN ENTERA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
ALAN GUTIÉRREZ RUIZ

TUTOR:
DR. JAVIER GARCÍA GARCÍA
FACULTAD DE CIENCIAS, UNAM

CIUDAD DE MÉXICO, NOVIEMBRE DE 2016

Hoja de datos del jurado

1. Datos del alumno
   Gutiérrez
   Ruiz
   Alan
   alangr@ciencias.unam.mx
   Universidad Nacional Autónoma de México
   Maestría en Ciencia e Ingeniería de la Computación
   306159363

2. Datos del tutor
   Dr.
   Javier
   García
   García

3. Datos del sinodal 1
   Dr.
   Luis Bernardo
   Morales
   Mendoza

4. Datos del sinodal 2
   Dra.
   Hanna
   Jadwiga
   Oktaba

5. Datos del sinodal 3
   Dr.
   Sergio
   Rajsbaum
   Gorodezky

6. Datos del sinodal 4
   M. en C.
   María Guadalupe Elena
   Ibargüengoitia
   González

Agradecimientos

Este trabajo va dedicado especialmente a mi mamá Paty, a quien la he visto como un claro ejemplo por hacer las cosas con entrega y corazón. Muchas gracias mamá.

De igual manera, le agradezco a mi familia por estar cerca de mi. Estos últimos dos años han sido difíciles pero sé que puedo contar con ustedes y ustedes conmigo. Le agradezco a mi papá Paulino por ser el soporte dentro de nuestra familia y a mis hermanos Pablo y Pamela, que aunque son menores que yo, créanme, les he aprendido mucho y he disfrutado mucho con ustedes.

A mi tutor Javier, que me ha brindado su apoyo y confianza desde que lo conocí. Realmente he aprendido mucho de usted, desde que fui su alumno y tesista en licenciatura, luego alumno y tesista en maestría, hasta ahora que he tenido la oportunidad de colaborar con ud. en sus proyectos y dando clases de ayudantía. Muchas gracias profe, a ud. y a su esposa Norma por brindarme un espacio en su casa y poder disfrutar de su compañía y de su hermosa persona.

A mis amigos que me han acompañado en esta etapa dentro de la UNAM también muchas gracias. A Fozz, Jonas, Vic, Peña, Maya, Maggi, Aida, Melvin, Oga, Rafa, Diana, Marta, Josh, Sebas, Cat, Fabi, Fausto y a todos los que me faltaron, gracias. Gracias porque hemos tenido muchos lindos momentos, algunos de ellos difíciles, otros de mucho aprendizaje y definitivamente muchos de ellos muy divertidos. De ustedes he aprendido que la vida va más allá de la escuela y que hay que disfrutar de ella.

A mi amiga de toda la vida Nidia, por apoyarme en mis momentos difíciles y a seguir adelante. Gracias por escucharme y estar ahí conmigo. Nos vemos poco pero siempre disfruto estar contigo, me haces reír y hasta aprender nuevas palabras. Eres genial.

A Luis, Karla, Ari, Mariana, Natali y Rubén gracias. Gracias porque verlos a ustedes me hizo darme cuenta que hay personas brillantes en todas las áreas de la UNAM (y en México). Aunque algunos de ustedes aún los conozco de poco tiempo, realmente me han marcado en el sentido de querer siempre seguir adelante, a soñar y alcanzar tus metas, incluso si es fuera de este país. Con ustedes también aprendí a que asimiliar nuevos idiomas es fácil y divertido cuando lo haces con amigos.

A Isaías y a Angie, porque con ustedes empezó esta etapa de la maestría. Los conocí cuando ustedes iban saliendo de ella y de alguna manera me apoyaron e inspiraron para dar el último paso y aplicar al programa. Gracias chicos.

A mis profesores, a todos ellos, algunos de ellos mis sinodales en esta tesis. Muchas gracias. Gracias a la Maestra Lupita Ibargüengoitia y a los Doctores Sergio Rajsbaum, Hanna Oktaba y Luis Morales por darme muchas de las mejores clases que tuve y por apoyarme con sus consejos y observaciones en mi trabajo aquí presentado.

A mis tíos y tías por estar cerca de nuestra familia y apoyar cuando se requería. Gracias especialmente a mi tía Chelis, a mi tía Maria, a mi tío Marco y a mi tío José.

Gracias a la UNAM, a la Facultad de Ciencias, al programa del Posgrado en Ciencia e Ingeniería de la Computación y al Conacyt porque me han permitido seguir con mis estudios y poder retribuir ese mismo esfuerzo en la sociedad. El trabajo aquí presentado va con todo mi empeño y dedicación por apoyar una tarea específica en el sector salud, los trasplantes de órganos.

A Daniel Scognamiglio, a Jose Ángel y a Raúl Rivera por ser parte fundamental de nuestro equipo de trabajo para la realización de este trabajo. Muchas gracias.

Finalmente, quiero agradecer a un persona muy especial que conocí en mi etapa de licenciatura y que de igual manera, me inspiró a seguir con mis estudios de maestría: a mi profe René Villeda. Gracias René porque contigo aprendí que las clases se disfrutan, que las clases son fáciles cuando te apasionan y que hay que estar cerca de los alumnos, disfrutar de su compañía, atender a sus inquietudes y seguir aprendiendo de ellos. Gracias a ti me acerqué a mi tutor Javier y es por tu trabajo que yo intento hacer lo mismo ahora con mis alumnos. Muchas gracias René.

Contents

Chapter 1

Introducción

La salud es una condición de bienestar física y mental de la cual gozamos todos y cada uno de los seres vivos. Con el paso del tiempo, y de acuerdo a una serie de eventos, ésta puede irse deteriorando hasta llegar a un punto en que la única salida posible sea el fallecimiento. Con el apoyo de las ciencias médicas es posible recuperar la salud, ya sea por medio de reposo, medicinas, tratamientos, cirugía, trasplante de órganos, entre otros. El trabajo que se presenta en esta tesis está relacionado, particularmente, con el trasplante de órganos en México.

De acuerdo al Centro Nacional de Trasplantes (CENATRA) de la Secretaría de Salud, en su Reporte Nacional 2014 de la Donación de Trasplantes: en México, para el 31 de Diciembre del 2014 había 7,293 receptores en espera de córnea; 11,302 en espera de riñón; 394 en hígado y 42 más en corazón. Por otra parte y para el mismo año, se llevaron a cabo 3,204 trasplantes de córnea; 2,610 de riñón; 131 de hígado y 52 de corazón. Para los años 2013 y 2012 las cifras son muy similares, con un ligero incremento hacia 2014. Esto es porque, año con año, hay menos donantes que nuevos receptores.

Con respecto a los datos anteriores, y aclarando que la salud siempre es un tema de consideración para la sociedad -y para cada persona-, se llevó a cabo este trabajo para la elaboración de un software que permita encontrar una óptima asignación de órganos para un conjunto de donantes y receptores. Dicho software conformó parte de un proyecto desarrollado en 2015-2016 por la Facultad de Ciencias de la UNAM para el Centro Nacional de Trasplantes de la Secretaría de Salud, con el fin de apoyar sus actividades diarias y su proceso de trasplante de órganos.

El software desarrollado, contempló programas bilaterales de donación de órganos, hasta ahora poco conocidos en México. Para ejemplificar, supóngase el caso de donación de riñón. Este tipo de programas permiten a los pacientes receptores, acompañados de donantes vivos aunque biológicamente incompatibles[1], intercambiar donantes a través de ciclos o de cadenas de trasplantes.

En un **ciclo** con dos parejas donante-receptor, cada receptor recibe el órgano del donante de la otra pareja. De manera más general y teniendo más de dos parejas donante-receptor, se pueden crear ciclos donde se asigne a cada donante un receptor de tal manera que todos los individuos salgan beneficiados de la mejor manera.

En cambio, una **cadena** de donación inicia por algún donante indirecto[2] entregando su órgano a alguno de los receptores de las parejas donante-receptor, el donante relacionado al receptor beneficiado entrega su órgano al receptor de alguna otra pareja donante-receptor, y así sucesivamente hasta llegar a que el último donante pueda entregar su órgano a algún receptor que no pertenezca a ninguna de las parejas donante-receptor (es decir, un receptor en lista de espera).

La idea de formar estos ciclos y cadenas es clara: incrementar el número de trasplantes a realizar pero, sobretodo, incrementar el grado de compatibilidad entre donante y receptor de cada trasplante. Al programa de donación de órganos que sólo contempla una cadena, se le denomina programa Dominó; al programa que sólo contempla dos o tres ciclos se le conoce como Cruzado.

El objetivo del software fue: obtener una determinada combinación de ciclos y cadenas de trasplantes independientes entre sí y de longitud variable, tal que el grado de compatibilidad donante-receptor fuese máximo para cada asignación. Dicho problema quedó reducido a un problema de combinatoria, donde, para encontrar la mejor solución, había que buscar sobre todas las soluciones posibles. Una búsqueda exhaustiva para obtener la solución al problema hubiese tomado complejidad exponencial. Sin embargo, la solución aquí descrita trabaja con complejidad polinomial en el caso promedio, gracias al enfoque que se le dió, apoyándonos en [1] por utilizar Programación Entera [11] para encontrar la solución óptima.

[1] Algún amigo o familiar dispuesto a donar su riñón, aunque incompatible por razones médicas.
[2] Donante vivo altruista o cadáver no relacionado al receptor del órgano.

1.1 Objetivo

Describir el análisis e implementación de dos procesos desarrollados para la asignación óptima de órganos. Sean k, $k2$, q y $q2$ números enteros positivos.

- El primer proceso, el Proceso Completo, trata de resolver el problema de encontrar la combinación óptima de ciclos y cadenas para trasplantes entre un conjunto de donantes indirectos, parejas incompatibles y receptores en lista de espera. Dicha solución deberá tener a lo más $k2$ ciclos, a lo más $q2$ cadenas, cada ciclo con tamaño a lo más k y cada cadena con tamaño a lo más q.
- El segundo proceso, el Proceso Cruzado, trata de resolver el problema de encontrar la combinación óptima de ciclos para trasplantes entre un conjunto de parejas incompatibles. Dicha solución deberá tener a lo más $k2$ ciclos, cero cadenas, y ciclos de tamaño a lo más dos.

Nótese que el segundo proceso es un subcaso del primero. Sin embargo, se llevó a cabo su programación aparte, por motivos de optimización que se verán más adelante (en la Sección 3.1).

1.2 Motivación

Tuvimos dos motivos a considerar para tomar este trabajo como tesis de posgrado. La primera fue, el hecho de trabajar un problema con complejidad NP-duro y poder establecer una solución rápida y eficiente en un tiempo mucho menor, por medio de Programación Entera. La segunda fue precisamente, llevar a cabo una solución óptima e innovadora en México. Una solución que permita, de la mano de la Secretaría de Salud -por medio del Centro Nacional de Trasplantes-, mejorar el bienestar en cuestiones de salud dentro de nuestro país.

1.3 Trabajo relacionado

La idea de un intercambio renal entre una pareja donante receptor incompatible y otra, fue primeramente propuesta por Rapaport [17] en 1986 y después por Ross et al. [18] en 1997.

En Estados Unidos[3] desde el año 2000 a la fecha, se han llevado a cabo varios avances, entre los cuales se destaca lo siguiente:

[3]Estados Unidos país pionero en este sector de investigación.

- En 2000 se realiza el primer trasplante de riñón cruzado en Estados Unidos. Un ciclo de tamaño dos, por parte del Hospital de Rhode Island [29].
- En 2005 se establece un primer algoritmo de asignación de riñón para ciclos de tamaño dos [21]. Un año después se proponen nuevas soluciones que integraban ciclos pequeños y cadenas de tamaño a lo más tres [23][20].
- Hasta este momento cada trasplante de órganos dentro de un ciclo o cadena de trasplantes debía hacerse de manera simultánea por cuestiones logísticas y de incentivos[4].
- En 2007 se lleva a cabo la primera cadena larga no-simultánea [5]. En 2012 las cadenas largas no-simultáneas se vuelven estándar debido a su éxito [19][2].

Para 2015 en México, suceden dos eventos importantes con respecto a los programas bilaterales de donación de órganos. Uno, el Hospital Civil de Guadalajara realiza el primer intercambio de riñón (ciclo de tamaño dos) en México [28]. Y dos, se lleva a cabo un programa pionero llamado SIATRE[3][32], con el objetivo de obtener cadenas de trasplantes, apoyándose del algoritmo de Búsqueda en Anchura (BFS por sus siglas en inglés) para su funcionamiento. Con estos dos primeros pasos y aunado al presente trabajo, se pretende colocar a México a la vanguardia en este sector a nivel América Latina.

Descripción de los capítulos

En el Capítulo 2, se presenta la definición fomal del problema de Asignación de Órganos. En el Capítulo 3, se describen los procesos Completo y Cruzado, la justificación de sus respectivos diseños, análisis y descripción de su implementación. En el Capítulo 4, se presenta una demostración para indicar la igualdad de soluciones que los programas *Chainscycles* y SIATRE obtienen con parámetros equivalentes. *Chainscycles*, el nombre del programa desarrollado, y SIATRE, un programa similiar y precursor del presente trabajo. En el Capítulo 5, se encuentran los resultados generados por el sistema y sus respectivos análisis, además de la descripción del hardware y sofware utilizados para las pruebas. Y finalmente, en el Capítulo 6, se presenta el análisis de los resultados, las conclusiones y trabajo a futuro.

[4]Se trata este punto más adelante en la Sección 2.1

[5]Todos los trasplantes involucrados dentro de una cadena no-simultánea no necesariamente se realizan de manera simultánea.

Chapter 2

Antecedentes y definiciones

Este capítulo contiene las siguientes definiciones: Problema de Asignación de Órganos, trasplante, donante, donante indirecto, receptor, pareja incompatible, Proceso Completo, Proceso Cruzado, Programación Lineal y método Simplex.

Empezando por lo básico. ¿Qué es un trasplante? En este proyecto, un trasplante se refiere a trasladar un órgano, tejido o un conjunto de células de una persona (donante) a otra (receptor). El donante del órgano o tejido a trasplantar no necesariamente debe ser una persona con vida. Si una persona donadora sufre muerte cerebral, sus órganos pueden ser conservados por medio de diversos métodos con la intención de que su funcionamiento no se vea afectado y sea de utilidad para otro receptor que los requiera. La lista de órganos y tejidos trasplantables incluye: pulmón, corazón, riñón, hígado, páncreas, intestino, estómago, piel, córnea, médula ósea, sangre, hueso, entre otros, siendo el riñón el órgano más comúnmente trasplantado a nivel mundial.

De acuerdo a los requerimientos del sistema realizado, el proceso de trasplante es (de manera básica) como sigue:

1. El médico diagnostica un padecimiento que ha afectado gravemente un órgano o tejido específico de un paciente, quien es enviado a un especialista para determinar si requiere un trasplante de órgano.

