Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Técnicas de programación lineal entera para la optimización de la recolección de residuos reciclables en el Municipio de Morón

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Francisco B. Wesner

Directores: Javier Marenco y Guillermo A. Durán

Buenos Aires, 2015

TÉCNICAS DE PROGRAMACIÓN LINEAL ENTERA PARA LA OPTIMIZACIÓN DE LA RECOLECCIÓN DE RESIDUOS RECICLABLES EN EL MUNICIPIO DE MORÓN

El Problema del Cartero Chino consiste en encontrar una ruta de costo mínimo para atravesar cada arco de una red o grafo al menos una vez y volver al vértice desde el cual se empezó. Diversos problemas pueden ser modelados como Problemas del Cartero Chino, tales como la recolección de basura, la programación de las rutas de patrulla de la policía, o las rutas de autobuses escolares. El Problema del Cartero Chino está bien resuelto cuando el grafo original contiene sólo arcos o sólo aristas, sin embargo, el Problema del Cartero Chino Mixto es NP-completo.

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 reciclables en el Municipio de Morón, en la Provincia de Buenos Aires, Argentina. Este trabajo se enmarca dentro de un convenio entre la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires y la Municipalidad de Morón. Este problema corresponde a una variante del Problema del Cartero Chino Mixto. Al considerar que no se puede "girar en U" ni girar a la izquierda en las esquinas con semáforos sin giro permitido, se agregan restricciones adicionales.

Para cumplir con el objetivo se resuelve un modelo de programación lineal entera construido en base al problema. Para agilizar la ejecución del programa, se implementó un algoritmo que acelera la búsqueda uniendo los subciclos generados en la solución del modelo. También se desarrolló un algoritmo de búsqueda tabú para mejorar la distribución del área de recolección de cada camión. En general se obtuvieron buenos resultados, logrando realizar la recolección en todas las cuadras de cada sector del municipio dentro del intervalo de tiempo establecido.

Palabras clave: Grafos, Transporte, Problema del cartero chino, Ruteo de vehículos, Programación lineal entera, Metaheuristicas, Búsqueda tabú

INTEGER LINEAR PROGRAMMING TECHNIQUES TO OPTIMIZE THE COLLECTION OF RECYCLABLE WASTE IN MORÓN, BUENOS AIRES

The Chinese Postman Problem is to find a least cost way to traverse each arc of a network at least once and to return to the starting vertex. Diverse problems such as garbage collection, police patrol scheduling and school bus routing can be modelled as Chinese Postman problems. The Chinese Postman Problem is well solved when the original graph contains only arcs or only edges, however, the Mixed Chinese Postman Problem is NP-complete.

The aim of this thesis is to design and implement algorithms based on integer linear programming to optimize municipal recyclable waste collection in Morón, Buenos Aires, Argentina. This work is part of an agreement between the School of Exact and Natural Sciences, University of Buenos Aires and the Municipality of Morón. This problem corresponds to a variant of the mixed Chinese postman problem. Considering that you cannot u-turn or turn left at the corners with traffic lights, additional constraints are added.

In order to reach the goal, an integer linear programming model was built. To accelerate the execution of the program, an algorithm that joins the subtours generated by the solution of the model was implemented. A Tabu search algorithm was also developed to improve the distribution of the collection area of each truck. We obtained good results, doing the collection in all blocks of each area within the established time frame.

Keywords: Graphs, Transportation, Chinese postman problem, Vehicle routing, Integer linear programming, Metaheuristics, Tabu search

AGRADECIMIENTOS

A mi familia, que sin su soporte incondicional no me hubiera sido posible llegar a este punto.

A mis amigos, que me acompañaron durante todos estos años de carrera.

A mis directores de tesis, Guillermo A. Durán y Javier Marenco, por su guía, consejo y tiempo.

A Gustavo Braier, quien hizo posible este proyecto.

A la gente del muncipio de Morón que ayudó en el proyecto aportando datos y demás.

Al jurado, por haberse dedicado a la lectura y corrección de esta tesis.

A mi familia

Índice general

1. INTRODUCCIÓN

1.1. Descripción del problema

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 reciclables en el Municipio de Morón, en la Provincia de Buenos Aires. Este trabajo se enmarca dentro de un convenio entre la Facultad de Ciencias Exactas y Naturales de la Universidad de Buenos Aires y la Municipalidad de Morón.

El municipio se encuentra dividido en cinco localidades, y a su vez cada localidad está dividida en hasta 8 cuadros. Un *cuadro* es un conjunto de manzanas que es recorrido por un camión determinado durante la recolección. Los nombres de las localidades son: Castelar, Morón, Haedo, Palomar y Villa Sarmiento. Cada localidad es recorrida un día prefijado de la semana, y cada camión está abocado a recorrer un cuadro. Dado que no existe un mecanismo de contenedores, cada camión debe recorrer todas las cuadras de su cuadro, respetando los sentidos de las calles y los giros en las esquinas con semáforos. Este problema corresponde a una variante del problema del cartero chino mixto, que es NP-completo [13]. Al considerar que no se puede "girar en U" ni girar a la izquierda en las esquinas con semáforos sin giro permitido, se agregan restricciones adicionales.

Previo a la realización de este trabajo, cada recorrido era decidido por el conductor del camión mientras se realizaba la recolección. Como consecuencia, cambiaban continuamente y, en la mayoría de los casos, la recolección no se realizaba en algunas cuadras. Los residuos que quedaban sin recolectar eran levantados luego por el recorrido nocturno como residuos generales. La cantidad de quejas de los vecinos debido a que no se realizaba la recolección de residuos reciclables en su cuadra, alertó a la municipalidad para que luego se iniciara este proyecto de optimización de los recorridos.

Las causas de la no-recolección de algunas de las cuadras se debe a la falta de tiempo. Los camiones salen del Obrador Municipal[1] a las 14hs, realizan la recolección, y llevan los residuos al Centro de acondicionamiento de Materiales[2], terminando alrededor de las 19hs. Debido a que luego los camiones son usados en la recolección nocturna y por cuestiones gremiales no pueden partir del obrador más temprano, debe haber un estricto cumplimiento del horario.

La construcción de los cuadros en los que se divide cada localidad fue realizada "a ojo" por el personal de la municipalidad. A medida que se realizaban las recolecciones, los cuadros se fueron ajustando para balancear el tiempo que se tarda en cada recorrido, y reducir el número de cuadras que se dejan sin recolectar. Como objetivo adicional de la tesis se incluye la modificación de estos cuadros mediante una heurística de búsqueda tabú, con la finalidad de equilibrar el costo de recorrer cada cuadro.

[1] Pola 1398, Localidad de Morón, Provincia de Buenos Aires

[2] Av. Juan Álvarez de Arenales 60, Localidad de Morón, Provincia de Buenos Aires

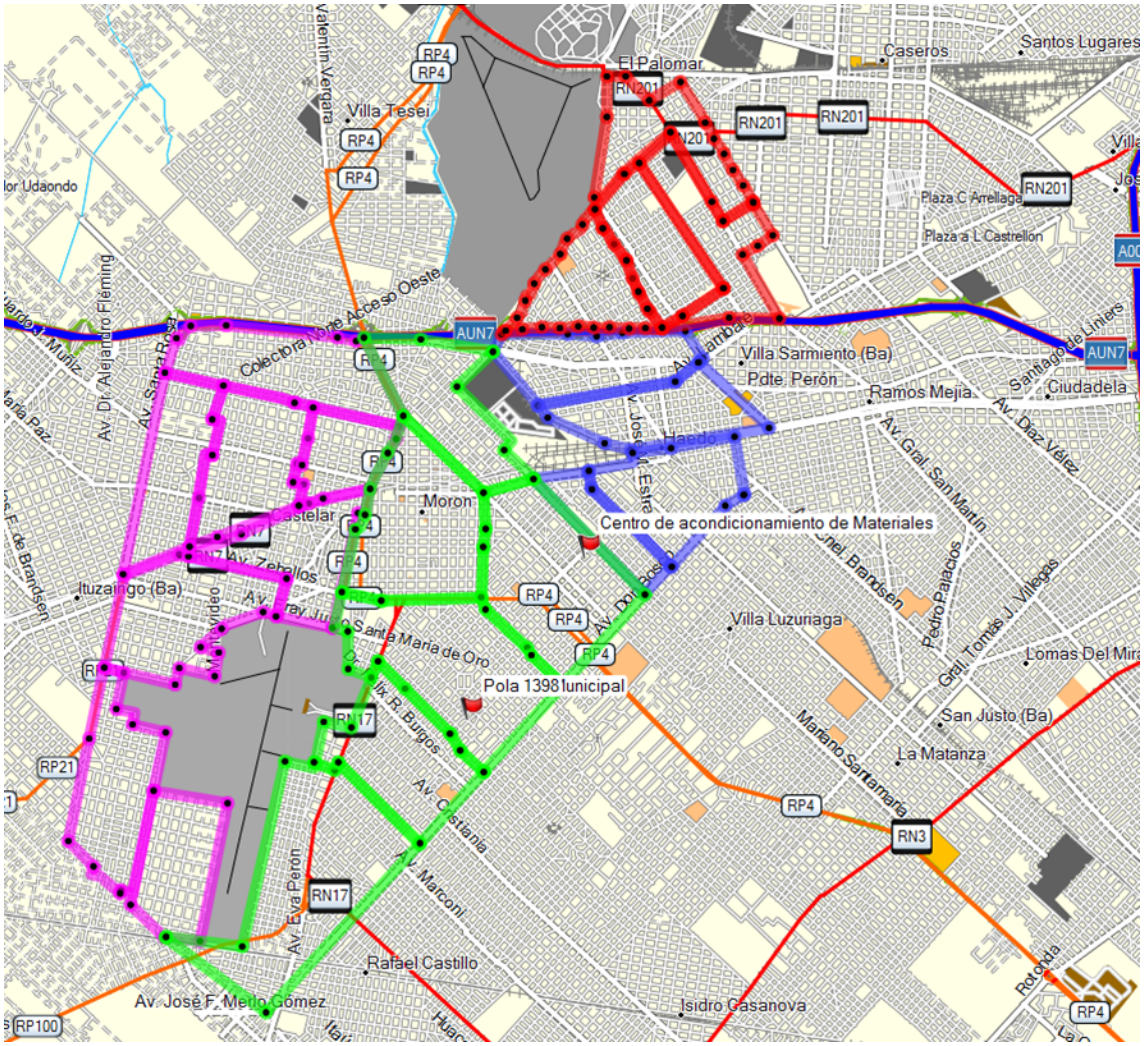


Fig. 1.1: Partido de Morón dividido en cuadros según el proyecto de recolección de residuos reciclables[3]. Los cuadros de distinto color son recolectados durante días diferentes de la semana.

