Universidad Nacional de Rosario

Tesina de grado
para la obtención del grado de
Licenciado en Ciencias de la Computación

Optimización de recorridos en ciudades. Una aplicación al sistema de recolección de residuos sólidos urbanos en el Municipio de Concordia.

Autor:
Federico A. Bertero

Directora:
Graciela L. Nasini

Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Av. Pellegrini 250, Rosario, Santa Fe, Argentina

22 de septiembre de 2015

Resumen

En esta tesis se estudia la aplicación de técnicas de la Investigación Operativa y de Sistemas de Información Geográfica en el diseño de recorridos en ciudades. En particular, se aplican estas técnicas a las rutas que deben realizar los vehículos encargados de la recolección de residuos en la Ciudad de Concordia, Entre Ríos.

Dado que los vehículos están obligados a circular por todas las cuadras que componen cada zona de recolección, el problema del diseño de rutas, óptimas en cantidad de kilómetros realizados, se encuadra en las variaciones del clásico Problema del Cartero Chino (Chinese Postman Problem). Las restricciones asociadas a las normas específicas de circulación en la ciudad de Concordia (sentidos de las calles, los giros permitidos, entre otros) definen una variante específica de este problema general que debe ser estudiada y modelada.

Para ello, en el desarrollo de la tesis se utilizan herramientas de la Teoría de Grafos y diversos modelos Programación Lineal Entera. La aplicación de estos modelos a las instancias específicas de la Ciudad de Concordia implica el contacto con servidores de información geográfica, el uso de algoritmos para la resolución de programas de Programación Lineal Entera y el estudio de diversos formatos para la representación de los resultados obtenidos.

Adicionalmente, con el fin de brindar una interfaz amigable y práctica a la Subsecretaria de Higiene Urbana de la Municipalidad de Concordia, se desarrolla una herramienta que permite consultar las rutas obtenidas en forma de listados de direcciones y de animaciones.

Finalmente, se considera el problema de, entre todas las posibles formas de recorrer una ruta óptima en cantidad de kilómetros, determinar la que implique la menor cantidad de giros en las esquinas. Se obtiene un modelo de este problema como instancias del Problema del Viajante de Comercio (TSP) y se aplica a las rutas mínimas obtenidas.

Palabras clave: Grafos, Problema del cartero chino, Problema del viajante de comercio, Programación lineal entera, Recolección de residuos.

Agradecimientos

Quiero agradecer principalmente a mi directora, Graciela, por todas las enseñanzas, el tiempo y las oportunidades que me brindó estos últimos años.

A Guillermo y Javier, por darme la oportunidad de formar parte del proyecto en el que se basa esta tesis.

A los jurados, Daniel y Pablo, por la rápida evaluación.

A todos los profesores que forman parte de la LCC, por las ganas y gran dedicación que ponen en la carrera.

A los grandes amigos que hice a lo largo de estos años en la facultad, Steve, Qk, Tincho, Marian, Juli, Fepi, Lu, Emi, Ariel, Nati, Alejo, Mel, Nico, Lilo, Lulo, Leo, Eze, Isma, Adriel, Toro, Campesino, Buho y a los que seguro estoy olvidando nombrar, por las largas horas de estudio y tiempo libre.

A mis amigos de la vida, Chino, Jochi, Tronco, Hippie, Santi, Tota, Gonza, Galy, Lechuga, Lues (ambas), Julia, Stefi, Angie, Vir, Vicko y Ney, por estar siempre.

A Vir, mi novia, por toda la paciencia, el cariño y las distracciones.

Y por ultimo pero no menos importante, a mi familia por el apoyo que me dio en todos estos años y en especial a mis padres que me permitieron dedicarme a estudiar sin tener que preocuparme por nada más.

A mi familia

Índice general

Capítulo 1

Introducción

1.1. Marco general

El diseño de sistemas de recolección de residuos urbanos eficientes se ha convertido en una prioridad para los gobiernos de las principales ciudades de todo el mundo debido a preocupaciones con respecto a la contaminación, la salud pública y el medio ambiente, así como también a los impactos presupuestarios de transporte, costos de operación y mano de obra de estos sistemas.

En esta tesis se describe el trabajo realizado para mejorar, utilizando técnicas de optimización matemática, los gastos asociados a los recorridos semanales que deben realizar los camiones recolectores de residuos y su personal en la Ciudad de Concordia, Entre Ríos, Argentina.

Concordia, oficialmente San Antonio de Padua de la Concordia, es un municipio de la provincia de Entre Ríos, República Argentina. Comprende la localidad del mismo nombre, la localidad de Osvaldo Magnasco y un área rural. Está compuesto en la actualidad por 109 barrios y posee una superficie de 20.006 ha, mientras que su Planta Urbana, 2.307 ha. El ejido de la ciudad de Concordia se encuentra emplazado sobre la orilla hidrográfica derecha del río Uruguay y comprende actualmente un casco urbano principal y una serie de barrios periféricos a este. Con 152.282 habitantes, según el censo nacional del año 2010, la ciudad de Concordia ocupa el segundo lugar entre las ciudades mas pobladas de la provincia de Entre Ríos, siendo superada únicamente por la ciudad de Paraná. Es además la de mayor población de la cuenca del río Uruguay.

El servicio de recolección de residuos sólidos urbanos está a cargo de la Subsecretaría de Higiene Urbana (SHU), quien también se encarga de limpieza, barrido y desmalezado de espacios públicos. La Dirección de Recolección, dependiente de la SHU, realiza la recolección de residuos en todo el ejido urbano, tanto de los residuos domiciliarios como de los contenedores ubicados en diversos barrios. El tratamiento y disposición final de los residuos y el resto de los servicios está a cargo de la Dirección de Higiene Urbana.

El municipio cuenta con un depósito de basura ubicado a 12 km del ejido urbano

llamado *Campo Abasto* que cuenta con una superficie de 243 hectáreas y una planta de separación y tratamiento de los residuos reciclables, principalmente materiales como vidrio, papel, cartón y botellas plásticas. Para el servicio de recolección se dispone de 14 camiones ubicados en un depósito contiguo a la SHU donde se realizan las tareas de mantenimiento.

La recolección está organizada en 23 *zonas*. Una zona es un conjunto de manzanas cuyas cuadras deberán ser recorridas por un camión determinado durante un turno determinado. Toda la información respecto a la organización del sistema se encuentra solamente almacenado en formato *papel*. En la Figura 1.1 se muestra la digitalización del mapa en papel donde se encuentra esta información. Los números identifican el camión encargado de la recolección mientras que el color indica el turno en que se realiza.

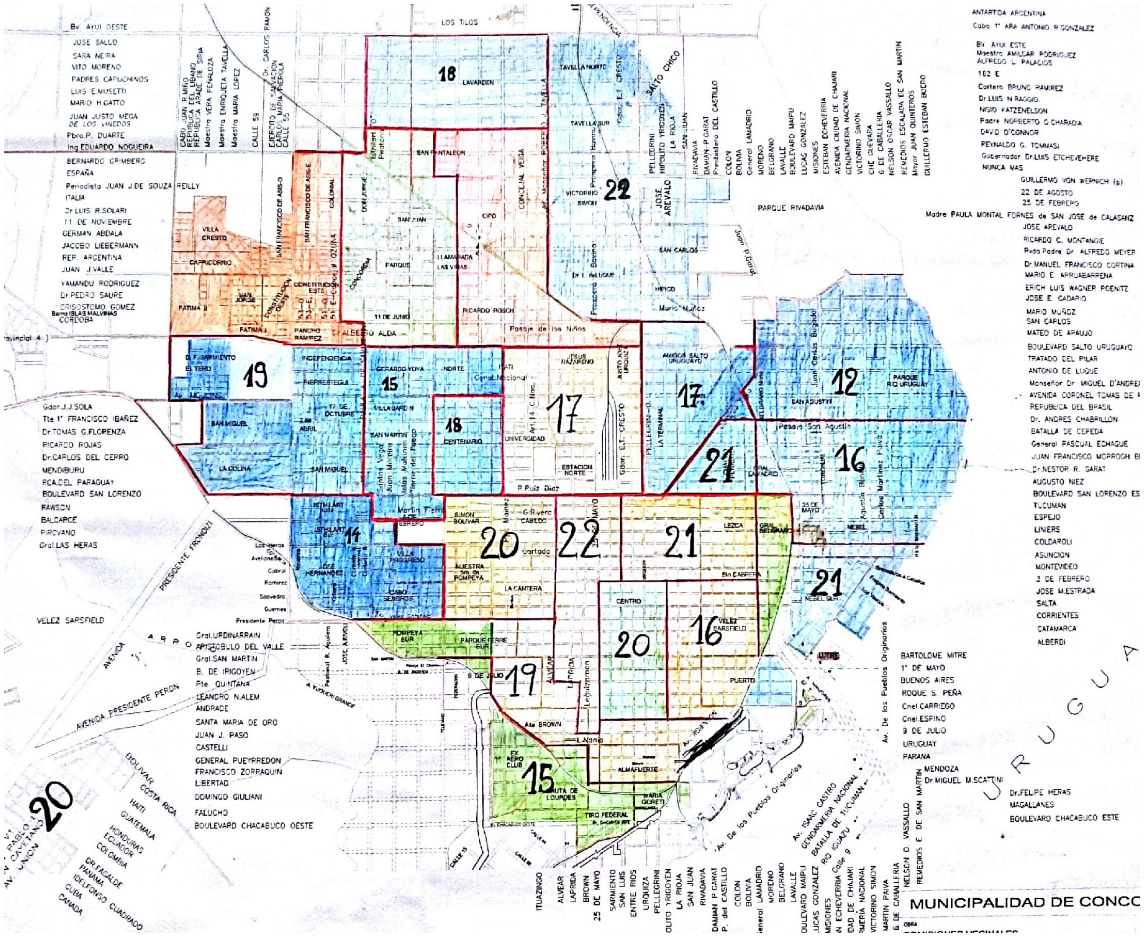


Fig. 1.1: Zonas de recolección del Municipio de Concordia.

La recolección domiciliaria se realiza en 4 turnos:

- Turno Mañana: de 07:30 a 10:30 hs.
- Turno Media mañana: de 10:30 a 13:30 hs.
- Turno Tarde: de 13:30 a 16:30 hs.

- Turno Noche: de 18:30 a 22:30 hs.

Hasta el año 2014, todos los residuos domiciliarios se depositaban en el frente de los domicilios y la recolección se realizaba una vez por día. A finales de dicho año, el municipio modificó el sistema de recolección en la zona centro (Número 20, con color verde en el mapa) ubicando un contenedor en cada una de sus cuadras y recorriendo esta zona tres veces al día.

Todo camión con una zona asignada, inicia su recorrido en la SHU, se dirige a la zona correspondiente y, después de recorrer la misma, va al Campo del Abasto donde realiza la descarga y procede a retornar a la SHU. En los dos casos (residuos en los frentes domiciliarios o contenedores en el medio de cada cuadra) se requiere que los recorridos que realicen los camiones pasen por todas las cuadras de la zona, con excepción de las cuadras sin salida en las cuales el camión recolector no ingresa y espera en la esquina que los empleados realicen la recolección a pie y la lleven al camión.

Sin embargo, la SHU deja a criterio de los conductores la definición de este recorrido interno en cada zona. En consecuencia, los mismos cambian continuamente y, en la mayoría de los casos, no se recorren algunas cuadras. De esta manera, la única información certera con la que se cuenta respecto a cada uno de los recorridos en las diferentes zonas es las esquinas de la zona donde se inicia y se termina la recolección.

Como consecuencia de las reuniones mantenidas con el personal de la SHU en las que se analizó el sistema de recolección y las características del contexto, surgió como uno de sus objetivos el de mejorar los costos relacionados con el consumo de combustible, las horas de trabajo de los empleados y el desgaste de los camiones. Así, el problema se centró en diseñar los recorridos de manera de optimizar la distancia y de esta forma, obtener en consecuencia una mejora en tiempos, consumo de combustible y desgaste del vehículo.

También se resolvió, en esta primer etapa, no analizar la posibilidad de rezonificación y trabajar con 14 de las 23 zonas ya definidas por la SHU, correspondientes al área más urbana de la ciudad.

Este trabajo se realizó en el marco del Convenio "Estudio para la optimización de recorridos aplicado a los sistemas de recolección de RSU y reciclables en Municipios" entre el Instituto de Cálculo de la Facultad de Ciencias Exactas y Naturales (FCEyN) de la Universidad de Buenos Aires y la Secretaría de Asuntos Municipales del Ministerio del Interior y Transporte de la Nación. Dicho convenio plantea realizar un trabajo académico para mejorar, utilizando técnicas de optimización matemática, ciertos aspectos a definir de la recolección de residuos en los Municipios de Salta, Tucumán, Concordia y Bariloche. Participaron cuatro grupos, cada uno compuesto por un estudiante de grado y un director y con un municipio asociado, con quien realizar el trabajo descripto en el convenio. Esta tesis describe el trabajo realizado junto y para el Municipio de Concordia y el estudio de algunos nuevos problemas derivados.

1.2. Teoría de grafos y Problemas de Recorridos

La Teoría de Grafos modela numerosos problemas de diseño y optimización de mundo real. La mayoría de los autores ubica el nacimiento de la Teoría de Grafos en el *Problema de los siete puentes de Königsberg*, planteado por Leonhard Euler en 1736.

La ciudad de Königsberg cuenta con dos islas en el río Pregel que se unen a la tierra firme mediante siete puentes, como se muestra en la figura 1.2.

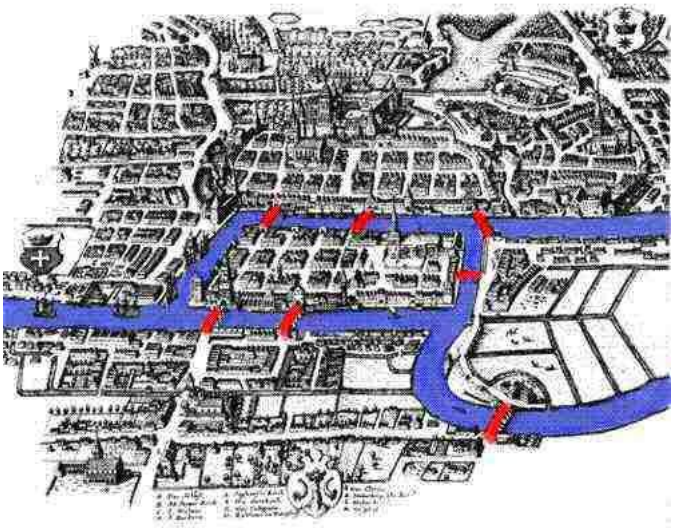


Fig. 1.2: Los Siete puentes de la ciudad de Königsberg.

La pregunta que plantea Euler es si sería posible dar un paseo empezando en un punto cualquiera de tierra firme y volviendo al mismo punto después de haber cruzado cada puente exactamente una sola vez.

Euler enfoca el problema representando cada parte de tierra por un punto y cada puente, por una línea, uniendo los puntos que se corresponden (ver figura 1.3).

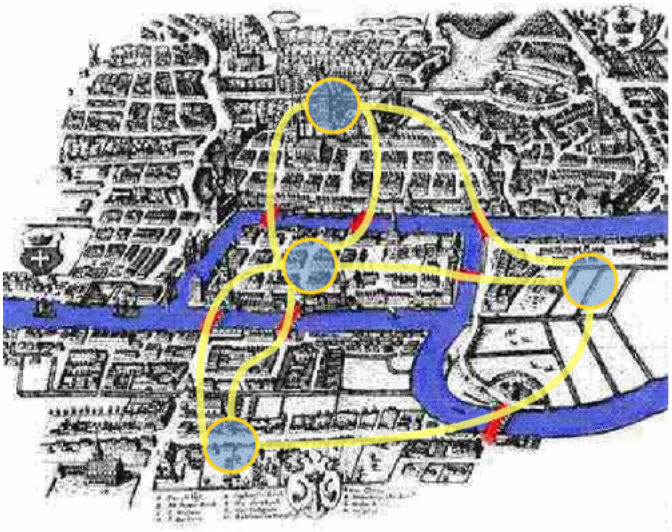


Fig. 1.3: El grafo que modela los Siete puentes de Königsberg.

Entonces, el problema anterior se puede trasladar a la siguiente pregunta: ¿se puede realizar el dibujo sin repetir las líneas y volviendo el punto donde se inició el mismo?

Euler demostró que no era posible puesto que hay puntos en la figura para los cuales el

número de líneas que inciden no es par, condición necesaria para entrar y salir del mismo y luego regresar al punto de partida por caminos distintos.

La pregunta de Euler puede generalizarse a cualquier *dibujo de puntos y líneas que unan algunos pares de puntos*, dibujos que son nuestros actuales *grafos*. Los puntos del dibujo se denominan *vértices* y las líneas, *aristas*. El número de aristas incidentes en un punto se denomina *grado del vértices*. Los recorridos que pasan exactamente una vez por todas las líneas y vuelven al inicio son nuestros actuales *circuitos eulerianos* en honor a Euler y su resultado general que se enuncia el siguiente teorema: *Un grafo posee un circuito euleriano si y sólo si es conexo y todos sus vértices tienen grado par.*

A partir de este problema original surgen muchas variantes del mismo. ¿Qué sabemos si algunas de las aristas se pueden recorrer en un sólo sentido? Si el grafo no tiene un circuito euleriano, ¿cuál será el mínimo número de aristas que deberemos repetir? Y si cada arista tiene un costo, ¿cuál conjunto de aristas será el más conveniente para repetir de manera que el costo a adicionar sea mínimo? Todas estas preguntas y muchas otras variaciones han sido motivo de estudio en el área de la Optimización Combinatoria.

Claramente el problema a resolver en el Municipio de Concordia está asociado a una de estas variantes relativas a recorridos en grafos que deben garantizar recorrer todas sus aristas.

A continuación, presentamos las definiciones formales y resultados previos que usaremos en el desarrollo de la tesis.

1.3. Definiciones y resultados previos

Para toda terminología y notación sobre grafos no definidas en este trabajo, se sigue [9].

Un *grafo* G consiste en un par ordenado $G = (V, E)$, donde V es un conjunto finito no vacío de *vértices* o *nodos* y E es un conjunto de pares no ordenados vw con $v, w \in V$ y $v \neq w$, denominados *aristas*. Un *multigrafo* es un grafo $G = (V, E)$ donde E es un multiconjunto de pares no ordenados vw con $v, w \in V$. Es decir, puede tener aristas múltiples entre los mismos nodos. Para cada $uv \in E$, decimos que u y v son *adyacentes* o *vecinos*.

Un *grafo dirigido* o *digrafo* es un grafo $G = (V, A)$ donde $A \subseteq \{(a, b) \in V \times V : a \neq b\}$, es un conjunto de pares ordenados de elementos de V. Dado un arco $(a, b) \in A$, a es su *nodo inicial* y b su *nodo final*. Similarmente, un *multidigrafo* es un grafo $G = (V, A)$ donde A es un multiconjunto de pares ordenados (v, w) con $v, w \in V$. Dado un digrafo $G = (V, A)$, llamamos *grafo subyacente* al grafo con mismo conjunto de vértices que G y una arista por cada arco de G.

Un *grafo mixto* es una tupla $G = (V, E, A)$ donde V es un conjunto finito no vacío de *vértices* o *nodos*, E es un conjunto de pares no ordenados vw con $v, w \in V$, denominados *aristas* y A es un conjunto de pares ordenados (v, w) con $v, w \in V$, denominados *arcos*.

Observemos que un grafo $G = (V, E)$ (resp. un digrafo $G = (V, A)$) puede pensarse como un grafo mixto $G = (V, E, A)$ donde A es vacío (resp. E es vacío).

Así, dado un grafo mixto $G = (V, E, A)$, para todo $v \in V$ definimos como *grado de*

v y lo notamos $\delta(v)$ al número de aristas de E incidentes en v. También definimos *grado de entrada* de v, denotado $\delta_-(v)$, como el número de arcos de A que tienen a v como su vértice final y *grado de salida* de v, denotado $\delta_+(v)$, como el número de arcos de A que tienen a v como su nodo inicial. También, para todo $v \in V$ definimos $\Gamma_-(v)$ (resp. $\Gamma_+(v)$) al conjunto de arcos tales que v es su extremo final (resp. extremo inicial).

Se llama *camino* en un grafo (resp. digrafo) a una secuencia de vértices tal que existe una arista (resp. un arco) entre cada vértice y el siguiente. Se dice que dos vértices están conectados si existe un camino que comience en uno y termine en otro, de lo contrario estarán desconectados. Dos vértices pueden estar conectados por varios caminos. Un grafo G es *conexo* si para todo par de vértices distintos v y w de G existe un camino de v a w. Un digrafo es conexo si su grafo subyacente lo es. Un *ciclo* o *circuito* es un camino cuyo vértice inicial y final son el mismo.

Un *ciclo euleriano* es aquel que contiene todas las aristas de un grafo exactamente una vez. Un grafo o digrafo que contiene un circuito euleriano se dice que él mismo es euleriano.

Con estas definiciones, el Teorema de Euler puede enunciarse formalmente de la siguiente manera:

Teorema: Sea un grafo o multigrafo $G = (V, E)$. Entonces, G es euleriano si y sólo si G es conexo y $\delta(v)$ es par, para todo $v \in V$.

Tenemos también una caracterización similar para digrafos eulerianos:

Teorema: Sea un digrafo o multidigrafo $G = (V, A)$. Entonces, G es euleriano si y sólo si G es conexo y $\delta_+(v) = \delta_-(v)$ para todo $v \in V$.

Muchas otras aplicaciones requieren de circuitos que sólo garanticen pasar por todos los vértices del grafo. Un *ciclo hamiltoniano* es aquel que pasa por todos los vértices de un grafo exactamente una vez. También en este caso, un grafo o digrafo que contiene un circuito hamiltoniano se dice que él mismo es hamiltoniano. Sin embargo, a diferencia del caso euleriano, no se conocen caracterizaciones sencillas de los grafos hamiltonianos.

Tal como en el caso de los puentes de Konigsberg, los recorridos de los camiones recolectores de residuos de la ciudad de Concordia pueden interpretarse como recorridos en un grafo cuyos vértices son las esquinas de la zona y las aristas o arcos, las cuadras. Dependiendo de las restricciones de circulación, la ciudad se modelará con un grafo, un digrafo o un grafo mixto. Esto grafos en general no serán eulerianos, como veremos más adelante. Esto nos lleva al problema de optimización asociado en el cual nos preguntamos cómo recorrer todas las cuadras de manera que las cuadras adicionales necesarias para realizar el circuito *sumen lo menos posible*.

A este problema de optimización se lo denomina *Problema del Cartero Chino* y tendrá diferentes características dependiendo de que se trate de un grafo dirigido, no dirigido o mixto. En todas las variantes tenemos una función *costo* (que puede estar medido en distancia, tiempo o cualquier otro atributo) asociado a cada arco o arista. Para simplificar la presentación formal de los mismos, empezaremos definiendo el caso mixto.

Problema del Cartero Chino Mixto.

Dado un grafo mixto $G = (V, E, A)$ y una función de costos sobre cada uno de los elementos de E y A, determinar un circuito de G que contiene toda arista y arco de G al menos una vez y tal que la suma de los costos de las aristas utilizadas sea mínimo.

El *Problema de Cartero Chino* planteado originalmente [6] corresponde al caso *no dirigido*, esto es, a las instancias del problema anterior para las cuales A es vacío. El denominado *Problema de Cartero Chino Dirigido*, corresponde a las instancias para las cuales E es vacío.

A diferencia del caso no dirigido, los casos dirigido o mixto pueden no tener solución. Para que exista un circuito euleriano el grafo debe ser *fuertemente conexo* (para cada par de vértices u y v existe un camino de u hacia v y un camino de v hacia u).