2. El paciente se somete a una serie de exámenes médicos que permiten determinar sus condiciones específicas y la utilidad o no del trasplante.

3. Los casos son puestos a consideración y, en su caso avalados, por médicos especialistas del comité interno de trasplantes de un hospital.

4. Una vez que el paciente es aceptado ingresa sus datos en una base de datos electrónica. Los criterios a considerar para llevar a cabo el trasplante son:
   - la antigüedad en la base de datos electrónica,
   - la urgencia ante la inminente pérdida de vida,
   - la oportunidad o viabilidad del trasplante,
   - la compatibilidad entre el órgano donado y el receptor,
   - entre otros criterios de índole médica.
5. En caso de haber donantes disponibles, el sistema inicia un proceso de asignación de órganos y se llevan a cabo los trasplantes correspondientes.
6. El hospital da seguimiento a aquellos pacientes involucrados en los trasplantes.

2.1 Problema de Asignación de Órganos

La siguiente definición de problema resalta por su importancia: el problema lo definiremos en términos del Problema de Asignación Renal (KEP por sus siglas en inglés). Sin embargo, se puede extender a la forma generalizada de Asignación de Órganos.

Dentro del Problema de Asignación Renal, se tiene involucrado lo siguiente:

- Receptores. Hay dos tipos: receptores en lista de espera y receptores provenientes de parejas incompatibles.
- Donantes. Hay dos tipos: donantes indirectos (donantes vivos altruistas y cadáveres) y donantes provenientes de parejas incompatibles.
- Listas de compatibilidad. Por cada pareja (donante, receptor) se tiene una relación de compatibilidad de trasplante.

El problema es el siguiente: dados los conjuntos S, P y T; los tipos de cadenas SPT, SP y ST; y los ciclos PP; el **objetivo** es encontrar la combinación de ciclos y cadenas ajenos entre sí que represente la solución óptima de asignación de órganos, conforme al grado de compatibilidad de los trasplantes, donde:

- S es el conjunto de nodos sólo con salida que representa a la lista de donantes indirectos,

- P es el conjunto de nodos de entrada y salida que representa la lista de parejas incompatibles,
- T es el conjunto de nodos sólo con entrada que representa la lista de receptores en espera,
- SPT el tipo de cadenas que empiezan con un nodo en S, pasan por nodos en P y terminan con un nodo en T (i.e. una cadena SPT empieza con un donante indirecto de S y le otorga su órgano a un receptor en P, la pareja de éste último a su vez le otorga su órgano a otro receptor en P, y así sucesivamente hasta que un donante de pareja incompatible en P le done a un receptor en lista de espera en T. De manera análoga pasa con SP, ST y PP),
- SP el tipo de cadenas que empiezan con un nodo en S, pasan por nodos en P y terminan con un nodo en P[1],
- ST el tipo de cadenas que empiezan con un nodo en S y terminan con un nodo en T,
- PP el tipo de ciclos que empiezan con un nodo en P, posiblemente pasan por por nodos en P y terminan con un nodo en P

Véanse las Figuras 2.1, 2.2, 2.3 y 2.4 para hacer una mejor visualización del problema.

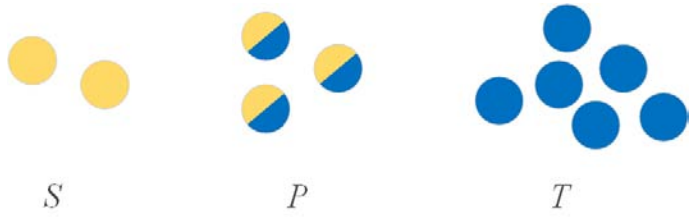


Figure 2.1: Actores del Problema de Asignación Renal. El conjunto S se refiere a los donantes indirectos, P a las parejas incompatibles y T a los receptores en lista de espera. Los donantes en P son del mismo color que los del conjunto S; de manera análoga con los receptores en P y T.

[1]Nótese que en este tipo de cadenas, el último nodo deja libre a un donante del conjunto P, que más adelante (en otra asignación renal) pudiera reinterpretarse como si fuese un nodo del conjunto T.

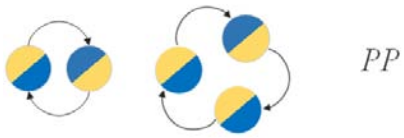


Figure 2.2: Ejemplos de ciclos. El primero de tamaño dos y el segundo de tamaño tres. Hay un solo tipo de ciclos: PP.

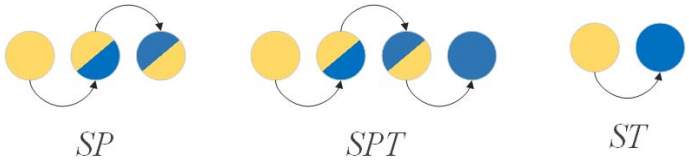


Figure 2.3: Ejemplos de cadenas. Las cadenas son de tamaño 3, 4 y 2, respectivamente. Hay tres tipos de cadenas: SP, SPT y ST.

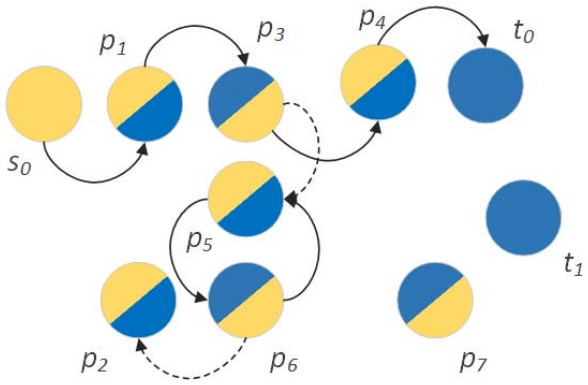


Figure 2.4: Ejemplo de un problema KEP visto desde una gráfica. Las flechas punteadas significan posibilidad de trasplante y las flechas bien definidas indican la solución óptima (la cual está conformada por la cadena s_0-p_1-p_3-p_4-t_0 y el ciclo p_5-p_6). Nótese que puede haber nodos sin vértices de salida o entrada.

Otras consideraciones del problema

En un trasplante se llevan a cabo dos cirugías, una donde se extrae el órgano y otra en donde se recibe. Para asegurar que cada receptor reciba su riñón antes de que su respectiva pareja done su riñón, los trasplantes involucrados en un ciclo son llevados a cabo de manera simultánea. De no ser así, si un donante ya no está

disponible para el trasplante a un receptor del cual su pareja ya ha donado su riñón: no sólo ese receptor no recibe el riñón sino que ya no le es posible participar en futuros intercambios (ver ejemplo en Figura 2.5).

Debido a que los trasplantes involucrados en un ciclo deben llevarse a cabo de manera simultánea, y a que el número de quirófanos y equipos médicos disponibles para hacer los trasplantes en un centro médico puede ser limitado, entonces se tiene una **nueva limitante**: restringir el número máximo de trasplantes que se pueden hacer de manera simultánea en un ciclo. Dicho de otra forma: se acota el tamaño máximo en los ciclos.

De manera análoga al hecho de limitar el número de ciclos, y para llevar a cabo un sistema más personalizable para el cliente, se añadieron **tres parámetros más**: el número máximo de ciclos permitidos, el tamaño máximo en cadenas y el número máximo de cadenas permitidas.

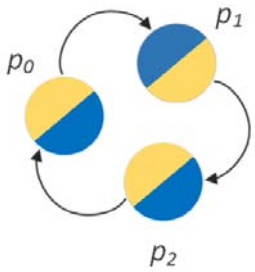


Figure 2.5: Ejemplo de ciclo de tamaño tres. Un ejemplo de ruptura dentro de este ciclo sería cuando: el donante de p_2 ya no puede donar, entonces el receptor de p_0 ya no obtiene su riñón y el receptor de p_2 se queda sin pareja para una nueva asignación, es decir se convierte en receptor en lista de espera. Un ciclo de tamaño tres involucra tres trasplantes simultáneos, equivalentes a seis cirugías simultáneas (en tres de ellas se extrae el órgano y en otras tres se recibe).

Para finalizar con la descripción del problema, se añade una **última condición** de acuerdo a lo siguiente: supóngase que por alguna razón se deba añadir a un elemento -ya sea donante indirecto, pareja incompatible o receptor en lista de espera- en la asignación de trasplantes de manera obligada; entonces la solución a obtener debe contener dicho elemento.

Problema de Intercambio de Órganos Cruzado

Hasta ahora ya se describió el problema en su totalidad. El problema de Intercambio de Órganos Cruzado sólo se refiere a un problema derivado del anterior. Únicamente consiste en fijarse en el conjunto de parejas incompatibles y obtener ciclos excluyentes, normalmente de tamaño 2 ó 3, tal que la suma de las compatibilidades por cada trasplante sea máxima.

Se hace mención particular de este subproblema pues, se llevó a cabo el desarrollo de un segundo proceso. El Proceso Cruzado resuelve el problema descrito sólo para ciclos de tamaño 2 y, el Proceso Completo resuelve -como su nombre lo dice- el problema completo (ciclos y cadenas de cualquier tamaño).

Por otra parte cabe aclarar que, en la práctica es muy común llevar a cabo trasplantes con ciclos de tamaño dos, por cuestiones de regla de negocio.

2.2 Programación Lineal

De acuerdo a [6], la Programación Lineal es una de las más desarrolladas y utilizadas ramas del área de Investigación de Operaciones. Se refiere a la optimización de asignación de recursos disponibles hacia actividades competentes, bajo una serie de restricciones impuestas por la naturaleza del problema a ser estudiado. Estas restricciones pueden ser de tipo financieras, tecnológicas, organizacionales, de marketing, entre otras[2].

En términos generales, la Programación Lineal se puede ver como una representación matemática que pretende calcular la mejor asignación posible de recursos disponibles, utilizando exclusivamente modelos con funciones lineales.

Un problema de Programación Lineal se expresa por medio de una función objetivo, un conjunto de variables y una serie de restricciones. La función objetivo es una expresión que se pretende, ya sea, maximizar o minimizar.

[2]De hecho, los orígenes de la Programación Lineal se remontan a la Segunda Guerra Mundial, donde surge como un modelo matemático para planificar los gastos a fin de reducir los costos al ejército y aumentar las pérdidas del enemigo.

A continuación se da la estructura de un problema de Programación Lineal.

$$\begin{aligned} \text{minimizar} \quad & z = c^T x \\ \text{sujeto a} \quad & Ax \leq b, \qquad (2.1) \\ & x \geq 0, \qquad (2.2) \end{aligned}$$

Donde x representa el vector de variables a ser determinadas; c y b son vectores de coeficientes conocidos; A una matriz conocida de coeficientes; juntos (c, b y A) representan la expresión lineal de las restricciones; finalmente, c^T una matriz transpuesta de coeficientes, representa la expresión a ser minimizada (o maximizada, según sea el caso).

Las desigualdades $Ax \leq b$ y $x \geq 0$ son las restricciones que especifican un politopo convexo sobre el cual la función objetivo, será optimizada.

En este contexto, y para llevar a cabo la minimización, se dice que, dos vectores son comparables cuando ambos tienen las mismas dimensiones. Si cada entrada en el primero es menor o igual a la correspondiente entrada del segundo, entonces se puede decir que el primer vector es menor o igual al segundo.

2.2.1 El método Simplex

En 1947, el estadounidense George Bernard Dantzig desarrolló el método Simplex [8] para resolver de manera general el Problema de Programación Entera.

Este método es un proceso iterativo que permite ir mejorando la solución en cada paso. La razón matemática de esta mejora radica en que el método consiste en caminar del vértice de un poliedro a un vértice vecino de manera que, se disminuya el valor de la función objetivo (o aumente, según sea el caso del problema).

El método Simplex se puede ver como una forma ordenada de escanear vértices. Dado que el número de vértices que presenta un poliedro solución es finito, siempre se hallará solución, en caso de existir.

Análisis del método

Este método es muy eficiente en la práctica. De acuerdo a su complejidad, el peor caso toma $O(nm2^n)$ (Klee-Minty [24]), aunque su caso promedio es polinomial (Karloff [11]), con n el número de variables y m el número de restricciones.

2.2.2 Variantes de Programación Lineal

- La **Programación Lineal Entera Mixta** (MILP por sus siglas en inglés) involucra problemas en donde sólo algunas de las variables están restringidas a ser números enteros, mientras que otras no.
- Si todas las variables necesitan ser enteras, se llama **Programación Entera** (IP por sus siglas en inglés).
- Si todas las variables necesitan ser 1 o 0, se llama **Programación Lineal 0-1**.

Los casos en que se requiere que la solución óptima se componga únicamente de valores enteros para las variables, conllevan a un nuevo problema pues, muchas veces, la solución del programa lineal truncado está lejos de ser el óptimo entero. La resolución a este problema se obtiene analizando las posibles alternativas de valores enteros de esas variables en un entorno alrededor de la solución obtenida considerando las variables reales [11].

El Problema de Asignación de Órganos puede representarse como un Problema en Programación Entera y resolverse mediante el método Simplex. Se justifica la selección del método Simplex dentro del proyecto, debido a su extraordinaria eficiencia computacional y robustez. Esto último pudo corroborarse dentro de uno de los experimentos realizados en este trabajo donde, se compara el desempeño de dicho método para resolver un problema NP-difícil tal como el **Problema del Agente Viajero** (TSP por sus siglas en inglés), en comparación con otros tipos de modelos computacionales tal como el Algoritmo de Ramificación y Poda [12].

El Problema del Agente Viajero, formulado por los matemáticos W.R. Hamilton y Thomas Kirkman en el siglo XVIII, se enuncia de la siguiente manera: *Dada una lista de ciudades y el costo de viaje entre cada par de ellas, ¿cuál es la ruta más económica de viajar, de tal manera que se visite cada ciudad exactamente una vez y se regrese al mismo punto de origen?*

TSP se puede modelar por medio de Programación Lineal [5]:

Sea $G(V, E)$ una gráfica dirigida.
Notación: Sea $\delta(v)$ el conjunto de aristas que entran o que salen de v, y $\delta(A)$ el conjunto de aristas que entran o que salen de v, para cada $v \in A$.

$$\text{minimizar} \quad \sum_{e \in E} x_e w_e \tag{2.3}$$

$$\text{sujeto a} \quad \sum_{e \in \delta(S)} x_e \geq 2 \qquad \forall S \subset V, S \neq \emptyset, \tag{2.4}$$

$$\sum_{e \in \delta(v)} x_e = 2 \qquad \forall v \in V, \tag{2.5}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E. \tag{2.6}$$

Explicación:

(2.3) La función objetivo indica que se desea minimizar la suma de los pesos de las aristas seleccionadas en el tour final.

(2.4) Se asegura que al menos dos aristas de cada corte $\delta(S)$ deban estar en la solución.

(2.5) Se garantiza que exactamente dos aristas están conectadas para cada vértice.