1.2. Definiciones

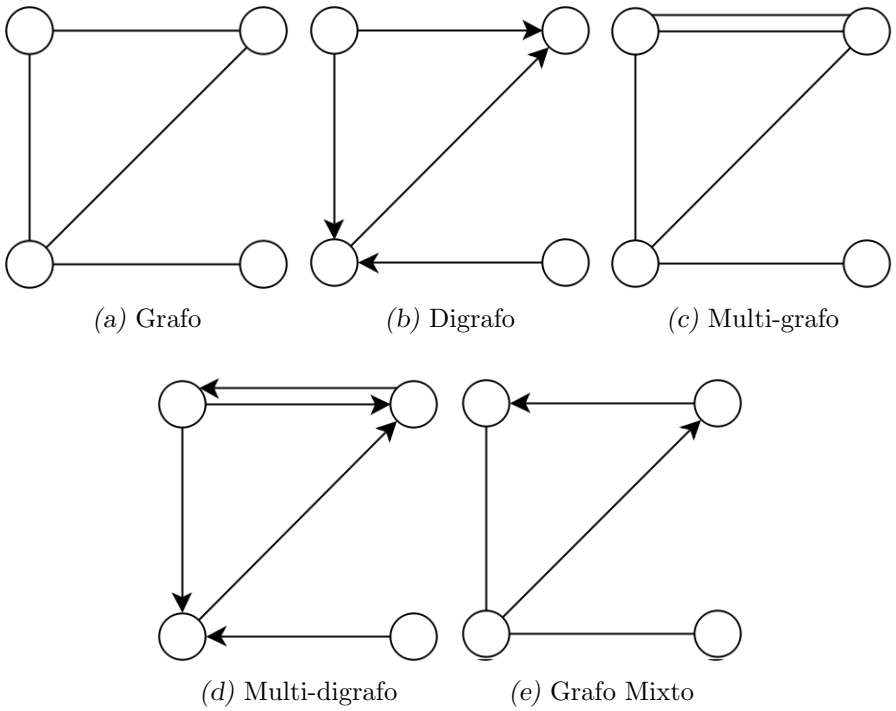
A continuación enunciamos algunas definiciones que usaremos en el desarrollo de la tesis [7].

Un *grafo* G consiste en un par ordenado $G = (V, E)$, donde:

- $V \neq \emptyset$ es un conjunto finito no vacío de *vértices* o *nodos*
- E es un conjunto de pares no ordenados (v, w) con $v, w \in V, v \neq w$, denominados *aristas*.

Para cada $(v, w) \in E$, decimos que v y w son *adyacentes* o vecinos, y que v es adyacente

[3] Mapa proporcionado por la municipalidad de Morón. Por un error en el mapa, no fue incluida la localidad de Villa Sarmiento



(a) Grafo *(b)* Digrafo *(c)* Multi-grafo

(d) Multi-digrafo *(e)* Grafo Mixto

Fig. 1.2: Ejemplos de diferentes tipos de grafos

a w. Por simplicidad, también se puede escribir vw para referirnos a la arista (v, w).

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), a es su *nodo inicial* o *cola*, y b su *nodo final* o *cabeza*.

Un *grafo mixto* es una tupla $G = (V, E, A)$ donde

- $V \neq \emptyset$ es un conjunto finito no vacío de *vértices* o *nodos*
- E es un conjunto de pares no ordenados (v, w) con $v, w \in V$, denominados *aristas*.
- A es un conjunto de pares ordenados (v, w) con $v, w \in V$, denominados *arcos*.

Un *multigrafo* o pseudografo es un grafo $G = (V, E)$ donde E es un multiconjunto de pares no ordenados (v, w) con $v, w \in V$. Es decir, puede tener aristas múltiples (aristas que relacionan los mismos nodos).

Un *multidigrafo* es un multigrafo cuyas aristas son dirigidas.

Un grafo $G = (V, E)$ diremos que es un *grafo ponderado* si tiene asociada una función $w : E \to R$ llamada *función de ponderación*. La imagen de cada arista (o arco) determinada por los vértices v_i y v_j la llamaremos *peso* o *costo* de la arista y lo denotaremos por w_{ij}.

Se llama *camino* en un grafo a una secuencia de vértices dentro del grafo tal que existe una arista entre cada vértice y el siguiente. Se dice que dos vértices están conectados si existe un camino que vaya de uno a otro, de lo contrario estarán desconectados. Dos vértices pueden estar conectados por varios caminos. El número de aristas dentro de un camino es su longitud.

Si el camino está definido sobre un grafo ponderado, la longitud del camino queda

determinada por la suma de los pesos de las aristas que unen los vértices del camino.

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. Análogamente, se define *camino euleriano* como un camino que pasa por cada arista una y sólo una vez.

Por último, dado un digrafo $G = (V, A)$ definimos $\delta^-(v)$ para $v \in V$ como el conjunto de arcos que tienen a v como su cabeza. Análogamente, definimos $\delta^+(v)$ como el conjunto de arcos que tienen a v como su cola.

1.3. Teoría de complejidad computacional

La teoría de la computación se centra en la clasificación de los problemas computacionales de acuerdo a su dificultad [14] [16]. Para poder referirnos a problemas como "difíciles" y a problemas de dificultad "equivalente", es necesario comprender algunos términos más básicos que se explican a continuación.

Un *problema de decisión* es aquel en donde las respuestas posibles son "sí" o "no". Comúnmente se define a un problema de decisión describiendo su conjunto de instancias cuya respuesta es "sí" y aquellas cuyas respuesta es "no". Por ejemplo, el problema de decisión "dado un natural n, ¿es n primo?" tiene como conjunto de instancias a los naturales $\mathbb{N}$ y como conjunto de instancias "sí" a $\{p \in \mathbb{N} : p \text{ es primo}\}$.

Un *algoritmo* A 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 $f_A = O(g)$, 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 $f_A = O(g)$ decimos que el tiempo de ejecución de A es $O(g)$. 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 lineal o polinomial será considerado eficiente, mientras que uno exponencial como ineficiente.

Dado un problema de decisión P, decimos que A es un algoritmo para P, si A toma como entrada al conjunto de instancias de P, y produce como salida "1" para las entradas pertenecientes al conjunto de instancias "sí" de P, y "0" para las pertenecientes al conjunto de instancias "no".

Decimos que un problema P es *polinomial* si existe un algoritmo polinomial A para P. Llamamos $\mathcal{P}$ al conjunto de todos los problemas de decisión polinomiales. Definimos como $\mathcal{NP} - completo$ al conjunto de todos los problemas de decisión cuyas instancias en donde la respuesta es "sí" pueden ser verificadas en tiempo polinomial.

Un problema P es *polinomialmente reducible* a otro problema Q si existe una función f computable en tiempo polinomial que transforma instancias de P en instancias de Q de modo tal que $f(x)$ es una instancia de respuesta "sí" de Q si y sólo si x es una instancia de

respuesta "sí" de P. Así, resolver P no puede ser más difícil que resolver Q. Un problema P es $\mathcal{NP} - hard$ si cada problema en $\mathcal{NP}$ es reducible a P. Si ademas $P \in \mathcal{NP}$, decimos que P es $\mathcal{NP} - completo$.

Una de las mayores preguntas aún sin resolver en la ciencia de la computación es si $\mathcal{NP} = \mathcal{P}$ o no. Sin embargo la mayoría de los investigadores cree que son diferentes. Muchos problemas importantes se sabe que son $\mathcal{NP}-$hard o $\mathcal{NP}-$completos. Estos problemas se consideran "difíciles" de resolver.

1.4. Problemas relacionados de la literatura

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 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)?.

Existen diversos problemas de optimización en la literatura que son muy conocidos debido a que muchos problemas teóricos y del mundo real pueden ser modelados en este esquema general. A continuación mencionaremos algunos de ellos que tienen relación con el trabajo presentado en esta tesis.

1.4.1. El Problema del Viajante de Comercio

El Problema del Viajante de Comercio (Travelling Salesman Problem en inglés, o simplemente TSP) [5] responde a la siguiente pregunta: Dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y regresa a la ciudad origen?. La variante de decisión equivalente es un problema $\mathcal{NP} - hard$.

Este problema es uno de los problemas de optimización más estudiados. Aunque el problema es computacionalmente complejo, una gran cantidad de heurísticas y métodos exactos son conocidos, de manera que, algunas instancias desde cien hasta miles de ciudades pueden ser resueltas en forma óptima.

El TSP puede ser modelado a través de un grafo ponderado no dirigido, de manera que las ciudades sean los vértices del grafo, los caminos son las aristas y las distancias de los caminos son los pesos de las aristas. Es un problema de minimización que comienza y termina en un vértice especifico y se visita el resto de los vértices exactamente una vez. Con frecuencia, el modelo es un grafo completo (cada par de vértices es conectado por una arista). Si no existe camino entre un par de ciudades, se añade arbitrariamente una arista muy pesada para completar el grafo sin afectar el recorrido óptimo.