Los grafos asociados a las zonas de recolección suelen presentar el problema de no ser fuertemente conexos. Basta con pensar en una esquina *ángulo de la zona* en la cual confluyan dos cuadras con direcciones que ambas llegan o salen de esa esquina. En esos casos, no tendremos mas remedio que utilizar cuadras aledañas a la zona, que serán recorridas no para realizar la recolección de residuos en ellas sino para poder seguir circulando sobre la zona a recorrer. (ver figura 1.4).

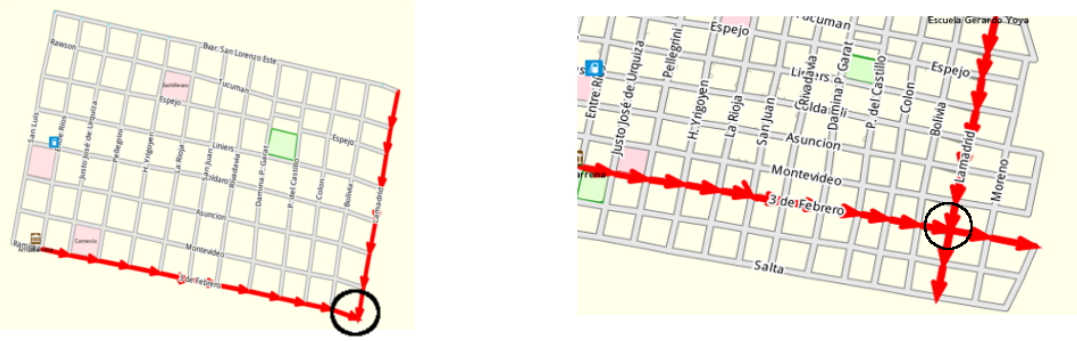


Fig. 1.4: Zona de recolección *no fuertemente conexa.*

En ese caso, deberemos considerar un *grafo extendido* que incluya esquinas y cuadras aledañas a la zona por las cuales el circuito *puede* pasar (a diferencia de las cuadras de la zona por las cuales *debe* pasar).

Esta situación corresponde al denominado *Problema del Cartero Rural*, que se enuncia a continuación:

El Problema del Cartero Rural

Dado un grafo mixto $G = (V, E, A)$, $R \subseteq E \cup A$ y una función costo sobre $E \cup A$, determinar un circuito que pase por todos los elemento de R al menos una vez y tenga costo mínimo.

Más allá de que algunos casos prácticos requieren servir todos los arcos de la red, la mayoría de las aplicaciones de la vida real son modeladas como un problema del cartero rural.

Es interesante tener en cuenta que, aún en caso de que el problema del Cartero Chino Mixto sea factible en el subgrafo correspondiente a los arcos obligados a ser recorridos, muchas veces permitir que se usen arcos no obligatorios baja el costo del circuito.

A pesar de la similitud en el enunciado de estos problemas, cada uno de ellos presenta características muy específicas que se reflejan en la dificultad y estrategias de resolución. Claramente, en todos los casos, si el grafo cuyas aristas queremos recorrer es euleriano, el problema de optimización es trivial ya que el costo del circuito óptimo será la suma de las aristas a recorrer.

Esto no sucede en el caso de los circuitos hamiltonianos. Si el grafo es hamiltoniano, existen muchas formas de recorrer sus vértices (distintos órdenes) y cada uno de ellos tendrá un costo asociado al costo de las aristas que se utilicen. Esto da lugar a otro clásico problema de optimización asociado a recorridos en grafos que se denomina *Problema del Viajante de Comercio* y que mencionaremos como TSP.

Problema del Viajante de Comercio (TSP)

Dado un grafo mixto hamiltoniano $G = (V, E, A)$ y una función costo sobre $E \cup A$, determinar un circuito hamiltoniano de G de costo mínimo.

El TSP es uno de los problemas *más duros* de resolver en el área. En la próxima sección formalizaremos este tipo de conceptos asociados a la *complejidad computacional de problemas*.

1.4. Complejidad Computacional de problemas

La Teoría de la Complejidad Computacional se centra en la clasificación de los problemas de acuerdo a la dificultad de su resolución.

Para poder referirnos a conceptos como *problemas difíciles* o *problemas de dificultad equivalentes*, es necesario comprender alguna nociones básicas de esta teoría ([8], [19]).

En primer lugar, la Teoría de Complejidad Computacional se desarrolla sobre los *problemas de decisión*. Un *problema de decisión* es aquel en donde las respuestas posibles son *si* o *no*. El problema se define entonces a partir de un conjunto posible de entradas (I) y una pregunta. El problema de decidir si un número entero es múltiplo de otro puede ser plateado como un problema de decisión de la siguiente manera:

ENTRADA: a, b par ordenado de números enteros.

PREGUNTA: ¿Es a múltiplo de b?

En este caso, el conjunto de entradas I es el de todos los pares ordenados de números enteros. Se llama *instancia* de un problema al problema resultante al elegir una entrada particular dentro de I. Por ejemplo, una instancia del problema anterior sería: *Dados a=4 y b=2, ¿es 4 múltiplo 2?*.

Un *algoritmo* A para aun problema Π puede definirse como una lista de instrucciones elementales bien definidas que toman una entrada (de un conjunto de entradas I) y eventualmente produce una salida (perteneciente a un conjunto de salida O). El tiempo de ejecución de A es una función $f_A : \mathbb{N} \to \mathbb{N}$ donde $f_A(i)$ es el número máximo de instrucciones elementales necesarias para procesar una entrada de tamaño i. Debido a la dificultad para calcular la función f_A, decimos que f_A es del orden de g, donde $g : \mathbb{N} \to \mathbb{N}$, y lo denotamos $f_A = \mathcal{O}(g)$, si existe una constante $C \in \mathbb{R}$ y una constante $N \in \mathbb{N}$ tal que

$f_A(n) \leq Cg(n)\ \forall n \in \mathbb{N}, n > N$. Si $f_A = \mathcal{O}(g)$ decimos que el tiempo de ejecución es $\mathcal{O}(g)$. Si g es una función lineal, polinomial o exponencial, decimos que A es un algoritmo lineal, polinomial o exponencial respectivamente.

El orden de un algoritmo se usa como medida de su eficiencia. Un algoritmo es considerado *eficiente* sólo en el caso que sea polinomial.

Dado un problema de decisión Π, decimos que A es *un algoritmo para* Π, si A toma como entrada al conjunto de instancias de Π y produce como salida *si* para las entradas pertenecientes al conjunto de instancias *si* y *no* para las pertenecientes al conjunto de instancias *no*.

Decimos que un problema Π es *polinomial* si existe un algoritmo polinomial A para Π. Llamamos $\mathcal{P}$ al conjunto de todos los problemas de decisión polinomiales.

Decimos que un problema pertenece a la clase $\mathcal{NP}$ si las instancias en donde la respuesta es *si* pueden ser *certificadas* en tiempo polinomial. Por ejemplo, no se conocen algoritmos polinomiales que permitan resolver el problema de reconocimiento de grafos hamiltonianos. Sin embargo, toda instancia de respuesta *si*, esto es, todo grafo que sí es hamiltoniano, tiene como *certificado* de su condición un orden de recorrido de los nodos. Para verificar la validez de ese certificado, basta confirmar que el grafo contiene un circuito que recorre sus vértices en ese orden. Y esta verificación puede hacerse en tiempo polinomial. De esta manera, el problema de reconocimiento de grafos hamiltonianos está en la clase $\mathcal{NP}$, aunque no se sabe si está en $\mathcal{P}$. Claramente, todo problema en $\mathcal{P}$ también está en $\mathcal{NP}$ ya que el algoritmo de certificación polinomial es el mismo que resuelve el problema y por lo tanto $\mathcal{P} \subset \mathcal{NP}$. ¿Será posible que $\mathcal{P} = \mathcal{NP}$?

Uno de los Problemas de Milenio[1] consiste en demostrar lo que la comunidad científica da por cierto desde hace muchos años y es que $\mathcal{P} \neq \mathcal{NP}$. Esto es, que hay problemas para los cuales no existiría un algoritmo polinomial que lo resuelva. O sea, que hay problemas *más difíciles* que otros. Para *comparar dificultades* de problemas se utilizan las *reducciones*.

Un problema Π es *reducible* a un problema Π' si existe un algoritmo que transforma cada instancia de Π en una instancia de Π' tal que la instancia de Π tiene respuesta *si* si y solo si su correspondiente instancia en Π' tiene respuesta *si*. Si el algoritmo de reducción es polinomial se dice que Π se *reduce polinomialmente* a Π'. Claramente, si $\Pi' \in \mathcal{P}$, Π también está en $\mathcal{P}$. En general, Π' es *al menos tan difícil como* Π.

Decimos que Π es $\mathcal{NP}$-completo si $\Pi \in \mathcal{NP}$ y todo problema de $\mathcal{NP}$ es polinomialmente reducible a Π. Por lo dicho anteriormente, si existiera un problema $\mathcal{NP}$-completo que estuviera en $\mathcal{P}$, todos los problemas de $\mathcal{NP}$ estarían en $\mathcal{P}$ y $\mathcal{NP}$ coincidiría con $\mathcal{P}$. Así, el problema del milenio mencionado anteriormente podría reformularse como *demostrar que ningún problema $\mathcal{NP}$-completo puede ser resuelto polinomialmente* lo cual considera a los problemas $\mathcal{NP}$-completos como los problemas de la clase $\mathcal{NP}$ *más duros de resolver*.

La mayoría de las aplicaciones nos llevan a problemas que no son problemas de decisión sino a problemas de *optimización* ya que la solución buscada está asociada al máximo o al mínimo de una cierta función objetivo.

Un problema de optimización puede ser representado de la siguiente forma: dada una función $f : X \to \mathbb{R}$ donde X es un conjunto de números reales, se intenta encontrar un

[1] `https://es.wikipedia.org/wiki/Problemas_del_milenio`

elemento $x_0 \in X$ tal que $f(x_0) \leq f(x)\ \forall x \in X$ (problema de minimización) o tal que $f(x_0) \geq f(x)\ \forall x \in X$ (problema de maximización).

Al no ser problemas de decisión, en principio no se pueden clasificar como $\mathcal{P}$ o $\mathcal{NP}$, sin embargo un problema de optimización puede ser fácilmente transformado en uno de decisión respondiendo a la siguiente pregunta: dado $x_0 \in \mathbb{R}$, ¿existe $x \in X$ tal que $f(x) \leq x_0$ (minimización) o $f(x) \geq x_0$ (maximización)?, donde x_0 pasa a formar parte de la instancia del problema de decisión.

No es difícil probar que existe un algoritmo polinomial que resuelve el problema de optimización si y sólo si existe un algoritmo polinomial que resuelve el problema de decisión asociado. Cuando el problema de decisión asociado es $\mathcal{NP}$-completo, decimos que el problema de optimización es $\mathcal{NP}$-difícil.

En 1965, Edmonds[5] presentó un algoritmo polinomial para resolver el problema del Cartero Chino no dirigido, basado en el algoritmo para encontrar todos los caminos mínimos en un grafo enunciado por Floyd[7] y Warshal[21].

Si el grafo de entrada para el problema del Cartero Chino es dirigido, el problema puede ser reducido a un problema de flujo mínimo en redes[6].

De esta manera, tanto el caso dirigido como el no dirigido resultan ser problemas polinomiales.

A diferencia del Cartero Chino dirigido y no dirigido, no se conocen algoritmos polinomiales que resuelvan el Problema del Cartero Chino mixto. Más aún, Papadimitrou[8] demostró que el problema es $\mathcal{NP}$-*difícil*.

Tal como lo hemos mencionado, el diseño de recorridos óptimos para las zonas de recolección de residuos del Municipio de Concordia corresponden a instancias del Problema de Cartero Rural. En este caso, tanto el caso dirigido como no dirigido son $\mathcal{NP}$-*difíciles* [14].

La Programación Lineal Entera es la herramienta que se ha consolidado como la más eficiente a la hora de resolver con optimalidad grandes instancias de problemas $\mathcal{NP}$-difíciles. En la próxima sección se presentan formalmente las definiciones y resultados preliminares asociados a esta rama de la optimización.

1.5. Programación Lineal Entera y problemas NP-difíciles

Se denomina *Programación Lineal* al problema de optimización donde la función objetivo es una función lineal en $\mathbb{R}^n$ y el dominio $X \subset \mathbb{R}^n$ de optimización se define como el conjunto solución de un número finito de desigualdades lineales que llamaremos restricciones[4]. Cada restricción define un subespacio de $\mathbb{R}^n$ y el conjunto X de soluciones factibles, intersección de estos subespacios, es lo que geométricamente se define como un *poliedro* de $\mathbb{R}^n$.

Como disciplina matemática, la Programación Lineal es bastante joven. Sus inicios se remontan al año 1947, cuando G. B Dantzig diseña el *método Simplex* para la resolución de instancias de programación lineal asociadas a problemas de planeamiento de la fuerza aérea norteamericana. Básicamente, este método avanza en el poliedro de soluciones factibles desde un punto extremo hasta otro adyacente, de tal manera que el valor de la función

objetivo aumente, o en el peor de los casos, permanezca igual.

Lo que siguió al año 1947 fue un período de rápido desarrollo en esta nueva disciplina. Pronto se hizo evidente que una amplia gama de problemas aparentemente no relacionados, asociados a la gestión de la producción, podían ser modelados como instancias de Programación Lineal y, más importante aun, resueltos haciendo uso del método Simplex. Tales problemas habían sido tradicionalmente abordados mediante un enfoque de prueba y error guiado únicamente por la experiencia y la intuición. El uso de la Programación Lineal provocó, en la mayoría de los casos, un aumento considerable en la eficiencia de toda la operación, mejora que hasta entonces iba exclusivamente de la mano de avances tecnológicos.

Como el cálculo desarrollado en el siglo XVII a partir de la necesidad de resolver los problemas de la mecánica, la Programación Lineal fue desarrollada en el siglo XX a partir de la necesidad de resolver problemas de gestión. Sin embargo, otras profundas influencias estimularon la evolución del nuevo campo desde sus inicios. La economía fue una de ellas: ya en 1947, TC Koopmans comenzó señalando que la Programación Lineal proporciona un excelente marco para el análisis de las teorías económicas clásicas. Y fue el 14 de octubre de 1975, la fecha en la que la Academia Real Sueca de Ciencias otorgó el Premio Nobel de la Ciencia Económica para T. C. Koopmans y L. V. Kantorovich por su trabajo en la teoría de la asignación óptima de recursos.

Ahora, ¿es el método Simplex un algoritmo eficiente para resolver problemas de Programación Lineal? En general sí aunque en algunos casos puede darse lo contrario. Como se dijo, el método recorre los puntos extremos de un poliedro y el número de estos puede llegar a ser exponencial en el tamaño del problema. Así, en 1972, V. Klee y G. Minty[13] desarrollaron un problema lineal de n variables para el cual Simplex debería examinar cada uno de los vértices extremos del poliedro que resulta ser un número exponencial.

Queda preguntarse: ¿es la Programación Lineal un problema polinomial?, es decir, ¿existe algún algoritmo que siempre resuelva un programa lineal en tiempo polinomial? La respuesta es si ya que, en el año 1979, L.G. Khachiyan [12] presentó el *algoritmo del elipsoide* para resolver problemas de programación lineal, y demostró que es un algoritmo polinomial. El algoritmo del elipsoide es iterativo y, en principio, se aplica a un problema del siguiente tipo: dado un conjunto de desigualdades lineales del tipo $Ax \leq b$, ¿tiene una solución?. En cada iteración se construye un elipsoide que contiene siempre una solución del problema anterior (si la hay). El elipsoide construido en cada iteración es siempre "más pequeño" que el anterior, de manera que, después de un determinado número de iteraciones se encuentra una solución o se concluye que tal solución no existe.

Luego, en 1984, N. Karmarkar[11] propuso un nuevo enfoque para resolver problemas de Programación Lineal, basado en un algoritmo que sigue trayectorias en el interior del conjunto convexo de soluciones factibles. Este fue llamado *método de punto interior* y en lugar de avanzar de un punto extremo a otro por los bordes del conjunto convexo, hasta alcanzar la solución óptima (si existe), excava por el interior del conjunto para determinar una solución óptima.

Dado que estos últimos dos métodos presentados no están restringidos a la dirección de los bordes ni a sus longitudes, es razonable suponer que podrían ser significativamente más rápidos que el método Simplex. A pesar de esto y del ejemplo presentado por Klee y Minty, el algoritmo Simplex resulta ser muy eficiente en la práctica. Se sabe que la

complejidad promedio para resolver un Programa Lineal crece en forma lineal en el número de restricciones y en forma logarítmica en el número de variables.

Después de 70 años de su nacimiento, acompañado del gran avance que se logró en los últimos 20 años en cuanto a performance, el método Simplex es el algoritmo base en la mayoría de programas de resolución (o *solvers*, como se los llama habitualmente) de instancias de Programación Lineal. Para instancias muy grandes, algunas de estas herramientas de optimización incluyen la opción de mezclar el algoritmo Simplex con técnicas provenientes de los algoritmos de puntos interiores. Entre los solvers de PL mas populares podemos nombrar a CPLEX[10], XPRESS[2], SOPLEX[3] y GLPK[4].

Hoy en día, y mediante el uso de alguno de los solvers antes nombrados, se resuelven instancias de programación lineal con miles de variables y miles de restricciones en pocos minutos[5].

La *Programación Lineal Entera* (PLE) es un problema de optimización con las mismas características de la Programación Lineal pero con la restricción adicional de que las variables deben tomar valores enteros. Así, un problema de PLE puede escribirse de la siguiente forma:

$$
\begin{aligned}
&\textit{minimizar} \sum_{j=1}^{n} c_j x_j \\
&\textit{sujeto a} \\
&\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m \\
&\qquad x_j \in \mathbb{N}_+, \qquad\qquad j = 1, \ldots, n
\end{aligned}
$$

La PLE nos permite modelar muchas más situaciones que la Programación Lineal. De hecho, gran parte de los problemas de Optimización Combinatoria pueden modelarse como instancias de Programación Lineal Entera[22]. Sin embargo, la resolución de los problemas será mucho más costosa ya que, mientras que la Programación Lineal está en la clase $\mathcal{P}$, la PLE es $\mathcal{NP}$-difícil.

Una *relajación lineal* de un problema de PLE es un problema lineal tal que el conjunto de soluciones enteras de su dominio es igual al conjunto de soluciones enteras del problema de PLE. En particular, el problema lineal que se obtiene al eliminar las restricciones que fuerzan a las variables a ser enteras, es una relación lineal. Claramente, el valor óptimo de la relajación lineal será una cota del valor óptimo del problema entero. Un problema de PLE puede tener muchas relajaciones lineales y una relajación se considera *mejor* que otra si la cota que provee es más ajustada. Una relajación lineal puede ser *mejorada* con la adición de nuevas desigualdades lineales, válidas para todas las soluciones enteras pero que no son satisfechas por todas las soluciones de la relajación anterior. Estas desigualdades se denominan *cortes*.

La técnica más eficiente para la resolución de instancias de PLE es la denominada

[2] FICO Xpress-Optimizer. `http://www.fico.com/en/products/fico-xpress-optimization-suite`
[3] The Sequential object-oriented simPlex. `soplex.zib.de/`
[4] GNU Linear Programming Kit. `www.gnu.org/software/glpk/glpk.html`
[5] http://plato.asu.edu/ftp/lpsimp.html

ramificación y corte [15].

En este método se resuelven varias relajaciones del problema donde los cortes se utilizan para la obtención de mejores cotas. La ramificación es una partición sucesiva del dominio en subdominios más pequeños donde se supone que la resolución del problema es más sencilla. Las buenas cotas sirven para descartar zonas del dominio donde seguramente no estará la solución óptima y hacer que el árbol de particionamiento sea más pequeño.

Existe al día de hoy una gran variedad de solvers de PLE de propósito general con un alto nivel de madurez (e.g., CPLEX[10], Gurobi[6], SCIP[7]). Si bien actualmente es posible resolver instancias de problemas que hasta hace unos años parecían inalcanzables, este tipo de algoritmos sigue encontrando limitaciones al intentar resolver instancias reales, que suelen ser de gran escala.

En los algoritmos para la resolución de PLE, a diferencia del caso de la Programación Lineal, no podemos hablar de un comportamiento *promedio* en términos del tamaño de la entrada. La dificultad de resolución de una instancia depende fuertemente de su *estructura*. Así, se puede encontrar que un solver capaz de resolver instancias realmente grandes, no sea capaz de obtener una solución óptima sobre instancias con una estructura combinatoria particular, relativamente pequeñas.

Esto ha llevado al diseño de solvers específicos para problemas particulares de Optimización Combinatoria, basados también en las técnicas generales para la PLE pero que utilizan la estructura particular del problema a resolver. De esta manera, estos solvers específicos logran resolver instancias muchos más grandes de este problema específico que las que resolvería un solver de PLE general.

Tal es el caso del software Concorde[1] para el TSP. Concorde es un programa realizado por David Applegate, Robert Bixby, Vasek Chvatal y William Cook en el Instituto de Tecnología de Georgia (Georgia Institute of Technology) que permite resolver instancias del TSP en forma exacta mediante programación lineal entera. Para eso se le adjunta un paquete para resolver problemas de PLE como Cplex o QSOpt (de libre uso y desarrollado por los mismos autores). Puede resolver instancias de hasta 1000 elementos en forma eficiente.

Para la resolución de los problemas de PLE que surgen a lo largo de esta tesis, se hizo uso del solvers general de PLE, CPLEX[10] de la empresa IBM.

También, para la resolución del problema planteado en el capítulo 5, utilizamos el software Concorde.

1.6. Estructura de la tesis

Tal como se mencionó, el objetivo de esta tesis es diseñar e implementar algoritmos basados en programación lineal entera para optimizar los recorridos de los camiones de recolección de residuos en el Municipio de Concordia.

Para ellos nos planteamos los siguientes objetivos específicos:

- Hacer uso de los sistemas de información geográfica disponibles, para la correcta

[6] GUROBI Optimizer. `www.gurobi.com/`
[7] Solving Constraint Integer Programs. `http://scip.zib.de/`

creación y manipulación de instancias del problema correspondientes a las zonas de recolección.

- Proponer diferentes modelos para las distintas situaciones que se plantean en las diferentes zonas de la ciudad donde se realiza la recolección de residuos.
- Implementar y evaluar la performance de diferentes algoritmos con el fin de utilizar el mas adecuado.
- Desarrollar una herramienta para la toma de decisiones que integre tanto la creación y modelización de cada una de las instancias como el uso de los algoritmos nombrados en el item anterior.
- Integrar los conocimientos relacionados con los items anteriores en una solución que implique una mejora en la elección de los recorridos de los vehículos encargados de la recolección de residuos en Concordia.

Finalmente, la tesis que se presenta a continuación está estructurada de la siguiente forma:

Capítulo **Construcción de instancias**. Se muestra como se generan los grafos que representan las distintas zonas en base a la información obtenida de un servidor de información geográfica.

Capítulo **Modelos de PLE y su resolución**. Se desarrolla como, una vez obtenidos los grafos que representan las zonas de recolección, se obtienen recorridos eulerianos mínimos (en cantidad de kilómetros) para cada uno de ellos.

Capítulo **Construcción de soluciones para la Municipalidad de Concordia**. Se presentan los diferentes formatos utilizados para la representación de los resultados y una herramienta desarrollada para el fácil acceso a estos por parte del personal del ente encargado de la recolección.

Capítulo **Minimizando cantidad de giros en los recorridos**. Se propone un método para manipular los recorridos óptimos obtenidos con el fin de minimizar la cantidad de giros que estos realizan en las esquinas.

Capítulo **Detalles de la implementación**. Se explican en detalle los algoritmos implementados.

Capítulo **Experiencias computacionales y resultados**. Se comparten los resultados obtenidos al correr los diferentes algoritmos propuestos en cada una de las instancias correspondientes a las zonas de recolección.