(2.6) La bandera x_e de la arista e, indica con un 1 si esa arista se toma en cuenta en la solución final, 0 en caso contrario.

2.3 SMBD y UDF

Los Sistemas Manejadores de Bases de Datos (SMBD) son sistemas de cómputo que permiten a los usuarios a crear y dar mantenimiento a Bases de Datos[9].

Las Funciones Definidas por el Usuario (UDF por sus siglas en inglés) son un mecanismo que proveen los Sistemas Manejadores de Bases de Datos [9] para extender la funcionalidad de los mismos. Una UDF ofrece ciertas ventajas, dos de las cuales serían: uno, mejorar el tiempo de procesamiento de los datos, y dos, evitar la exposición de los mismos a ambientes inseguros [16].

Chapter 3

Diseño e implementación del proceso

Este capítulo describe los procesos Completo y Cruzado, su diseño, complejidad computacional e implementación en el programa denominado *Chainscycles*.

3.1 Diseño por Programación Lineal

De manera formal, una instancia del Programa de Asignación de Órganos (Ver Figura 3.1) contiene:

- una gráfica $G(V, E)$, donde:
  - $V = S \cup P \cup T$, donde:
    - $S = \{s_0, s_1, \ldots, s_{m-1}\}$ es el conjunto de m donantes indirectos,
    - $P = \{p_0, p_1, \ldots, p_{n-1}\}$ es el conjunto de n parejas incompatibles,
    - $T = \{t_0, t_1, \ldots, t_{r-1}\}$ es el conjunto de r pacientes en lista de espera,
  - E es el conjunto de aristas dirigidas entre cada vértice del conjunto S hacia cada vértice en P y T, y de cada vértice en P hacia cada vértice en P y T. Los pesos de estas aristas estan dadas por las matrices W_{SP}, W_{ST}, W_{PP} y W_{PT}, donde:
    - W_{SP} es la matriz de compatibilidad (de tamaño $m \times n$) entre donantes indirectos y parejas compatibles.
    - W_{ST} es la matriz de compatibilidad (de tamaño $m \times r$) entre donantes indirectos y receptores en lista de espera.

  - W_{PP} es la matriz de compatibilidad (de tamaño $n \times n$) entre donantes y receptores de las parejas compatibles.
  - W_{PT} es la matriz de compatibilidad (de tamaño $n \times r$) entre donantes de las parejas incompatibles y receptores en lista de espera.
  - Nota: el valor de las entradas (i, j), de cualquiera de estas matrices está dada por números enteros positivos, los cuales hacen referencia al nivel de compatibilidad entre el donante de la fila i con el receptor de la columna j; una entrada con valor cero indica que no existe compatiblidad.

- un parámetro k como cota máxima para el tamaño de los ciclos,
- un parámetro $k2$ como cota máxima para el número de ciclos permitidos,
- un parámetro q como cota máxima para el tamaño de las cadenas,
- un parámetro $q2$ como cota máxima para el número de cadenas permitidas.

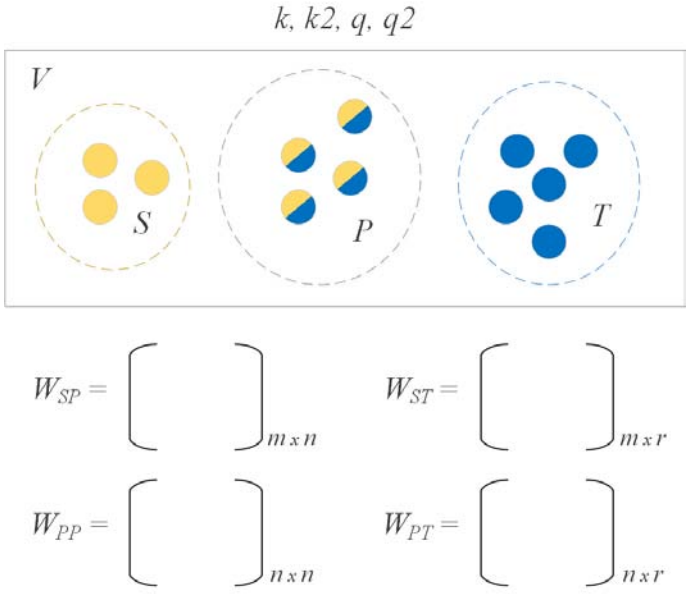


Figure 3.1: Parámetros de una instancia de Problema de Asignación de Órganos. $V = S \cup P \cup T$ el conjunto de nodos conformado por donantes indirectos, parejas incompatibles y receptores en lista de espera; W_{SP}, W_{ST}, W_{PP}, W_{PT} las matrices de adyacencia entre los nodos; k, $k2$, q y $q2$ cotas de restricción.

3.1.1 Modelo del Proceso Completo

Para conseguir el objetivo presentado en la Sección 2.1 se presenta el siguiente modelo en Programación Lineal Entera (inspirado en [1]).

Notación: Para cada v en V, sea $\delta^-(v)$ el conjunto de aristas que entran a v y $\delta^+(v)$ el conjunto de aristas que salen de v.

$$\text{maximizar} \quad \sum_{e \in E} x_e w_e \tag{3.1}$$

$$\text{sujeto a} \quad \sum_{e \in \delta^+(v)} x_e \leq 1 \qquad \forall v \in S, \tag{3.2}$$

$$\sum_{e \in \delta^-(v)} x_e \leq \sum_{e \in \delta^+(v)} x_e \qquad \forall v \in P, \tag{3.3}$$

$$\sum_{e \in \delta^+(v)} x_e \leq 1 \qquad \forall v \in P, \tag{3.4}$$

$$\sum_{e \in \delta^-(v)} x_e \leq 1 \qquad \forall v \in T, \tag{3.5}$$

$$\sum_{e \in \delta^+(v)} x_e \leq q2 \qquad v \in S, \tag{3.6}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E, \tag{3.7}$$

$$w_e \in \mathbb{R}^+ \qquad \forall e \in E. \tag{3.8}$$

Explicación:

(3.1) La función objetivo indica que se desea maximizar el peso de las aristas seleccionadas en los ciclos y cadenas solución.

(3.2) Para cada vértice en S, el número de aristas que salen es a lo más uno.

⇒ Para cada donante indirecto, el número de veces a donar es a lo más uno.

(3.3) Para cada vértice en P, el número de aristas que entran debe ser menor o igual al que salen.

⇒ Para cada pareja incompatible, el número de veces que se recibe un órgano es a lo más el número de veces que se dona.

(3.4) Para cada vértice en P, el número de aristas que salen es a lo más uno.

⇒ Para cada pareja incompatible, el número veces a donar es a lo más uno.

(3.5) Para cada vértice en T, el número de aristas que entran es a lo más uno.

⇒ Para cada receptor en lista de espera, el número veces veces que se recibe un órgano es a lo más uno.

(3.6) El número máximo de salidas por parte de los vértices en S es $q2$.

⇒ El número máximo de veces a donar por parte de los donantes no directos es $q2$. Cada donante sólo puede donar a lo más una vez (por la Restricción 3.2). Por lo que, el número máximo de cadenas es a lo más $q2$.

(3.7) La bandera x_e de la arista e, indica con un 1 si esa arista se toma en cuenta en la solución final, 0 en caso contrario.

⇒ La bandera x_e de la arista $e(i, j)$ indica con un 1 si en la asignación renal se lleva a cabo el trasplante de órgano del donante i al paciente j, 0 en caso contrario

(3.8) Los pesos de las aristas pueden ser cualquier número real positivo.

⇒ Las compatibilidades entre los pacientes sólo pueden ser representadas por números reales positivos.

Introduciendo datos y ejecutando este modelo se obtienen un conjunto de cadenas (a lo más $q2$ cadenas, de acuerdo a la Restricción 3.6) y ciclos disjuntos entre sí, los cuales representan una serie de trasplantes. Dicho conjunto también garantiza que la suma de cada una de las compatibilidades en cada emparejamiento sea máxima.

¿Porqué éste modelo entrega ciclos y cadenas? Nótese que en la Restricción 3.7, x_e indica una bandera para cada $e \in E$. Es decir, se tiene una matriz binaria en x. Lo único que se tiene que hacer para obtener los ciclos y cadenas es interpretar dicha matriz. En cualesquier entrada (i, j) que haya un 1 significa que esa arista está dentro de un ciclo o alguna cadena. Posteriormente, nos fijamos en la fila j para encontrar algún otro 1 (sólo puede haber un 1 en ese renglón, pues así se tienen las restricciones en el modelo); de existir un 1 en la entrada (j, h), significa

que esa arista está dentro del mismo ciclo o cadena. Se continúa del mismo modo, hasta que la última entrada $(i*, j*)$ contenga un 1 y que $j* = i$ -esto implicaría que se tiene un ciclo-; o que se contenga un 0 -significaría que se tiene una cadena con un último elemento en $j*$-. Cada que se encuentra un 1, se guarda el índice del vértice referido y, posteriormente, se actualiza la entrada con un 0. Se obtienen todos los ciclos y cadenas cuando la matriz x contiene únicamente ceros.

Todo va bien hasta ahora, sin embargo, a este modelo aún le falta cumplir con tener ciclos de tamaño a lo más k, tener a lo más $k2$ ciclos y tener cadenas de tamaño a lo más q. Estas últimas tres restricciones no se ponen en el modelo anterior a propósito. No se introducen, con el fin de *relajar* el problema. De no hacerlo así, el número de restricciones sería muy largo y contraproducente. A continuación se da una explicación más a detalle.

Supóngase que se desea añadir la restricción de tener sólo ciclos de tamaño a lo más k. Esto implicaría que dentro del modelo se sabe cómo *identificar* un ciclo, lo cual es imposible si lo que se tienen son ciclos y cadenas. Suponiendo que las cadenas y ciclos se encuentran por separado, habría que crear un número exponencial de restricciones con respecto a k para poder obtener ciclos con tamaño a lo más k[1]. Aparte de esto último, habría que agregar que, al unir tanto ciclos y cadenas, se tendría que afrontar un esfuerzo computacional extra, al tener que asegurar la condición de que sigan siendo excluyentes entre ellos. El hecho de tener la oportunidad de identificar ciclos durante el proceso en que se desarrolla el método de Simplex, sería un tema a debatir. Un primer obstáculo sería modificar el código de la biblioteca de funciones de Programación Entera[2]. Suponiendo que no hay problema con ello, se mejoraría el tiempo de búsqueda del método en algunos casos, dado lo siguiente: si sólo se encuentran ciclos que no cumplen con la regla (ser menores a k), entonces el programa terminaría pronto; de haber ciclos que cumplen con la regla, el programa seguiría ejecutándose hasta encontrar la solución deseada, sin embargo, surgen dos detalles: uno, que en cada paso, habría que hacer la verificación de que se siga cumpliendo con la regla; y dos, que cada verificación toma tiempo cuadrático con respecto al número de nodos que se tengan.

Involucrar las restricciones correspondientes para $k2$ y q al modelo de Programación Entera, implicaría un problema análogo al anterior (para k).

[1] Al menos una restricción por cada tamaño de ciclo, del 1 a k, más cada una de las combinaciones de agrupamiento de ciclos

[2] Se utilizó un software de Programación Entera para resolver el problema.

Entonces, ¿cómo lidiar con aquellas restricciones que involucran a k, $k2$ y q?. Lo que se hizo fue lo siguiente: omitir las restricciones correspondientes a estos parámetros y correr el programa con el modelo tal cual se propuso originalmente. Al terminar la ejecución y obtener el resultado, lo único que había que hacer era interpretar la matriz y determinar si aquella solución encontrada cumplía o no con lo especificado por los parámetros k, $k2$ y q. Esta determinación resulta ser mucho más sencilla que incluir desde un principio las restricciones correspondientes. Si la solución cumplía con lo especificado, entonces el proceso terminaba ahí, de lo contrario se introducía la solución arrojada de vuelta al modelo pero ahora como una restricción; una restricción que obligase que, la solución a obtener debía ser diferente a la anterior. Para lograr esto último hubo que hacer uso de la operación lógica XOR. La idea es como sigue a continuación:

1. La solución que arroja la biblioteca de Programación Entera es un arreglo de bits. Este arreglo binario es el equivalente a la matriz binaria x mencionada en la Restricción 3.8, pero linealizada. Ver Figura 3.2.

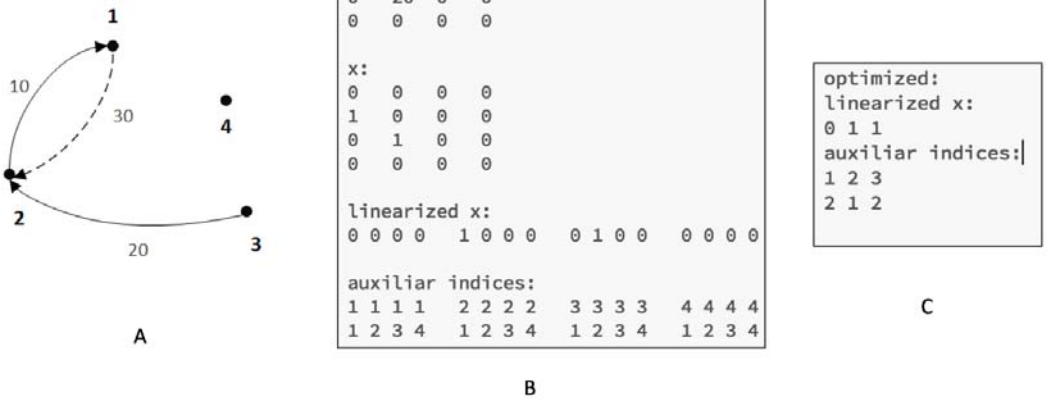


Figure 3.2: En la parte A se presenta una pequeña gráfica. En la parte B se muestra la matriz de pesos en w con respecto a dicha gráfica; en x una matriz solución por parte de Programación Entera; posteriormente se muestra la misma matriz x, pero linealizada, la cual cuenta con dos arreglos auxiliares que muestran de forma ordenada las coordenadas originales. Finalmente, en la parte C se muestra la misma x linealizada, pero sólo conservando información de los elementos relevantes (al tratarse de una matriz no completa, varias aristas no están necesariamente definidas, por lo que se omiten los valores de las aristas inexistentes).

2. Para marcar una solución como nueva, se utiliza la operación XOR entre cada elemento del arreglo actual con el correspondiente -de acuerdo al orden- del arreglo anterior.

3. Si la suma de los elementos resultado del XOR es mayor a 0, significa que los arreglos son diferentes. Ver Figura 3.3.