1.4.2. El Problema del Cartero Chino

El Problema del Cartero Chino [4] consiste en encontrar el circuito más corto que visite cada arista de un grafo (conectado), es decir, que pase al menos una vez por cada arista del grafo, volviendo al punto (o nodo) de partida. Cuando el grafo posee un circuito euleriano (un circuito que alcance toda arista exactamente una vez), ese circuito es una solución óptima.

En 1965, Edmonds [3] presentó un algoritmo polinomial para resolver el problema del cartero chino, basado en el algoritmo para encontrar todos los caminos mínimos en un grafo enunciado por Floyd [6] y Warshall [18].

1.4.3. El Problema del Cartero Chino Mixto

Dado un grafo mixto G, un circuito de G resuelve el problema del cartero chino para ese grafo G si contiene cada arista y arco de G al menos una vez. Para que exista tal circuito, el grafo en cuestión G 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).

A diferencia del cartero chino dirigido y no dirigido, no se conocen algoritmos polinomiales que resuelvan el problema del Cartero Chino Mixto en forma exacta. Papadimitrour [13] demostró que que el problema de decisión equivalente es $\mathcal{NP} - hard$.

1.5. Programación lineal entera

La programación lineal es una técnica matemática que consiste en una serie de métodos y procedimientos que permiten resolver problemas en los que se exige maximizar o minimizar funciones lineales que se encuentran sujetas a determinadas limitaciones lineales, que llamaremos restricciones. [1]

En esencia la programación lineal consiste en un problema de optimización en donde tanto la función objetivo como las restricciones pueden ser expresadas como una serie de expresiones lineales en función de las variables de decisión. Si el problema tiene n variables, entonces las restricciones definen un conjunto de semiespacios en el espacio de n dimensiones. Estos planos marcan los limites de una región que define el conjunto de las posibles soluciones del problema.

En general, un problema de programación lineal se puede escribir de la siguiente forma:

Maximizar (o minimizar):

$$\sum_{i=1}^{n} c_i x_i$$

Sujeto a:

$$\sum_{i=1}^{n} a_{1i} x_i \sim b_1$$

$$\vdots$$

$$\sum_{i=1}^{n} a_{mi} x_i \sim b_m$$

donde $\sim$ puede ser $\leq$,$\geq$, o $=$,
a_{ji}, b_j, c_i con $1 \leq i \leq n$ y $1 \leq j \leq m$ son constantes,
y x_i con $1 \leq i \leq n$ son las variables de decisión.

Con el término "programación lineal entera" nos referiremos a problemas de programación lineal, pero en los que las variables están restringidas a tomar valores enteros.

Los problemas de programación lineal entera nos permiten modelar muchas más situaciones que la programación lineal, pero a cambio la resolución de los problemas será mucho más costosa, presentando, en general, un costo computacional mucho más elevado que el de la programación lineal. Los problemas de programación lineal pueden resolverse utilizando un algoritmo polinomial [10], mientras que los problemas de programación lineal entera corresponden a la case de problemas $\mathcal{NP} - completo$ [17]. Llamamos *relajación* de un problema de programación lineal entera, al problema que se obtiene al eliminar las restricciones que fuerzan a las variables a ser enteras. Es decir, permitimos que las variables puedan tomar valores no enteros.

En la práctica, la mayoría de los problemas de programación lineal entera se resuelve mediante el uso de la técnica denominada *branch and cut* [11]. Este método se basa en resolver primero la relajación del problema usando el *algoritmo simplex* [2]. Si la solución no es entera, es usada para formular una nueva restricción que sea satisfecha por la, todavía desconocida, solución entera, pero no por la solución no entera calculada. Esta restricción se suele llamar *de corte*, y en general todas las soluciones enteras cumplen esa desigualdad. Este proceso se repite hasta que se obtiene una solución entera. Si no encontramos ninguna solución, la rama actual llega a su fin. Luego de evaluadas todas las ramas posibles, la mejor cota obtenida es la solución al problema. Para un modelo de maximización, la relajación continua nos proporciona una cota superior del valor óptimo del modelo de programación entera asociado, mientras que en un problema de minimización, nos proporciona una cota inferior. Se utilizan estas cotas para eliminar todas aquellas ramas que no lleven a soluciones óptimas, reduciendo el número de ramas a explorar.

1.6. Metaheurísticas

Las metaheurísticas pueden definirse como métodos diseñados para encontrar "buenas" soluciones a problemas de optimización combinatoria difíciles de resolver de forma exacta. Son métodos iterativos que exploran el espacio de soluciones buscando eficientemente una solución cercana a la óptima. [12]

Existe una gran variedad de metaheurísticas basadas en por ejemplo heurísticas clásicas, inteligencia artificial, biología evolutiva, etc., que han demostrado su poder a la hora de obtener soluciones de gran calidad en tiempos de ejecución bajos. A continuación veremos algunas de ellas.

1.6.1. Búsqueda Local

La búsqueda local es una metaheuristica que se basa en el concepto de explorar la vecindad de una solución inicial (no necesariamente óptima). Más precisamente, comienzan con una solución inicial y reiteradamente se mueven bajo algún criterio a una solución vecina hasta cumplir con algún criterio de terminación. Estas soluciones son evaluadas de acuerdo a la función que se busca optimizar. Al finalizar el algoritmo, se obtiene como resultado la mejor solución entre todas las evaluadas a lo largo de las iteraciones.

El óptimo local que produce la búsqueda puede estar muy lejos del óptimo global del problema, ya que solamente explora el espacio de soluciones cercanas a la solución inicial. Además la solución encontrada depende en gran medida del método de generación de vecinos y de la solución que se usa como inicial.

1.6.2. Búsqueda Tabú

La búsqueda tabú es una metaheuristica perteneciente a la clase de técnicas de búsqueda local. La búsqueda tabú aumenta el rendimiento del método de búsqueda local mediante el uso de estructuras de memoria: una vez que una potencial solución es determinada, se la marca como "tabú" de modo que el algoritmo no vuelva a visitar esa posible solución. De esta forma continua la búsqueda de soluciones en donde la búsqueda local clásica podría quedar estancada (iterando entre las mismas soluciones una y otra vez).

Existen variantes de este método que en lugar de prohibir una solución, prohíben las soluciones que comparten cierta característica o atributo que ha cambiado en las últimas iteraciones, reduciendo aún mas el espacio de vecinos de una solución. De esta forma se evita durante algunas iteraciones las soluciones "demasiado parecidas", expandiendo el área de búsqueda.

1.6.3. Otras técnicas

Existen otras técnicas metaheurísticas más avanzadas que no detallaremos pero vale la pena mencionar. Por ejemplo, Simulated Annealing es un método probabilístico inspirado en el proceso de recocido del acero (proceso que consiste en calentar y luego enfriar el material lentamente para variar sus propiedades físicas). Esta técnica al igual que las

anteriores parte de una solución inicial que es modificada mientras se realiza la búsqueda.

Otro ejemplo interesante son los algoritmos genéticos, inspirados en el proceso de selección natural. A diferencia de los anteriores, se mantiene un conjunto de posibles soluciones del problema. El algoritmo repetidamente modifica la "población" de posibles soluciones, seleccionando en cada paso individuos que usará como "padres" para producir a los individuos de la próxima generación. A medida que nacen diferentes generaciones, la población evoluciona hacia una solución mejor.

1.7. Resumen del trabajo

El objetivo de esta tesis consiste en implementar un algoritmo basado en técnicas de programación lineal entera para una variante del problema del cartero chino mixto, y aplicarlo a los cuadros del Municipio de Morón, con la intención de que los recorridos obtenidos sean utilizados en la práctica.

A continuación listaremos el software utilizado en la realización del trabajo.

Se usaron los datos de OpenStreetMaps(OSM)[4], una base de datos de mapas del mundo de licencia libre. La municipalidad nos proveyó de sus datos para corregir la información errónea o desactualizada (que existía en cantidad en los datos de OSM), y agregar otros datos de interés, como la posición de los semáforos del partido. Para realizar estas correcciones e incorporación de información se usó el programa JOSM[5], un editor de archivos .osm (los cuales se pueden obtener de OSM) que provee una interfaz gráfica sencilla de usar para la edición de los mapas (así como la adición de etiquetas a las cuadras o esquinas del mapa).

El trabajo se realizó en su mayoría utilizando Python[6], un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. El archivo .osm tiene la estructura del lenguaje XML, por lo que su procesamiento desde Python es sencillo.

Para la resolución del modelo de programación lineal entera se usó SCIP[7], un solver no comercial para problemas de programación lineal. Como interfaz de programación entre SCIP y Python se usó la biblioteca de uso libre NumberJack[8], que permite trabajar con SCIP de forma sencilla e intuitiva.

Por último, se creó una pequeña aplicación que muestra los recorridos en forma de animación. Con ese propósito se usó la biblioteca Python Imaging Library[9] que provee diversas capacidades para trabajar con imágenes en Python.

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

- En el capítulo "Preliminares" se explica cómo se obtuvieron los mapas para el problema, cómo se corrigieron los datos con errores, y cómo se representa el mapa del partido.