Capítulo **Conclusiones y próximos pasos**. Se enuncian las conclusiones obtenidas del trabajo realizado y se señalan posibles próximos pasos.

Capítulo **Apéndice** se muestra como ejemplo la ruta óptima obtenida para una de las zonas de recolección.

Capítulo 2

Construcción de Instancias

Tal como fue mencionado en la Introducción, el problema de diseño de los circuitos de recolección de residuos de la ciudad de Concordia será abordado como instancias particulares del Problema del Cartero Rural. A tal efecto, cada zona de recolección y sus cuadras aledañas deberán ser modeladas por un grafo, y sus aristas (las cuadras) tener un costo asociado (la distancia).

Esta etapa de modelado requiere en primer lugar un proceso de obtención de la información cartográfica digital asociada a cada una de las zonas de recolección con las que trabajamos y el pre-procesamiento realizado sobre la misma.

Posteriormente, a partir de esta información, se define la estructura de datos utilizada para mantener la información de cada una de las instancias del problema.

2.1. Sistemas de Información Geográfica

Un sistema de información geográfica (GIS) es cualquier sistema de información que permite la organización, almacenamiento, manipulación, análisis y modelización de grandes cantidades de datos procedentes del mundo real que están vinculados a una referencia espacial.

Los GIS se presentan en forma de herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones.

En la web podemos encontrar un buen número de servicios de geolocalización donde algunos de los más conocidos son *Google Maps* o *Microsoft Bing Maps*. Ambos servicios ofrecen gran cantidad de información y están presentes en innumerables páginas web y aplicaciones, sin embargo, ofrecen un acceso muy restringido a la información haciendo imposible su almacenamiento y manipulación. Esto hace que se utilicen alternativas libres y abiertas. Un proyecto que cada día tiene más fuerza en este ámbito es *OpenStreetMap*.

OpenStreetMap[1] es un proyecto colaborativo para crear mapas libres y editables. Con una infraestructura que actualmente se aloja en la University College de Londres, el proyecto ofrece a los usuarios un GIS en constante crecimiento que es de libre acceso y que, además, cuenta con un juego de APIs sin restricciones, para facilitar su integración en todo tipo de servicios.

En la figura 2.1 podemos ver una imagen correspondiente a un pequeño fragmento del mapa de Concordia brindado por el Municipio y, de fondo, la herramienta que ofrece la pagina del proyecto para la selección y de un área sobre el mapa (aplicada en este caso a una de las zonas del mapa de la SHU).

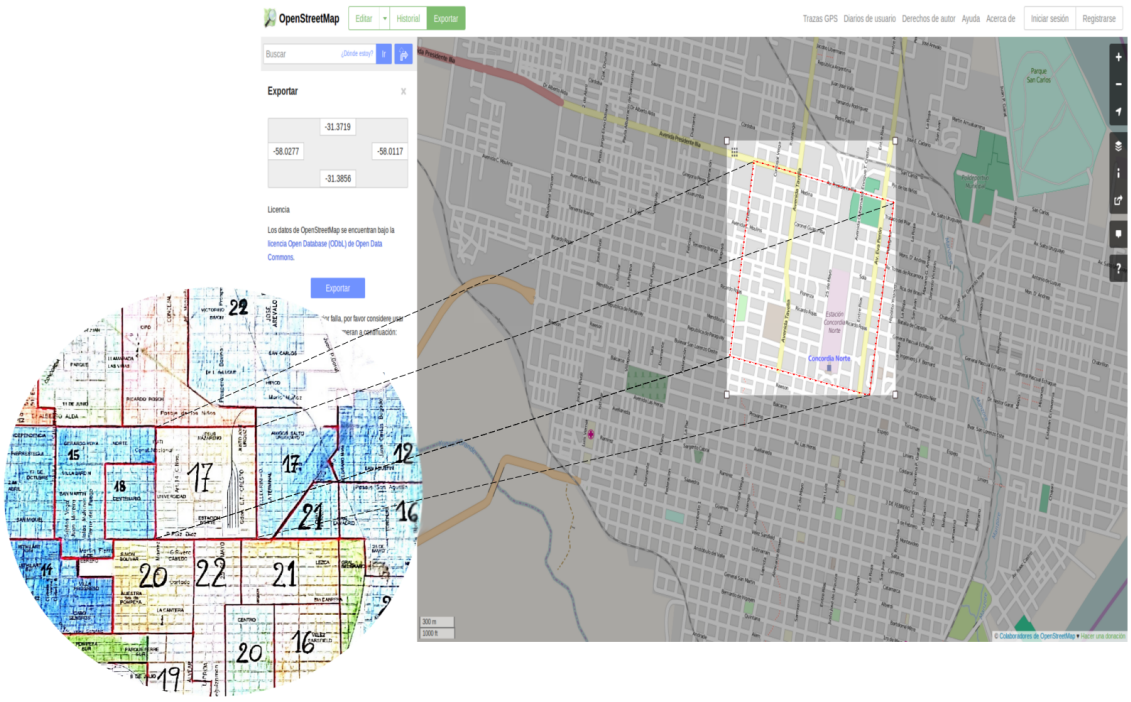


Fig. 2.1: Zona de recolección en el mapa brindado por la SHU y su área correspondiente en OpenStreetMap

Una vez seleccionada la zona, la información de la misma se exporta a un archivo en formato OSM.

Un archivo OSM es un archivo XML[2] que, básicamente, consiste en una lista de instancias de las primitivas de datos *node* y *way*.

Las instancias de *node* consisten en un simple punto en el espacio definido por su latitud, longitud y un identificador único y con ellas se puede representar elementos como un semáforo, una escuela o un radar. En este proyecto se utilizan para la representación de los puntos formados por la intersección de dos calles, es decir, las esquinas.

[1] `http://www.openstreetmap.org/`

[2] Lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible.

Las instancias de *way* se utilizan para la representación de una o varias calles adyacentes y contienen una lista ordenada de *nodes* que representan las esquinas por la que está compuesto dicho segmento. Para mayores precisiones sobre los detalles técnicos, ver Sección 6.1.

El formato de la herramienta para exportar datos de OpenStreetMap, requiere seleccionar un área más amplia que la estrictamente necesaria para representar una determinada zona de recolección. Esto se ve a la izquierda de la figura 2.2, donde al abrir el archivo OSM asociado a la zona con número 17 en el mapa, con el editor de datos cartográficos JOSM (Java OpenStreetMap Editor)[3], vemos que el archivo contiene muchos datos que están fuera de los límites de la zona en cuestión. Estos datos superfluos deberán ser descartados en un proceso de edición que se realiza manualmente.

Para esta edición se utiliza el editor antes nombrado, JOSM, ya que permite identificar fácilmente los elementos correspondientes a las instancias de *node* y *way* del archivo OSM, haciendo mas fácil la creación, modificación y eliminación de estos objetos.

A la derecha de la figura 2.2 se puede ver la representación cartográfica de los datos de la zona luego del proceso de edición.

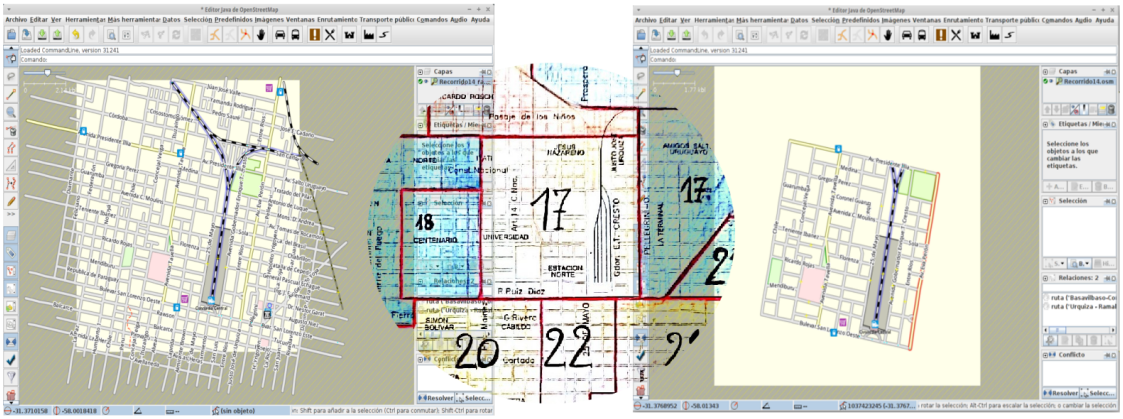


Fig. 2.2: Antes y después de la edición del archivo OSM correspondiente a la zona antes exportada.

2.2. Manejo de fallas del mapa

Como es usual en este tipo de base de datos formadas de manera colaborativa y que contienen información de la realidad, existen fallas en el mapa.

Se verificaron muchas de las calles, especialmente las que se encuentran en los recorridos de recolección con los que tratamos. Para eso se comparó con otras bases de datos cartográficas como por ejemplo, las antes nombradas, Google [4] y Microsoft [5].

Para comprobar sentidos de las calles se utilizaron las fotos que provee el servicio de Google, donde se pueden ver los autos y a partir de ellos identificar el sentido.

[3] Editor para los datos obtenidos del sitio de OpenStreetMap, https://josm.openstreetmap.de
[4] `maps.google.com`
[5] `maps.bing.com`

2.3. Zonas extendidas

Tal como fue mencionado en la introducción, en algunos casos puede ser más efectivo tomar una cuadra exterior a la zona que se está recorriendo, en lugar de mantener el recorrido siempre dentro de esta. En general, estos casos se dan en zonas con muchas cuadras de una sola mano, en los que para llegar a una esquina es necesario realizar muchas vueltas. Estas se podrían evitar al tomar una calle fuera la zona. En otros casos, como el de la figura 2.3, sin el uso de una cuadra exterior a la zona, estaríamos ante una zona sobre la cual no sería posible hallar un recorrido. Imaginemos al momento de realizar un recorrido sobre el área representada en el mapa de la izquierda, una vez que se llegue a la esquina resaltada, ya sea por 3 de Febrero o Lamadrid, no es posible salir de ella por el sentido de sus calles incidentes.

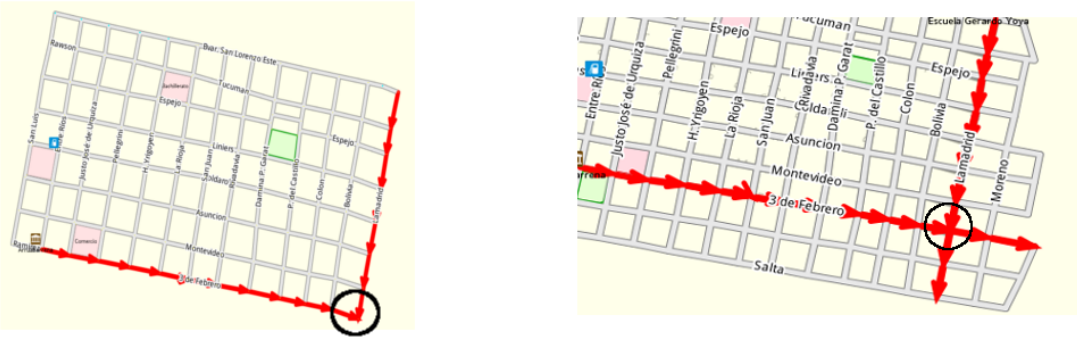


Fig. 2.3: Zona sin extensión y parte de la zona extendida asociada.

Teniendo en cuenta los casos recién nombrados, por cada zona de recolección creamos dos mapas digitales relacionados. Uno que se adapta de forma perfecta a la zona y otro que mantiene cierto borde sobre ella, es decir, un mapa que representa una *zona extendida*. El primero, se obtiene mediante el proceso que se describió en las ultimas dos secciones y el segundo sigue el mismo proceso excepto al momento de la edición, cuando no se descartan todos los datos que están fuera de los límites de la zona.

De esta manera se hace posible identificar ciertas cuadras como opcionales, es decir, cuadras sobre las cuales no es necesario realizar la recolección, pero que pueden usarse para reducir el costo del recorrido o para hacer posible encontrar un recorrido para la zona. A estas se las llama *calle o cuadra extendida*.

2.4. Del mapa a una instancia del Problema del Cartero Rural

Tal como se mencionó, las zonas a recorrer se modelan con un grafo. Para cada una de las zonas se utilizó un grafo mixto $G = (V, E, A)$ que se construye a partir de los datos que se encuentran en los archivos OSM correspondientes a la zona que coincide de manera perfecta con el área descripta en el mapa físico y a la zona extendida.

El conjunto V se construye a partir de las instancias *nodes* asociadas a una esquina, así cada elemento en V se corresponde a una instancia de este tipo de datos con dicha característica. Por cada nodo se guarda la información de su instancia de node asociada, un identificador único e información sobre la latitud y longitud del punto que representa

en el mapa.

Los conjuntos E y A se construyen a partir de las instancias *ways*. Más específicamente, cada elemento de E se corresponde a una instancia de *way*, siempre que esta represente una calle con sentido estricto de circulación. Por otro lado, cada elemento de A se corresponde a una instancia de *way*, siempre que esta represente una calle con doble sentido de circulación. Por cada arco o arista se almacena un identificador único de calle, su nombre, los identificadores de sus nodos inicio y final, la cantidad de manos que posee, su sentido de circulación (simple o doble), su longitud, si se trata de una calle sin salida y si es una *calle extendida*.

La mayoría de los datos asociados a los elementos de los conjuntos V, E, A, se pueden obtener de manera directa o muy simple, a partir de la información disponible en los archivos OSM.

Para los detalles acerca de cómo se crean estos grafos y cómo se determinan estos valores, ver el Capítulo 6.

Una vez obtenido el grafo a partir del archivo OMS, se realiza un pre-procesamiento asociado a las restricciones del sistema de recolección. En particular, como las calles sin salida no se recorren, se eliminan los arcos y aristas que poseen un vértice extremo con grado 1.

Además, conocemos las esquinas en las que los camiones inician y terminan el recorrido de la zona y nunca coinciden. En consecuencia, el recorrido buscado es un *camino euleriano*. Sin embargo, el agregar al grafo un arco de costo nulo que vaya del vértice asociado a la esquina final al nodo asociado a la esquina inicial, nos permite modelarlo como circuito. Más adelante veremos cómo es que este arco *ficticio* lo hace posible.

Técnicamente, sea $G = (V, E, A)$ el grafo que representa una determinada zona de recolección y $v_{inicio}, v_{fin} \in V$, los nodos asociados a las esquinas donde se comienza y finaliza el recorrido, se agrega al conjunto de arcos, A, el elemento (v_{fin}, v_{inicio}), llamado *arco ficticio*.

Finalmente, una instancia del Problema del Cartero Rural requiere de una función costo asociada a arcos y aristas del grafo. Definimos la función costo $c : E \cup A \to \mathbb{R}$ donde, si $(v_i, v_j) \in E \cup A$, $c((v_i, v_j))$ retorna el dato distancia de la calle formada por las esquinas asociadas a los vértices v_i y v_j, siempre que (v_i,v_j) sea diferente a (v_{fin}, v_{inicio}), es decir, siempre que no se trate del arco ficticio. En este ultimo caso, la distancia asociada es 0, es decir $c((v_{fin}, v_{inicio})) = 0$.

Capítulo 3

Modelos de Programación Lineal Entera y su resolución

3.1. Modelo Inicial

Tal como fue mencionado, la obtención de recorridos mínimos en kilómetros realizados para cada una de las zonas de recolección de residuos de la ciudad de Concordia fue planteada como la resolución de una instancia del Problema del Cartero Rural (construidas en el Capítulo 2).

También se ha mencionado que el Problema del Cartero Rural es $\mathcal{NP}$-difícil y que la herramienta más eficiente para la resolución de grandes instancias es su modelización como programa lineal entero (ver Capítulo 1.6).

El primer modelo de PLE con que se trabajó fue el presentado en [17], con algunas modificaciones que permitieron reducir el número de variables.

Sea $G = (V, E, A)$ un grafo mixto conexo con un conjunto de vértices V, un conjunto de arcos A y un conjunto de aristas E. En el conjunto de vértices tenemos identificados dos de ellos, v_{inicio} y v_{fin}, que corresponden respectivamente al inicio y fin del recorrido.

Definimos $\hat{E}$ y $\check{E}$ como conjuntos de arcos, donde se almacenan copias dirigidas de los elementos en E que buscan representar los dos sentidos de una calle doble mano. Así, si la arista $ij \in E$, el arco $ij \in \hat{E}$ y el arco $ji \in \check{E}$. Si e es un arco orientada de $\hat{E}$ o $\check{E}$, entonces $\tilde{e}$ es su arco orientado opuesto. Es decir, si $e = ij$ entonces $\tilde{e} = ji$.

Definimos el conjunto de arcos opcionales $\bar{A}$ que representa las calles que componen la zona extendida, es decir, aquellas calles que no estamos obligados a recorrer, y la función de costo $c : \bar{A} \cup A \cup \hat{E} \cup \check{E} \to \mathbb{R}_+$ la cual, dado un arco dirigido, retorna su longitud asociada.

En el modelo se definen las variables naturales x, donde la variable x_{ij} representa la cantidad de veces que se recorre el arco ij. Así, el primer modelo de PLE es el siguiente:

$$\textit{minimizar} \sum_{ij \in A \cup \hat{E} \cup \check{E}} c_{ij} x_{ij}$$

sujeto a

$$x_a \geq 1, \qquad \forall a \in A \qquad (1)$$

$$x_e + x_{\tilde{e}} \geq 1, \qquad \forall e \in \hat{E} \qquad (2)$$

$$\sum_{a \in \Gamma_-(v)} x_a = \sum_{a \in \Gamma_+(v)} x_a, \qquad \forall v \in V \qquad (3)$$

$$x_{ts} = 1 \qquad (4)$$

$$x_a \in \mathbb{Z}_+, \qquad \forall a \in A \cup \hat{E} \cup \check{E} \cup \bar{A} \qquad (5)$$

La función objetivo es minimizar la suma de los costos asociados a las cuadras recorridas, es decir, la distancia total recorrida.

Las restricciones en (1) indican que se debe pasar, al menos una vez, por cada cuadra con un único sentido de circulación (arcos en A) mientras que las restricciones en (2) indican que, por cada cuadra doble mano (aristas en E), se debe recorrer al menos una vez una de sus copias dirigidas.

Vale aclarar que estas restricciones no incluyen las calles de la zona extendida, dándoles la posibilidad, a diferencia de las calles de la zona estricta, de no ser recorridas.

Las restricciones en (3) son denominadas *restricciones de conservación de flujo* que obliga a que la cantidad de veces que se entra a un nodo sea igual a la cantidad de veces que se sale del mismo, haciendo que la solución retornada corresponda a un circuito.

La restricción (4) asegura que el arco ficticio, cuyo costo es nulo, sea recorrido exactamente una vez en el circuito. De esta manera, si borramos el arco ficticio del circuito euleriano asociado a la solución del modelo, obtenemos un camino euleriano del mismo costo, que va desde el vértice inical al final.

Finalmente, la restricción (5) obliga a que las variables tomen valores enteros, los cuales indican la cantidad de veces que se recorren los arcos.

3.1.1. Construcción de un recorrido a partir de la solución

A partir de la solución del PLE, se tiene la información de por qué cuadras se debe pasar más de una vez a los efectos de poder asegurar recorrer las cuadras de la zona de recolección con costo mínimo en kilómetros.

Resuelto el modelo, se obtiene como solución una lista de pares $(x\#i\#j, n)$ donde el primer elemento del par, $x\#i\#j$, hace referencia a la variable $x_{i,j}$ y el segundo elemento, n, indica la cantidad de veces que se utiliza el arco (v_i, v_j).

A partir de la lista de pares obtenida se construye un multidigrafo $G_{sol} = (V_{sol}, A_{sol})$ de la siguiente manera:

- V_{sol} es el conjunto de vértices que son extremos de algún arco que es recorrido al menos una vez. Esto es, $v_i \in V_{sol}$ si hay un par en la lista de la forma $(x\#i\#j, n)$ con $n \geq 1$.

- Por cada elemento $(x\#i\#j, n)$ se agregan en A_{sol}, n aristas (v_i, v_j).

Al construir los V_{sol} y A_{sol} a partir de la solución obtenida del modelo se garantiza que el multidigrafo G_{sol} posee circuitos eulerianos que pasan exactamente una vez por el arco (v_{fin}, v_{inicio}) y tal que, eliminando este y realizando una simple modificación, G_{sol} pasa a poseer un camino euleriano de costo mínimo que comienza en el nodo v_{inicio} y finaliza en el nodo v_{fin}. En el capitulo 6 se darán detalles de esto.

Sin embargo, la solución del modelo no provee el orden en el cual ir avanzando de manera de recorrer el grafo euleriano. Existen algoritmos sencillos, incorporados en las bibliotecas de grafos como rutinas que, ingresando un grafo euleriano, construyen el orden de recorrido del mísmo.

Se utilizó una implementación de uno de los algoritmos estándar para esta tarea, denominado Algoritmo de Hierholzer [6], en cual se verá en detalle también en el Capítulo 6.

3.2. Giros en U

Los recorridos propuestos por el Algoritmo de Hierholzer presentaban un problema. En las soluciones relacionadas a zonas donde existen calles doble mano, aparecen comportamientos como el de la figura 3.1. En la imagen de la izquierda se representa el grafo asociado a la zona donde las calles son todas doble mano pero la del medio debe ser recorrida en ambos sentidos (por eso hay dos arcos en lugar de una arista).

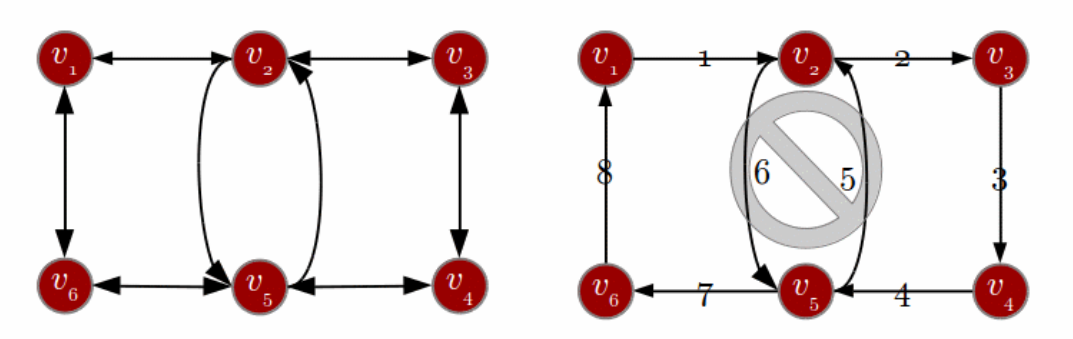


Fig. 3.1: A la izquierda, un grafo sobre el cual se buscara un recorrido eureliano. A la derecha, un posible recorrido propuesto para este.

En la imagen de la derecha, donde se propone un recorrido sobre el grafo, se recorre una calle en una determinada dirección e inmediatamente después se recorre esa misma calle en la dirección opuesta. Esto corresponde a lo que habitualmente se denomina un *giro en U* y las normas de tránsito de las ciudades prohíben este tipo de giros en la mayoría de las esquinas. Algunas excepciones se pueden encontrar en avenidas con semáforos de muchos tiempos o en boulevares, pero no es el caso de Concordia donde los giros en U están prohibidos en todas las esquinas.

En un primer intento por solucionar este inconveniente, se trabajó con una *rutina de mejora* la cual, a partir de la solución propuesta por el Algoritmo de Hierholzer, realiza intercambios *factibles* en el orden propuesto de manera de disminuir la cantidad de giros en U. Por supuesto, el objetivo final era el de obtener una solución sin giros en U, lo cual como veremos, no fue siempre posible.

3.2.1. Reordenando listas

El Algoritmo de Mejora consiste en una modificación en el orden de recorrido propuesto por cada giro en U encontrado en la solución.