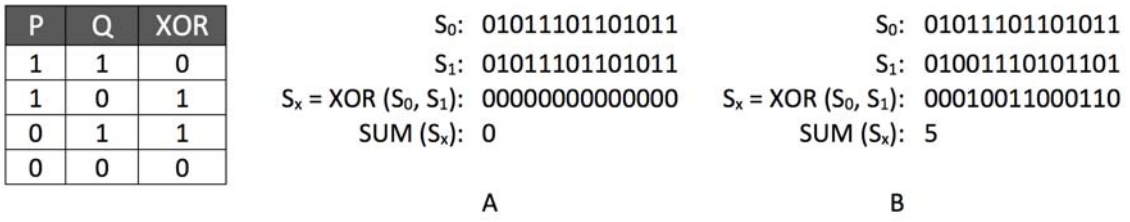


Figure 3.3: Tabla XOR y ejemplo de uso. En la parte A se puede ver que los arreglos S_0 y S_1 son iguales, por lo que la suma de los elementos resultado de XOR es 0. En la parte B las arreglos son distintos por 5 elementos.

Durante este punto del diseño del programa surgió un problema: la operación lógica XOR no es una operación definida dentro de la Programación Lineal. Hubo que hacer unos ajustes para poder hacer uso de ella . Si p y q son valores dentro de $\{1, 0\}$, entonces se puede obtener dicha operación de la siguiente manera: $\text{XOR}(p, q) = \overline{p}q + p\overline{q}$. Entiéndase $\overline{p}$ por p negada.

Otro problema a considerar fue que la operación lógica NOT (o negación) tampoco es una operación definida dentro de la Programación Lineal. Sin embargo, se pudo hacer uso de ella por medio de una ecuación: sea p un valor dentro de $\{1, 0\}$, entonces se puede obtener $\overline{p}$ con: $p + \overline{p} = 1$.

Recapitulando lo anterior: dentro de nuestro modelo de Programación Lineal, para obtener una nueva solución, diferente a las α anteriores, hay que agregar α restricciones tal como se indica en la Ecuación 3.9.

$$\sum_{e \in E} \overline{x_e^i} x_e + x_e^i \overline{x_e} \geq 1, \forall i \in \{1, \ldots, \alpha\}, \tag{3.9}$$

Donde x_e^i es el valor de una constante, que representa el valor de x_e en la i-ésima solución.

Por último, para considerar aquellos donantes o receptores de forma obligada en la solución, habría que modificar ligeramente el modelo presentado. De hecho, a cada vértice (ya sea en S, P ó T) que deba incluirse, habría que modificar su respectiva restricción en 3.2, 3.4 ó 3.5. En particular, bastaría cambiar la relación de desigualdad (en este caso menor o igual que 1) por una de igualdad (igual a 1). La justificación de este cambio es, forzar al modelo a que al menos una arista salga o entre al vértice en cuestión. Ejemplo: supóngase que se debe incluir al donante indirecto $v_0 \in S$. La Restricción 3.2 se modificaría, para ese vértice en particular, de la siguiente manera:

$$\sum_{e \in \delta^+(v_0)} x_e = 1, v_0 \in S$$

3.1.2 Modelo del Proceso Cruzado

El siguiente modelo responde al problema de encontrar sólo ciclos de tamaño dos (tal como se presentó en la Sección 2.1).

Notación: Para cada v en V, sea $\delta(v)$ el conjunto de aristas que entran a v unión el conjunto de aristas que salen de v. Es decir $\delta(v) = \delta^-(v) \cup \delta^+(v)$.

Sea $E^* = \{e(u,v) | u, v \in P, u \neq v\}$.

$$\text{maximizar} \quad \sum_{e \in E^*} x_e w_e \tag{3.10}$$

$$\text{sujeto a} \quad \sum_{e \in \delta(v)} x_e \leq 2 \qquad \forall v \in P, \tag{3.11}$$

$$x_{e(u,v)} - x_{e(v,u)} = 0 \qquad \forall u, v \in P, \tag{3.12}$$

$$x_e \in \{0,1\} \qquad \forall e \in E^*, \tag{3.13}$$

$$w_e \in \mathbb{R}^+ \qquad \forall e \in E^*. \tag{3.14}$$

Explicación:

(3.10) La función objetivo indica que se desea maximizar el peso de las aristas seleccionadas en los ciclos solución.

(3.11) Para cada vértice en P, el número de aristas que salen o entran es a lo más dos.

⇒ Para cada pareja indirecta, el número de veces a donar o recibir órgano es a lo más dos. Con las Restricción siguiente (3.12) aseguramos que sólo se recibe un órgano y sólo se dona un órgano por pareja.

(3.12) Para dos vértices cualesquiera u, v en P, las banderas x para las aristas de u a v y de v a u deben tener el mismo valor. Los únicos dos valores disponibles son 1 ó 0 (por la Restricción 3.13). Por lo que: o se añaden ambas aristas a la solución (con 1) o se descartan de ella (con 0).

⇒ Para cualesquiera dos parejas incompatibles u, v: Sólo si el donante de la pareja u le dona su órgano al receptor de la pareja v, entonces el donante de v le dona al receptor de u.

(3.13) La bandera x_e de la arista e, indica con un 1 si esa arista se toma en cuenta en la solución final, 0 en caso contrario.

⇒ La bandera x_e de la arista $e(i, j)$ indica con un 1 si en la asignación renal se lleva a cabo el trasplante de órgano del donante i al paciente j, 0 en caso contrario

(3.14) Los pesos de las aristas pueden ser cualquier número real positivo.

⇒ Las compatibilidades entre los pacientes sólo pueden ser representadas por números reales positivos.

3.2 Complejidad del proceso

Proceso Completo

El cálculo de la complejidad del proceso considera dos cosas: uno, qué tan complejo es el problema a resolver por medio del programa Simplex en una iteración, y dos, cuál es el número de iteraciones que tomará el proceso hasta encontrar la solución deseada, en el peor caso. Nótese que esta iteración no se refiere a la iteración que realiza el método Simplex de manera interna, sino que, una vez que se obtiene un resultado por parte de Simplex, se hace una interpretación de los datos para saber si se cumple con las condiciones dadas por las variables k, $k2$ y q.

Tomando en cuenta el modelo del Proceso Completo, sea:

- $cols = 2(nz\_wsp + nz\_wst + nz\_wpp + nz\_wpt)$, el número de columnas a utilizar en Simplex, donde:

  - nz_wsp es el número de entradas mayores a cero, en la matriz W_{SP},
  - nz_wst es el número de entradas mayores a cero, en la matriz W_{ST},
  - nz_wpp es el número de entradas mayores a cero, en la matriz W_{PP},
  - nz_wpt es el número de entradas mayores a cero, en la matriz W_{PT}.

- $rows = m + n + n + r + cols/2 + 1 + \alpha$, el número de restricciones a utilizar en Simplex para la $(\alpha + 1)$-ésima iteración, donde:
  - m, la cardinalidad de S, indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.2,
  - n, la cardinalidad de P, indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.3,
  - n, la cardinalidad de P, (de nuevo, pero ahora) indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.4,
  - r, la cardinalidad de T, indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.5,
  - $cols/2$, indican el número de restricciones a agregar para poder cumplir con la Restricción 3.7; estas restricciones permitirán obtener el valor de una variable negada dentro de Simplex,
  - 1, indica que se utiliza una sola restricción para cumplir con el número de sub-restricciones a utilizar para cumplir la Restricción 3.6,
  - α, indica el número de restricciones a añadir para indicar que la próxima solución debe ser diferente a las α anteriores.

La constante $cols$ es el número de aristas que se tomarán en cuenta multiplicado por dos. Es decir, se requieren dos variables por cada arista e; una, para la bandera x_e, y la segunda para x_e negada. Esto último es con el fin de poder utilizar la operación XOR dentro de Simplex (Restricción 3.9).

La variable $rows$, a diferencia de $cols$, es de valor dinámico en cada iteración. Esto es debido a que, en cada iteración, se agrega una nueva restricción para indicar que se desea una solución diferente a las α anteriores.

Vale la pena volver a recalcar que, en caso de tener vértices *forzados* a aparecer en la solución, no hace falta añadir más restricciones, sino modificarlas.

Simplex, en el peor de los casos, tiene complejidad $O(2^{rows})$. En la práctica, el número de variables (o columnas) de *cols* está entre *rows* (el número de restricciones) y $10rows$; y el número de pasos para encontrar la solución está entre *rows* y $4rows$. Rara vez, se tiene un número de pasos mayor a $10rows$. Es decir, tenemos un **tiempo polinomial** [11][10][25].

El número de iteraciones que tomará el proceso hasta encontrar la solución deseada en el peor caso, será igual al número de todas las combinaciones posibles en las que se pueden formar un conjunto de ciclos y cadenas de trasplantes. Este peor caso va de la mano con la densidad de las matrices de adyacencia y se puede establecer por medio de los números de Bell[3].

Dado un conjunto de n elementos, se puede obtener el número total de posibles soluciones, por medio de los números de Bell [31]:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

Considérese la Figura 3.4. En ella se tiene un conjunto de cinco elementos, y a partir de ellos, se contemplan todas las posibilidades que se pueden obtener para formar conjuntos de ciclos o cadenas, de tal manera que no se repita un elemento. De acuerdo a los números de Bell, el número total de dichas posibilidades es cincuenta y dos.

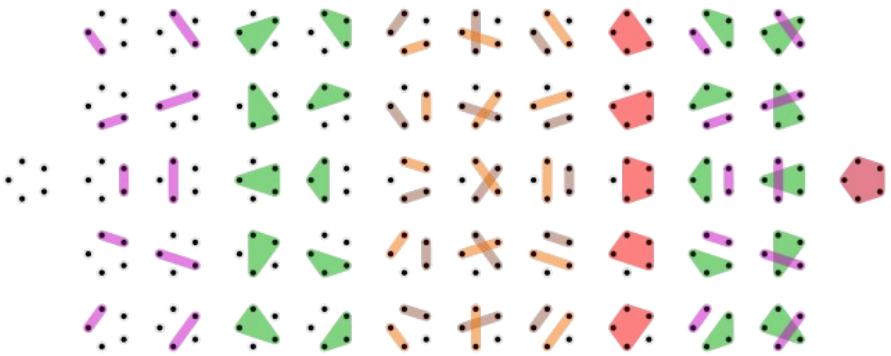


Figure 3.4: Cincuenta y dos particiones de un conjunto de tamaño cinco.

[3]Dichos números también relacionados con el Problema de Partición de un conjunto [4]

Proceso Cruzado

La complejidad del Proceso Cruzado es análoga a la anterior. Las principales diferencias son que el número de filas ($rows$) y columnas ($cols$) son de menor tamaño. Lo que implica un menor costo computacional, de entrada. Tomando en cuenta el modelo del Proceso Cruzado, sea:

- $cols = 2nzp\_wpp$, el número de columnas a utilizar en Simplex, donde:
  - nzp_wpp es el número de entradas (i, j), en la matriz W_{PP} tal que si la entrada (i, j) es mayor a cero, entonces (j, i) también. Esto último se hace para descartar todas aquellas parejas de parejas incompatibles que no puedan establecer un ciclo de tamaño dos. En el peor caso se tiene que: $nzp\_wpp = (n)(n-1)$, donde n es el número de parejas incompatibles.
- $rows = nzp\_wpp/2 + n + cols/2 + \alpha$, el número de restricciones a utilizar en Simplex para la $(\alpha + 1)$-ésima iteración, donde:
  - $nzp\_wpp/2$, (nzp_wpp definido hace un momento) indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.11,
  - n, la cardinalidad de P, indica el número de sub-restricciones a utilizar para cumplir con la Restricción 3.12,
  - $cols/2$, indican el número de restricciones a agregar para poder cumplir con la Restricción 3.13; estas restricciones permitirán obtener el valor de una variable negada dentro de Simplex,
  - α, indica el número de restricciones a añadir, para restringir que la próxima solución deba ser diferente a las α anteriores.

3.3 Implementación de Chainscycles

Chainscycles[4] es el nombre del programa desarrollado en este trabajo. El desarrollo de *Chainscycles* se llevó a cabo en el lenguaje de programación C con ayuda de la biblioteca SYMPHONY[5] (escrita en C++). Posteriormente, se implementó una función externa (o Función Definida por el Usuario, UDF) en el Sistema Manejador de Bases de Datos Oracle 12c para poder llamar al programa con los datos reales. Por último, se elaboró la interfaz gráfica correspondiente, para que el usuario pudiera acceder a la aplicación.

[4]Registro ante INDAUTOR 03-2015-120312293400-01 (ver Figura A.5 del Anexo A).
[5]https://projects.coin-or.org/SYMPHONY/

3.3.1 Programación en lenguaje C

A continuación se presenta la firma del método *Chainscycles*. Dicho método se utilizó para obtener los ciclos y cadenas solución, ya sea por medio del Proceso Completo o el Proceso Cruzado[6].

```
void chainscycles
(
        int s_cardinality,      //m
        int p_cardinality,      //n
        int t_cardinality,      //r
        unsigned long int * s_indices,
        unsigned long int * p_indices,
        unsigned long int * t_indices,
        int ** w_sp,
        int ** w_st,
        int ** w_pp,
        int ** w_pt,
        char * s_forced_indices,
        char * p_forced_indices,
        char * t_forced_indices,
        int max_size_of_cycle,     //k
        int max_num_of_cycles,  //k2
        int max_size_of_chain,     //q
        int max_num_of_chains,  //q2
        int n_requested_solutions,
        int max_n_iterations,
        int id_process,
        char * previous_process_data,
        char ** solution_string
);
```

Con respecto al diseño de los procesos Completo y Cruzado, presentados en el inicio de esta Sección 3.1, este método tiene como parámetros de entrada distintos elementos, tales como:

- La cardinalidad de los conjuntos S, P y T.
- Tres arreglos de tipo entero -grande- que guarden las etiquetas de identificación (tales como llaves primarias de una base de datos) para cada elemento de los conjuntos S, P y T. Internamente, para cada conjunto, se consideran los índices

[6]Indicando por medio de la combinación: $q2 = 0$ y $k = 2$ en los parámetros del método, se solicita la funcionalidad del Proceso Cruzado. Se utiliza el Proceso Completo en caso contrario.

de sus elementos desde el cero hasta el número de su respectiva cardinalidad menos uno.

- Cuatro arreglos bidimensionales de tipo entero para representar las matrices W_{SP}, W_{ST}, W_{PP} y W_{PT}. La dimensión de estas matrices está determinado por la cardinalidad de los conjuntos S, P y T.

- Tres arreglos de tipo caracter[7] para marcar qué elementos deben estar forzosamante en la solución. Se utilizan símbolos de verdadero o falso únicamente. Hay un arreglo por cada conjunto y la cardinalidad es la misma correspondiente.

- La cota máxima para el tamaño de los ciclos.

- La cota máxima para el número de ciclos permitidos.

- La cota máxima para el tamaño de las cadenas.

- La cota máxima para el número de cadenas permitidas.

- Número de soluciones requeridas. De existir, es posible que el programa entregue más de una solución.

- Máximo número de iteraciones. Debido a que el programa itera de manera necesaria hasta que se cumplan ciertas condiciones, es posible restringir dicho número con esta variable.