[4] http://www.openstreetmap.org/
[5] https://josm.openstreetmap.de/
[6] https://www.python.org/
[7] http://scip.zib.de/
[8] http://numberjack.ucc.ie/
[9] http://www.pythonware.com/products/pil/

- En el capítulo "Resolución del problema" se explica en detalle cómo se resolvió el problema. Se desarrollan todas las soluciones implementadas en orden, desde el primer modelo que se construyó, pasando por los problemas que se fueron presentando, cómo se resolvieron y las decisiones que se tomaron, hasta llegar a la solución final.
- En el capítulo "Detalles de la implementación" se explican detalles de los algoritmos implementados.
- En el capítulo "Resultados" se muestran los resultados obtenidos a partir de experimentar con diferentes cuadros, cálculo de las mejoras obtenidas, etc.
- En el capítulo "Conclusiones" se muestra una valoración personal del trabajo realizado, y si se cumplieron o no los objetivos.
- En el último capítulo "Apéndices" se muestra un ejemplo de una ruta calculada.

2. PRELIMINARES

2.1. Definiciones adicionales

A continuación enunciaremos algunas definiciones que usaremos a lo largo de la tesis, sobre todo en el proceso de zonificación.

Desde el punto de vista de la recolección de residuos, llamaremos *zona* a un sector del partido en donde se realiza la recolección un día determinado. Por ejemplo, los Viernes se realiza la recolección en la zona de Castelar. En la Figura 1.1 se pueden ver las diferentes zonas del partido de distinto color. En general, una zona es una de las localidades del partido, como por ejemplo, la zona de Castelar se corresponde a la localidad de Castelar.

Cuando se realiza la recolección de residuos, una zona puede dividirse en varios *cuadros*. La recolección es realizada por camiones diferentes, en forma paralela, en todos los cuadros de una zona particular. Por ejemplo, la zona de Castelar se divide en 8 cuadros, cada uno de los cuales es recorrido al mismo tiempo, por un camión diferente, durante los días Viernes.

Llamaremos *distribución* de una zona a la forma en que se divide la zona en cuadros. Puede representarse como un conjunto de cuadros. En otras palabras, la distribución de una zona determina a qué cuadro va a pertenecer cada cuadra de la zona. También puede verse como el resultado obtenido a partir de un proceso de zonificación.

Para una distribución dada, una cuadra puede pertenecer al borde del cuadro, por lo que las llamaremos *cuadras borde*. Un borde puede pertenecer a un cuadro (cuando se encuentra en el borde de la zona) o a dos (cuando se encuentra en el límite entre dos cuadros). Las cuadras borde que pertenecen a dos cuadros son recorridas por dos camiones diferentes. La municipalidad de Morón estableció que cada camión realiza la recolección en su lado (vereda) de la cuadra.

Llamaremos costo de un cuadro a la longitud del recorrido óptimo de dicho cuadro (en km). Una distribución también tiene un costo, el cual se calcula como el máximo entre los costos de los cuadros que la componen. Elegimos esta forma de calcular su costo debido a que todos los cuadros correspondientes a una zona son recolectados al mismo tiempo, por lo tanto, el tiempo de recolección de toda la zona es igual al máximo entre los tiempos de recolección de cada cuadro.

2.2. Obtención de datos

Como primer paso para la construcción del mapa del partido, se buscó en OpenStreetMap(OSM) la zona correspondiente al partido de Morón (Figura 2.1). Desde la misma web de OSM se puede exportar el sector a un archivo .osm[1] que luego sera abierto con el programa de edición JOSM.

[1] http://wiki.openstreetmap.org/wiki/.osm

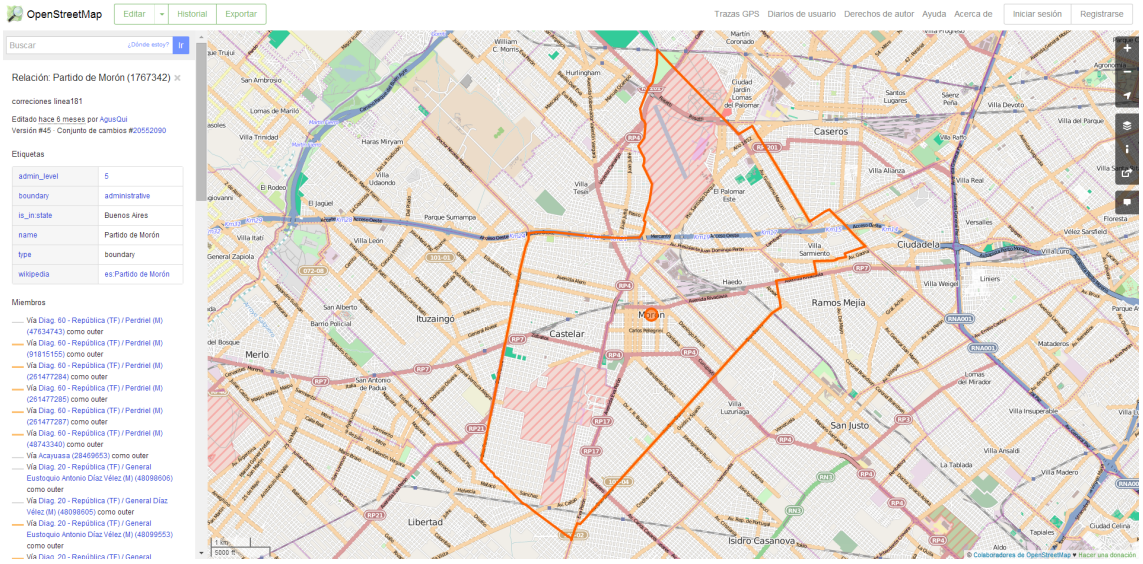


Fig. 2.1: Partido de Morón, provincia de Buenos Aires, visto desde OpenStreetMap

Un archivo .osm es un archivo XML de los datos primitivos (nodos, segmentos y relaciones) que son usados en el modelo de datos de OpenStreetMap. Su estructura es la siguiente:

- Un encabezado XML que indica la codificación (UTF-8) para el archivo.
- Un encabezado OSM que indica la versión de la API y el generador del archivo (por ejemplo, una herramienta de edición).
- Un bloque de nodos, en donde, para cada nodo, se guarda su ubicación en el sistema WGS84[2], y etiquetas (tags) usadas para informar diversas características del nodo (por ejemplo, si se encuentra un semáforo en esa ubicación).
- Un conjunto de segmentos o aristas que representan las cuadras. Un segmento está definido por dos nodos, uno para cada extremo del segmento. Los segmentos también tienen un conjunto de etiquetas que nos informan de, por ejemplo, el nombre de la cuadra, si es de doble o de un solo sentido, etc.
- Un conjunto de relaciones. Una relación es un conjunto de nodos o de aristas. No usaremos esta característica en la tesis.

A continuación vemos un ejemplo de un archivo .osm simplificado, para que el lector tenga una idea de cómo está formado.