La idea es que, recorriendo la solución posición por posición, una vez que halla una subsecuencia de la forma v_iv_j,v_jv_i, se busca de seguir por el vértice v_j. Dado que no existen vértices con grado uno de salida y de entrada, debe existir un arco alternativo por el cual evitar el que produciría el giro en U. Luego, dadas las condiciones de flujo que rigen sobre los vértices, debe existir una forma de volver al vértice v_j y ahí si, tomar el arco v_j.

Si nuestra solución tiene la forma:

$$v_1v_2,\ v_2v_3,\ \ldots,\ v_iv_j,\ v_jv_i,\ \ldots,\ v_jv_s,\ \ldots,\ v_mv_j$$

Podemos ver que tenemos el giro en U $v_iv_j,\ v_jv_i$. Este se puede evitar colocando la porción de la solución que se corresponde con el ciclo $v_jv_s,\ \ldots,\ v_mv_j$ entre los arcos v_iv_j y v_jv_i, obteniendo así la nueva solución:

$$v_1v_2,\ v_2v_3,\ \ldots,\ v_iv_j,\ v_jv_s,\ \ldots,\ v_mv_j,\ v_jv_i,\ \ldots$$

Un ejemplo de la utilización de este método se puede ver en la figura 3.2 donde los números sobre los arcos indican el orden en que son recorridos en el circuito euleriano. En el primer ordenamiento, los arcos 1 y 2 producen un giro en U. Aplicando el algoritmo de mejora se obtiene el segundo ordenamiento donde no se producen giros en U.

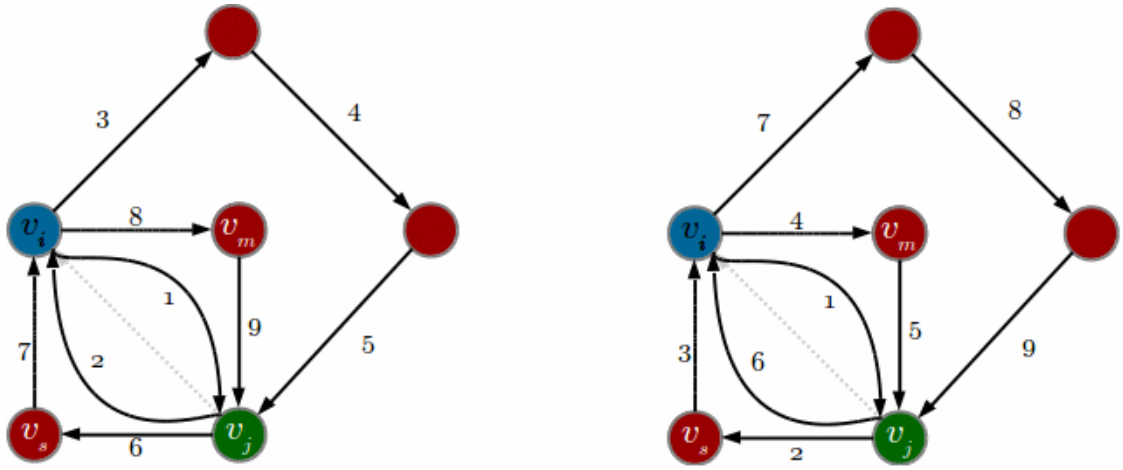


Fig. 3.2: Digrafo Euleriano, orden con giro en U y reordenamiento sin giro en U

Sin embargo, la figura 3.3 podemos ver un ordenamiento para el mismo digrafo euleriano en el cual el algoritmo de mejora no corrige el comportamiento que buscamos erradicar. Cuando el algoritmo detecta la situación planteada por los arcos 8 y 9, debería intercambiar el 9 con otro arco que salga desde el extremo final del arco 8. Sin embargo, el único arco posible es el arco 3, que ya fue recorrido.

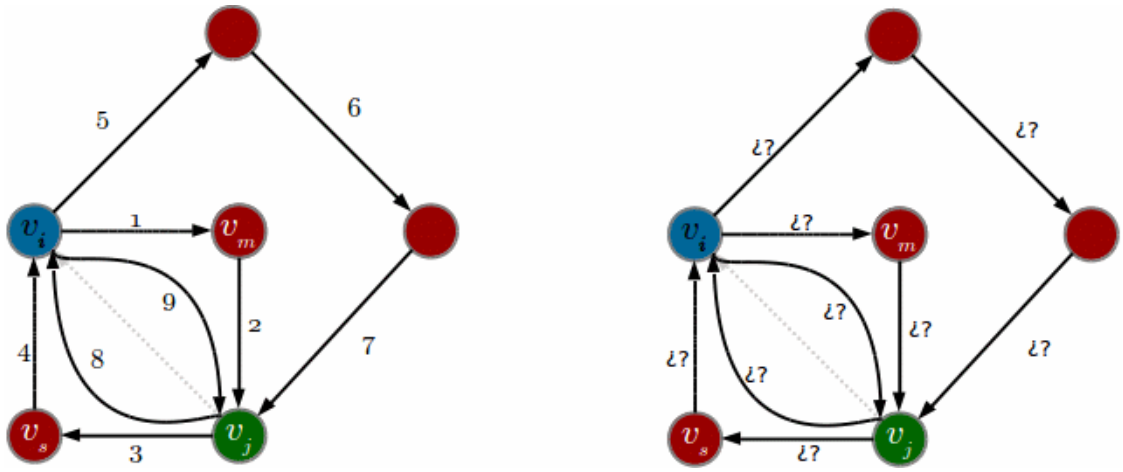


Fig. 3.3: Digrafo Euleriano, orden con giro en U.

Existen digrafos eulerianos cuyos circuitos eulerianos no son posibles de recorrer sin realizar *giros en U*. Esto nos obligó a modificar el modelo de manera de asegurarnos que las soluciones propuestas se puedan recorrer sin realizar giros prohibidos.

3.2.2. Modificando el modelo

Para poder modelar la restricción de no permitir giros en U, es necesario incluir en la información del grafo que modela nuestra ciudad, el hecho de que la posibilidad de ir de una esquina A a una esquina B no solo depende de la existencia de una calle que las une sino también del punto desde el cual se llegó a la esquina A. La prohibición de giro en U requiere poder modelar la restricción "para poder ir de la esquina A a la esquina B, debe existir una calle que una estas esquinas y además se debe haber llegado a A desde una esquina diferente a B".

De poder modelar este tipo de restricciones, también se podrían modelar prohibiciones de giros específicos, como la prohibición de giro a la izquierda en esquinas de avenidas con semáforo. Veremos esto en más detalle a partir de la presentación del nuevo modelo en grafos.

El grafo que nos permite modelar estas situaciones requiere que cada esquina y cada cuadra estén representados por más de un vértice y más de un arco. Diremos que cada esquina está representada por un "supernodo" (conjunto de vértices) y cada calle por una "superarista" (conjunto de aristas). Este nuevo grafo dirigido $SG = (SV, SA)$, el cual llamamos *super grafo*, se construye a partir del grafo original $G = (V, A)$ de la siguiente manera:

- El conjunto SV está formado por los arcos dirigidos y las copias dirigidas de las aristas del grafo G, es decir, $SV = A \cup \hat{E} \cup \check{E}$. La denominación del conjunto como *supernodos* se debe a que en este nuevo grafo, cada vértice v_j de G (esquina de la zona a modelar) queda representada por todos los vértices de SG de la forma v_iv_j correspondientes a los arcos de G que llegan a v_j. Cada uno de estos vértices de SG de la forma v_iv_j indican que estamos en la esquina correspondiente a v_j y llegamos a ella viniendo desde la esquina correspondiente a v_i.

- Todo vértice v_iv_j en SV debe unirse con todo vértice de la forma v_jv_k siempre y cuando el camino $v_iv_jv_k$ sea un camino posible en el grafo G, o sea, si podemos ir desde la esquina asociada a v_j hacia la esquina asociada a v_k siendo que a la esquina de v_j llegamos desde la esquina asociada a v_i.

  Así, un vértice v_iv_j de SG se interpreta como *llego a* v_j *desde* v_i y un arco v_kv_i, v_iv_j como *recorro* v_iv_j *después de haber recorrido* v_kv_i.

  Entonces, si sólo queremos prohibir los giros en U, los caminos no admitidos son los de la forma $v_iv_jv_i$ y por ende, no existirá una arista entre los vértice v_iv_j y v_jv_i de SV teniendo:

$$SA = \{((v_i, v_j), (v_k, v_l)) : (v_i, v_j) \in SV \ \wedge \ (v_k, v_l) \in SV \ \wedge \ v_j = v_k \ \wedge v_i \neq v_l\}$$

La construcción de este nuevo grafo hace muy fácil la tarea de evitar los giros prohibidos a la izquierda. Basta con borrar los arcos $((v_iv_j)(v_jv_k))$ de SA que representen el llegar a v_j desde v_i y luego girar a la izquierda mediante un giro considerado prohibido para llegar a v_k.

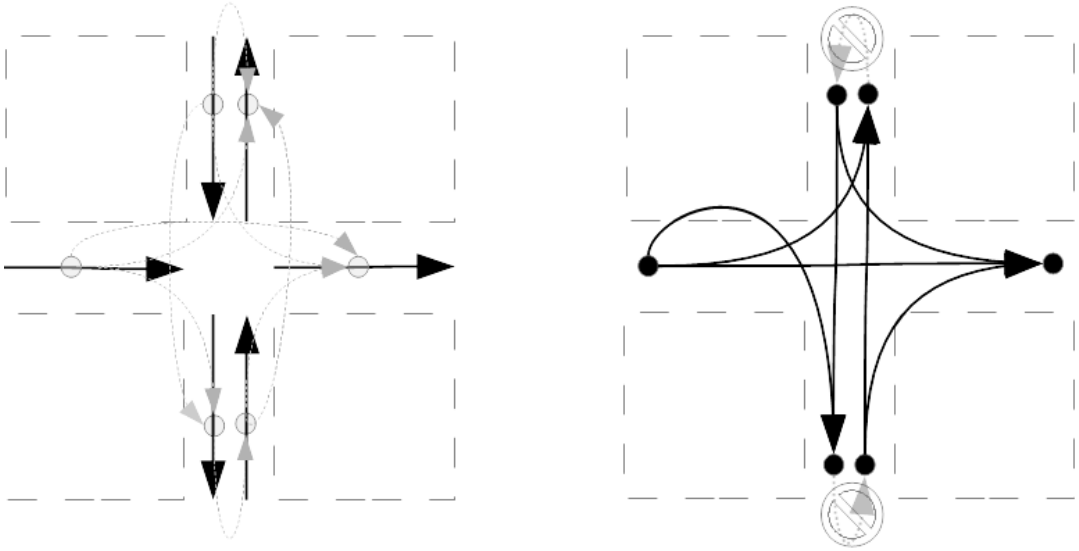


Fig. 3.4: Supergrafo asociado a un fragmento de zona de 4 manzanas.

3.3. Nuevo Modelo de PLE

A continuación se presenta un segundo modelo de programación lineal entera basado en el supergrafo $SG = (SV, SA)$. Aun se tiene en cuenta el conjunto de calles expandidas $\bar{A}$ y el grafo mixto original $G = (V, E, A)$ ya que nos permite identificar, a través de los conjuntos E y A, las restricciones de circulación de las calles.

La variable $sx_{(i,j)(j,k)}$ representa la cantidad total de veces que se llega al arco (v_j, v_k) desde el arco (v_i, v_j).

Otro punto a tener en cuenta con este nuevo grafo, es que se pierde la identificación entre esquina y vértice, lo que produce una dificultad al momento de seleccionar los vértices de inicio y fin del recorrido. Dada la esquina por la que se desea iniciar el recorrido, ahora hay que seleccionar un arco del antiguo grafo G como vértice inicio. Esta dificultad fue resuelta con la información provista de la SHU respecto a por cuál calle llegaban a los puntos de inicio y finalización del recorrido. De esta manera, en cada zona tenemos identificadas dos cuadras a_s, a_t (vértices de SG) por las cuales se llega a estos puntos y agregamos a SA el arco (a_t, a_s) para, en forma similar al modelo anterior, modelar el problema como una instancia del Problema de Cartero Rural que busca un circuito (no un camino) euleriano.

$$\textit{minimizar} \sum_{(i,j)\in SV} c_{ij}\Big(\sum_{(k,i):(k,i)(i,j)\in SA} sx_{(k,i)(i,j)}\Big)$$

sujeto a

$$\sum_{(k,i):(k,i)(i,j)\in SA} sx_{(k,i)(i,j)} \geq 1, \qquad \forall (i,j) \in (SV \cap A) \quad (1)$$

$$\sum_{(k,i):(k,i)(i,j)\in SA} sx_{(k,i)(i,j)} + \sum_{(k,j):(k,j)(j,i)\in SA} sx_{(k,j)(j,i)} \geq 1, \quad \forall (i,j) \in (SV \cap \hat{E}) \quad (2)$$

$$\sum_{(k,i):(k,i)(i,j)\in SA} sx_{(k,i)(i,j)} = \sum_{(j,k):(i,j)(j,k)\in SA} sx_{(i,j)(j,k)}, \qquad \forall (i,j) \in SV \quad (3)$$

$$sx_{(a_t,a_s)} = 1 \quad (4)$$

$$sx_{(i,j)} \in \mathbb{Z}_+, \qquad \forall (i,j) \in (SV \cup \bar{A}) \quad (5)$$

Las restricciones de este modelo tienen por objetivo los mismos que las restricciones del modelo anterior. Solo hay que pensar que cada arco ij de G ahora tiene asociado, en SG, un conjunto de *superarcos* que representan las distintas formas de llegar a i para recorrer la cuadra ij.

3.3.1. Construcción de un recorrido a partir de la solución

Ahora, veremos como se interpreta la solución obtenida por el modelo recién presentado. Esta vez se obtiene una lista de pares $(sx\#i\#j\#j\#k, n)$ donde el primer elemento del par, $sx\#i\#j\#j\#k$, hace referencia al uso del arco (v_j, v_k) de G luego de pasar por el arco (v_i, v_j), también de G, y el segundo elemento, n, indica la cantidad de veces que se utiliza el camino (v_i, v_j, v_k).

A partir de la lista de pares obtenida se construye un multidigrafo $SG_{sol} = (SV_{sol}, SA_{sol})$ de la siguiente manera:

- SV_{sol} es el conjunto de supernodos que son extremos de algún superarco que es recorrido al menos una vez. Esto es, $(v_i, v_j) \in SV_{sol}$ si hay un par en la lista de la forma $(sx\#i\#j\#j\#k, n)$ con $n \geq 1$.
- Por cada elemento $(sx\#i\#j\#j\#k, n)$ se agregan n aristas $((v_i, v_j), (v_j, v_k))$.

Al igual que G_{sol}, el multidigrafo SG_{sol} posee varias caminos que pasan por todas las aristas del multidigrafo exactamente una vez y alguno de ellos se puede encontrar mediante un procedimiento análogo al descripto en la sección 3.1.1.

Solo hay que tener en cuenta que el orden de recorrido obtenido es un listados ordenado en las aristas de SG_{sol}. Para llevar esto a las cuadras de la zona de la ciudad, basta reemplazar en este listado cada superarco $((v_i, v_j), (v_j, v_k))$ de SA por la arista (v_j, v_k) de A.

Sin embargo, las soluciones de este modelo no siempre representan los recorridos requeridos ya que el grafo SG_{sol} puede resultar no conexo como puede verse en la figura 3.5. En la misma se representa una parte de una zona y una posible solución con subciclos.

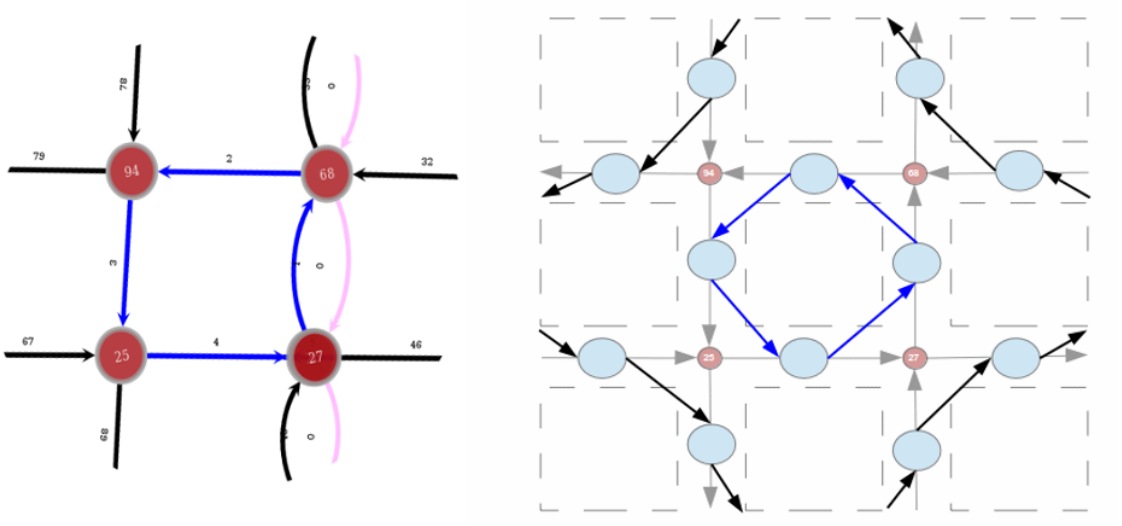


Fig. 3.5: Representación parcial de G y SG_{sol} no conexo

Claramente, las sucesiones de arcos indicadas en la figuras corresponden a soluciones factibles del PLE 3.3 ya que verifican todas las restricciones del mismo y sin embargo no representan un recorrido posible para un camión recolector. Gracias a las restricciones de conservación de flujo en los supernodos (restricciones (3)) sabemos que todas las soluciones factibles del modelo corresponderán a la unión de ciclos. En los casos en que no representan un único ciclo, diremos que hemos obtenido *subciclos*.

En las siguientes secciones presentaremos diferentes métodos utilizados para evitar subciclos.

3.4. Algoritmo de Unión para evitar subciclos

Del análisis de las soluciones obtenidas, se observó que, en muchas de ellas, dos subciclos distintos *pasaban por la misma esquina* y que estos podían ser evitados mediante un simple *intercambio de decisiones tomadas en la esquina* que lograra su unión en una solución equivalente, esto es, con mismo valor de función objetivo.

Un simple ejemplo puede verse en la figura 3.6, donde luego de un intercambio de arcos se logra adjuntar al recorrido principal uno de los subciclos presentes en la solución.

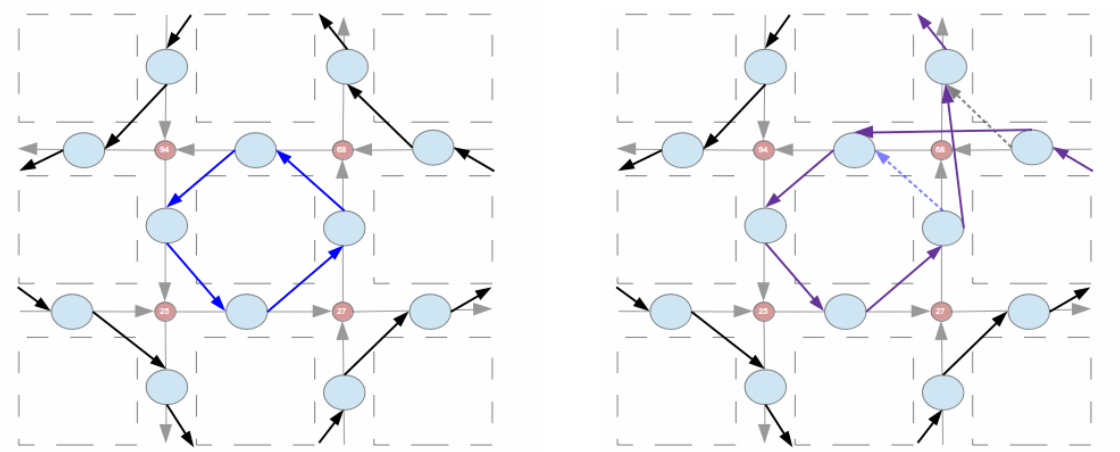


Fig. 3.6: Antes y después de una iteración del Algoritmo de Unión sobre el fragmento de zona antes presentado.

A la esquina representada por el nodo con numero 68, estamos llegando desde el este y desde el sur. Cuando llegamos desde el este, se dobla a la derecha y cuando llegamos desde el sur, se dobla a la izquierda. En la parte derecha de la figura se indica el cambio: seguir derecho en ambos casos une el subciclo a la solución integral sin cambiar las cuadras que se recorren.

Con esta idea, implementamos el algoritmo de unión de subciclos, el cual podemos ver en detalle en la sección 6.3.

Este algoritmo es muy sencillo y eficiente y resolvió muchos de los casos estudiados. Pero no siempre es posible evitar con este método los subciclos, como puede verse en el ejemplo sencillo de la figura 3.7. En esta figura se presentan el grafo G y el grafo SG_{sol} como subgrafo de SG. No se representan todas las aristas de SG para mayor claridad en la imagen.

En este caso, al detectar que dos de los subciclos pasan por la esquina v_5 (representada por los nodos v_2v_5 y v_4v_5), el algoritmo intentaría cambiar la decisión de uno de los subciclos (color verde) de ir hacia la esquina v_6 (representada por los nodos v_5v_6 y v_1v_6) por la decisión de ir hacia la esquina v_2, a la que se dirige el otro subciclo. Sin embargo esto implicaría realizar un giro en U cosa que no está permitido en el grafo SG.

De esta manera, nuevamente, resultó imprescindible modificar el modelo de manera que la condición de conexidad esté impuesto desde las restricciones.

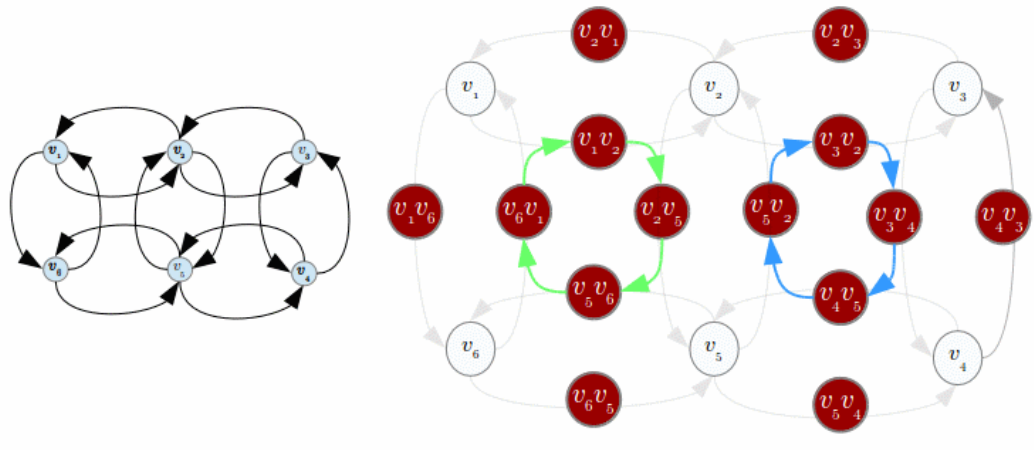


Fig. 3.7: Instancia donde *el Algoritmo de Unión* no logra obtener una única componente conexa.