- Un número entero positivo para identificación del proceso. Se tiene programada la funcionalidad de ejecutar el programa una primera vez y obtener cierta cantidad de soluciones, y posteriormente ejecutar otra vez para obtener los siguientes resultados (diferentes a los anteriores). Para esto, fue necesario el apoyo de la memoria secundaria. Con el valor -1 se indica que no se desea utilizar dicha funcionalidad.

- Una cadena de caracteres que indica la ruta a un archivo donde se puedan guardar las soluciones obtenidas del proceso indicado.

- Un apuntador a memoria para guardar la solución obtenida en una cadena de caracteres.

[7]El lenguaje de programación C no tiene tipo de dato booleano. La unidad más pequeña de memoria es el tipo caracter (char).

El resultado del programa se presenta de acuerdo a una gramática libre de contexto, la cual se presenta en el Código A.1 del Apéndice A.

De manera general, el método *Chainscycles* hace lo siguiente:

1. Validación de los parámetros de entrada. Se hace validación de números enteros positivos, acceso de escritura a disco o factibilidad del problema.
2. Preanálisis y reducción de matrices de compatibilidad. Hay algunos nodos que tal vez nunca se relacionen con algún otro. Estos se pueden descartar.
3. Lectura de datos previos desde memoria secundaria, en caso de haber.
4. Asignación de memoria para las estructuras de datos que se utilizarán.
5. Configuración de los datos con respecto al modelo de Programación Lineal y, de acuerdo a los lineamientos de la biblioteca de SYMPHONY.
6. Ejecución del programa SYMPHONY y obtención de resultados.
7. Interpretación de la solución obtenida y validación con respecto a las restricciones con respecto a k, $k2$ y q.
8. Se repite desde el paso 5, aclarando que ahora al modelo se le han integrado las restricciones correspodientes para que la próxima solución sea diferente a las ya obtenidas. Se termina el proceso hasta que se haya encontrado el número de soluciones pedidas, se haya agotado el número de iteraciones permitidas o el software de SYMPHONY indique que ya no hay soluciones posibles.
9. Entrega de la cadena resultado en la dirección del apuntador *solution_string*.
10. En caso de requerirse, se guardan las soluciones obtenidas en memoria secundaria.
11. Liberación de memoria solicitada.

En el Apéndice A se puede ver un ejemplo del uso de *Chainscycles* (figuras A.1, A.2, A.3 y A.4).

3.3.2 Funciones Definidas por el Usuario (UDF)

Se llevó a cabo la implementación de una UDF dentro del Sistema Manejador de Bases de Datos para que sirviera como interfaz intermedia entre la aplicación final y el programa *Chainscycles*. A continuación se describe lo que se llevó a cabo:

- Elaboración del modelo de la base de datos, incluyendo las tablas respectivas para donantes indirectos, parejas incompatibles y pacientes en lista de espera.
- Implementación de función de compatibilidad de trasplante de órgano (una por cada órgano) entre un donante y un receptor.
- Implementación en lenguaje C, de la función externa *conv* para el acceso a las funciones de la biblioteca de *Chainscycles*.
- Desarrollo de función que llame a *conv* para cada tipo de trasplante de órgano. Es decir, una función *conv_riñón* para trasplantes de riñón, *conv_corazón* para trasplantes de corazón, *conv_córnea* para córneas.

En la Figura 3.5 se puede ver la arquitectura empleada.

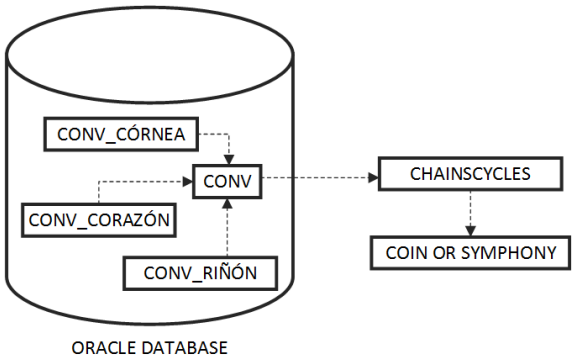


Figure 3.5: Las funciones CONV_RIÑÓN, CONV_CORAZÓN y CONV_CÓRNEA son dependientes de la función externa CONV, que a su vez, depende de *Chainscycles* y ésta última de la biblioteca COIN OR SYMPHONY.

Chapter 4

Demostración de la igualdad de soluciones entre los programas SIATRE y Chainscycles

En este capítulo se presenta una demostración para mostrar la igualdad de soluciones que los programas SIATRE y *Chainscycles* obtienen con parámetros equivalentes. Recordando que *Chainscycles* es el nombre del programa desarrollado, y SIATRE [3] un programa precursor del presente trabajo. Ambos buscan resolver el Problema de Asignación Renal, el primero utilizando Programación Entera, y el segundo obteniendo resultados de forma constructiva.

Se pretende a continuación, demostrar la igualdad entre las soluciones que ambos programas obtienen con los siguientes parámetros:

- Un conjunto de n parejas incompatibles P.
- Un donante indirecto denotado por λ.

Y las siguientes condiciones:

- Que no admita ciclos.
- Los arcos no tendrán peso asignado.

Vimos conveniente comparar las soluciones de nuestro programa con las de el sistema SIATRE dado que el objetivo que buscan ambos programas es el mismo para los parametros señalados.

Para poder realizar la demostración es necesario hacer un pequeño análisis de los dos métodos a los que se recurre por cada uno de los programas, para que de este modo, podamos establecer cómo se realiza la construcción de las soluciones por programa.

SIATRE

El programa SIATRE recurre al algoritmo *Búsqueda en Anchura* (BFS por sus siglas en inglés) como principal método para obtener su solución.

BFS es un algoritmo de búsqueda sin información que expande y examina sistemáticamente todos los nodos de una gráfica para "descubrir" cada vértice alcanzable desde un nodo raíz [7]. Por ello sabemos que recorrerá todos los nodos, explorando las posibles adyacencias entre nodos para poder establecer las cadenas de compatibilidad que den solución al problema.

Proposición 4.0.1 *El programa SIATRE construye todas las cadenas posibles dado un conjunto de nodos, un listado de compatibilidad y un único donante indirecto* λ.

Demostración Sea $P = \{p_1, p_2, \ldots, p_n\}$ el conjunto de nodos de parejas incompatibles y un vértice λ que denota al donante indirecto.

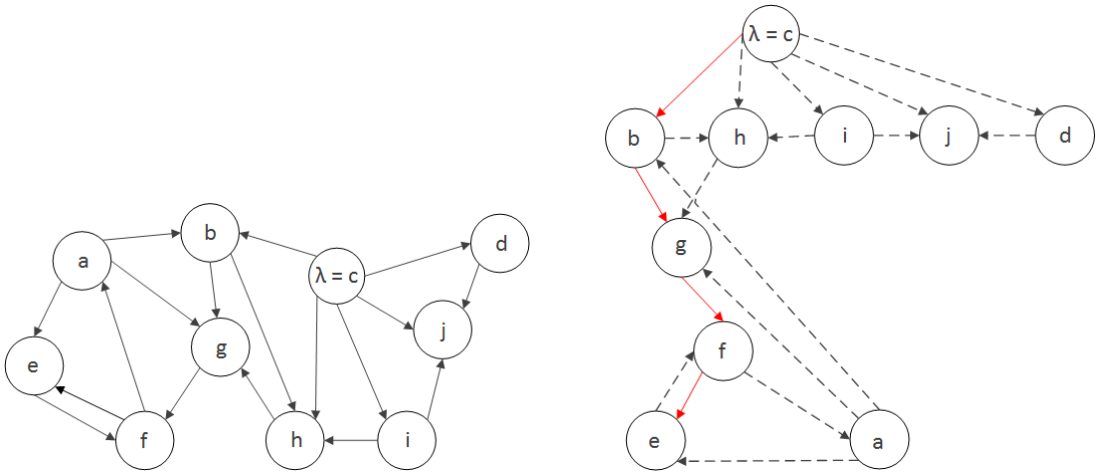
Dado que queremos que λ esté presente e inicie todas las cadenas que surjan, se introduce λ como nodo raíz del árbol que construye el algoritmo BFS. Éste último explora a partir del listado de compatibilidad W_{PP} entre nodos del conjunto P para encontrar los $p_i \in P$ que pueden ser alcanzados desde λ por medio de cadenas (trayectorias), formando de esta manera, las cadenas solución. BFS asegura que se recorran todas las posibilidades, agotando los nodos de P sin repetición por cadenas-trayectorias. ∎

Sea $Y = \{Y_1, Y_2, \ldots, Y_\beta\}$ el conjunto que contiene sin repetición todas las posibles cadenas solución, donde $Y_i = [\lambda, p_1^*, p_2^*, \ldots, p_t^*]$, con $i \in \mathbb{N}$, $p_h^* \in P$, $\{h, t\} \subset \{1, 2, \ldots, n\}$. En la Figura 4.1 se presenta un ejemplo de solución obtenida mediante el programa SIATRE.

Chainscycles

El programa *Chainscycles* es descrito a lo largo del Capítulo 3.

Al finalizar la ejecución del método Simplex, que pretende resolver un problema en Programación Lineal, se obtienen los valores solución de las distintas variables



(a) Gráfica original. (b) Ejemplo cadena $Y_i = [c, b, g, f, e]$

Figure 4.1: Ejemplo de cadena obtenida, por medio de BFS.

utilizadas, en caso que el problema sea factible de resolver. En el caso del programa *Chainscycles*, se obtiene una matriz binaria que indica las adyacencias utilizadas en la cadena de nodos.

Proposición 4.0.2 *El método Simplex, dentro del programa Chainscycles, revisa todas las combinaciones para formar cadenas, descartando repeticiones.*

Demostración El método Simplex siempre devuelve la solución óptima para problemas de Programación Lineal factibles, pues cumple con el Teorema Fundamental de la Programación Lineal (Teorema 4.0.3) [7].

Dado lo anterior, suponiendo el caso de tener un problema factible, entonces Simplex obtiene la óptima solución. Para la segunda mejor solución, *Chainscycles* introduce la solución anterior como restricción y Simplex obtiene dicha segunda mejor solución. Para la i-ésima mejor solución, de igual manera, *Chainscycles* introduce las $(i-1)$ soluciones anteriores como restricción para que Simplex obtenga dicha i-ésima mejor solución.

Sea $Z_i = [\lambda, p_1^*, p_2^*, \ldots, p_d^*]$, con $p_h^* \in P$, $\{h, d\} \subset \{1, 2, \ldots, n\}$, la i-ésima mejor cadena-solución obtenida en la i-ésima iteración.

Simplex devolverá una solución siempre y cuando el problema sea factible. *Chainscycles* termina de recorrer todas las posibles cadenas solución cuando el método Simplex determine que ya no hay más, pues es infactible. ∎

Sea $W = \{W_1, W_2, \ldots, W_\gamma\}$ el conjunto que contiene todos los posibles conjuntos de cadenas solución , donde $W_j = \{Z_1, Z_2, \ldots, Z_\omega\}$, con $Z_i = [\lambda, p_1^*, p_2^*, \ldots, p_d^*]$, con $p_h^* \in$, $\{i, j\} \subset \mathbb{N}$, $\{h, d\} \subset \{1, 2, \ldots, n\}$. Nótese que $|W_j| = 1\ \forall j \in \{1, \ldots, \omega\}$, ya que una solución está compuesta únicamente de una cadena solución, al haber un único donante indirecto (por las condiciones descritas al inicio de este Capítulo).

Nota: El método Simplex, dentro del programa *Chainscycles*, no sólo revisa todas las combinaciones para formar cadenas, sino que revisa todas las combinaciones de ciclos y cadenas, descartando repeticiones. Tal como se vió en la Sección 3.2 del Capítulo 3, el número total de posibles combinaciones de ciclos y cadenas, en el peor caso se puede expresar por medio de los números de Bell.

Teorema 4.0.3 (Teorema Fundamental de la Programación Lineal [7]) *Cada problema de Programación Lineal tiene las siguientes tres propiedades:*

1. *Si el problema no tiene solución óptima entonces es no acotado o infactible.*
2. *Si tiene una solución factible, tiene una solución básica factible.*
3. *Si el problema tiene solución óptima, tiene una solución básica factible óptima.*

Igualdad de soluciones para SIATRE y Chainscycles

Proposición 4.0.4 *Las soluciones que los programas SIATRE y Chainscycles devuelven son iguales.*

Demostración Sean Y y W los conjuntos de soluciones que arrojan los programas SIATRE y *Chainscycles*, respectivamente.

- $Y = \{Y_1, Y_2, \ldots, Y_\beta\}$, donde $Y_i = [\lambda, p_1^*, p_2^*, \ldots, p_t^*]$, con $i \in \mathbb{N}$, $t \in \{1, 2, \ldots, n\}$.
- $W = \{W_1, W_2, \ldots, W_\gamma\}$, donde $W_j = \{Z_1, Z_2, \ldots, Z_\omega\}$, con $|W_j| = 1\ \forall j \in \{1, \ldots, \omega\}$, y $Z_i = [\lambda, p_1^*, p_2^*, \ldots, p_d^*]$, con $\{i, j\} \in \mathbb{N}$, p_h^*, $\{h, d\} \subset \{1, 2, \ldots, n\}$.

Nótese que $|Y| = \beta$ y $|W| = \gamma$. Sea Z el conjunto donde se encuentran todas las Z_i. Es decir:

- $Z = \{Z_i |\ Z_i \in W_j\ \forall W_j \in W\}$.

Tenemos que $|Z| = \gamma$; pues por cada $W_j \in W$, tenemos que $|W_j| = 1$.

Para esta demostración basta asegurar que $Y = Z$. Esta última demostración es por contradicción. Sea $Y \neq Z$. Esto implica dos opciones: existe al menos un elemento en Z que no está en Y ó existe al menos un elemento en Y que no está en Z.

Para la primera opción tenemos que Z es un conjunto que contiene al menos un elemento que no está en Y. Es decir, existe al menos una cadena solución que sí contempla W pero no Y. Contradicción. Tanto Y como W contienen todas las posibles cadenas para un conjunto de parejas incompatibles y un único donante indirecto. Por lo que, no puede ser que a Y le falte un elemento que sí esté contemplando un conjunto en W. El caso en que Y es un conjunto que contiene un elemento que no está en Z, es análogo.

Por lo tanto Y y Z son iguales y tenemos que los conjuntos de soluciones que arrojan los programas SIATRE y *Chainscycles* son idénticos. ∎

Para terminar este capítulo, cabe aclarar que, la comparación tanto teórica como práctica de ambos programas, SIATRE y *Chainscycles*, fue productiva y exitosa por dos razones: uno, nos permitió contar con la tranquilidad de saber que por dos vías distintas se encontraba la misma solución; y dos, permitió la comparación total de los resultados, es decir, se pudo verificar que a ninguno de los dos le faltase contemplar alguna solución para las pruebas realizadas.