[2] Sistema de coordenadas geográficas mundial que permite localizar cualquier punto de la Tierra (sin necesitar otro de referencia) por medio de tres unidades dadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <node id="100" lat="54.0901746" lon="12.2482632" user="userID"/>
 <node id="101" lat="54.0906309" lon="12.2441924" user="userID"/>
 ...
 <way id="1000" user="userID"
  <nd ref="100"/>
  <nd ref="101"/>
  <tag k="highway" v="residential"/>
  <tag k="name" v="StreetName"/>
 </way>
...
</osm>
```

Como se puede ver, en este archivo .osm se visualizan dos nodos con identificador 100 y 101, y una cuadra residencial que los une.

Una vez obtenido el archivo .osm de la zona del partido, se pasa al proceso de limpiado de los datos. Este proceso fue realizado a mano mediante la utilización del programa JOSM (Java OpenStreetMap Editor), un editor para los datos obtenidos a partir del sitio de OpenStreetMap. La limpieza consta de los siguientes pasos:

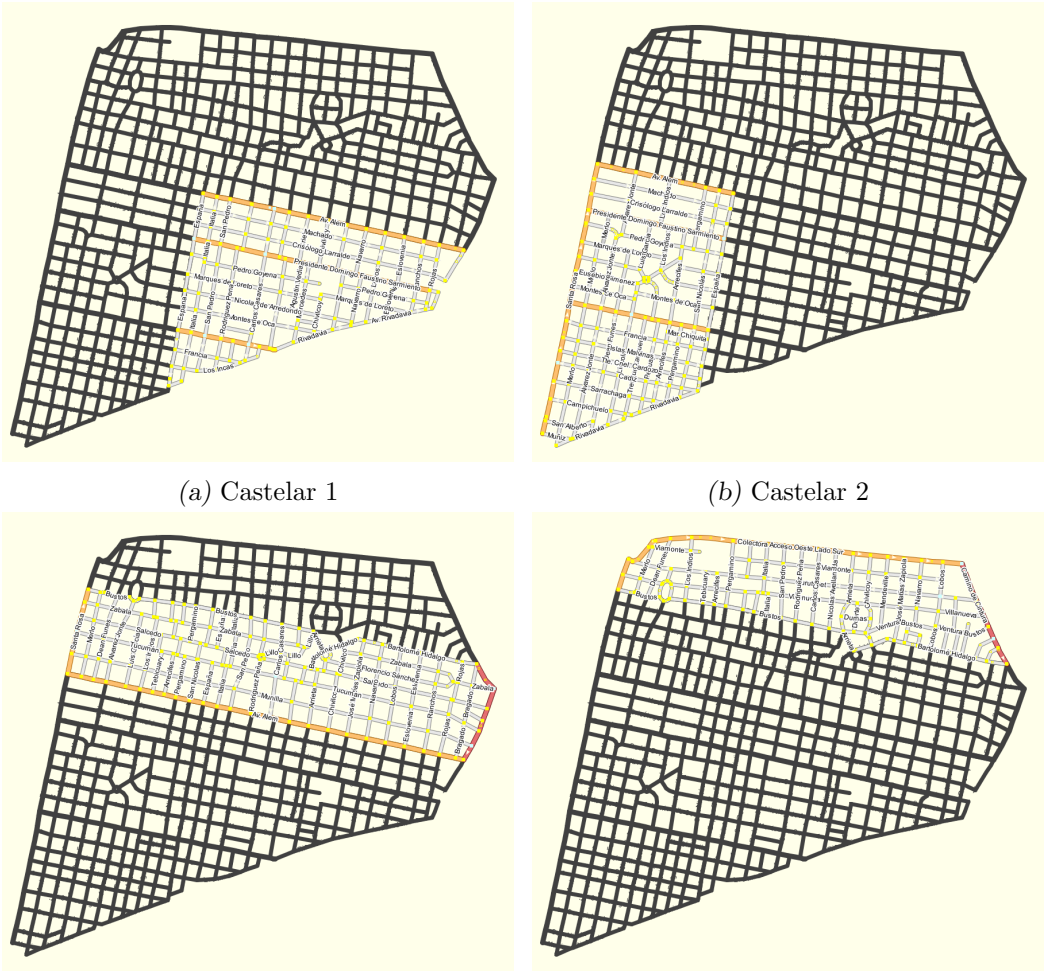
- Recorte de cada cuadro del partido como un archivo .osm particular. Se descartan los datos que están fuera de los límites del cuadro.
- Adición de la información proporcionada por la municipalidad, como por ejemplo, los sentidos de las calles.
- Corrección de datos erróneos, también utilizando datos provistos por la municipalidad.
- Adición de información necesaria para la realización de la tesis (por ejemplo, se establece una etiqueta para saber si una esquina puede ser "inicial", es decir, que el recorrido puede empezar en esa esquina. Se verán más detalles a lo largo de la tesis).

El objetivo de este proceso es obtener un conjunto de archivos *osm*, cada uno de los cuales representa un cuadro del partido. Estos archivos serán procesados más adelante por el programa que se implementa en esta tesis. En la Figura 2.2 se ve el resultado obtenido al finalizar la limpieza para los cuadros del norte de Castelar.

2.3. Modelo del mapa

Cada cuadro en los que se divide el partido es representado por un grafo mixto $G = (V, E, A)$, que se construye a partir del archivo .osm procesado como se vio en la sección anterior.

Los nodos (conjunto V) del grafo representan las esquinas en el mapa. Para cada nodo se guardan sus coordenadas geográficas y si hay un semáforo en esa esquina. Las aristas (E) y los arcos (A) del grafo representan las cuadras doble mano y de sentido único, respectivamente. Cada cuadra se identifica por los nodos que componen sus dos extremos. Además cada cuadra tiene un nombre y un sentido (puede ser de una mano, o doble-mano).



(a) Castelar 1 *(b)* Castelar 2

(c) Castelar 3 *(d)* Castelar 4

Fig. 2.2: Cuadros del sector norte de Castelar después de la limpieza de los datos

La mayoría de las cuadras puede representarse por un segmento entre sus dos esquinas. Sin embargo, hay cuadras que tienen una forma más compleja. Estas cuadras están representadas en el mapa por una lista de segmentos que se conectan entre sí de tal forma que la figura se asemeja a la forma de la cuadra. Un ejemplo de esta situación se ve en la Figura 2.3 en donde la cuadra con forma curva en Nicolás de Arredondo, es representada como la unión de varios segmentos rectos.

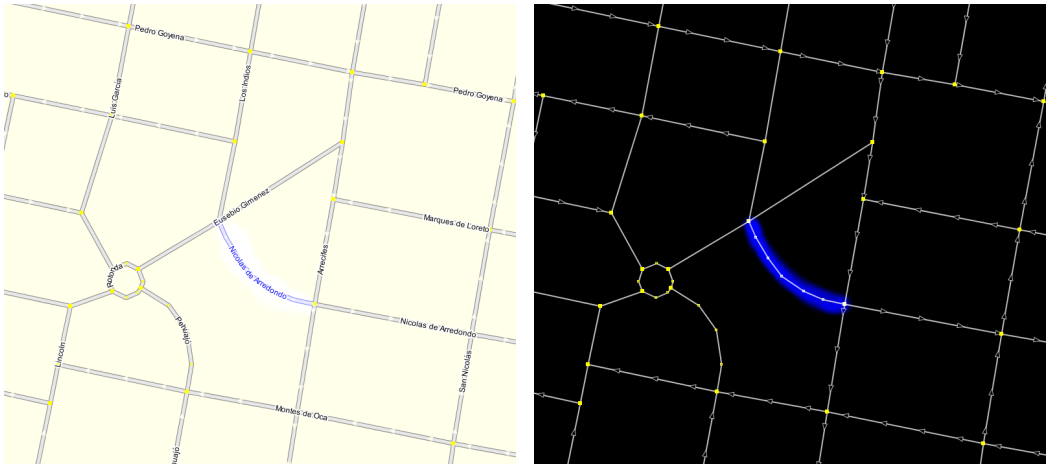


Fig. 2.3: Representación de una cuadra curva (resplandor azul) como la unión de 6 segmentos rectos

Por último, para calcular la distancia entre dos coordenadas geográficas se utilizó la siguiente fórmula [8]. Dados los puntos $p_1 = (lat_1, lon_1)$ y $p_2 = (lat_2, lon_2)$, donde lat_1 y lat_2 indican la latitud de p_1 y p_2, respectivamente, y lon_1 y lon_2 indican la longitud de los puntos p_1 y p_2, respectivamente, sobre la superficie terrestre.

Definimos :

$$a(p_1, p_2) = \left(sen\left(\frac{lat_2 - lat_1}{2}\right)\right)^2 + cos(lat_1) * cos(lat_2) * \left(sen\left(\frac{lon_2 - lon_1}{2}\right)\right)^2 \quad (2.1)$$

Entonces la distancia entre p_1 y p_2 es:

$$dist(p_1, p_2) = 6371000 * atan2(\sqrt{a}, \sqrt{1-a}) \quad (2.2)$$

Donde la función *atan2* es la función arcotangente con dos argumentos. Para cualquier par de números reales x e y distintos de cero, $atan2$ se define como el ángulo en radianes entre el eje x positivo de un plano y el punto dado por las coordenadas (x, y).

El valor 6371000 se debe al radio de la Tierra en metros.

3. RESOLUCIÓN DEL PROBLEMA

3.1. Estrategia de resolución

Para resolver el problema planteado, una vez obtenidos los datos del mapa de cada cuadro, se construye un modelo de programación lineal entera que luego será resuelto por el solver no comercial SCIP[1]. Cada cuadro es resuelto de manera independiente al resto. A continuación se detallan los modelos que se fueron utilizando a lo largo de la implementación de la solución junto con los problemas que fueron surgiendo y cómo se solucionaron.

3.2. El modelo inicial

El primer modelo de programación lineal entera que se usó para tratar de resolver el problema planteado fue construido en base a los presentados en la bibliografía [9] [15], con la modificación de que buscamos un camino, en lugar de un ciclo.