3.5. Mejorando el modelo de PLE

El modelo de PLE teóricamente correcto para evitar los subciclos consiste en agregar restricciones que aseguren que la solución sea conexa. La conectividad se traduce en pedir que, cualquiera sea la subzona que yo elija, debe haber un arco que conecte este subzona con el resto de la zona. Esto, en términos del grafo requiere una restricción por cada subconjunto de vértices (asociados a cada subzona). Sea $\mathcal{P}(SV)$ el *conjunto potencia* del conjunto de *super vértices*, el conjunto de restricciones que debemos agregar al modelo 3.3 es el siguiente:

$$\sum_{(k,i)\,:\,(k,i)\in SC \wedge (i,j)\notin SC} sx_{(k,i)(i,j)} \geq 1, \quad \forall SC \in \mathcal{P}(SV)$$

Cada una de estas restricciones indica que, por cada subconjunto de vértices debe utilizarse al menos un arco con inicio en ese subconjunto (subzona) y final fuera del mismo. Las restricciones de conservación de flujo aseguran que si una solución satisface la restricción asociada a un subconjunto SV también satisface la restricción asociada a su complemento. Por esta razón, podemos sólo incluir las restricciones asociadas a subconjuntos de cardinal a lo sumo la mitad del cardinal de SV. Del mismo modo, como el grafo SG es conexo, podemos evitar las asociadas a subconjuntos de cardinal 1. A pesar de estas simplificaciones, el número de restricciones es exponencial lo cual hace que el modelo sea inmanejable a los efectos prácticos.

En estos casos, el conjunto de restricciones se incluye en la formulación como *cortes*. La idea es que no todas las restricciones serán activas en la solución óptima por lo cual sólo se incorporan las necesarias y esto se decide en forma iterativa como se detalla a continuación:

Inicio: Sea Π el programa lineal entero presentado en 3.3.

1. Se resuelve el modelo Π.
2. Si la solución obtenida no posee subciclos, ir al paso 4.
3. Se actualiza el modelo Π agregando las restricciones necesarias para evitar los subciclos presentes en la actual solución y se retorna al paso 1.
4. Se generan los archivos de salida para el recorrido óptimo hallado y se termina.

Claramente, este procedimiento converge y se puede probar que realizará a lo sumo un número de iteraciones polinomial en el tamaño del grafo SG. Sin embargo, este número puede ser grande y además cada iteración será *más dura* que la anterior ya que se resuelve un PLE más grande, debido al agregado de restricciones.

Por esta razón, se decidió integrar las dos estrategias de eliminación de subciclos como se describe en la sección siguiente.

3.6. Estrategia final de resolución

En forma sintética, la estrategia final consiste en agregar las restricciones de eliminación de subciclos presentes no en la solución brindada por la resolución del PLE sino en la solución mejorada por el algoritmo de unión de subciclos.

Así, el algoritmo utilizado (y que se puede ver en mayor detalle en el Algoritmo 8) fue el siguiente:

Inicio: Sea Π el programa lineal entero presentado en 3.3.

1. Se resuelve el modelo Π.
2. Si la solución obtenida no posee subciclos, ir al paso 6.
3. Se aplica a la solución obtenida el algoritmo de Unión de Subciclos (Algoritmo 7).
4. Si la solución obtenida no posee subciclos, ir al paso 6.
5. Se actualiza el modelo Π agregando las restricciones necesarias para evitar los subciclos presentes en la actual solución y se retorna al paso 1.
6. Se generan los archivos de salida para el recorrido óptimo hallado y se termina.

Con este algoritmo se pudieron resolver todas las instancias correspondientes a las 14 zonas de recolección del Municipio de Concordia con las que se trabajo. Para ver detalles sobre los tiempos de resolución y otros datos, ver la Tabla 7.1.

Capítulo 4

Construcción de soluciones para la Municipalidad de Concordia

En este capítulo se muestra cómo, una vez obtenidos los recorridos óptimos correspondientes a las zonas de recolección, estos se representan en distintos formatos para facilitar su consulta y uso por parte del personal de Subsecretaria de Higiene Urbana de la Municipalidad de Concordia.

4.1. Generación de archivos de salida

Cada recorrido obtenido por el programa se representa internamente como lista de pares de nodos (arcos) de la forma:

$$[(v_{inicio}, v_i), (v_i, v_j), \ldots, (v_k, v_{final})]$$

siendo v_{inicio} el vértice asociado a la esquina de inicio del recorrido y v_{final} el vértice asociado a la esquina donde se finaliza el mismo.

Claramente, la mejor forma de presentar un recorrido para su fácil implementación es en texto como una lista calles y acciones a realizar, como la siguiente:

```
Inicio: Eva Peron y Moullins
-------------------------------------------
Por Eva Peron hacer 2 cuadras, hasta Chabrillon, y girar a la derecha.
Por Chabrillon hacer 3 cuadras, hasta Hipolito Yrigoyen, y girar a la derecha.
                .
                .
                .
Por Pellegrini hacer 3 cuadras, hasta Chabrillon, y girar a la izquierda.
Por Chabrillon hacer 1 cuadra.
-------------------------------------------
Final: Hipolito Yrigoyen y Chabrillon
```

Dado que por cada arco tenemos información sobre el nombre de la calle que tiene asociado y por cada vértice tenemos información sobre su ubicación geográfica, crear una lista como la antes mencionado es realmente simple. El gran problema de este método es que los datos utilizados para construir la lista son los obtenidos del servidor de datos cartográficos OpenStreetMap y, como se mencionó en la sección 2.2, al ser un proyecto que se basa en la participación de los usuarios para llenar sus bases de datos, en algunas de las zonas, la diferencia o falta de información sobre cada calle es notoria. En consecuencia, pueden surgir listas de la siguiente forma:

```
Inicio: Lamadrid y Chavrillon
-------------------------------------------
Por Chabrillon hacer 2 cuadras, hasta -/-, y girar a la izquierda.
Por -/- hacer 4 cuadras, hasta Antonio de Luque, y girar a la derecha.
                .
                .
                .
Por -/- hacer 9 cuadras, hasta Maip, y girar a la derecha.
Por Maip hacer 3 cuadras, y girar a la derecha.
                .
                .
                .
-------------------------------------------
Final: San Lorenzo y Maipu
```

Los caracteres -/- representan una calle donde el dato asociado a su nombre no está disponible. Esto hace que el método resulte insuficiente para representar un recorrido y que se deba buscar otro para complementarlo.

Si se tuviese una imagen del grafo que representa el mapa, y el recorrido en forma de números sobre los arcos (indicando en que momento se recorre la calle asociada a este), no habría problema al no tener el dato del nombre o que exista alguna problema en este.

Para ello, dada una zona con grafo G y lista solución S, se asignó a cada elemento (v_i, v_j) de S un entero positivo n_{ij} que indica el momento en que se recorre la cuadra cuyas esquinas son las asociadas a los vértices v_i y v_j. Luego, imprimiendo esta información sobre los arcos de G, podemos obtener una imagen como la de la izquierda de la figura 4.1 que muestra solo un fragmento de la solución para una zona determinada. La imagen a la derecha de la figura se construye como una imagen auxiliar para relacionar cada arco a una determinada calle y cada vértice a una esquina. Si se colocan ambos datos sobre los arcos, (el nombre de calle y el numero que identifica el momento de recorrido) en la misma imagen, esta se torna muy cargada de información y se dificulta mucho su lectura.

En las imágenes de la figura 4.1, se utilizaron colores mas claros y con un poco de transparencia para los arcos asociados a las calles extendidas de la zona. En la imagen de la izquierda, el número 0 sobre los arcos indica que dicho arco no fue utilizado por la solución.

Si bien este es el formato más simple de implementar y compartir, dado el tamaño de las zonas sobre las que se trabajó, puede resultar muy compleja la consulta a las imágenes asociadas a sus recorridos.

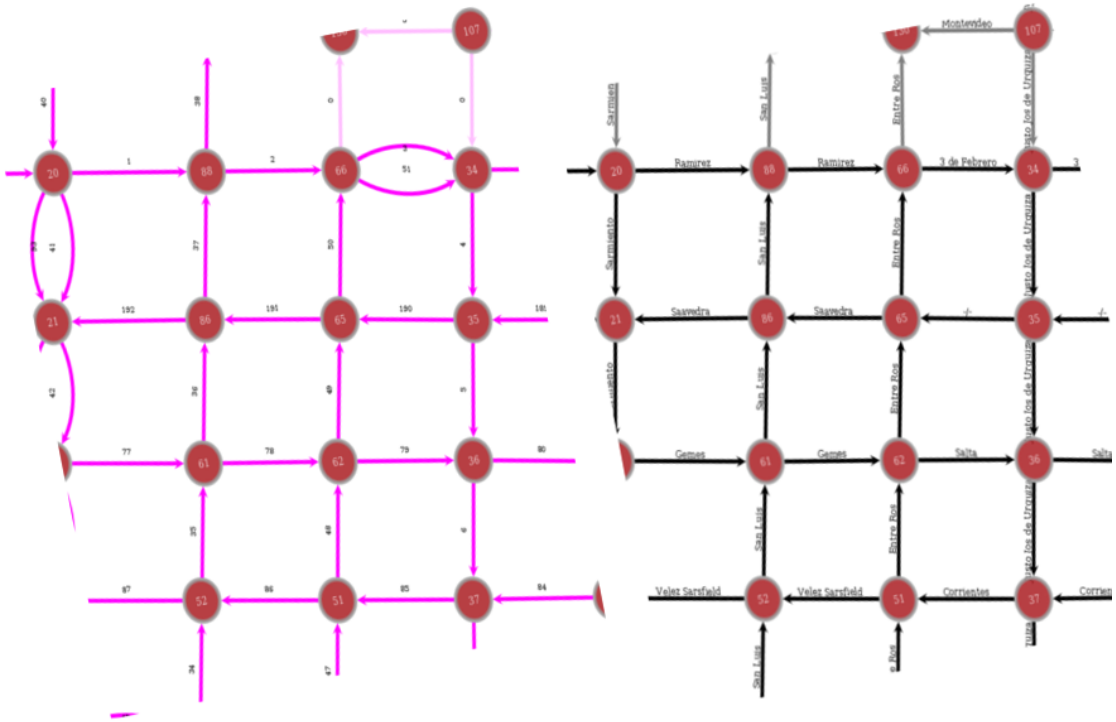


Fig. 4.1: A la derecha, fragmento del grafo que representa una determinada zona. A la izquierda, grafo asociado a un fragmento del recorrido óptimo para la zona.

Se decidió entonces, crear a partir de los recorridos óptimos, archivos compatibles con dispositivos GPS. Cargando estos archivos en algún sistema de navegación presente en los camiones recolectores, el personal encargado de la conducción solo tendría que seguir el rumbo indicado por el sistema.

Para esto se hizo uso del formato GPX o GPS eXchange Format (Formato de Intercambio GPS), un esquema XML el cual establece un mecanismo estándar para el intercambio y almacenamiento de información de mapas en dispositivos GPS, teléfonos inteligentes y computadoras.

Se puede utilizar para describir un conjunto de puntos (waypoints), recorridos (tracks), y rutas (routes). Los *puntos de ruta* son sólo un conjunto de ubicaciones sin orden predeterminado. Un *recorrido* está formado por una sucesión de puntos, normalmente próximos entre sí, que representan el camino que hay entre un punto de partida y otro de llegada. Una *ruta* es el camino en línea recta que hay que seguir para ir desde un Waypoint a otro y desde éste al siguiente y así sucesivamente hasta alcanzar el último punto.

Dado que un recorrido de recolección se puede describir como un *track* donde la secuencia ordenada de locaciones es el conjunto de esquinas que compone el recorrido, podemos generar un archivo GPX por cada uno de ellos y, de esta manera, una vez que los camiones dispongan de la tecnología necesaria, utilizar estos para que la descripción del recorrido se efectue de manera automática a medida que este se va realizando.

Un archivo GPX que describe el recorrido de una determinada zona de recolección tiene la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx ...>
    <trk>
        <trkseg>
            <trkpt lat="-31.378498" lon="-58.0135846">
            </trkpt>
            <trkpt lat="-31.3793551" lon="-58.0137747">
            </trkpt>
            <trkpt lat="-31.3802254" lon="-58.0139678">
            </trkpt>
                .
                .
                .
            <trkpt lat="-31.3803942" lon="-58.0129615">
            </trkpt>
        </trkseg>
    </trk>
</gpx>
```

Donde la etiqueta *trk* indica que el archivo estará compuesto por una lista de recorridos, cada uno de ellas asociado a una instancia de *trkseg*, la cual representa una secuencia de locaciones (particularmente, dado que creamos un archivo por cada recorrido, en cada uno de ellos habrá una única instancia de este tipo de datos). La etiqueta *trkpt* se utiliza para representar cada una de las locaciones pertenecientes a la secuencia, en nuestro caso, se utilizará para representar cada una de las esquinas por las que pasa el recorrido calculado.

Como se puede ver, solo es necesario crear una instancia del tipo de datos *trkpt* para cada uno de los elementos que pertenecen a la lista solución, es decir, si la lista tiene la forma $[(v_s, v_i), (v_i, v_j), \ldots, (v_k, v_t)]$, entonces crearemos una instancia del tipo *trkpt* para los nodos v_s, v_i, v_j, $\ldots$,v_k y v_t. Los únicos datos que necesitamos al momento de crear la instancia son la latitud y longitud del punto que representa la esquina asociada al nodo, ambos disponibles como se muestra en la sección 6.1.

El problema aquí es que los camiones que se utilizan para la recolección en el Municipio de Concordia aun no cuentan con ningún sistema de navegación al que se le puedan cargar los archivos asociados a los recorridos óptimos y sea el encargado de guiar al conductor (se nos comentó el interés por la incorporación de algunos dispositivos GPS que se podrían usar para este propósito).

En consecuencia, se decidió a partir de los archivos GPX y utilizando la aplicación *GPX Animator*[1], generar animaciones de los recorridos en formato de vídeo, las cuales pueden ser vistas y consultadas desde cualquier dispositivo que solo cumpla con la característica de reproducir vídeo.

Además, GPX Animator nos permite utilizar como fondo para el recorrido mapas de cualquier servidor publico TMS[2], como *CloudMade*, *MapQuest* o *OpenStreetMap*, por lo que las animaciones obtenidas resultan muy fáciles de comprender.

[1] http://zdila.github.io/gpx-animator/

[2] `https://en.wikipedia.org/wiki/Tile_Map_Service`

Se puede descargar una de las animaciones asociadas a los recorridos óptimos en `https://www.dropbox.com/s/eep0cev3h5xnx5z/r15.mp4?dl=0` mientras que en el Apéndice A se muestran varias capturas a dicha animación.

Finalmente, haciendo uso de la libreria WxPython[3] se implementó una herramienta que consiste en una interfaz de usuario para la consulta de los resultados y la exportación de los recorridos obtenidos a los distintos formatos que fuimos mencionando a lo largo de este capítulo. La idea de esta herramienta es la clasificación de toda la información para el fácil acceso a la misma por parte del personal de la Subsecretaría de Higiene Urbana.

En la figura 4.2 podemos ver una captura de la aplicación desarrollada. A la izquierda, las zonas agrupadas en forma de árbol según el momento del día en que se realiza la recolección sobre ellas. Una vez que se selecciona algún ítem del mas bajo nivel del árbol (una zona), se detallan los barrios que la componen, se resaltan las calles de la mísma sobre el mapa de la derecha y se habilitan las opciones que posibilitan la obtención de su recorrido óptimo asociado.

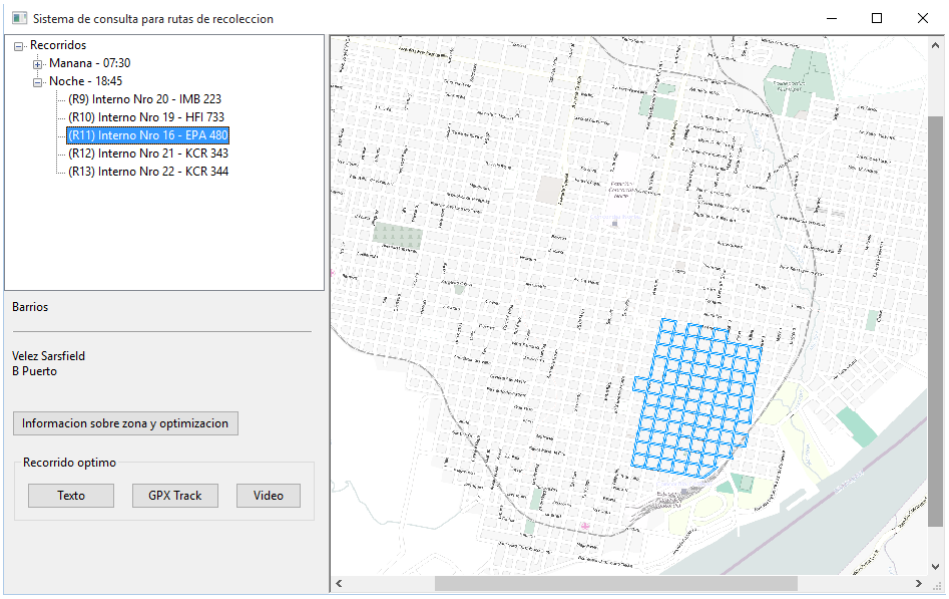


Fig. 4.2: Captura de herramienta desarrollada para la consulta de los recorridos calculados.

[3] `http://www.wxpython.org/`

Capítulo 5

Minimizando giros en los recorridos

En este capítulo abordamos el problema de obtener, entre todos los órdenes posibles para recorrer las aristas de un circuito euleriano, aquel que minimice la cantidad de giros realizados. Reducimos este problema al problema de Viajante de Comercio (TSP) y resolvemos el mísmo sobre las instancias correspondientes a las zonas de recolección de residuos de la ciudad de Concordia.

5.1. Introducción

Una vez finalizado el trabajo asociado al convenio de colaboración entre el Instituto del Calculo y el Ministerio del Interior, observamos que, en general, las rutas resultantes realizaban un gran número de giros en las esquinas.

Ya obtenidas las soluciones óptimas en kilómetros recorridos, las cuales vienen expresadas en términos de qué cuadras deberán ser recorridas adicionalmente a aquellas obligatorias donde se debe realizar la recolección, construimos un grafo euleriano (G_{sol} o SG_{sol}, dependiendo de con cuál modelo estemos trabajando). Sin embargo, existe mas de un circuito euleriano en el grafo.

En la figura 5.1 se muestra un ejemplo de cómo diferentes circuitos eulerianos sobre el mismo grafo euleriano, determinan diferentes cantidades de giros y giros en U. Los números sobre las aristas indican el orden en que son recorridos.

En el primer recorrido, la cantidad de giros es de 11 y contiene además, 1 giro en U (el cual, recordamos, consideramos prohibido). En el segundo recorrido, se reduce la cantidad de giros a 9 y no hay giros en U.

Esto lleva a plantearnos la posibilidad de obtener, entre todas las formas de recorrer el circuito euleriano, aquella que minimizara la cantidad de giros.

Para ello, es necesario asociar un costo a cada decisión respecto a *cuál es la cuadra siguiente a recorrer*. Si la cuadra siguiente *sigue derecho* (misma calle por ejemplo), no deberá ser penalizada. En cambio si *dobla* (cambia de calle) deberíamos poner un costo

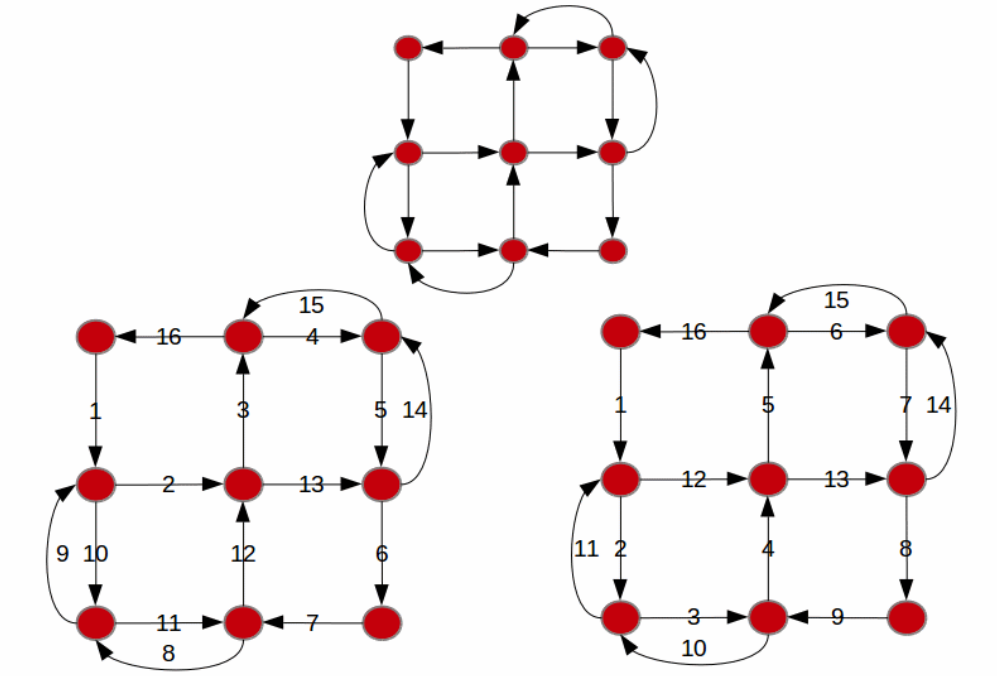


Fig. 5.1: Distintos órdenes de recorridos de un grafo euleriano.

significativo. O sea, la función objetivo a minimizar debía estar relacionada con el ángulo que formaban una cuadra y su siguiente. Obsérvese que este mismo criterio podría haberse utilizado con las soluciones originales donde no modificábamos el grafo para prohibir los giros en U, penalizando fuertemente cuando el ángulo entre una cuadra y la siguiente fuera cercano a 180 grados.

Esto motivó el estudio de este nuevo problema y su aplicación a las soluciones obtenidas para el municipio de Concordia en las distintas etapas del trabajo.

5.2. Minimización de Giros y el TSP

El nuevo problema a estudiar tiene como entrada un grafo euleriano que vamos a suponer dirigido ya que las instancias correspondientes a las zonas de recolección de residuos en Concordia fueron así modeladas. Presentamos a continuación algunas definiciones asociadas a grafos dirigidos que serán de ayuda para plantear más claramente el nuevo problema.

Dado un grafo dirigido $G = (V, A)$, decimos que un par ordenado de arcos a_1, a_2 son *consecutivos* si existe un $v \in V$ tal que $a_1 \in \Gamma_-(v)$ y $a_2 \in \Gamma_+(v)$, es decir, si el final de a_1 coincide con el origen de a_2. Llamamos $\mathcal{C}(G)$ al conjunto de pares de arcos consecutivos en G.

Supongamos $G = (V, A)$ euleriano y que los elementos de A están indexados de la forma $\{a_1, \ldots, a_m\}$. Una elección de recorrido del grafo euleriano de G puede verse como una permutación $\pi : \{1, \ldots, m\} \to \{1, \ldots, m\}$ de manera que $a_{\pi(i)}$ es el arco que se recorre en el i-ésimo lugar. Claramente, no toda permutación representa maneras posibles

de recorrer el grafo euleriano, es condición necesaria y suficiente que $(a_{\pi(i)}, a_{\pi(i+1)}) \in \mathcal{C}(G)$ para todo $i = 1, \ldots, m-1$ y $(a_{\pi(m)}, a_{\pi(1)}) \in \mathcal{C}(G)$. Llamamos $\mathcal{P}(G)$ al conjunto de todas las permutaciones π que verifican esta condición.

Para evaluar *el costo en giros* de una permutación $\pi \in \mathcal{P}(G)$, se define una función costo $f : \mathcal{C}(G) \to \mathbf{R}$ tal que para cada par de arcos consecutivos (a_1, a_2), $f(a_1, a_2)$ representa el costo asociado a pasar de la arista a_1 a la arista a_2. El costo de una forma de recorrer el grafo, es decir, el costo de $\pi \in \mathcal{P}(G)$ lo notamos $\zeta(\pi)$ y se define como la suma de los costos de sus *cambios de cuadras*, esto es,

$$\zeta(\pi) = f(a_{\pi(m)}, a_{\pi(1)}) + \sum_{i=1}^{m-1} f(a_{\pi(i)}, a_{\pi(i+1)}).$$

Con estas definiciones, el nuevo problema a analizar puede ser formalizado de la siguiente manera:

Problema de recorrido euleriano de costo mínimo en giros (PRECMG)

ENTRADA: $G = (V, A)$ un digrafo euleriano, $f : \mathcal{C}(G) \to \mathbf{R}$

PROBLEMA: determinar $\pi \in \mathcal{P}(G)$ tal que $f(\pi)$ sea mínimo.

Dada una instancia del PRECMG definida por un digrafo euleriano $G = (V, A)$ y una función de costos de giros $f : \mathcal{C}(G) \to \mathbf{R}$, construimos un digrafo $\tilde{G} = (\tilde{V}, \tilde{A})$ donde $\tilde{V} = A$ y $(a_i, a_j) \in \tilde{A}$ si y sólo si $(a_i, a_j) \in \mathcal{C}(G)$ y definimos un costo c sobre los arcos de $\tilde{G}$ de manera que $c(a_i, a_j) = f(a_i, a_j)$. En la Figura 5.2 se muestra un ejemplo de esta transformación asociada al grafo euleriano de la derecha.

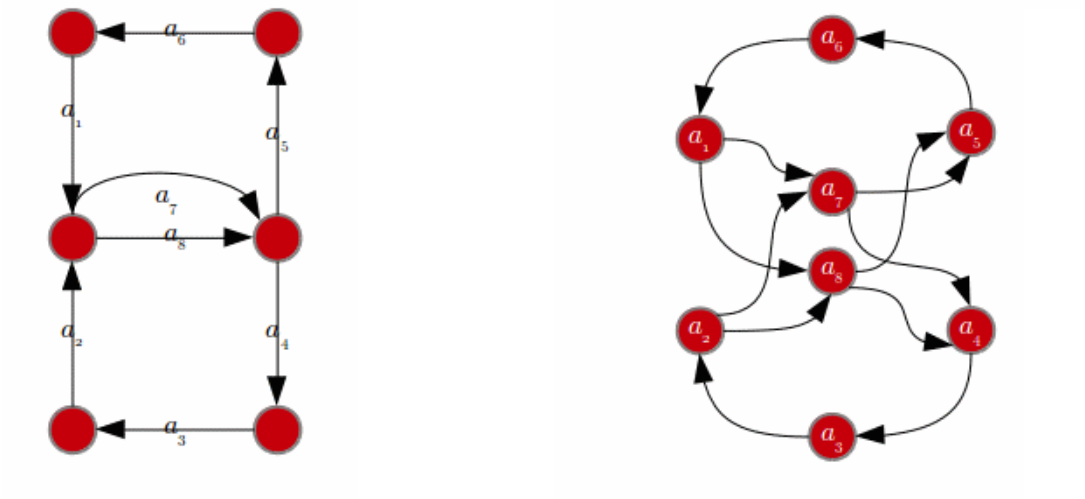


Fig. 5.2: A la izquierda, el digrafo eureliano $G = (V, A)$ y su función de costos sobre giros, f. A la derecha, el digrafo resultado de la transformación, $\tilde{G} = (\tilde{V}, \tilde{A})$, y su función de costos sobre arcos, c.

De la definición de $\mathcal{P}(G)$ se deriva fácilmente que para todo $\pi \in P(G)$, existe un circuito hamiltoniano $H(\pi)$ de $\tilde{G}$ tal que $f(\pi) = c(H(\pi))$ y recíprocamente, para todo circuito hamiltoniano H de $\tilde{G}$, existe un $\pi(H) \in \mathcal{P}(G)$ tal que $f(\pi(H)) = c(H)$.

Por lo tanto, el PRECMG puede reducirse polinomialmente al TSP dirigido.

Esto nos permite resolver cualquier instancia del PRECMG como una instancia del TSP dirigido, problema para el cual se cuenta con la poderosa herramienta de resolución Concorde (mencionada en la Sección 1.5). Si bien el Concorde resuelve instancias del TSP simétrico (o no dirigido), existen transformaciones polinomiales del TSP dirigido al simétrico.

En la Sección 6.4 se muestran los detalles técnicos.

5.3. Resolviendo las instancias de Concordia

Como fue mencionado, la motivación principal que llevó al estudio de este nuevo problema fue la de obtener formas de recorrer las soluciones óptimas en términos de longitud total del recorrido para las zonas de recolección de Concordia, de manera de minimizar los giros en las esquinas.

En esta sección detallamos cómo construimos instancias de PRECMG a partir de los recorridos óptimos obtenidos para las zonas de recolección.

Claramente, cada zona definirá una instancia del PRECMG donde el digrafo $G = (V, A)$ será G_{sol} o SG_{sol} dependiendo de si trabajamos con las soluciones del Modelo 3.1 o del Modelo 3.3.

Para la definición de la función objetivo tendremos en cuenta el ángulo que forman una cuadra y su consecutiva en el circuito.

Como las cuadras se definen por pares de esquina y cada esquina está representada internamente por sus dos coordenadas geográficas latitud y longitud, podemos encontrar las componentes del vector que definen un inicio y un final de cuadra. Esto es, si cada esquina i tiene coordenadas (x_i, y_i), la cuadra a_1 que se recorre de la esquina 1 a la esquina 2 se representa por el vector $\vec{a}_1 = (x_2 - x_1, y_2 - y_1)$. Si después de recorrer esta cuadra, pasamos a la cuadra a_2 que empieza en la esquina 2 y termina en la esquina 3, esta nueva cuadra está representada por el vector $\vec{a}_2 = (x_3 - x_2, y_3 - y_2)$. El ángulo que forman estos dos vectores puede caracterizarse por el valor del coseno del mísmo, que sabemos puede obtenerse realizando el producto escalar entre $\vec{a}_1$ y $\vec{a}_2$ normalizados al dividirlos por sus módulos.

Así, si $\vec{a}_1$ y $\vec{a}_2$ no cambian de dirección, el valor del coseno será 1 mientras que si se realiza un giro de 90 grados, será 0 y en caso de girar en U, resultará -1.

Como la función objetivo penaliza los giros, el costo de recorrer a_2 inmediatamente después de a_1 deberá ser una función no decreciente en función de $1 - cos(\hat{a_1, a_2})$.

En la figura 5.3 se puede ver gráficamente el comportamiento de la función coseno dependiendo el ángulo que forma la esquina intersección de las cuadras a_1 y a_2.

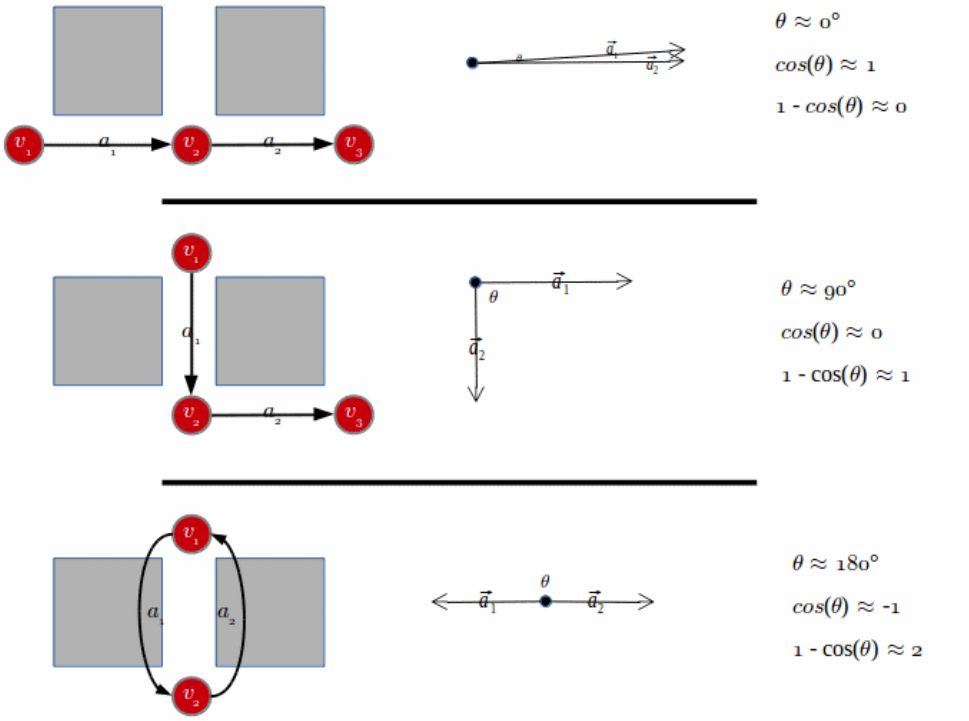


Fig. 5.3: Valor de la función coseno aplicada a tres posibles ángulos formados por intersección de las cuadras a_1 y a_2.

Capítulo 6

Detalles de la implementación

En este capítulo se presentan los detalles técnicos acerca de la implementación de los procedimientos y algoritmos que fueron utilizados a lo largo de esta tesis. Para esto, clasificaremos cada uno de ellos según su objetivo específico, dentro de uno siguientes grupos:

- Lectura del archivo OSM asociado a cada zona de recolección y almacenamiento de los datos en estructuras internas.
- Implementación y resolución de los modelos de PLE.
- Técnicas y algoritmos para evitar giros en U y subciclos.
- Minimización de giros.

Y en cada una de las primeras cuatro secciones del capítulo se abordará uno de estos aspectos.

En la ultima sección se describirá el Algoritmo Final propuesto para la resolución del Problema del Cartero Rural en instancias asociadas a zonas urbanas, el cual integra los algoritmos enunciados y detallados en las secciones previas.

6.1. Lectura de archivos OSM y almacenamiento de datos

Como se dijo en la Sección 2.1, los archivos OSM están compuestos por dos listas, una de instancias del tipo de datos *node* y otra de instancias del tipo de datos *way*, donde cada una de ellas se corresponde, respectivamente, a un conjunto de esquinas y elementos y a un conjunto de calles.

Cada instancia *node* está compuesta por la siguiente información:

- **id**: Identificador único asociado

- **lat**: Coordenadas de latitud en grados
- **lon**: Coordenadas de longitud en grados
- **tags**: Conjunto de pares "Clave-Valor"

Y un ejemplo de estas podría ser:

```
<node id="25496583" lat="51.5173639" lon="-0.140043" ...>
    <tag k="highway" v="traffic_signals"/>
</node>}
```

En este caso, el conjunto de pares "clave-valor" está compuesto solamente por el par

```
"tag k=highway" v="traffic_signals"
```

que indica que dicha instancia esta asociada a un semáforo.

Otro tipo de información que se vuelca en estos pares podría ser el de lugares como escuelas, bares, puntos turísticos, estacionamientos para bicicletas, etc.

Cuando una instancia está relacionada a una esquina, el conjunto de pares "clave-valor" es vacío.

Por otro lado, las instancias de *way* contienen la siguiente información:

- **id**: Identificador único asociado
- **nds**: Lista de nodos ordenados que componen el segmento
- **tags**: Conjunto de pares "Clave-Valor"

A continuación, una instancia de *way* que representa un segmento (compuesto por una o varias calles) y donde uno de los puntos que forma este segmento es la instancia de *node* del ejemplo anterior.

```
<way id="5090250" ...>
  <nd ref="25496583"/>
  .
  .
  .
  <tag k="highway" v="residential"/>
  <tag k="name" v="Clipstone Street"/>
  <tag k="oneway" v="yes"/>
</way>
```

Para identificar las restricciones de circulación de cada calle se utiliza la información que contiene la instancia *way* asociada en sus pares clave-valor.

En la figura 6.1 podemos ver como:

- Una calle con sentido único de circulación tiene asociada: una instancia de *way* donde la clave *oneway* tiene valor *yes* y un arco en el grafo mixto, es decir un elemento de A.
- Una avenida o boulevard tiene asociada: una instancia de *way* donde la clave *oneway* tiene valor *no* y la clave *lanes* tiene valor *2* y dos arcos (sobre los mismos nodos pero con distinta dirección) en el grafo mixto, es decir dos elementos de A.
- Una calle con sentido doble de circulación tiene asociada: una instancia de *way* donde la clave *oneway* tiene valor *no* y un arista en el grafo mixto, es decir un elemento de E.

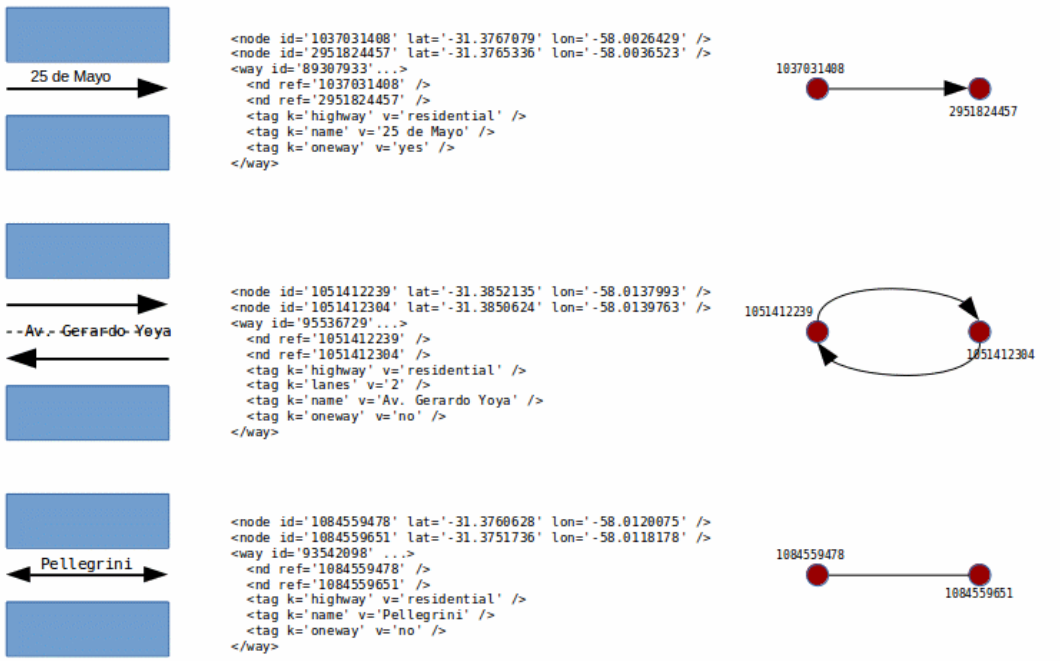


Fig. 6.1: Asociación entre distintos fragmentos de código OSM y su representación en un grafo.

A partir de los datos de estos archivos y haciendo uso de la librería *graph-tool*[1], construimos un grafo mixto $G = (V, E, A)$ donde cada elemento de V se corresponde con una instancia de *node*, cada elemento de E se corresponde a una instancia *way* que represente una calle con doble sentido de circulación y cada elemento de A se corresponde con una instancia *way* que represente una calle con un único sentido de circulación.

Por cada elemento de V, es decir cada esquina, se almacena un identificador único de nodo y los datos relacionados a su ubicación geográfica: las coordenadas de latitud y longitud. Toda esta información se obtiene de manera directa del fragmento de código OSM asociado.

Para cada elemento de E o A, es decir cada calle, se almacena un identificador único de calle, su nombre, los identificadores de sus nodos inicio y final, la cantidad de manos que

[1] Librería para el lenguaje de programación Python la cual permite la creación y manipulación eficiente de grafos, https://graph-tool.skewed.de/

posee, su sentido de circulación (simple o doble), su longitud, si se trata de una calle sin salida y si es una calle *calle extendida*, esto es, una calle que pertenece a la zona extendida pero no a la original (ver Sección 2.3).

Algunos de esos valores se obtienen de manera directa de la información contenida en el fragmento de código OSM asociado y otros requieren de cierto procesamiento sobre la mísma, como veremos en detalle a continuación.

- Cálculo de Longitudes:

  Para calcular la longitud del segmento formado por las esquinas e_1 y e_2, con sus respectivas coordenadas geográficas (en grados) (lat_{e_1}, lon_{e_1}) y (lat_{e_2}, lon_{e_2}), se utiliza el Algoritmo 1, el cual se basa en la fórmula matemática para medir distancias sobre el planeta Tierra. [2].

Algoritmo 1: Longitud del Segmento

Entrada: (lat_1, lon_1) y (lat_2, lon_2), coordenadas geográficas, en grados, asociadas a los puntos e_1 y e_2.

Salida : Distancia en kms del segmento limitado por los puntos e_1 y e_2.

$R \leftarrow 6371$ `// radio de la tierra en kilómetros`
$grados\_a\_radianes \leftarrow \pi/180$

$\phi_1 \leftarrow (90 - lat_1) * grados\_a\_radianes$ `// latitud de` e_1 `en radianes`
$\phi_2 \leftarrow (90 - lat_2) * grados\_a\_radianes$ `// latitud de` e_2 `en radianes`

$\lambda_1 \leftarrow lon_1 * grados\_a\_radianes$ `// longitud de` e_1 `en radianes`
$\lambda_2 \leftarrow lon_2 * grados\_a\_radianes$ `// longitud de` e_2 `en radianes`

$\Delta\lambda \leftarrow \lambda_1 - \lambda_2$

`// Dados dos puntos por sus coordenadas esféricas` $(1, \phi_1, \lambda_1)$ `y` $(1, \phi_2, \lambda_2)$`, el coseno de la longitud del arco formado por la conexión de los puntos es`
$cos \leftarrow (\sin(phi_1) * \sin(phi_2) * \cos(\lambda_1 - \lambda_2) + \cos(\phi_1) * \cos(\phi_2))$

`// Y la longitud del arco (en kms) es`
$longitud \leftarrow R * \arccos(cos)$

return *longitud*

- Identificación de calles sin salida:

  Todas las calles sin salida poseen doble sentido de circulación (esto es obvio ya que una vez que se entra, para salir, se debe utilizar la misma calle en el sentido contrario), luego, cada una de estas será vinculada con un elemento del conjunto E, es decir con una arista.

  Desde la Subsecretaria de Higiene nos informaron que a la calles sin salida el camión recolector no ingresa y espera en la esquina que los empleados realicen la recolección

[2] `http://www.johndcook.com/lat_long_details.html`

a pie y la lleven al camión. Por lo tanto, decidimos eliminar estas aristas de los grafos asociados a las zonas y así, no tener en cuenta las calles sin salida al momento de la optimización.

El Algoritmo 2, se utiliza para determinar y eliminar las aristas de un grafo mixto $G = (V, E, A)$ que se corresponden con una calle sin salida.

Algoritmo 2: Eliminar Calles Sin Salida

Entrada: E, conjunto de aristas del grafo G.

Salida : E, conjunto de aristas del grafo G.

for $e \in E$ **do**

$nodo\_extremo_1 \leftarrow verticeInicio(e)$
$nodo\_extremo_2 \leftarrow verticeFinal(e)$

if $(gradoEntrada(nodo\_extremo_1) == gradoSalida(nodo\_extremo_1) == 1) \ \vee \ (gradoEntrada(nodo\_extremo_2) == gradoSalida(nodo\_extremo_2) == 1)$ **then**

$E \leftarrow E \setminus \{e\}$

return E

- Identificación de calles *extendidas*:

  Finalmente, el algoritmo 3, se utiliza para determinar si un arco o arista representa una calle extendida.

Algoritmo 3: Determinación Calle Extendida

Entrada: $G = (V, E, A)$ grafo mixto de la zona original, $G_{ext} = (V_{ext}, E_{ext}, A_{ext})$ grafo mixto de la zona extendida y e elemento del conjunto $(E_{ext} \cup A_{ext})$.

Salida : *Verdadero*, si se trata de una calle sin salida; *Falso*, en caso contrario.

$calles\_zona\_original = \{a : a \in (E \cup A)\}$;
$calles\_zona\_extendida = \{a : a \in (E_{ext} \cup A_{ext})\}$;

if $e \in (calles\_zona\_extendida \setminus calles\_zona\_original)$ **then**
return Verdadero
else
return Falso

6.2. Implementación y resolución de los modelos de PLE

Los modelos de PLE presentados en las secciones 3.1 y 3.3 fueron implementados en ZIMPL [23], un lenguaje para traducir el modelo matemático de una problema de programación PLE a archivos con formato .lp o .mps, los cuales pueden ser leídos y resueltos por la mayoría de los optimizadores disponibles.

Para la implementación del modelo se requiere de los conjuntos A, E y E_{ext} y las variables $nodo_{inicio}$ y $nodo_{final}$ que representan, respectivamente, la esquina por donde se

comienza y finaliza el recorrido. A modo de ejemplo, a continuación presentamos el archivo ZIMPL con el cual se ingresa el Modelo 3.1.