Para la elaboración de las demostraciones, se agradece el apoyo del estudiante de la carrera de Matemáticas Daniel Scognamiglio Moreno.

Chapter 5

Resultados

Este capítulo presenta las pruebas realizadas en el sistema, así como, sus respectivos análisis y herramientas utilizadas.

Las pruebas realizadas se presentan en cuatro subsecciones. En la primera subsección "Pruebas de desempeño en SYMPHONY" se compara el desempeño de la herramienta de Programación Entera SYMPHONY al momento de resolver un problema NP-difícil. La prueba se enfocó en resolver un problema similiar al de Asignación de Órganos. Se escogió el Problema del Agente Viajero como objetivo a resolver y como modelo computacional para comparación de desempeño al Algoritmo de Ramificación y Poda (*Branch and Bound*) [12].

En la segunda subsección "Pruebas en base de datos real" se muestran los tiempos obtenidos al ejecutar el programa *Chainscycles* sobre una base de datos real, aunque pequeña de tamaño; la tercera "Pruebas en base de datos sintética", de manera similar, indica los tiempos que se tomaron al ejecutar *Chainscycles* sobre una base de datos sintética aunque a mayor escala que la base de datos real; y finalmente, la cuarta subsección "Proceso Completo vs Proceso Cruzado" muestra las diferencias en cuanto a tiempos entre ejecutar el Proceso Completo y el Proceso Cruzado sobre el mismo conjunto de datos.

5.1 Herramientas utilizadas y descripción del conjunto de datos

Herramientas utilizadas

- Lenguaje de programación C, C++. Compilador GCC 4.2
- Sistema Manejador de Bases de Datos Oracle 12c
- Biblioteca para Programación Lineal COIN OR SYMPHONY 5.6.11

Dentro de los requerimientos no funcionales del sistema se solicitó el uso del SMBD de Oracle.

Por otra parte, la biblioteca SYMPHONY se eligió como herramienta para la Programación Lineal por tres motivos: uno, por ser *open source*; dos, por tener una fuerte comunidad activa la cual permite su continuo mejoramiento y soporte; y tres, por tener un buen rendimiento en sus resultados [13].

El uso del lenguaje C se debió a dos razones principalmente: una, por compatibilidad y manejo de la biblioteca SYMPHONY; y dos, por la necesidad de un programa intermediario (UDF) entre el SMBD y el programa *Chainscycles*.

En el Apéndice A se pueden ver las capturas de pantalla del sistema desarrollado (figuras A.6, A.7 y A.8).

Descripción del conjunto de datos

El conjunto de datos fue tomado de una base de datos real, para el caso particular de trasplante renal. Dentro de ella, se definió una función para determinar la compatibilidad donante-receptor (tal como en [3]), por medio de propiedades médicas tales como número y relación de antígenos, anticuerpos, edad, sexo y grupo sanguíneo[1].

[1]En el presente trabajo no se dá más información por motivos de confidencialidad, únicamente basta saber que la función de compatibilidad entrega un número entero positivo, donde a mayor valor, mayor compatibilidad.

5.2 Resultados y análisis

Las pruebas se llevaron a cabo en una PC con Intel(R) Core(TM) CPU @ 2.20GHz, 8 Gb de memoria RAM y 256 Gb de HDD.

5.2.1 Pruebas de desempeño en SYMPHONY

En la Tabla 5.1 se presentan los tiempos obtenidos para TSP con respecto a dos programas solución. El primero, usando Programación Lineal Entera, y el segundo la técnica de Ramificación y Poda [12]. En dichas pruebas se utilizaron los problemas TSP para 17 ciudades de Gröetschel, 26 ciudades de Fricker y 42 ciudades de Dantzig, entre otras [30].

Table 5.1: Tiempos resultado para el problema TSP con Ramificación y Poda, y con Programación Entera

n	$n!$	**Ramificación y Poda**	**Programación Entera**
5	120	0.005s	0.025s
17	3.55e+14	3.559s	2.817s
26	4.03e+26	10min 15s	54.431s
37	1.37e+43	3min 38s	14.867s
42	1.40e+51	-	-

En esta prueba, como en las siguientes, los casos donde se marca un guión (-) indica que el proceso fue **detenido** después de dos días de ejecución.

Los resultados indican que para 5 y 17 ciudades, el algoritmo de Ramificación y Poda es más rápido. La razón del porqué el procedimiento de Programación Entera es más lento al principio es porque, éste último necesita construir estructuras de datos que le permitan interpretar el modelo del problema. Sin embargo, vemos que conforme el problema se agranda, el procedimiento de Programación Entera supera en tiempo al algoritmo de Ramificación y Poda.

Los resultados en el caso de 37 ciudades, el tiempo es menor que el caso de 26 ciudades. La complejidad del problema determina el tiempo de ejecución. Es decir, el caso de 26 ciudades tenía un espacio de búsqueda menor, pero había un número mayor de soluciones más cercanas al óptimo que con respecto al número de soluciones cercanas al óptimo del problema de 37 ciudades. En el caso de 42 ciudades, ambos procedimientos fueron detenidos después de dos días de ejecución, el problema era

muy grande.

5.2.2 Pruebas en base de datos real

Se hicieron dos pruebas sobre una base de datos real, la cual contó con 20 parejas incompatibles, 1 donante indirecto y una matriz de compatibilidad dispersa.

La primera prueba fue obtener la solución óptima para el problema de asignación de riñón. El resultado fue bueno, pues nuestro programa *Chainscycles* no tardaba más de 1 segundo en resolver.

En la segunda prueba se hizo una comparación de nuestro programa *Chainscycles* con el programa SIATRE [3].

El programa *Chainscycles* puede producir soluciones con más de una cadena y más de un ciclo, mientras que, SIATRE sólo permite soluciones con una cadena y cero ciclos. Dado lo anterior, se optó por que el objetivo fuese obtener todas las cadenas solución para el conjunto de datos de 20 parejas incompatibles y 1 donante indirecto. Para asegurar que las condiciones de ambos programas fuesen las mismas, hubo que configurar los parámetros de entrada del programa *Chainscycles* para admitir solo resultados con una sola cadena y un solo donante indirecto. Los resultados de la prueba fueron satisfactorios, ya que, ambos programas presentaron las mismas soluciones.

Se obtuvieron distintos tiempos de ejecución. En SIATRE la prueba se realizó en 1min 15s. En el caso de *Chainscycles*, la prueba duró 4min 35s. Esto último, como ya se ha mencionado, se debió a que para cada siguiente solución, el programa introduce las soluciones anteriores como restricción al modelo del problema en Programación Entera. Lo cual hacía cada vez más dificil el problema y de ahí, el tiempo requerido para su solución.

En resumen, *Chainscycles* funciona mejor cuando no se busca obtener todas las soluciones posibles, sino las mejores disponibles. SIATRE es una buena herramienta cuando se trata de obtener todas las opciones posibles, cuando el número de elementos (donantes y receptores) es pequeño.

5.2.3 Pruebas en base de datos sintética

Dentro de esta sección se generaron datos de manera masiva para poder indentificar el desempeño del programa *Chainscycles*.

Como se vió en la Sección 3.1, la posibilidad de trasplante entre un paciente y un receptor esta representada por matrices de adyacencias. La densidad de éstas matrices se generó con respecto a una probabilidad de compatibilidad:

- Muy Alta (la densidad es 100%, todas las entradas de las matrices estan definidas),
- Alta (probabilidad de compatibildad de 50%),
- Media (probabilidad de 25%),
- Baja (12.5%) y
- Muy Baja (6.25%).

Se espera que en problemas reales, la densidad de las matrices sea muy baja (aproximadamente 6.25% del total disponible). Esto, debido al escaso número de compatibilidades entre donantes y receptores [2].

En la Tabla 5.2 se presentan las pruebas realizadas para diferentes cardinalidades de los conjuntos de Donantes Indirectos (S), Parejas Incompatibles (P) y Pacientes en Lista de Espera (T).

Table 5.2: Resultados en la Base de Datos sintética

			Densidad de las matrices				
$\|S\|$	$\|P\|$	$\|T\|$	**Muy alta**	**Alta**	**Media**	**Baja**	**Muy baja**
1	10	100	0.130s	0.042s	0.022s	0.016s	0.013s
2	20	200	0.294s	0.163s	0.074s	0.041s	0.020s
5	50	500	7.873s	2.663s	0.778s	0.305s	0.121s
10	100	1000	1min 52s	27.945s	7.993s	2.750s	0.812s
20	200	2000	-	17min 05s	1min 44s	27.393s	8.210s

Para cada densidad, los tiempos obtenidos van incrementando conforme se aumenta el número de elementos en S, P y T. Esto era de esperarse. Sin embargo,

[2]El número de compatibilidades es escaso de acuerdo a nuestra base de datos real, sin embargo, no se descarta que a futuro incremente.

nótese que cuando la densidad de las matrices es alta, entonces los tiempos son demasiado grandes. Esto es porque, dentro del programa *Chainscycles*, el modelo en Programación Entera está tomando en cuenta cada una de las posibles adyacencias. Por lo que, para matrices dispersas se tiene un buen desempeño, aún cuando el tamaño de estas matrices crece.

5.2.4 Proceso Completo vs Proceso Cruzado

En la Tabla 5.3 se presentan los resultados obtenidos al ejecutar los procesos Completo y Cruzado (Sección 3.1 del Capítulo 3), considerando sólo al grupo de las parejas incompatibles, compatibilidad 100% para todos los participantes, y una configuración para obtener sólo soluciones que contemplen ciclos de tamaño dos.

Sea $n = |P|$, el número de parejas incompatibles.

Table 5.3: Comparación de resultados entre Proceso Completo y Cruzado

n	**Proceso Completo**	**Proceso Cruzado**
4	0.034s	0.014s
6	13.103s	0.015s
8	1min 48s	0.015s
10	-	0.017s
32	-	0.096s
64	-	0.359s
128	-	7.743s
256	-	52.089s
512	-	10min 4.773s
1024	-	-

Para esta prueba, el Proceso Cruzado es más eficiente. La diferencia comienza desde la prueba para 8 elementos. Mientras al proceso Cruzado le toma menos de un segundo obtener la solución, el proceso Completo toma 12 segundos más. A continuación se indica porqué.

El proceso Completo es más lento en esta prueba porque siempre entrega la mejor solución. Esto quiere decir que primero obtiene, posiblemente, una solución que tenga ciclos o cadenas de tamaño casi igual al número de elementos que se tiene. Poco a poco, conforme a las iteraciones que tiene que hacer el programa para cumplir con las condiciones presentadas por los valores k, $k2$, q y $q2$, se van

obteniendo soluciones con ciclos o cadenas más pequeños que los anteriores. Es decir, si se tiene un conjunto con cinco elementos y lo que se pretende es obtener ciclos de tamaño dos, lo más probable es que primero se obtengan soluciones con cadenas o ciclos de tamaño cinco, luego las de tamaño cuatro, luego las de tamaño tres y por último las de tamaño dos. No hay un orden estricto en esto último, puesto que la compatibilidad entre donantes y receptores, es la que decide qué solución es mejor que otra.

En resumen, dado que el Proceso Completo tiene que visitar las mejores soluciones antes de obtener sólo las de ciclos de tamaño dos y a que, conforme el paso de las iteraciones el problema se vuelve más complejo (ya que se van añadiendo más restricciones al modelo), tenemos que el Proceso Completo es más lento que el Proceso Cruzado para esta prueba.

Por último y para concluir esta sección, se presenta la Figura 5.1. En ella se presentan los resultados obtenidos para matrices de compatibilidad desde 100 hasta 2000 elementos. La densidad de las matrices es muy baja, se tomó una probabilidad de compatibilidad de 6.25%, la cual se pretende que sea la más cercana a la de los problemas reales.

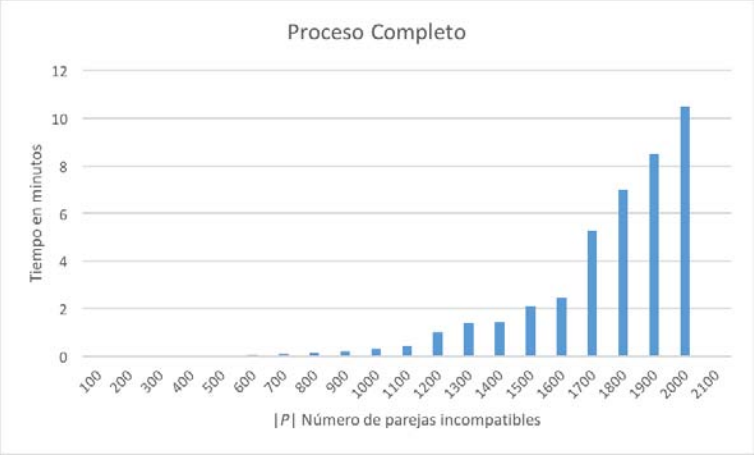


Figure 5.1: Resultados para matrices dispersas.

Se concluye que, el rendimiento del programa *Chainscycles* es mejor cuando se tienen matrices de compatibilidad dispersas para las compatibilidades de trasplante entre donantes y receptores.

Chapter 6

Conclusiones

Los objetivos de implementar satisfactoriamente los procesos Completo y Cruzado se cumplieron. Durante el trabajo realizado nos dimos cuenta que los problemas combinatorios pueden resultar un gran desafío, más que nada cuando el tamaño de los datos aumenta.

Los resultados indican un buen desempeño por parte de nuestro programa. Cabe señalar que nuestro conjunto de datos no es precisamente gigantesco, en cuanto a escala se refiere. Esto se debe a que, en la vida real, no se tiene tal histórico de datos en cuanto a donantes y receptores (no más de 13,000 registros [32]). Esto último es importante, pues permitirá al programa *Chainscycles* entregar resultados de manera casi instantánea y consecuentemente, en caso de órganos con período muy corto (tal vez horas) de vida, se puedan llevar a cabo las acciones correspondientes dentro del área médica.

En la práctica se espera que el tamaño de la base de datos de pacientes y receptores para trasplantes de órganos vaya creciendo. Sin embargo, cada determinado tiempo -cada semana, mes o meses, según el hospital- el número de pacientes y receptores irá decrementando, puesto que se habrán hecho las correspondientes asignaciones de órganos.

La aportación social del trabajo presentado fue, la posibilidad de tener en México un sistema que permita una correcta asignación de órganos con respecto al mayor grado de compatibilidad entre cada donante y receptor.

Durante el proceso de desarrollo, se tuvieron dos complicaciones a destacar.

Uno, llevar a cabo un modelo dinámico que se ajustase a los parámetros de entrada del programa SYMPHONY para poder resolver el problema. Y dos, la construcción de la Función Externa en el Sistema Manejador de Bases de Datos Oracle; la documentación de la herramienta implicó un gran desafío para mí, al no tener experiencia en esa tarea en particular.