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. Definimos E^- y E^+ como copias del conjunto E que representan los dos sentidos de las cuadras doble mano. Así, si $ij \in E$, entonces $ij \in E^-$ y $ji \in E^+$. Al darles dirección a estas aristas, podemos decir que E^- y E^+ son conjuntos de arcos. Si $e \in E$, llamaremos e^- y e^+ a los arcos que representan los dos sentidos de la arista e, en donde $e^- \in E^-$ y $e^+ \in E^+$.

Definimos $c : A \cup E^- \cup E^+ \to \mathbb{R} > 0$ como la función de costo para cada arista y arco del grafo. Para describir el modelo, definimos $\delta^-(v)$ para grafos mixtos como el conjunto de arcos $a \in A \cup E^- \cup E^+$ que tienen al nodo $v \in V$ como su cabeza, y $\delta^+(v)$ como el conjunto de arcos $a \in A \cup E^- \cup E^+$ que tienen a v como su cola.

Definimos el siguiente modelo de programación lineal entera. La variable x_{ij} representa la cantidad de veces total que se pasa por el arco o arista ij. Mientras que las variables s_v y t_v indican si un nodo v es el primero y/o el último del recorrido, respectivamente.

Minimizar:

$$\sum_{ij \in A \cup E^- \cup E^+} x_{ij} * c_{ij} \tag{3.1}$$

Sujeto a:

$$x_a \geq 1 \qquad \forall a \in A \tag{3.2}$$

$$x_{e^-} + x_{e^+} \geq 1 \qquad \forall e \in E \tag{3.3}$$

$$\sum_{a \in \delta^-(v)} x_a + s_v = \sum_{a \in \delta^+(v)} x_a + t_v \qquad \forall v \in V \tag{3.4}$$

[1] http://scip.zib.de/

$$\sum_{v \in V} s_v = 1 \tag{3.5}$$

$$\sum_{v \in V} t_v = 1 \tag{3.6}$$

$$x_a \in \mathbb{Z}^+ \qquad \forall a \in A \cup E^- \cup E^+ \tag{3.7}$$

$$s_v \in \{0, 1\} \qquad \forall v \in V \tag{3.8}$$

$$t_v \in \{0, 1\} \qquad \forall v \in V \tag{3.9}$$

La función objetivo es minimizar el costo total del recorrido (la suma de los costos de las cuadras por las que pasó). Si bien el objetivo real es minimizar el tiempo que se tarda en realizar el recorrido, al minimizar la distancia se consigue reducir el tiempo, siendo la distancia más fácil de trabajar.

La primera restricción (3.2) nos indica que se debe pasar por cada arco del grafo al menos una vez, mientras que la 3.3 se aplica a las aristas y obliga a la solución a pasar por cada arista al menos una vez (en alguna de las dos direcciones).

La restricción (3.4) impone que la solución sea efectivamente un camino, agregando una condición de flujo o continuidad en la solución del modelo. Así, la cantidad de veces que se "entra" en un nodo debe ser igual a la cantidad de veces que se "sale", a excepción del nodo inicial y final, en los que se entra y se sale una vez menos, respectivamente. Vale aclarar que $\delta^-(v) \subseteq A \cup E^- \cup E^+$ y $\delta^+(v) \subseteq A \cup E^- \cup E^+$, es decir, el camino puede estar formado tanto por arcos como por aristas.

Por último, las restricciones (3.5) y (3.6) aseguran que el nodo inicial y el final sean únicos (sólo se permite que s_v sea igual a 1 para un solo nodo, y de la misma forma para t_v).

Sin embargo este modelo tiene un problema, pues permite giros en U (pasar por una arista en una dirección e inmediatamente después volver por esa misma arista en la dirección contraria), con lo cual no cumple con los objetivos propuestos. Un ejemplo de este comportamiento se puede observar en la Figura 3.1.

3.3. Los giros en U

Los giros en U son prohibidos por la regulaciones de tránsito en la República Argentina, por lo tanto también deben ser prohibidos en nuestro recorrido. Para lograrlo se "expande" el grafo que representa al cuadro. Este proceso consiste en crear un grafo dirigido que contiene un arco por cada arco en el grafo original, dos arcos por cada arista del grafo original (uno por sentido), y arcos auxiliares que unen las transiciones permitidas de una cuadra a otra. En la Figura 3.2 se ve un ejemplo sencillo de cómo se expande una esquina del grafo en donde se cruza una calle doble-mano con una de sentido único.

De esta forma se agregan arcos al grafo por los que no es necesario que el recorrido pase. Estos arcos representan los giros validos para el recorrido, y los llamaremos *arcos auxiliares* (en la Figura 3.2, estos arcos se ven en líneas punteadas). Estos arcos auxiliares en principio no tienen costo, o más precisamente, su costo es equivalente a la realización del giro. Debido a que la municipalidad nos informó que los camiones no tienen problemas

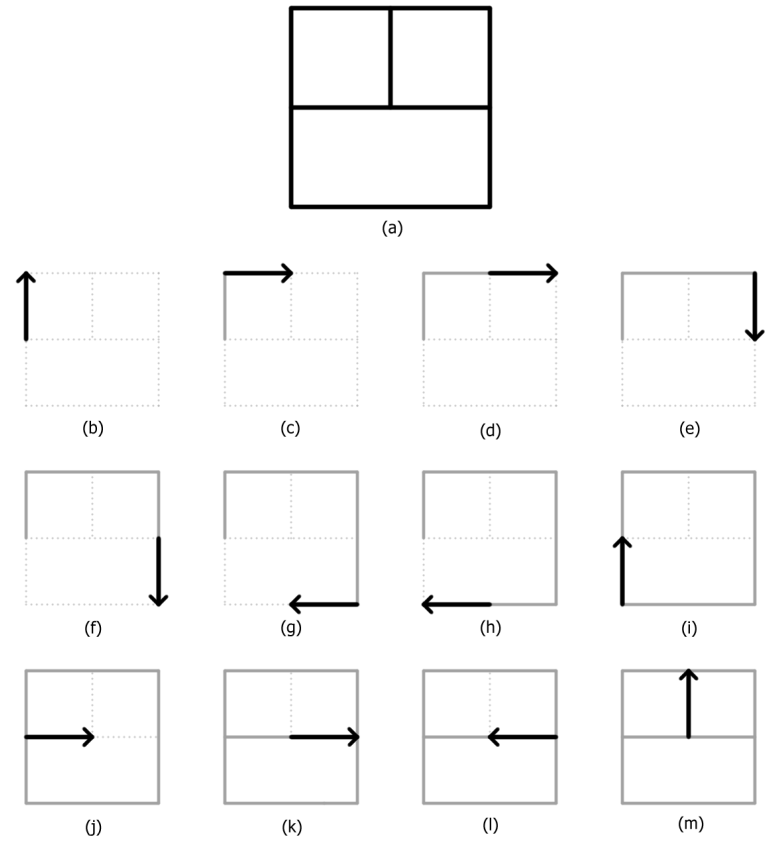


Fig. 3.1: Un ejemplo de cómo se realiza un giro en U en una solución del primer modelo sobre el grafo de la subfigura (a). El giro en U se realiza en las subfiguras (k) y (l).

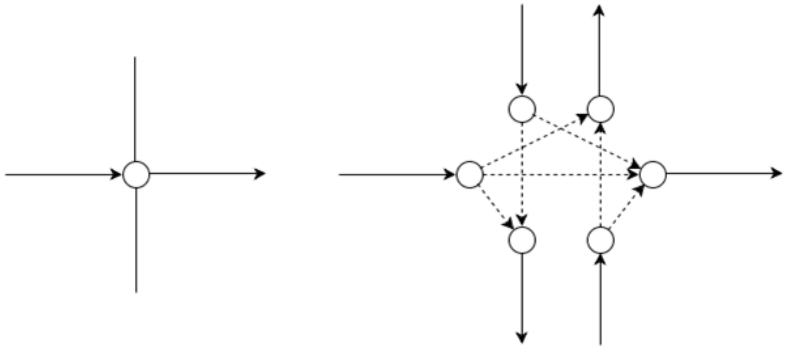


Fig. 3.2: Expansión de un nodo del grafo

al realizar giros en las esquinas, establecemos el peso de todos los arcos auxiliares en 1 (en principio se podría usar 0, pero debido a los problemas que pueden traer arcos con peso nulo, como por ejemplo, ciclos infinitos de costo 0, se optó por usar un peso igual a 1). De todas formas consideraremos que el costo de los arcos auxiliares es despreciable.

Dado el grafo mixto original $G = (V, E, A)$, obtenemos el digrafo expandido $G' = (V', X)$. Notar que en la expansión cambia el conjunto de vértices del grafo, debido a

que al expandir una esquina, ésta puede dividirse en varios nodos. Usaremos la notación a' para referirnos al arco en el digrafo expandido que se forma a partir del arco $a \in A$ (perteneciente al grafo mixto original). Análogamente, definimos e'^- y e'^+ para referirnos al arco que se forma a partir de e^- y e^+ (perteneciente al grafo mixto original.)