```
# Conjuntos
set Vertices := { 0 .. |V_ext};
set Arcos              := {read A as <1n,2n>};
set Aristas            := {read E as <1n,2n>};
set Aristas_Extendidas := {read E_ext \setminus E as <1n,2n>};
set Arcos_y_Aristas := Arcos union Aristas union Aristas_Extendidas

# Variables
var x[Arcos_y_Aristas] binary;

# Funcion de costo
param d[<i,j> in Arcos_y_Aristas] := read C as \"<1n,2n> 3n\" default 0;"

# Funcion objetivo
minimize fobj: sum <i,j> in Arcos_y_Aristas: (d[i,j] * x[i,j]);

# Restricciones

# Cada arista debe ser recorrida en alguna de sus direcciones

subto rest1:

forall <i,j> in Aristas with i < j: x[i,j] + x[j,i] >= 1;

# Cada arco debe ser recorrido al menos una vez

subto rest2:

forall <i,j> in Arcos: x[i,j] >= 1;

# Para todo vertice el grado de entrada debe ser igual al de salida.

subto rest4:

forall <i> in V: (sum <j> in V with <j,i> in Arcos_y_Aristas : x[j,i]) ==
                 (sum <j> in V with <i,j> in Arcos_y_Aristas : x[i,j]);

# Restricción para todos los arcos y aristas

subto rest5:

forall <i,j> in Arcos_y_Aristas: x[i,j] >= 0;

# Restricción especial para el arco ficticio (nodo_fin, nodo_inicio)
```

```
subto rest6:

x[nodo_fin, nodo_inicio] = 1;
```

El archivo ZIMPL correspondiente al Modelo 3.3 es similar.

Una vez resueltos los modelos por el solver, obtenemos los grafos eulerianos G_{sol} o SG_{sol} sobre los cuales es necesario encontrar un orden de recorrido de sus arcos. Tal como fue mencionado, para esto se utilizó el Algoritmo de Hierholzer [6], disponible en la librería de grafos utilizada.

Llamemos G al grafo sobre el cual aplicamos el algoritmo (que puede corresponder a G_{sol} o a SG_{sol}). La idea de este algoritmo es construir circuitos de arcos disjuntos en G, los cuales comparten vértices entre si. Cuando esos circuitos son unidos de manera adecuada, forman un circuito euleriano en G. A continuación presentamos el pseudocódigo correspondiente.

Algoritmo 4: Algoritmo de Hierholzer

Entrada: $G = (V, A)$ grafo y v_{inicio} nodo de G por el cual se desea iniciar el circuito.

Salida : Una lista ordena de los elementos de A.

1 Construir un circuito C_1 de G que comience por el nodo v_{inicio}. Marcar todas las aristas que componen C_1 y hacer $i = 1$.

2 Si C_1 contiene todos los arcos de A, entonces paramos y retornamos el circuito C_1.

3 Si C_1 no contiene todos los arcos de A, entonces sea v_i un nodo de C_1 incidente a alguna arista no marcada e_i.;

4 Construir un circuito Q_i que comience por el nodo v_i y use la arista e_i. Marcar las aristas de Q_i.;

5 Construir un nuevo circuito C_{i+1} uniendo el circuito Q_i a C_1 por el nodo v_i.;

6 Incrementar i en una unidad y volver al paso **2**.

Finalmente, una vez obtenido el circuito euleriano C de G, el cual comienza (y finaliza) en v_{inicio}, y recordando el arco ficticio, (v_{fin}, v_{inicio}), agregado en el grafo y obligado a estar presente una única vez en el recorrido óptimo; siendo C una lista de pares de vértices (arcos), tendrá la forma:

$$[(v_{inicio}, v_i),\ (v_i, v_j),\ \ldots,\ (v_k, v_{fin}),\ (v_{fin}, v_{inicio}),\ (v_{inicio}, v_l),\ \ldots,\ (v_m, v_{inicio})].$$

Tras eliminar el elemento correspondiente al arco ficticio y realizar una simple manipulación en el orden de la lista, obtenemos el camino buscado, es decir, un camino euleriano que comienza su recorrido en el nodo v_{inicio} y finaliza en el nodo v_{final}:

$$[(v_{inicio}, v_l),\ \ldots,\ (v_m, v_{inicio}),\ (v_{inicio}, v_i),\ (v_i, v_j),\ \ldots,\ (v_k, v_{fin})].$$

6.3. Evitando giros en U y sub-ciclos

En esta sección, veremos en detalle los algoritmos y técnicas utilizados para evitar los dos grandes problemas que surgen luego de interpretar las soluciones correspondientes a los dos modelos de PLE presentados.

De los caminos eulerianos obtenidos por el Algoritmo de Hierholzer sobre G_{sol} (solución del Modelo 3.1) surgió el primer gran problema, el de los *giros en U*. Para evitar este tipo de giros, se comenzó implementando el *Algoritmo de Mejora* descripto en la Sección 3.2.1. El detalle del mismo se presenta en el Algoritmo 5.

Algoritmo 5: Algoritmo de Mejora

Entrada: Camino eureliano solución como una lista ordenada, C, de pares de vértices.
Salida : Nuevo camino eureliano en la lista ordenada C

```
// Buscamos un giro en U.
for i ← 1 to |C| do
    (u, v) ← C[i]
    (v, w) ← C[i + 1]
    if u == w then
        // Buscamos un arco alternativo (v,_) por donde seguir luego de
           v y así evitar el giro en U
        for j ← i + 1 to |C| do
            (v_aux, w_aux) ← C[j]
            if v_aux == v then
                // Buscamos un arco (_,v) que, luego de tomado el arco
                   alternativo, vuelva a v para, ahí sí, volver a u
                for k ← j + 2 to |C| do
                    (s, v_aux) ← C[k]
                    // Si encontramos ambos, colocamos el circuito
                       (v,_)...(_,v) entre los arcos (u,v)(v,u)
                    if v_aux == v then
                        aux_tour ← C[j : k]
                        Se modifica C de manera que la sublista aux_tour se ubique
                        entre las posiciones i y i + 1
return C
```

Como se vió en los ejemplos de la Sección 3.2.1, el Algoritmo de Mejora puede fallar y la solución definitiva para este problema fue la de trabajar con el Modelo 3.3.

A partir de las soluciones obtenidas de este nuevo modelo se construye SG_{sol} donde aparece el otro problema, el de las estructuras llamadas *subciclos*.

Como vimos, esto puede solucionarse incluyendo en el modelo de PLE las restricciones que aseguran la conexidad de SG_{sol}. Si bien estas restricciones involucran un número exponencial de desigualdades, las mismas pueden manejarse incorporándolas al modelo

iterativamente, sólo cuando sean necesarias, a través de lo que se denomina Algoritmo de Cortes. Este algoritmo añade las restricciones al modelo de PLE a medida que aparecen subciclos en la solución. En el Algoritmo 6 presentamos los detalles de su implementación.

Algoritmo 6: Algoritmo de Cortes

Entrada: Grafo SG asociado a una zona de recolección.

Salida : Lista de pares de nodos que representa un circuito.

```
1  Resolver modelo de PLE asociado a SG
2  if Se encontró una solución S then
3      G_sol ← CrearMultiDiGrafoSolucion(S)
4      sub_ciclos ← SubCiclos(G_sol)
5      if |sub_ciclos| > 1 then
6          for sc ∈ sub_ciclos do
7              Agregar al modelo las restricciones necesarias para evitar subciclo sc.
8          Volver a la linea 1
9      else
10         return Primer elemento de la lista subciclos
11 else
12     return No se encontró solución.
```

Este método puede resultar poco eficiente ya en cada iteración resuelve un PLE y en determinadas zonas el número de iteraciones necesarias resulta muy alto, llevando el tiempo de cálculo a valores poco razonables.

La forma de eludir este inconveniente fue pre-procesando SG_{sol} con el Algoritmo de Unión de Subciclos. La idea es realizar un intercambio de superarcos en dos subciclos distintos que *pasen por la misma esquina* con el fin de unir los subciclos en un único circuito.

El algoritmo 7 describe en detalle este algoritmo.

La combinación de estos dos procedimientos se presenta en lo que denominamos Algoritmo Integrado y cuyo detalle es presentado en el Algoritmo 8.

Para la rutina encargada de obtener los subciclos en un digrafo, $SubCiclos()$, se utilizo la implementación de una versión mejorada del algoritmo de Tarjan[20] presentada por Nuutila y Soisalon-Soinen en 1994[16], disponible en la librería de grafos con la que se trabajó. Este algoritmo se utiliza para encontrar componentes fuertemente conexas en digrafos: en detalle, dado un digrafo G, retorna una lista de nodos por cada componente fuertemente conexa de G, es decir, una lista de listas de pares de nodos.

Algoritmo 7: Unión de Subciclos

Entrada: Lista *sub_ciclos* donde cada elemento representa un subciclo como una lista de pares de nodos y grafo a partir del cual se obtiene los subciclos, $G_{sol} = (V_{sol}, A_{sol})$.

Salida : Lista *sub_ciclos* donde cada elemento representa un subciclo como una lista de pares de nodos.

```
1  while |sub_ciclos| > 1 do
2      if Existen supernodos (u,v) ∈ C_i y (w,v) ∈ C_j con C_i y C_j subciclos distintos
       then
3          Salientes_(u,v) ← {Arcos de la forma ((u,v),(v,r)) usados en el subciclo C_i}
4          Salientes_(w,v) ← {Arcos de la forma ((w,v),(v,p)) usados en el subciclo C_j}
5          for ((u,v),(v,r)) ∈ Salientes_(u,v) do
6              for ((w,v),(v,p)) ∈ Salientes_(w,v) do
7                  if Existen superarcos ((u,v)(v,p)) y ((w,v)(v,r)) then
8                      A_sol ← (A_sol \ {((u,v),(v,r))}) ∪ {((u,v),(v,p))}
9                      A_sol ← (A_sol \ {((w,v),(v,p))}) ∪ {((w,v),(v,r))}
10                     sub_ciclos ← SubCiclos(G_sol)
11     else
12         return sub_ciclos
13 return sub_ciclos
```

Algoritmo 8: Algoritmo Integrado

Entrada: Grafo SG asociado a una zona de recolección.

Salida : Lista de pares de nodos que representa un circuito.

```
1  Resolver modelo de PLE asociado a SG
2  if Se encontró una solución S then
3      G_sol ← CrearMultiDiGrafoSolucion(S)
4      sub_ciclos ← SubCiclos(G_sol)
5      if |sub_ciclos| > 1 then
6          sub_ciclos → UnionSubCiclos(sub_ciclos)
7          if |sub_ciclos| > 1 then
8              for sc ∈ sub_ciclos do
9                  Agregar al modelo las restricciones necesarias para evitar subciclo sc.
10             Volver a la linea 1
11         else
12             return Primer elemento de la lista subciclos
13     else
14         return Primer elemento de la lista subciclos
15 else
16     return No se encontró solución.
```

6.4. Resolviendo la minimización de giros con Concorde

El algoritmo 9 describe los pasos a seguir para la minimización de giros en U y consiste en la creación de la instancia a partir del grafo solución, G_{sol}, la cual se corresponde con la reducción presentada en la sección 5.2, y su resolución, que solo se encarga de obtener una solución llamando al solver Concorde. Aunque parece simple, existe un inconveniente en este ultimo paso y se describe a continuación.

Algoritmo 9: Minimización De Giros

Entrada: Muldigrafo euleriano $G_{sol} = (V_{sol}, A_{sol})$.

Salida : Recorrido ordenado en forma de lista de los elementos de A_{sol}.

```
1  G̃ ← Crear instancia TSP a partir de G_sol
2  camino ← Resolver Instancia asociada a G̃
3  return camino
```

El optimizador Concorde solo resuelve TSP sobre grafos completos y simétricos (no dirigidos). Claramente, toda instancia del TSP proveniente de la reducción de una instancia del PRECMG (Sección 5.2), puede transformarse en una instancia sobre un grafo completo asignándole a los arcos no presentes en el grafo $\tilde{G}$ un costo infinito. Es decir, construimos el grafo completo $G_c = (V_c, A_c)$ y una función de costos $c_c : A_c \to \mathbf{R} \cup \infty$, a partir del grafo $\tilde{G} = (\tilde{V}, \tilde{A})$ y la función $c : \tilde{A} \to \mathbf{R}$, de la siguiente manera:

$$V_c = V$$
$$A_c = V_c \times V_c - \{(v, v) : v \in V_c\}$$

$$c_c(a) = \begin{cases} c(a) & \text{si } a \in \tilde{A} \\ \infty & \text{si } a \notin \tilde{A} \end{cases}$$

Sin embargo, no quedará simétrico (aun tenemos un grafo dirigido).

Para evitar este problema, utilizamos una conocida transformación polinomial (para, dado un grafo dirigido, obtener un grafo no dirigido) del TSP asimétrico a simétrico [18]. A continuación, describimos tal transformación:

Dado el grafo dirigido $G_c = (V_c, A_c)$ y la función de costo c_c, definimos el grafo no dirigido $G_{tsp} = (V_{tsp}, A_{tsp})$ con un nodo ficticio y un nodo real por cada nodo de G_c y la función de costo $c_{tsp} : A_{tsp} \to \mathbf{R} \cup \infty$:

$$F = \{f_i \ : \ f_i \ nodo \ ficticio \ asociado \ a \ v_i \ con \ v_i \ \in \ V_c\}$$
$$V_{tsp} = V_c \cup F$$
$$A_{tsp} = V_{tsp} \times V_{tsp} - \{(v, v) : v \in V_{tsp}\}$$

$$c_{tsp}((x, y)) = \begin{cases} \infty & \text{si } x \in V_c \wedge y \in V_c \\ \infty & \text{si } x \in F \wedge y \in F \\ -M & \text{si } \exists i \in \{1 \dots |V_c|\} / x = v_i \wedge y = f_i \\ c_c((v_i, v_j)) & \text{si } \exists i, j \in \{1 \dots |V_c|\} / x = v_i \wedge y = f_j \wedge i \neq j \end{cases}$$

donde M es un valor suficientemente grande.

Así, las aristas entre nodos reales tiene valor ∞, lo que implica que los caminos mínimos no utilizarán esas aristas. Lo mismo para las aristas entre nodos ficticios. Un camino mínimo en este grafo siempre alternará nodos ficticios con reales. Además, como las aristas entre un nodo real y su correspondiente ficticio tienen un valor $-M$ (con M relativamente grande), los caminos siempre utilizarán estas aristas. Como el camino mínimo no contendrá los arcos con peso infinito y sí contendrá los que tienen peso negativo, a continuación de un nodo real, siempre aparecerá su nodo ficticio en un camino mínimo.

El peso de las aristas entre un nodo real v_i y uno ficticio f_j con $i \neq j$ es el peso del arco que va de v_i a v_j en el grafo G_c. Esto implica que el conjunto de aristas del nodo real v_i en G_{tsp} representa los arcos de salida de v_i en G_c. Recíprocamente el conjunto de aristas del nodo ficticio f_j en G_{tsp} representa los arcos de entrada de v_j en G_c.

Entonces, un camino de G_{tsp} de la forma:

$$f_1, \ v_1, \ f_2, \ v_2, \ \dots, f_k, \ v_k$$

se interpreta como un camino:

$$v_1, \ v_2, \ \dots, \ v_k$$

en G_c.

A modo de ejemplo, en las figura 6.2 pasamos de un grafo dirigido G (sobre el cual queremos calcular un camino hamiltoniano mínimo), a G_c, un grafo dirigido completo tal que un circuito hamiltoniano mínimo nos permite calcular el camino hamiltoniano mínimo de G.

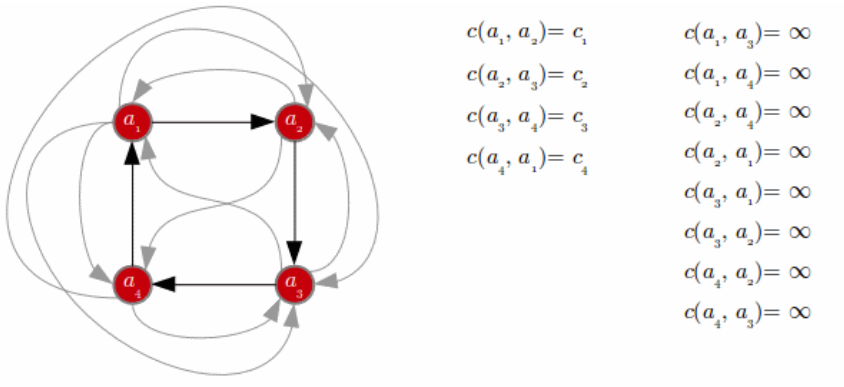


Fig. 6.2: Digrafo a digrafo completo, primer etapa de la transformación.

Luego, en la figura 6.3 pasamos a G_{tsp}, un grafo no dirigido completo. Si calculamos el circuito hamiltoniano mínimo de G_{tsp} podemos obtener fácilmente el circuito hamiltoniano mínimo de G_c .

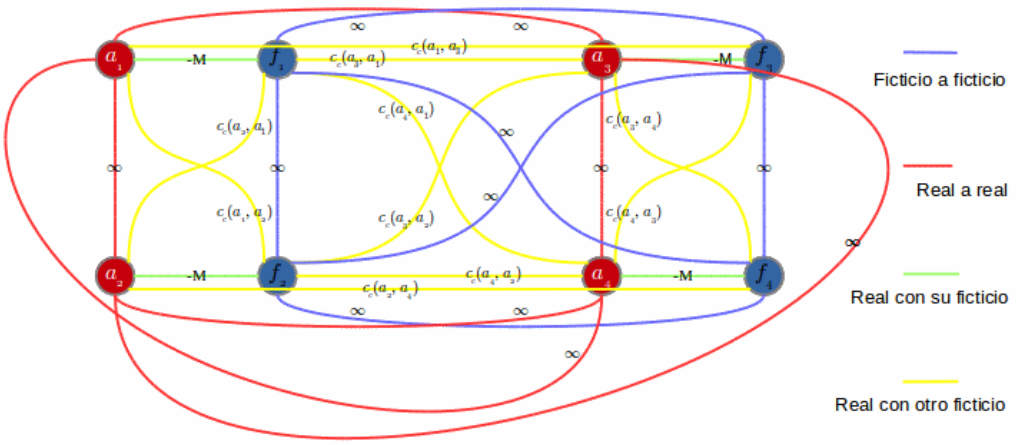


Fig. 6.3: Digrafo completo a grafo no dirigido completo, segunda y última etapa de la transformación.

De esta forma reducimos nuestro problema al Problema del Viajante de Comercio en un grafo completo no dirigido y podemos utilizar Concorde para obtener el resultado eficientemente.

Sobre el ejemplo planteado, se obtiene del Concorde, la secuencia de vértices:

$$f_1,\ a_1,\ f_2,\ a_2,\ f_3,\ a_3,\ f_4,\ a_4$$

La cual representa, sobre el grafo dirigido completo, el camino:

$$a_1,\ a_2,\ a_3,\ a_4$$

6.5. Algoritmo final

Tal como fue mencionado al final del Capítulo 5, la resolución del PRECMG sobre los grafos G_{sol} asociados a algunas de las zonas de recolección permitió obtener soluciones válidas (sin giros en U) sin ser necesaria la creación del supergrafo y del modelo asociado a éste (ver Tabla 7.2). Sin embargo, la cantidad de tiempo que requiere la creación y resolución del primer modelo de PLE en algunos casos es bastante alta y teniendo en cuenta que puede llegar a ser necesario (si luego de la minimización siguen existiendo giros en U) crear y resolver el segundo modelo de PLE (el cual también puede llegar a requerir mucho tiempo para su resolución), se decidió utilizar esta técnica para minimizar la cantidad de giros sobre la solución sin giros en U propuesta por el segundo modelo y así hacerla mas sencilla y amigable a la vista. Como podemos ver en las tablas 7.2 y 7.3, esto da muy buenos resultados.

En consecuencia, el que denominamos Algoritmo Final integra las técnicas presentadas en esta tesis resultando el más eficiente para la resolución de instancias del Problema del Cartero Rural en zonas urbanas.

En el Algoritmo 10 se presenta el detalle del Algoritmo Final.

Algoritmo 10: Algoritmo Final

Entrada: Archivos OSM, $entrada\_osm_{original}$ y $entrada\_osm_{expendida}$, correspondientes, respectivamente, a una determinada zona de recolección y a su zona extendida.

Salida : Camino eureliano dado en forma de lista de pares de vértices.