Trabajo a futuro

El modelo aquí presentado (Sección 3.1) se pudiera mejorar para el caso de obtener las cadenas más largas, tal como se sugiere en [1] y [5]. En palabras: Por medio de Programación Lineal, para cada vértice de las parejas incompatibles y pacientes en lista de espera, se corre el programa para resolver el modelo, añadiendo la restricción de que ese vértice debe estar en la solución. Una vez obtenida la mejor solución para cada vértice, se hace una transformación de la solución obtenida en un nuevo problema. Dicha transformación basada en el Método de Christofides [15]. Posteriormente, se resuelve para cada solución y se obtiene la mejor de ellas; la solución óptima al problema. Implementando esta heurística, también conocida como heurística MLP (*Modified LP relaxation heuristic*), se obtiene como solución un conjunto optimizado de las cadenas más largas. Los vértices no utilizados se pudieran emplear para una nueva búsqueda sólo de ciclos.

Appendix A

Chainscycles

Formato de la solución

La solución se entrega con respecto a la siguiente Gramática Libre de Contexto:

Código A.1: Gramática Libre de Contexto para *Chainscycles*

S → 0 | 1ZMZT	//CODIGO DE SALIDA. 0:ERROR, 1:OK
M →N	//NUMERO DE SOLUCIONES ENCONTRADAS
T →IZWZAZBZX | TZT	//DESCRIPCION DE SOLUCION
I → N	//NUMERO DE SOLUCION
W →N	//PESO TOTAL DE LA SOLUCION
A →N	//NUMERO DE CADENAS EN LA SOLUCION
B →N	//NUMERO DE CICLOS EN LA SOLUCION
X →FZLZVZE | XZX	//DESCRIPCION DE CADENA O CICLO
F → 0|1	//BANDERA CADENA O CICLO. 1:SP, 2:SPT, 3:ST, 4:PP
L → N	//CARDINALIDAD DE LA CADENA O CICLO
V →N	//PESO DE LA CADENA O CICLO
E →D | EYE	//CADENA O CICLO
N →D | DN	//NUMERO NATURAL
Z → '|'	//SIMBOLO FINAL. SEPARADOR PRIMARIO
Y → ','	//SIMBOLO FINAL. SEPARADOR SECUNDARIO
D → 0|1|2|3|4|5|6|7|8|9	//SIMBOLOS FINALES. DIGITOS

Ejemplo de uso

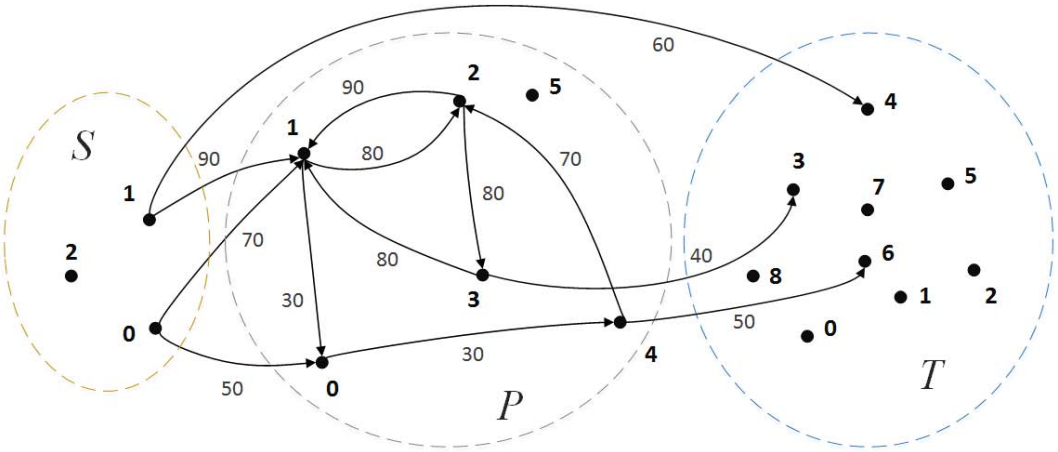


Figure A.1: Ejemplo del Problema de Asignación de Órganos. Los donantes no directos, las parejas incompatibles y los receptores en lista de espera quedan representados por índices distintos.

```
s_cardinality: 2                    w_sp:
p_cardinality: 5                    50 70 0 0 0
t_cardinality: 3                    0 90 0 0 0

p_indices: [ 0 1 2 3 4 ]            w_st:
s_indices: [ 0 1 ]                  0 0 0
t_indices: [ 3 4 6 ]                0 60 0

s_forced_indices: [ 1 0 ]           w_pp:
p_forced_indices: [ 0 0 0 0 0 ]     0 0 0 0 30
t_forced_indices: [ 0 0 0 ]         30 0 80 0 0

                                    0 90 0 80 0
max_size_of_cycle = 1000;           0 80 0 0 0
max_num_of_cycles = 1000;           0 0 70 0 0
max_size_of_chain = 1000;
max_num_of_chains = 1000;           w_pt:

                                    0 0 0
n_requested_solutions = 3           0 0 0
max_n_iterations = 10000            0 0 0
id_process = -1                     40 0 0
previous_process_data = "-1"        0 0 50
```

Figure A.2: Ejemplo del Problema de Asignación de Órganos. Parámetros de entrada.

```
1   OUTPUT:
2   1|3|1|430|2|1|2|4|130|0,0,4,6|3|2|60|1,4|4|3|240|2
    ,3,1|2|420|2|0|2|4|130|0,0,4,6|2|5|290|1,1,2,3,3|3
    |380|2|1|1|3|80|0,0,4|3|2|60|1,4|4|3|240|2,3,1
3
4   INTERPRETATION:
5
6   SOLUTION_1 (weight:430, chains:2, cycles:1)
7      CHAIN_SPT(size:4, weight:130) [ 0 0 4 6 ]
8      CHAIN_ST(size:2, weight:60) [ 1 4 ]
9      CYCLE_PP(size:3, weight:240) [ 2 3 1 ]
10
11  SOLUTION_2 (weight:420, chains:2, cycles:0)
12     CHAIN_SPT(size:4, weight:130) [ 0 0 4 6 ]
13     CHAIN_SPT(size:5, weight:290) [ 1 1 2 3 3 ]
14
15  SOLUTION_3 (weight:380, chains:2, cycles:1)
16     CHAIN_SP(size:3, weight:80) [ 0 0 4 ]
17     CHAIN_ST(size:2, weight:60) [ 1 4 ]
18     CYCLE_PP(size:3, weight:240) [ 2 3 1 ]
```

Figure A.3: Ejemplo del Problema de Asignación de Órganos. Salida del programa.

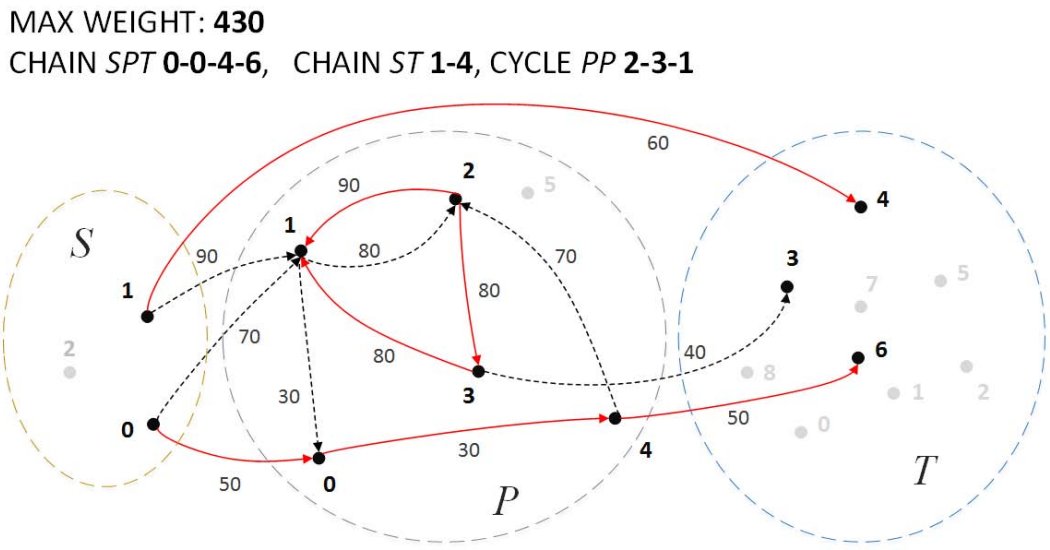
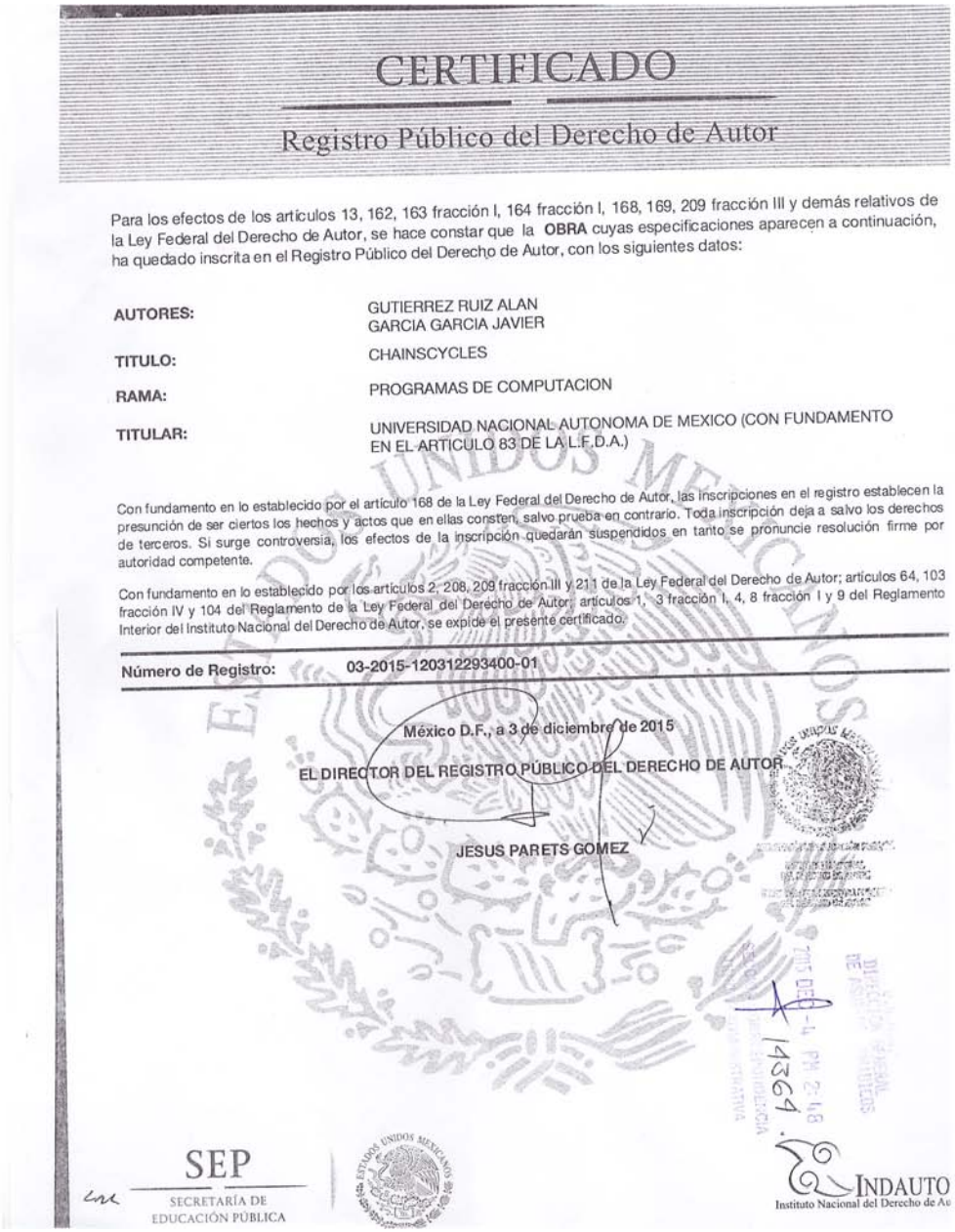


Figure A.4: Ejemplo del Problema de Asignación de Órganos. Solución óptima.

Registro ante INDAUTOR



CERTIFICADO

Registro Público del Derecho de Autor

Para los efectos de los artículos 13, 162, 163 fracción I, 164 fracción I, 168, 169, 209 fracción III y demás relativos de la Ley Federal del Derecho de Autor, se hace constar que la **OBRA** cuyas especificaciones aparecen a continuación, ha quedado inscrita en el Registro Público del Derecho de Autor, con los siguientes datos:

AUTORES:	GUTIERREZ RUIZ ALAN GARCIA GARCIA JAVIER
TITULO:	CHAINSCYCLES
RAMA:	PROGRAMAS DE COMPUTACION
TITULAR:	UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO (CON FUNDAMENTO EN EL ARTICULO 83 DE LA L.F.D.A.)

Con fundamento en lo establecido por el artículo 168 de la Ley Federal del Derecho de Autor, las inscripciones en el registro establecen la presunción de ser ciertos los hechos y actos que en ellas consten, salvo prueba en contrario. Toda inscripción deja a salvo los derechos de terceros. Si surge controversia, los efectos de la inscripción quedarán suspendidos en tanto se pronuncie resolución firme por autoridad competente.

Con fundamento en lo establecido por los artículos 2, 208, 209 fracción III y 211 de la Ley Federal del Derecho de Autor; artículos 64, 103 fracción IV y 104 del Reglamento de la Ley Federal del Derecho de Autor; artículos 1, 3 fracción I, 4, 8 fracción I y 9 del Reglamento Interior del Instituto Nacional del Derecho de Autor, se expide el presente certificado.