Debido a la forma en que se construye el digrafo expandido, el conjunto de arcos del digrafo X cumple la siguiente igualdad: $X = A' \cup E'^- \cup E'^+ \cup F$, en donde A', E'^- y E'^+ son los conjuntos formados a partir de A, E^- y E^+, respectivamente, luego de la expansión, y F es el conjunto de arcos auxiliares que se agregan al digrafo para representar los giros válidos en las esquinas.

A continuación se presenta un segundo modelo preliminar que presenta algunos problemas que se mencionan en la próxima sección. Este modelo está formado a partir del digrafo expandido $G' = (V', X)$:

Minimizar:

$$\sum_{ij \in X} x_{ij} * c_{ij} \tag{3.10}$$

Sujeto a:

$$x_{a'} \geq 1 \qquad \forall a \in A \tag{3.11}$$

$$x_{e'^-} + x_{e'^+} \geq 1 \qquad \forall e \in E \tag{3.12}$$

$$\sum_{a \in \delta^-(v)} x_a + s_v = \sum_{a \in \delta^+(v)} x_a + t_v \qquad \forall v \in V' \tag{3.13}$$

$$\sum_{v \in V'} s_v = 1 \tag{3.14}$$

$$\sum_{v \in V'} t_v = 1 \tag{3.15}$$

$$x_a \in \mathbb{Z}^+ \qquad \forall a \in X \tag{3.16}$$

$$s_v \in \{0, 1\} \qquad \forall v \in V' \tag{3.17}$$

$$t_v \in \{0, 1\} \qquad \forall v \in V' \tag{3.18}$$

Principalmente, el modelo cambia de la siguiente manera: se define la variable x también para el conjunto de arcos auxiliares, y se agregan estos arcos en el calculo del costo del recorrido (función objetivo 3.10). Se debe tener en cuenta que $\delta^-(v)$ y $\delta^+(v)$ también incluyen arcos auxiliares.

3.4. Los sub-ciclos

Sin embargo el segundo modelo propuesto presenta el problema de la aparición de sub-ciclos. Es decir, una solución del modelo puede contener ciclos no unidos al resto del recorrido. Esto se debe sencillamente a que las restricciones impuestas al modelo permiten esta clase de soluciones.

Por ejemplo, si se resuelve el modelo sobre el grafo de la Figura 3.3, se podría llegar a una solución como la de la Figura 3.4, en donde se forma un camino principal (azul) cuya secuencia de nodos es [4,1,2,3,6,9,8,7,4,5,2] y un subciclo (rojo) cuya secuencia es [5,6,9,8,5]. Tener en cuenta que cada nodo de la figura en realidad está compuesto por varios nodos debido a que el grafo se encuentra "expandido", pero por simplicidad fueron dibujados como un único nodo. Como se puede ver en la ampliación del nodo 5, el subciclo está separado del resto del recorrido. Una situación similar se podría observar en los nodos 6, 9 y 8.

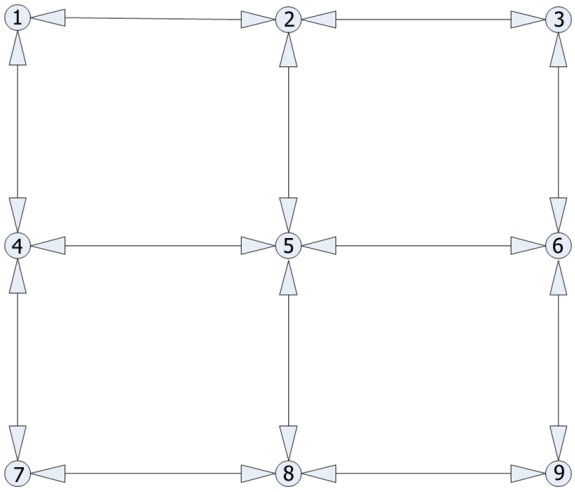


Fig. 3.3: Grafo que representa un cuadro simple formado por cuatro manzanas, cuyas cuadras son todas de doble sentido

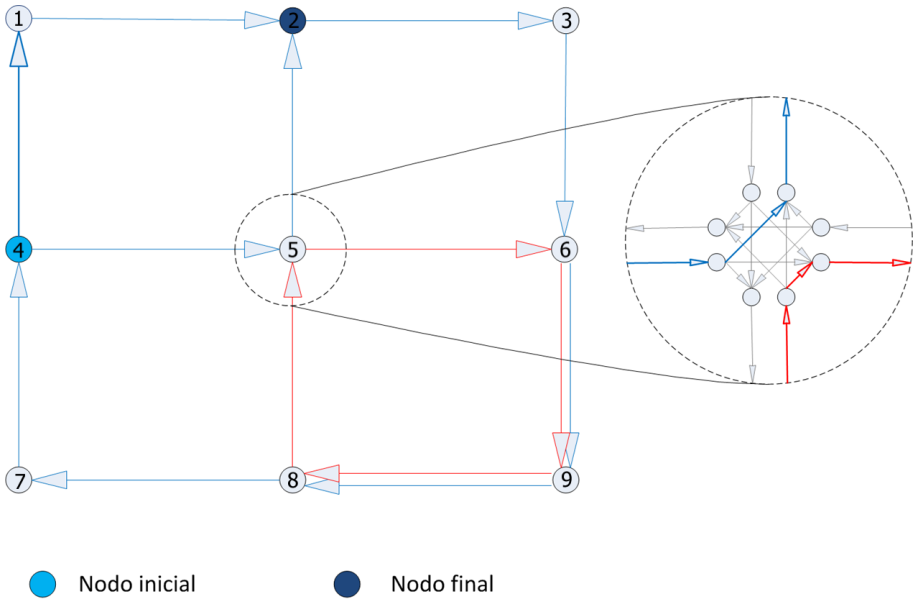


Fig. 3.4: Formación de un subciclo en una solución del modelo para el grafo de la Figura 3.3

Para solucionar este problema se puede agregar una restricción por cada subciclo que se puede formar. Esta restricción pide que haya al menos un arco ij que conecte los nodos que forman el subciclo con alguno de los demás nodos (es decir, que exista un arco ij

tal que i esté en el subciclo y j no, o viceversa). Esta condición debe cumplirse para cada subciclo posible en el grafo, dando lugar a una cantidad exponencial de restricciones. A continuación presentamos el último modelo construido, que incluye la restricción que prohíbe subciclos.

$$\sum_{ij \in X} x_{ij} * c_{ij} \tag{3.19}$$

Sujeto a:

$$x_a \geq 1 \qquad \forall a \in A' \tag{3.20}$$

$$x_{e'^-} + x_{e'^+} \geq 1 \qquad \forall e \in E \tag{3.21}$$

$$\sum_{a \in \delta^-(v)} x_a + s_v = \sum_{a \in \delta^+(v)} x_a + t_v \qquad \forall v \in V' \tag{3.22}$$

$$\sum_{v \in V'} s_v = 1 \tag{3.23}$$

$$\sum_{v \in V'} t_v = 1 \tag{3.24}$$

$$\sum_{\substack{ij \in X, \\ i \in S, j \notin S \ \vee \\ i \notin S, j \in S}} x_{ij} \geq 1 \qquad \forall S \in \mathcal{P}(V'),\ 0 < |S| < |V'|,\ S \textit{ subciclo} \tag{3.25}$$

$$x_a \in \mathbb{Z}^+ \qquad \forall a \in X \tag{3.26}$$

$$s_v \in \{0, 1\} \qquad \forall v \in V' \tag{3.27}$$

$$t_v \in \{0, 1\} \qquad \forall v \in V' \tag{3.28}$$

Donde $\mathcal{P}(V')$ se llama *conjunto potencia* y es el conjunto formado por todos los subconjuntos del V'.

Debido a la cantidad exponencial de restricciones, generar explícitamente todas las restricciones de este modelo es imposible en la práctica. En su lugar, se puede agregar las restricciones de a una, dinámicamente. Cada vez que se detecta un subciclo $S = [v_0, ..., v_n]$ en la solución, se agrega una restricción para evitar ese subciclo en particular, y se vuelve a resolver el modelo junto con la nueva restricción. Este proceso se repite hasta que se encuentre una solución válida, es decir, sin subciclos.

Sin embargo, aunque este algoritmo es posible de ser aplicado en la práctica, en algunos casos requiere un gran número de iteraciones hasta encontrar una solución sin subciclos. En este trabajo se implementó un método para reducir la cantidad de iteraciones necesarias para llegar a una solución válida. El método consiste en tratar de unir los subciclos al camino principal mediante un intercambio de arcos auxiliares. Por ejemplo, si tomamos la solución de la Figura 3.4, podemos intercambiar dos arcos auxiliares en el nodo 5 (como se puede ver en la Figura 3.5) y así unir el subciclo al camino principal, logrando el recorrido válido [4,1,2,3,6,9,8,7,4,5,6,9,8,5,2]. Debido a que su costo es el mismo para todos los arcos auxiliares, el camino luego del intercambio de arcos sigue manteniendo su optimalidad.