```
1  G ← Crear grafo a partir de los archivos entrada_osm_original y
   entrada_osm_expendida
2  SG ← Crear supergrafo asociado a G
3  Crear modelo de PLE asociado a SG
4  Resolver modelo de PLE asociado a SG
5  if Se encontró una solución S then
6      SG_sol ← CrearMultiDiGrafoSolucion(S)
7      sub_ciclos ← SubCiclos(SG_sol)
8      if |sub_ciclos| > 1 then
9          sub_ciclos ← UnionSubCiclos(sub_ciclos)
10         if |sub_ciclos| > 1 then
11             for sc ∈ sub_ciclos do
12                 Agregar al modelo las restricciones necesarias para evitar subciclo sc.
13             Volver a la linea 4
14         else
15             Se modifica G_sol teniendo en cuenta la modificación que realizo el
               algoritmo Union de Subciclos sobre la solución S
16             camino ← MinimizacionDeGiros(SG_sol)
17             Generar archivos de salida asociados a camino
18     else
19         camino ← MinimizacionDeGiros(SG_sol)
20         Generar archivos de salida asociados a camino
21 else
22     return No se encontró solución.
```

Capítulo 7

Experiencias computacionales y resultados

En este capítulo veremos los resultados obtenidos a partir de las experiencias llevadas a cabo para mostrar la efectividad de los algoritmos implementados.

Las experiencias fueron realizadas en una PC con procesador Intel Core i5-430M de 2.26GHz de velocidad de procesamiento y 4Gb de memoria RAM, corriendo con el sistema operativo Linux Ubuntu 14.04 de 64bits. Para la resolución de los modelos de programación lineal entera se utilizo el solver CPLEX[10] 12.5.1.0.

7.1. Resultados y comparación de los algoritmos para subciclos

En la primer tabla presentamos los resultados computacionales obtenidos de aplicar los dos algoritmos vistos en la Sección 6.3 para evitar subciclos, sobre las instancias correspondientes a las distintas zonas de recolección con las que se trabajaron.

La primer columna indica la zona utilizada para la experiencia. La segunda columna indica el numero de modelos de PLE que se tuvieron que crear y resolver para que el Algoritmo De Cortes (Algoritmo 6) obtenga una solución válida. Las últimas dos columnas corresponden a la cantidad de uniones que logró y no logró realizar, respectivamente, el Algoritmo Integrado (Algoritmo 8). Aclaramos que el valor de la última columna, número de uniones que no se lograron realizar, se corresponde con el número de modelos que tienen que ser creados y resueltos.

En la segunda tabla se presentan los tiempo de ejecución obtenidos al realizar las experiencias con las que se completó la tabla anterior.

Podemos ver entonces que, sin considerar el manejo de los datos de entrada, ni la creación y resolución del primer modelo de PLE ni tampoco el procesamiento posterior para una presentación amigable de las resultados, la obtención de una solución sin subciclos haciendo uso del:

- Algoritmo de Cortes: tuvo un promedio de 45 minutos de tiempo de ejecución, y la instancia más *dura* necesitó de 6 horas para obtener una solución válida.

Zona	Algoritmo de Cortes	Algoritmo Integrado	
	Modelos	Uniones	Fallos
1	0	0	0
2	3	2	0
3	4	3	1
4	3	2	0
5	5	11	3
6	2	3	0
7	7	10	2
8	1	1	0
9	4	6	1
10	1	2	0
11	3	3	0
12	6	7	0
13	2	4	0
14	2	2	0

Zona	Algoritmo de Cortes	Algoritmo Integrado
1	40 segs	6 segs
2	4 horas	4 segs
3	6 horas	10 segs
4	1 min 10 segs	1 seg
5	8 min 20 segs	2 mins
6	2 min 30 segs	6 segs
7	1 hora	1 min 15 segs
8	20 segs	2 segs
9	1 min 30 segs	6 segs
10	30 segs	4 segs
11	35 segs	2 segs
12	1 min 20 segs	2 segs
13	35 segs	4 segs
14	2 min	3 segs

- Algoritmo Integrado: tuvo un promedio de 18 segundos de tiempo de ejecución, y la instancia más *dura* necesitó 2 minutos para obtener una solución válida.

De los resultados expuestos en las primeras dos tablas, se puede ver que el Algoritmo Integrado es la técnica mas eficiente para la resolución de los subciclos y es por esto que es el algoritmo que se utiliza en la estrategia final de resolución.

7.2. Resultados de la minimización en cantidad de giros

En primer lugar, esta técnica se aplicó sobre los recorridos obtenidos para los grafos originales, es decir, aquellos grafos que contenían un vértice por esquina y un arco por calle.

Sobre las 14 instancias correspondientes a las zonas de recolección que se decidieron

optimizar, obtuvimos los siguientes resultados:

Zona	**Antes** de la minimización		**Después** de la minimización	
	Giros en U	Giros en las esquinas	Giros en U	Giros en las esquinas
1	57	117	15	84
2	37	123	11	97
3	35	198	1	181
4	10	78	0	54
5	27	187	5	157
6	42	193	3	169
7	50	230	6	210
8	0	83	0	48
9	11	129	1	75
10	5	154	3	90
11	2	99	2	37
12	4	111	0	46
13	2	102	0	56
14	32	174	5	121

Como se puede ver, en las zonas en las que aun persistían los giros en U, disminuye notablemente el número de giros en las esquinas. Sin embargo, no son soluciones factibles con lo cual, para esos casos, aplicamos este técnica pero sobre los caminos eulerianos obtenidos para los llamados *supergrafos* (vistos en la sección 3.2.2).

Cuando aplicamos la minimización por giros a las soluciones obtenidas del segundo modelo de PLE, ya no buscando evitar los giros en U, si no buscando reducir la mayor cantidad de giros en las esquinas posible. Logramos resultados muy satisfactorios y son los que se enuncian en la cuarta tabla. En la segunda columna de ella se muestra la cantidad de giros que contenía el recorrido, previo a la minimización en giros, y en la tercer columna, la cantidad de giros que contiene luego de ella. En la última columna se ve la cantidad de giros que logro reducir este algoritmo y el porcentaje que esta mejora representa.

Zona	**Antes** de la minimización	**Después** de la minimización	Mejora	
			Cantidad	Porcentaje
1	131	96	35	27
2	165	134	31	18
3	241	176	65	26
4	85	56	29	34
5	219	172	57	26
6	231	175	56	24
7	283	221	62	21
8	91	48	43	47
9	131	77	54	41
10	165	93	72	43
11	106	39	67	63
12	116	46	70	60
13	104	56	48	46
14	115	63	46	40

Claramente la aplicación de esta técnica sobre las soluciones ya válidas y mínimas respecto a cantidad de kilómetros, genera una importante mejora en la cantidad de giros. En consecuencia, en el algoritmo final utilizado para la obtención de los recorridos que se presentaran al Municipio, se aplica esta técnica de la manera antes enunciada, es decir, sobre los grafos solución correspondientes al segundo modelo de PLE.

7.3. Tiempos en la obtención de los recorridos óptimos

En la tabla final evaluamos la performance de cada una de las tareas mas importantes realizadas por el Algoritmo Final (Algoritmo 10) al aplicarlo sobre las instancias correspondientes a las zonas de recolección.

Las columna *Segundo Modelo* indica el tiempo que se necesitó para las tareas de creación y resolución del segundo modelo de PLE (Modelo 3.3), mientras que las columnas *Giros U*, *SubCiclos* y *Total*, indican el tiempo de ejecución de los algoritmos que se encargan de la minimización de giros (Algoritmo 9), evitar subciclos (Algoritmo 8) y la resolución total de la instancia (Algoritmo 10), respectivamente.

Tener en cuenta que el campo de la columna *Total* no coincide con la sumas de las demás columnas de la fila ya que en ellas no se incluye el tiempo necesario para la creación de los grafos, la generación de los archivos de salida y la impresión de algunos datos en pantalla.

Zona	Modelo PLE 2	Subciclos	Giros U	Total
1	7 segs	6 segs	4 segs	25 segs
2	6 mins 40 segs	4 segs	4 segs	7 mins
3	3 segs	10 segs	5 segs	26 segs
4	1 seg	3 segs	1 segs	9 segs
5	14 segs	2 mins	5 segs	2 min 20 segs
6	3 segs	12 segs	6 segs	30 segs
7	25 segs	1 min 15 segs	8 segs	2 mins
8	30 segs	2 segs	2 segs	8 segs
9	1 seg	6 segs	3 segs	15 segs
10	1 seg	4 segs	4 segs	17 segs
11	1 seg	3 segs	2 segs	10 segs
12	1 seg	5 segs	2 segs	14 segs
13	1 seg	4 segs	2 segs	11 segs
14	5 segs	7 segs	4 segs	14 segs

Como se ve, la mayoría de las instancias se logran resolver en un intervalo de tiempo corto, y la mayor cantidad de éste se encuentra en la creación y resolución de los modelos de programación lineal entera. Se puede deducir entonces que los algoritmos implementados realizan sus objetivos de manera eficiente y no influyen de manera determinante en el tiempo de ejecución.

Capítulo 8

Conclusiones y próximos pasos

8.1. Conclusiones

A lo largo del presente trabajo se estudió la posibilidad de utilizar técnicas de Investigación Operativa para diseñar recorridos de los camiones recolectores de residuos de la Municipalidad de Concordia que minimicen la cantidad de kilómetros recorridos.

El primer paso fue modelar el problema como un problema en grafos, representando a cada una de las zonas de recolección como un grafo mixto. Así, los recorridos de recolección de residuos en una determinada zona, que deben pasar por todas las cuadras de ella, no sería más que un camino euleriano sobre el mismo.

El problema es que todos los grafos asociados a las zonas de recolección no son eulerianos. Esto nos lleva al problema de decidir qué cuadras adicionales utilizar para cumplir el objetivo de recorrer todas las cuadras de la zona con la menor cantidad de kilómetros recorridos. Así, se incorporan cuadras aledañas a la zona y su grafo correspondiente con las distancias de sus arcos pasan a ser una instancia del Problema del Cartero Rural.

El Problema del Cartero Rural pertenece a la clase de problemas $\mathcal{NP}$-difíciles para los cuales su modelización como problema de Programación Lineal Entera resulta ser la técnica más eficiente para su resolución.

El primer modelo de Programación Lineal Entera utilizado (basado en la literatura existente) permitió obtener soluciones en un tiempo razonable pero no válidas: en muchas de las cuadras que se pueden recorrer en ambos sentidos, las soluciones utilizaban los llamados giros en U y, de esta manera, violaban las normas de tránsito de la Municipalidad de Concordia.

Este fue el motivo por el que se trató con una nueva estructura la cual llamamos *supergrafo* y no es más que una "expansión"del grafo que permite representar más información. Específicamente, resultaba necesario incluir el hecho de que la posibilidad de ir de una esquina A a una esquina B no solo dependa de la existencia de una calle que las una sino también del punto desde el cual se llegó a la esquina A.

Asociado a la nueva estructura que modela nuestra zona de recolección se debió implementar un nuevo modelo de Programación Lineal Entera.

A diferencia del modelo utilizado sobre el grafo original, el programa lineal entero que modelara un circuito euleriano en la zona de recolección, debía incluir restricciones que aseguraran la conexidad de la solución. De hecho, cuando estas restricciones no están presentes, las soluciones podían llegar a contener más de un circuito y a cada uno de ellos se los llamó subciclos. El problema es que las restricciones que aseguran la conexidad en la solución son un número exponencial y no pueden ser manejadas incorporándolas a todas ellas desde el inicio.

A través de la implementación de un algoritmo iterativo que, dado una solución obtenida por el segundo modelo de PLE, solo incorpora las restricciones necesarias para evitar los subciclos presentes en caso de que una unión "manual"de estos no sea posible, se resolvió de manera sumamente eficiente este problema.

De esta manera finalmente se obtuvieron los kilómetros mínimos necesarios para recorrer todas las cuadras de cada zona de recolección y cuántas veces debían ser recorridas cada una de las cuadras de la zona y de la zona aledaña considerada.

Una vez obtenidas estas soluciones óptimas, se implementó un algoritmo para determinar la manera de recorrer estas cuadras de manera de reducir la cantidad de giros. Para ello, el problema de minimizar giros se redujo a instancias del Problema del Viajante de Comercio (TSP). Estas soluciones resultan más *amigables* para su implementación por parte de la Subsecretaria de Higiene Urbana.

Finamente, para facilitar la consulta e implementación de los recorridos óptimos, se desarrolló una interfaz de usuario que hace muy simple esta tarea.

En resumen, el trabajo realizado en el marco de esta tesis permitió ofrecer a la Municipalidad de Concordia recorridos para los camiones recolectores de residuos que respetan la política de la SHU en lo que se refiere a la necesidad e importancia de recorrer todas las cuadras de cada zona y las normas de circulación en la ciudad. Estos recorridos realizan la menor cantidad de kilómetros posible con el consecuente ahorro en combustible, horas hombre, desgaste de camiones, etc y además, fueron optimizados en cantidad de giros realizados con el fin de hacerlos mas amigables para su lectura e implementación en las calles de la ciudad.

Los resultados obtenidos en esta tesis muestran el potencial que posee la integración de las diversas y variadas herramientas asociadas a ciencias como la informática y la matemática en la resolución y mejora de problemáticas actuales.

8.2. Trabajos a futuro

Los recorridos obtenidos en esta tesis ofrecen la garantía de que no pueden ser mejorados en términos de kilómetros totales recorridos ya que corresponden a una solución óptima del modelo de PLE. Sobre la solución óptima que brinda el modelo, resolviendo el problema de minimización de giros sobre el grafo G_{sol} asociado, obtenemos la forma de recorrer este grafo euleriano haciendo la menor cantidad de giros.

Sin embargo, el modelo de PLE tiene muchas soluciones óptimas. Todas ellas recorrerán la menor cantidad de kilómetros posibles. Pero cada una de ellas se traduciría en diferentes

grafos G_{sol}. El desafío sería lograr un modelo que integre estos dos objetivos de manera que, entre todos los circuitos óptimos en kilómetros recorridos, la optimización elija la solución tal que el SG_{sol} asociado pueda recorrerse con la menor cantidad de giros.

Un modelo posible a explorar es modificando la función de costos del Modelo 3.3 de manera que la misma incorpore la información de si ese arco se está recorriendo después de haber doblado o siguiendo derecho por la misma calle. Actualmente todos los arcos que componen el *superarco* correspondiente a una cuadra tienen el mismo costo. Sin embargo, cada uno de esos arcos tiene la información de *desde dónde se llega* a recorrerlo.

La idea seria entonces sumar un costo adicional a los arcos de SG que signifiquen haber cambiado de dirección. Este costo c_{ad} podría ser un porcentaje de la longitud de la cuadra que representa ese arco o un costo fijo.

Esto es, sabemos que en el grafo SG, cada cuadra entre las esquinas correspondientes a los nodos v_j y v_k tiene asociado un *superarco*, formado por varios arcos de la forma $((v_i, v_j), (v_j, v_k))$ que indican que esta cuadra se está recorriendo inmediatamente después de haber recorrido la cuadra entre las esquinas correspondientes a v_i y v_j. Actualmente, en el Modelo 3.3, todos estos arcos del *superarco* tienen el mismo costo c_{jk} en la función objetivo que corresponde a la longitud de la cuadra entre v_j y v_k.

Sin embargo, de la misma manera que lo hecho en el Capítulo 5 podemos calcular el coseno del ángulo que implica recorrer esa cuadra viniendo desde v_i. La propuesta es tomar como nuevo costo del arco $((v_i, v_j), (v_j, v_k))$ el que resulte de adicionar a c_{jk} el costo c_{ad} elegido multiplicado por 1 menos el coseno del ángulo.

Para la determinación de c_{ad} debería estudiarse el impacto en términos mecánicos (fuerza del motor, desgaste de cubiertas, etc) que implica un giro para camiones de ese porte.

Por último, queda pendiente el poder incorporar las soluciones a dispositivos que sean capaces de guiar a los conductores de los camiones de recolección a medida que van realizando el recorrido. Si bien hubo un comienzo de nuestro lado, creando archivos compatibles con la mayoría de los dispositivos GPS actuales, dado que los camiones con los que dispone la subsecretaria de higiene urbana no cuentan con ningún dispositivo capaz de utilizarlos, no se pudo probar si estos realmente cumplirían su objetivo.

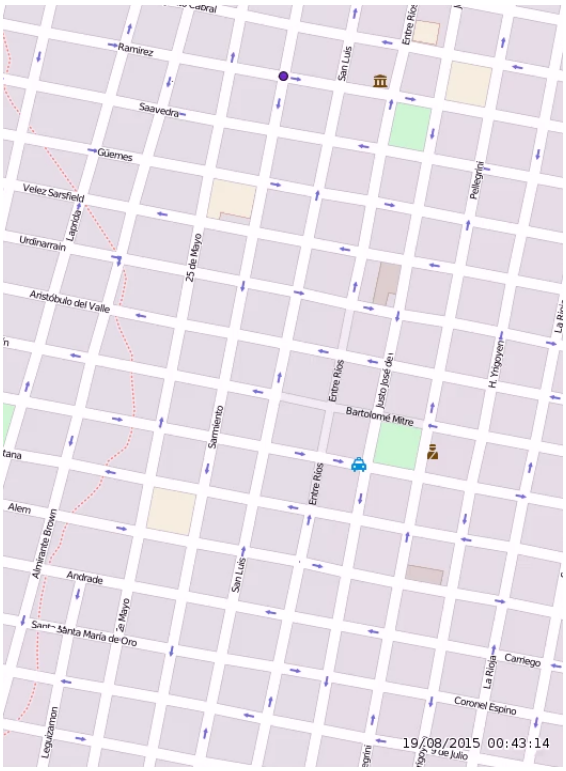
Apéndice

Apéndice A

Ejemplo de recorrido de recolección óptimo

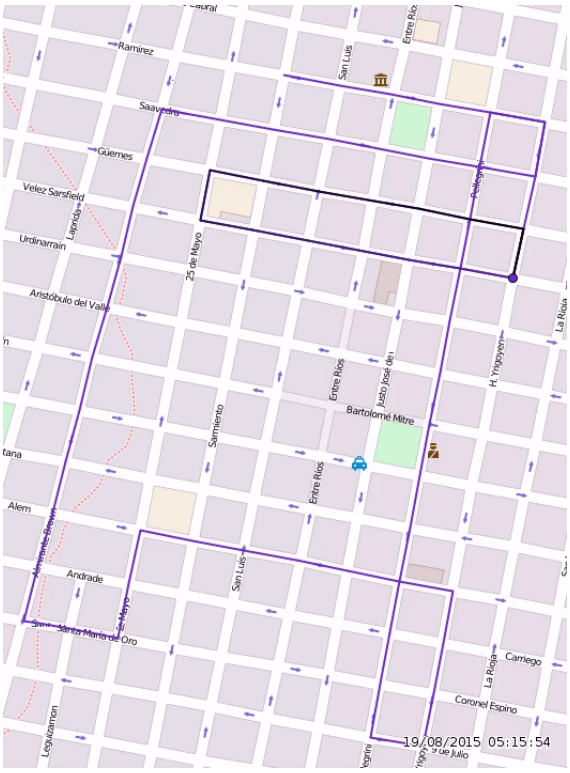
En este Apéndice mostramos el recorrido óptimo obtenido para la zona de recolección correspondiente al turno mañana, interno número 23, en dos de los formatos en los que se puede acceder al mismo: el listado de cuadras a recorrer y la traza sobre el mapa de la ciudad. Para mayor claridad se particiona el recorrido en tres etapas.

`Inicio: Ramirez y Sarmiento`



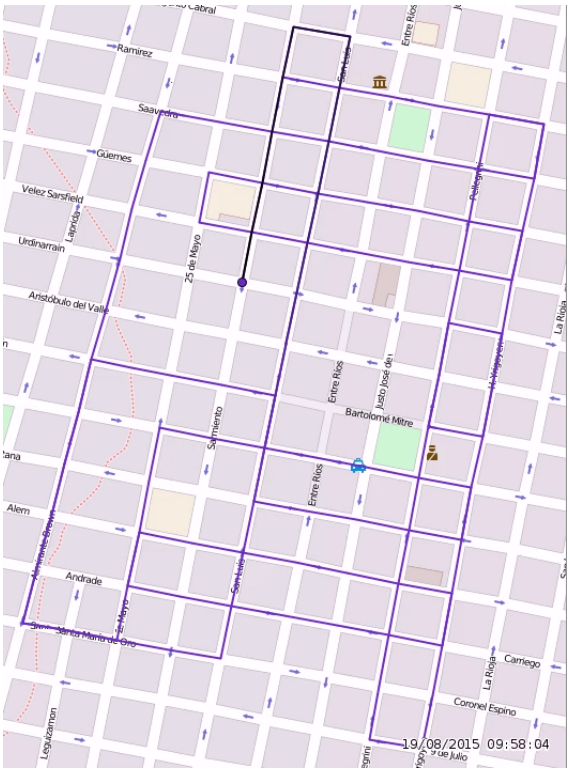
Primera parte:

Por Ramirez hacer 2 cuadras, hasta 3 de Febrero, y seguir derecho.
Por 3 de Febrero hacer 3 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 1 cuadras, hasta -/-, y girar a la derecha.
Por -/- hacer 3 cuadras, hasta Saavedra, y seguir derecho.
Por Saavedra hacer 4 cuadras, hasta Almirante Brown, y girar a la izquierda.
Por Almirante Brown hacer 10 cuadras, hasta Santa Mara de Oro, y girar a la izquieda.
Por Santa Mara de Oro hacer 2 cuadras, hasta 25 de Mayo, y girar a la izquierda.
Por 25 de Mayo hacer 2 cuadras, hasta Alem, y girar a la derecha.
Por Alem hacer 3 cuadras, hasta Roque Saenz Pea, y seguir derecho.
Por Roque Saenz Pea hacer 3 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 3 cuadras, hasta 9 de Julio, y girar a la izquierda.
Por 9 de Julio hacer 1 cuadras, hasta Pellegrini, y girar a la derecha.
Por Pellegrini hacer 12 cuadras, hasta 3 de Febrero, y girar a la derecha.
Por 3 de Febrero hacer 1 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 3 cuadras, hasta Corrientes, y girar a la derecha.
Por Corrientes hacer 3 cuadras, hasta Velez Sarsfield, y seguir derecho.
Por Velez Sarsfield hacer 3 cuadras, hasta 25 de Mayo, y girar a la derecha.
Por 25 de Mayo hacer 1 cuadras, hasta Gemes, y girar a la derecha.
Por Gemes hacer 3 cuadras, hasta Salta, y seguir derecho.
Por Salta hacer 3 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 4 cuadras, hasta Bartolom Mitre, y girar a la derecha.



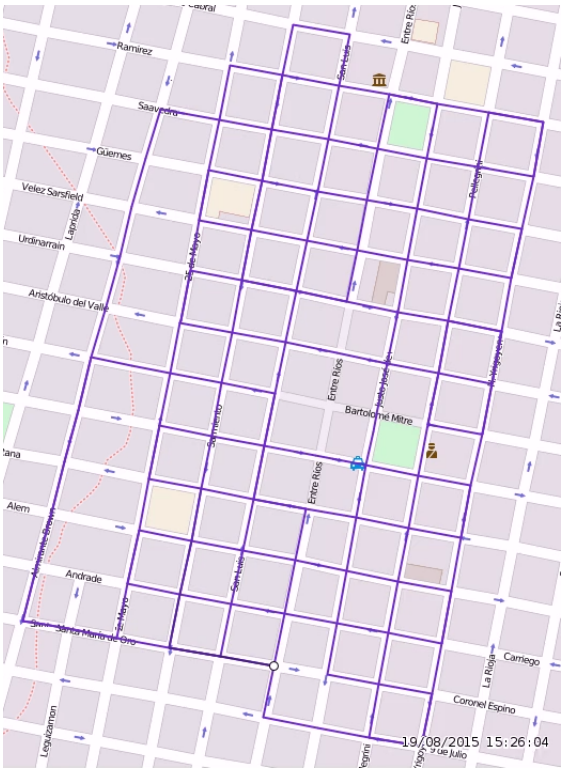
Segunda parte:

Por Bartolom Mitre hacer 1 cuadras, hasta Pellegrini, y girar a la derecha.
Por Pellegrini hacer 2 cuadras, hasta Catamarca, y girar a la derecha.
Por Catamarca hacer 1 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 6 cuadras, hasta Carriego, y girar a la derecha.
Por Carriego hacer 3 cuadras, hasta Andrade, y seguir derecho.
Por Andrade hacer 3 cuadras, hasta 25 de Mayo, y girar a la derecha.
Por 25 de Mayo hacer 3 cuadras, hasta Bernardo de Irigoyen, y girar a la derecha.
Por Bernardo de Irigoyen hacer 3 cuadras, hasta 1ro de Mayo, y seguir derecho.
Por 1ro de Mayo hacer 3 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 1 cuadras, hasta Buenos Aires, y girar a la derecha.
Por Buenos Aires hacer 3 cuadras, hasta Quintana, y seguir derecho.
Por Quintana hacer 1 cuadras, hasta San Luis, y girar a la derecha.
Por San Luis hacer 2 cuadras, hasta General San Martn, y girar a la izquierda.
Por General San Martn hacer 3 cuadras, hasta Almirante Brown, y girar a la izquierda.
Por Almirante Brown hacer 5 cuadras, hasta Santa Mara de Oro, y girar a la izquierda.
Por Santa Mara de Oro hacer 4 cuadras, hasta San Luis, y girar a la izquierda.
Por San Luis hacer 12 cuadras, hasta Sgto. Cabral, y girar a la izquierda.
Por Sgto. Cabral hacer 1 cuadras, hasta Sarmiento, y girar a la izquierda.
Por Sarmiento hacer 5 cuadras, hasta Urdinarrain, y girar a la izquierda.



Tercera parte:

Por Urdinarrain hacer 2 cuadras, hasta Catamarca, y seguir derecho.
Por Catamarca hacer 3 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 1 cuadras, hasta Alberdi, y girar a la derecha.
Por Alberdi hacer 3 cuadras, hasta Aristbulo del Valle, y seguir derecho.
Por Aristbulo del Valle hacer 3 cuadras, hasta 25 de Mayo, y girar a la derecha.
Por 25 de Mayo hacer 1 cuadras, hasta Urdinarrain, y girar a la derecha.
Por Urdinarrain hacer 3 cuadras, hasta Entre Ros, y girar a la izquierda.
Por Entre Ros hacer 4 cuadras, hasta 3 de Febrero, y giro a la derecha.
Por 3 de Febrero hacer 1 cuadras, hasta Justo Jos de Urquiza, y girar a la derecha.
Por Justo Jos de Urquiza hacer 11 cuadras, hasta Cnel. Espino, y girar a la derecha.
Por Cnel. Espino hacer 2 cuadras, hasta H. Yrigoyen, y girar a la derecha.
Por H. Yrigoyen hacer 1 cuadras, hasta 9 de Julio, y girar a la derecha.
Por 9 de Julio hacer 3 cuadras, hasta Entre Ros y girar a la derecha.
Por Entre Ros hacer 4 cuadras, hasta Quintana, y girar a la izquierda.
Por Quintana hacer 3 cuadras, hasta 25 de Mayo, y girar a la derecha.
Por 25 de Mayo hacer 6 cuadras, hasta Ramirez, y girar a la derecha.
Por Ramirez hacer 1 cuadras, hasta Sarmiento, y girar a la derecha.
Por Sarmiento hacer 11 cuadras, hasta Santa Mara de Oro, y girar a la izquierda.
Por Santa Mara de Oro hacer 2 cuadras, hasta Cnel. Espino, y seguir derecho.
Por Cnel. Espino hacer 1 cuadras.



Fin: Cnel. Espino y Justo Jos de Urquiza

Bibliografía

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, Princeton, New Jersey, 2006.

[2] E.J. Beltrami and L.D Bodin, *Networks and vehicle routing for municipal waste collection.* Networks, 4, 65-94, 1974.

[3] N. Christofides, *The optimum traversal of a graph.* Omega, 1, 719-732, 1973.

[4] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver, *Combinatorial Optimization.* John Wiley & Sons Inc., 1998

[5] J. Edmonds, *The chinese's postman problem.* Operations Research, 5:88–124, 1965.

[6] Jack Edmonds and Ellis L. Johnson, *Matching, Euler tours and the Chinese postman.* Mathematical Programming 5(1):88-124, 1973.

[7] R. W. Floyd, *Algorithm 97: Shortest path.* Communications of the ACM, 5:345, 1962.

[8] Garey, M and Johnson, D., *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, 1979.

[9] Harary, F., *Graph Theory.* Addison-Wesley, 1969.

[10] IBM ILOG CPLEX Optimization Studio, *http://www.ibm.com/software/commerce/optimization/cplex\discretionary{-}{}{}optimizer*

[11] N. Karmarkar, *A New Polinomial-time Algorithm for Linear Programing.* Combinatórica 4, pag. 373-395, 1984.

[12] L. G. Khachiyan, *Polinomian Algorithm in Linear Programing.* U.R.S.S. Computational Mathematics y Mathematical Physics, 20,pag 53-72, 1980.

[13] V. Klee and G. J. Minty, *How good is the simplex algorithm?.* Inequalities, III, Proc. Third Sympos., Academic Press, New York, 159-175, 1972.

[14] J. Lenstra and A. Rinnooy-Kan, *On General Routing Problems.* Networks, 6, 273-280, 1976.

[15] Mitchell, J. E., *Integer programming: branch and cut algorithms* In Encyclopedia of Optimization (pp. 1643-1650). Springer US., 2009.

[16] E. Nuutila and E. Soisalon-Soinen, *On finding the strongly connected components in a directed graph.* Information Processing Letters 49(1): 9-14, 1994.

[17] T. K. Ralphs, *On the Mixed Chinese Postman Problem.* Operations Research Letter vol. 14, pp. 123–127, 1993.

[18] Ratnesh Kumar and Haomin Li, *On Asymmetric Tsp: Transformation to Symmetric Tsp and Performance Bound.* Journal of Operations Research, 1994.

[19] Sanjeev, A. and Boaz, B., *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[20] R. Tarjan., *Depth-first search and linear graph algorithms.* SIAM Journal of Computing 1(2):146-160, 1972.

[21] S. Warshall, *A theorem on boolean matrices.* Journal of the ACM, 9:11–12, 1962.

[22] L. Wolsey and G. Nemhauser, *Integer Programming and Combinatorial Optimization.* John Wiley & Sons Inc., 1988

[23] ZIMPL (Zuse Institut Mathematical Programming Language), *http://zimpl.zib.de/*