Número de Registro: 03-2015-120312293400-01

México D.F., a 3 de diciembre de 2015

EL DIRECTOR DEL REGISTRO PÚBLICO DEL DERECHO DE AUTOR

JESUS PARETS GOMEZ

14364

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

INDAUTO
Instituto Nacional del Derecho de Au

Figure A.5: Registro de *Chainscycles*. Autores: Javier García García y Alan Gutiérrez Ruiz

Sistema desarrollado

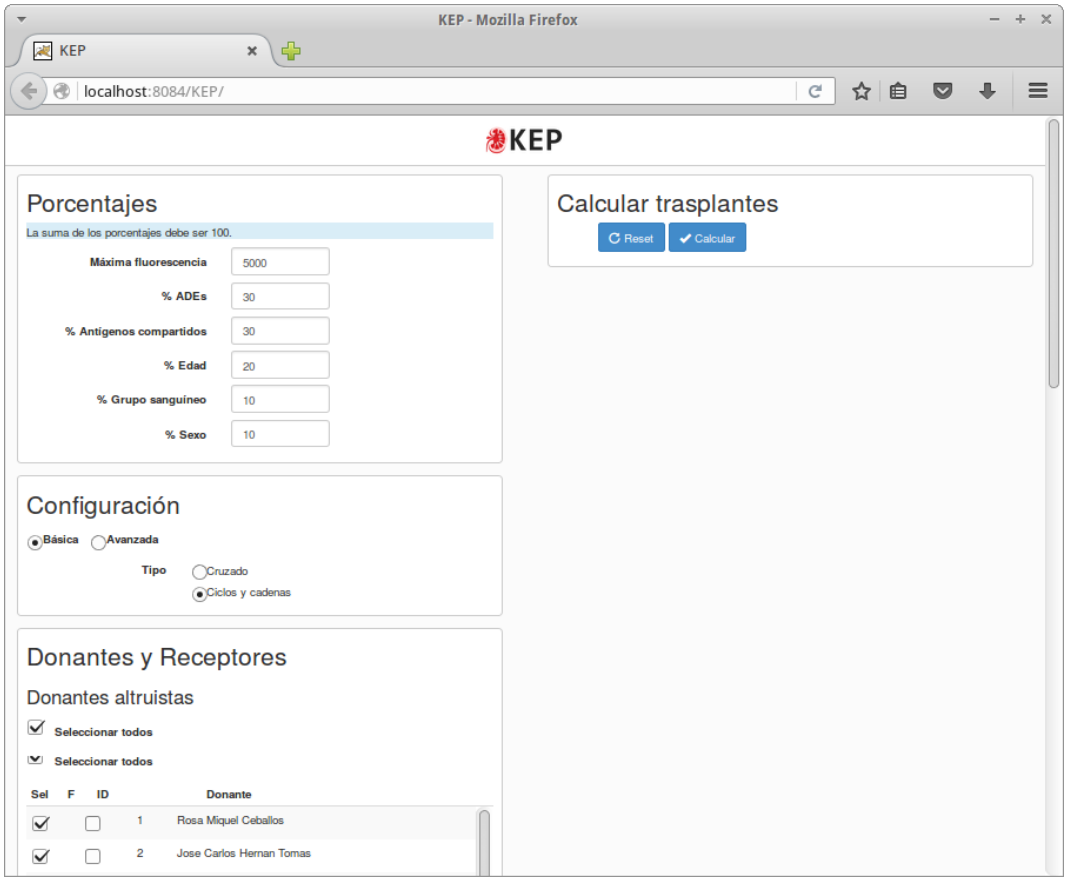


Figure A.6: Captura de pantalla del sistema desarrollado. Se tiene una tabla para configurar las preferencias en cuanto a las funciones de compatibilidad del órgano; una sección para seleccionar el Proceso Completo o el Cruzado; y tres tablas para la selección de los donantes indirectos, parejas incompatibles y receptores en lista de espera que serán considerados.

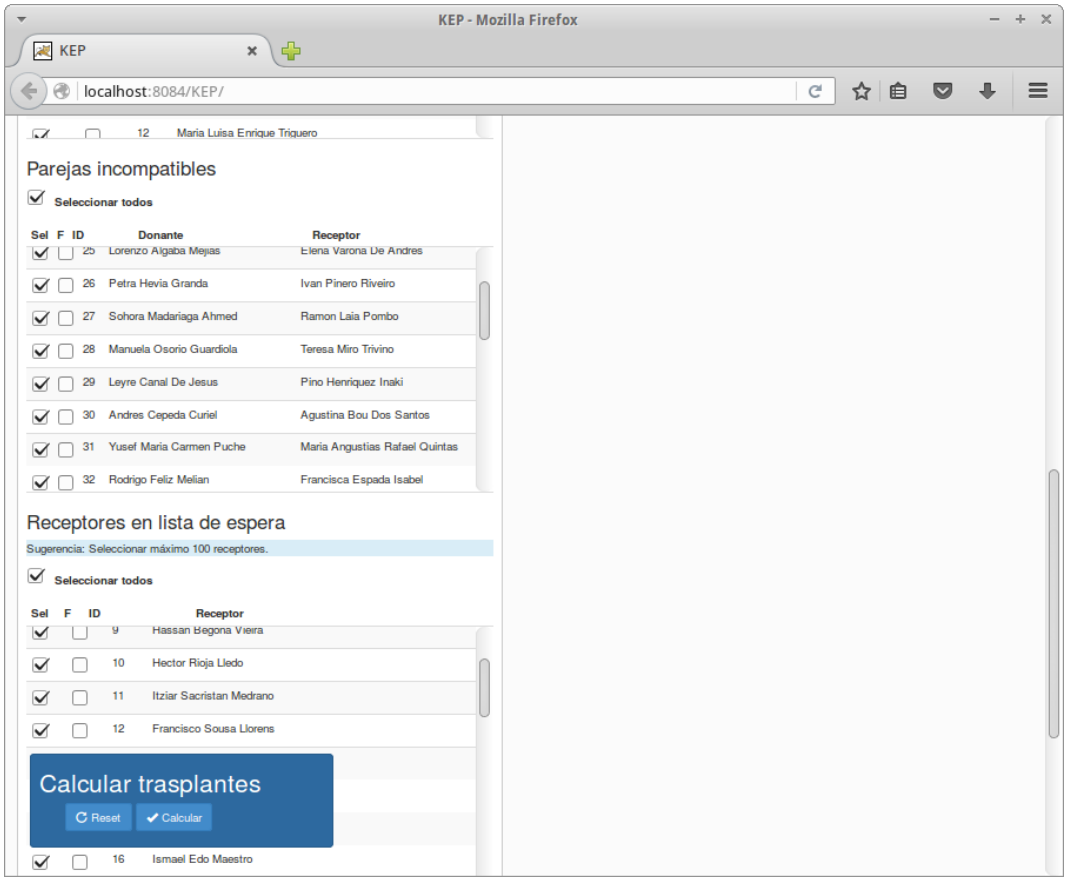


Figure A.7: Captura de pantalla del sistema desarrollado. Hay una celda de selección F a lado de cada donante y receptor, con el fin de indicar si se trata de un donante o receptor *forzado* (véase esta definición en la Sección 2.1).

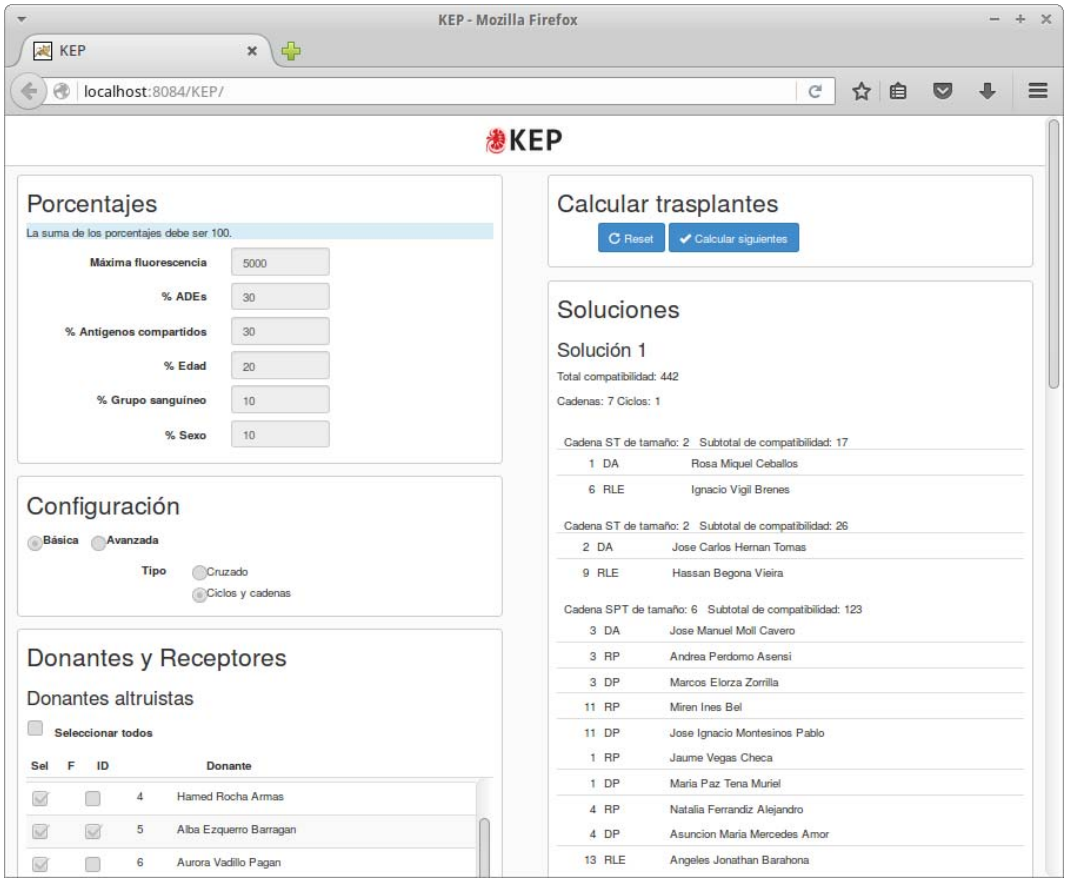


Figure A.8: Captura de pantalla del sistema desarrollado. Del lado derecho se muestran los ciclos y cadenas de la solución.

Bibliography

[1] Anderson R, Ashlagi I, Gamarnik D, Roth AE (2015) Finding long chains in kidney exchange using the traveling salesman problem. Proc Natl Acad Sci 112:663–668

[2] Ashlagi I, Gamarnik D, Rees MA, Roth AE (2012) The need for (long) chains in kidney exchange. (National Bureau of Economic Res, Cambridge, MA)

[3] J.A. Bautista Ruiz. Reporte de Trabajo Profesional. 2015. Sistema de Asignación Renal para el Instituto Nacional de Ciencias Médicas y Nutrición Salvador Zubirán. Facultad de Ciencias. UNAM

[4] Edward A. Bender, S. Gill Williamson. Foundations of Combinatorics with Applications. Dover (2006)

[5] Bienstock D, Goemans MX, Simchi-Levi D, Williamson D (1993) A note on the prize collecting traveling salesman problem. Math Program 59(1):413–420.

[6] Bradley, Stephen P., Arnoldo C. Hax, and Thomas L. Magnanti. Applied Mathematical Programming. Reading, MA: Addison-Wesley Publishing Company, 1977

[7] Cormen, Thomas H. Introduction To Algorithms. Cambridge, Mass.: MIT Press, 2009.

[8] Dantzig, G. B. 1951. Maximization of linear function of variables subject to linear inequalities. In Activity Analysis of Production and Allocation, T. C. Koopmans, Ed. 339–347.

[9] Elmasri, Ramez and Sham Navathe. Fundamentals Of Database Systems. 7th Edition. Pearson, 2015.

[10] Gerdts, Matthias. 2008. Linear Programming. Preliminary version. Institut für Mathematik Universität Würzburg

[11] H. Karloff. Linear Programming. Progress in Theoretical Computer Science. Birkhauser, 1991.

[12] Kreher, Donald L and Douglas R Stinson. Combinatorial Algorithms. Boca Raton, Fla.: CRC Press, 1999.

[13] Meindl, B. and Templ, M. (2012). Analysis of Commercial and Free and Open Source Solvers for Linear Optimization Problems. Technical report, Technische Universität Wien.

[14] Méndez Durán Antonio, Méndez Bueno J.Francisco, (2010) Epidemiología de la insuficiencia renal crónica en México. Diálisis y Transplante 10.1016/S1886-2845(10)70004-7 Elsevier

[15] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University (Pittsburgh, PA,1976).

[16] Carlos Ordonez, Naveen Mohanam, Carlos Garcia-Alvarado, Predrag T Tosic, Edgar Martinez. 2012. Fast PCA computation in a DBMS with aggregate UDFs and LAPACK. Proceedings of the 21st ACM international conference on Information and knowledge management.

[17] F.T. Rapaport, The case for a living emotionally related international kidney donor exchange registry,Transplant. Proc. 18 (1986) 5–9.

[18] L.F. Ross, D.T. Rubin, M. Siegler, M.A. Josephson, J.R. Thistlethwaite Jr., E.S. Woodle, Ethics of a paired-kidney-exchange program, New Eng. J. Med. 336 (1997) 1752–1755.

[19] Rees MA, etal. (2009) A non-simultaneous extended altruistic donor chain. New Engl J Med 360(11):1096–1101.

[20] Roth, Alvin E., Tayfun Sönmez, M. Utku Unver, Francis L. Delmonico, and Susan L. Saidman. 2006. Utilizing List Exchange and Undirected Good Samaritan Donation through 'Chain' Paired Kidney Exchanges. American Journal of Transplantation, 6(11): 2694 –2705.

[21] Roth, Alvin E., Tayfun Sönmez, and M. Utku Ünver. 2005a. Pairwise Kidney Exchange. Journal of Economic Theory, 125(2): 151– 88.

[22] Roth, Alvin E., Tayfun Sönmez, and M. Utku Ünver (2004) Kidney Exchange, Quarterly Journal of Economics, 119, 2, 457-488.

[23] Saidman, Susan L., Alvin E. Roth, Tayfun Sönmez, M. Utku Ünver, and Francis L. Delmonico. Increasing the Opportunity of Live Kidney Donation By Matching for Two and Three Way Exchanges. Transplantation, 81, 5, March 15, 2006, 773-782.

[24] Victor Klee and G. J. Minty. How good is the simplex algorithm. In O. Shisha, editor, Inequalities - III, pages 159–175. Academic Press, 1972.

[25] Zadeh, Norman. 1980. What is the Worst Case Behavior of the Simplex Algorithm? Stanford Univ CA. Dept of Operations Research, ADA089486.

[26] "Centro Nacional De Trasplantes :: México"
http://www.cenatra.salud.gob.mx/interior/trasplante_proceso_trasplante.html
Web. 24 Nov 2015

[27] "Informe Anual 2014 sobre donación y trasplante"
http://www.cenatra.salud.gob.mx/interior/trasplante_estadisticas.html
Web. 01 Dic 2015

[28] "Hospital Civil De Guadalajara"
http://www.hcg.udg.mx/pags/infAnuncio.php?IDBol=561&IDFot=1
Web. 03 Dic 2015

[29] "NBER Reporter 2012 Number 4: Research Summary"
http://www.nber.org/reporter/2012number4/roth.html
Web. 13 Ene 2016

[30] "TSP Data for the Traveling Salesperson Problem"
https://people.sc.fsu.edu/ jburkardt/datasets/tsp/tsp.html
Web. 20 Nov 2015.

[31] "Bell Number – From Wolfram Mathworld"
http://mathworld.wolfram.com/BellNumber.html
Web. 05 Feb 2016.

[32] "Desarrollan Algoritmo Que Podría Triplicar Trasplantes De Riñón — Gaceta Digital UNAM".
http://www.gaceta.unam.mx/20160613/desarrollan-algoritmo-que-podria-triplicar-trasplantes-de-rinon
Web. 23 Jun 2016.