Este método, a pesar de ser muy eficiente, no es aplicable a todos los subciclos. Por ejemplo, si tomamos el mapa de la Figura 3.6 y resolvemos el modelo sobre dicho grafo,

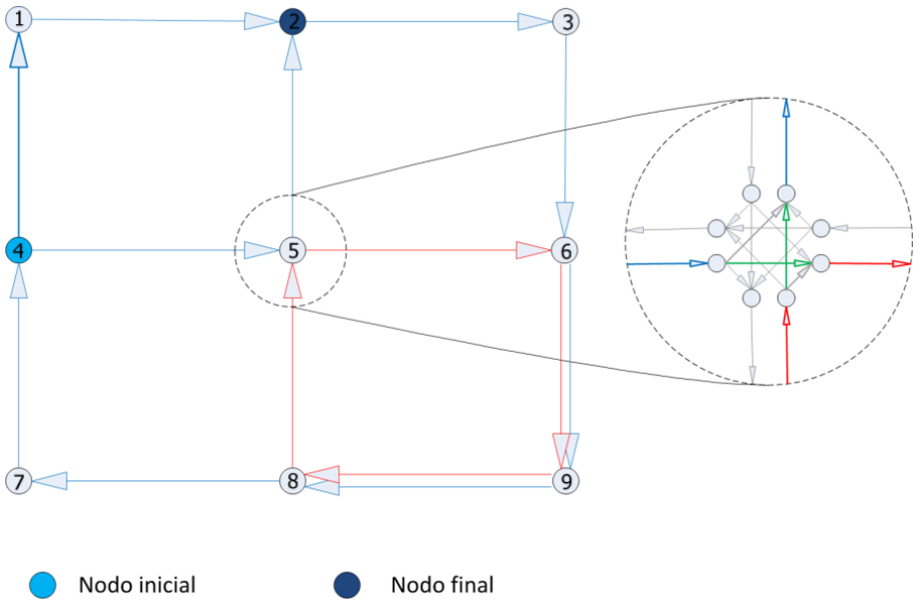


Fig. 3.5: Unión de subciclos para el recorrido de la Figura 3.3

podemos obtener como resultado el recorrido de la Figura 3.7. En este caso el subciclo (marcado en rojo) no se puede unir al recorrido principal (azul) por los métodos vistos. Debido a esto la unión de subciclos se combina con las restricciones dinámicas: si la solución no presenta subciclos, retorna la solución sin modificar; en cambio, si encuentra al menos un subciclo y si se puede aplicar el método para unir todos los subciclos, se aplica y se devuelve la solución; en el caso en que no se pueda aplicar, se agregan las restricciones para evitar esos subciclos particulares y se vuelve a resolver el modelo, repitiendo el procedimiento.

El objetivo de este proceso no es eliminar completamente la necesidad de agregar las restricciones para evitar subciclos, sino reducir el número de iteraciones hasta encontrar una solución válida (es decir, sin subciclos).

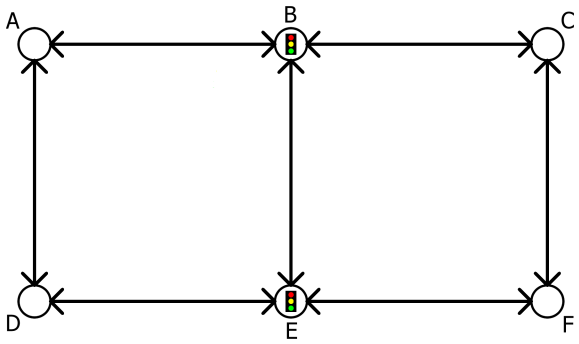
En la Figura 3.8 se ve un diagrama de flujo de este proceso iterativo.

3.5. Cuadras cerradas

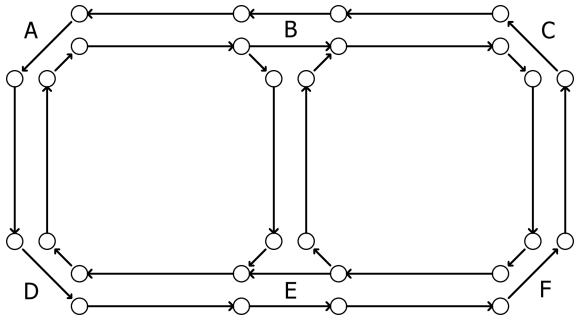
En las cuadras sin salida se deben aceptar los giros en U para que el recorrido no quede "estancado" en esas cuadras. Con este fin, se realizan pequeñas modificaciones al grafo agregando arcos auxiliares en esos nodos sin salida. En otras palabras, se permiten giros en U en cuadras cerradas (Figura 3.9).

3.6. Giros prohibidos en las esquinas con semáforos

En las esquinas con semáforos deben seguirse restricciones adicionales. Debido a las reglas de tránsito, desde una cuadra doble mano no se debe doblar a la izquierda si en esa esquina hay un semáforo común. Gracias a la forma en que se expande el grafo (Sección 3.3) agregar estas restricciones es sencillo, simplemente se suprimen del grafo los



(a)



(b)

Fig. 3.6: Ejemplo de grafo simple y el expandido correspondiente

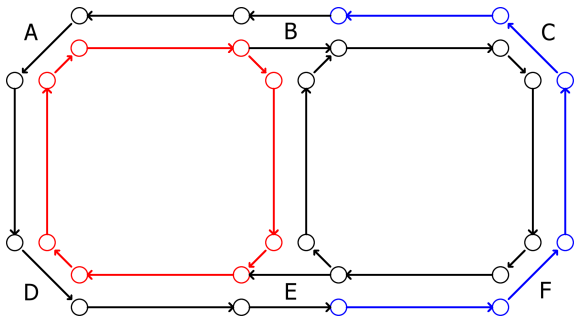


Fig. 3.7: Un recorrido con subciclos imposibles de unir mediante nuestro método, para el grafo de la Figura 3.6

arcos auxiliares que corresponden a los giros inválidos. Se puede ver en la Figura 3.10 cómo se eliminan los arcos auxiliares en el cruce de dos calles doble mano con semáforos.

3.7. Elección del punto de inicio

El punto de inicio del recorrido no se elige arbitrariamente, sino que se establecen ciertos criterios que deben cumplir los candidatos a "primer nodo". Debe estar en el borde del cuadro y del lado del cuadro más cercano al punto de salida (el obrador). Se

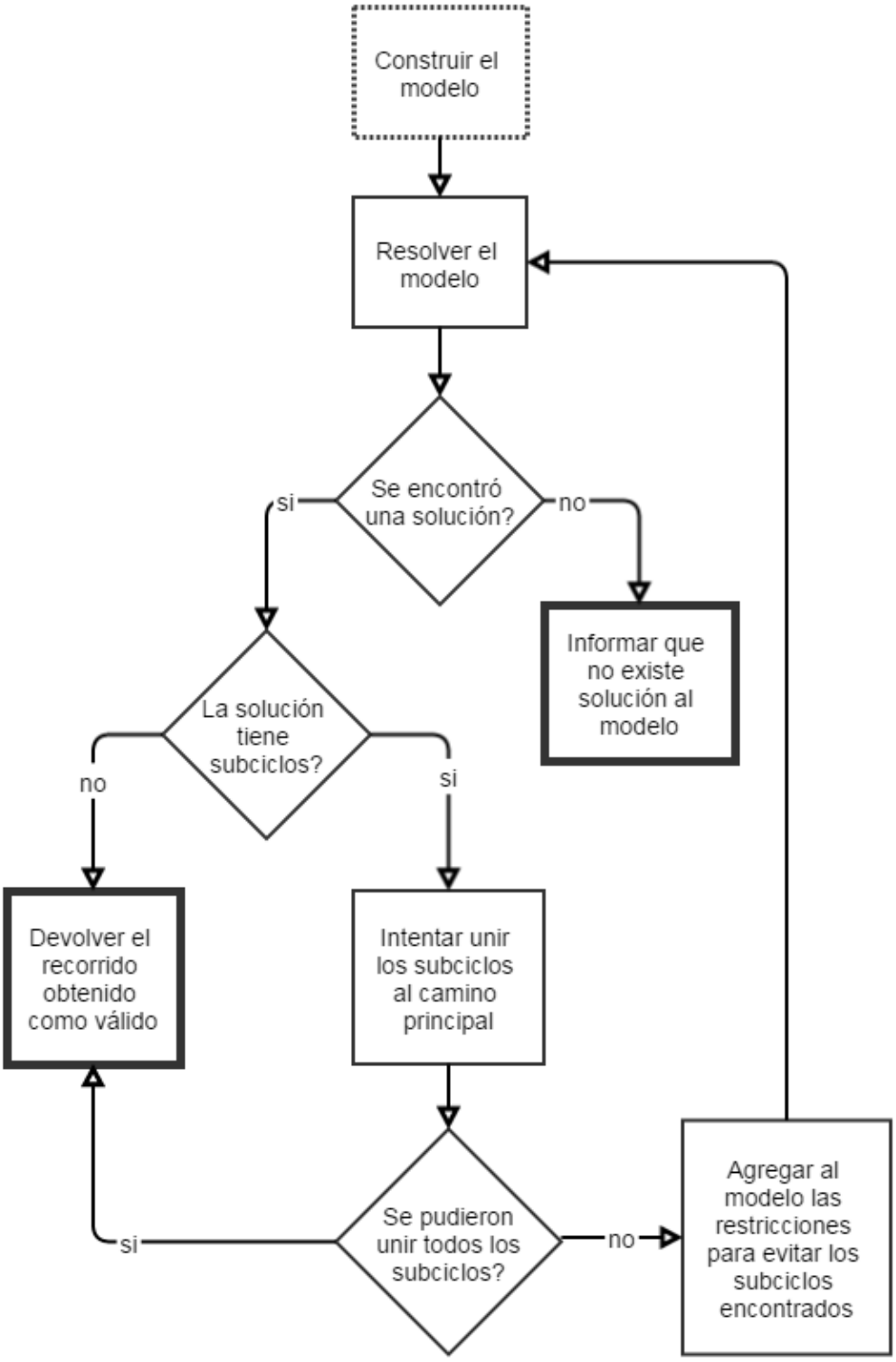


Fig. 3.8: Diagrama de flujo para el algoritmo principal que resuelve el problema de la generación de subciclos. Los procesos marcados con un borde más grueso son finales

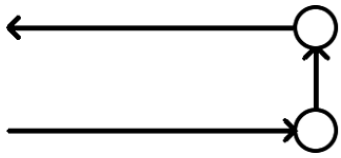
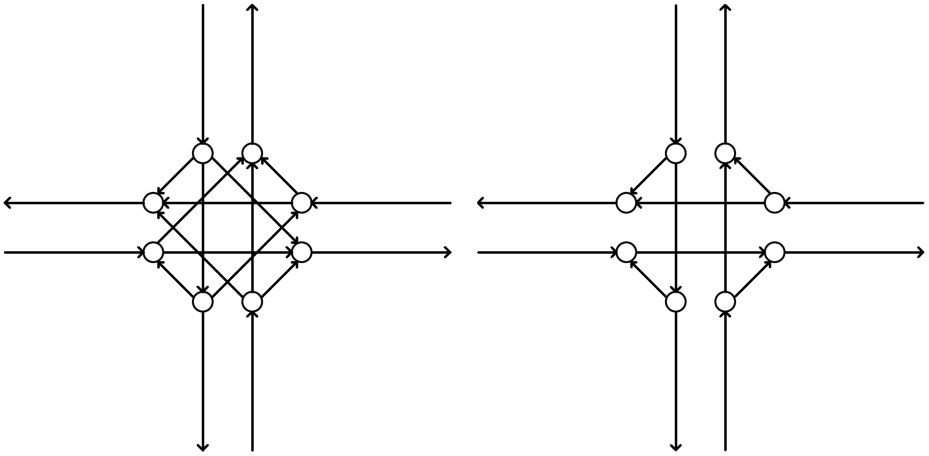


Fig. 3.9: Giros permitidos en una cuadra cerrada



(a) *(b)*

Fig. 3.10: Giros válidos en un cruce sin semáforos (a) y en un cruce con semáforos (b)

podría utilizar el algoritmo de Dijkstra para encontrar el nodo inicial óptimo, calculando la distancia desde el obrador a todos los nodos del cuadro, y agregar estos datos al modelo. Pero debido a que a la municipalidad de Morón le interesa el tiempo que pasa a partir de que se inicia el recorrido en el cuadro, y no a partir de que se sale del obrador, no se implementó esta funcionalidad.

3.8. Borde del cuadro

En algunos casos, puede ser más efectivo salir del cuadro en algún momento del recorrido y luego volver a entrar, en lugar de mantener el recorrido siempre dentro del cuadro. En general, estos casos se dan en cuadros con muchas cuadras de una sola mano, en los que para llegar a una esquina es necesario realizar grandes rodeos, que podrían ser más pequeños al realizar un rodeo fuera del cuadro.

Para resolver este problema se incluye un "borde" alrededor del cuadro, de cuadras por las que no es necesario realizar la recolección, pero que pueden usarse para reducir el costo del recorrido.

La inclusión de esta característica no produce cambios bruscos en el modelo, debido a que estas cuadras adicionales pueden ser tratadas de la misma forma que los arcos auxiliares. Se trata, simplemente, de marcar estas cuadras con una etiqueta que indica

que son "auxiliares", aunque teniendo en cuenta que conservan su peso original.

En la Figura 3.11 puede verse un cuadro perteneciente a la localidad de Villa Sarmiento. Las cuadras marcadas en líneas punteadas verdes son aquellas que tienen activada la etiqueta de auxiliar, y por lo tanto, pueden ser usadas pero no es necesario realizar la recolección en ellas. Como observación, se puede notar que no se agregaron cuadras auxiliares al sur y norte del cuadro. Esto se debe a que al sur limita con otro partido (fuera del partido de Morón) y por lo tanto no se tienen datos confiables de esa zona, mientras que al norte se encuentra una autopista y no hay cuadras que permitan el cruce.

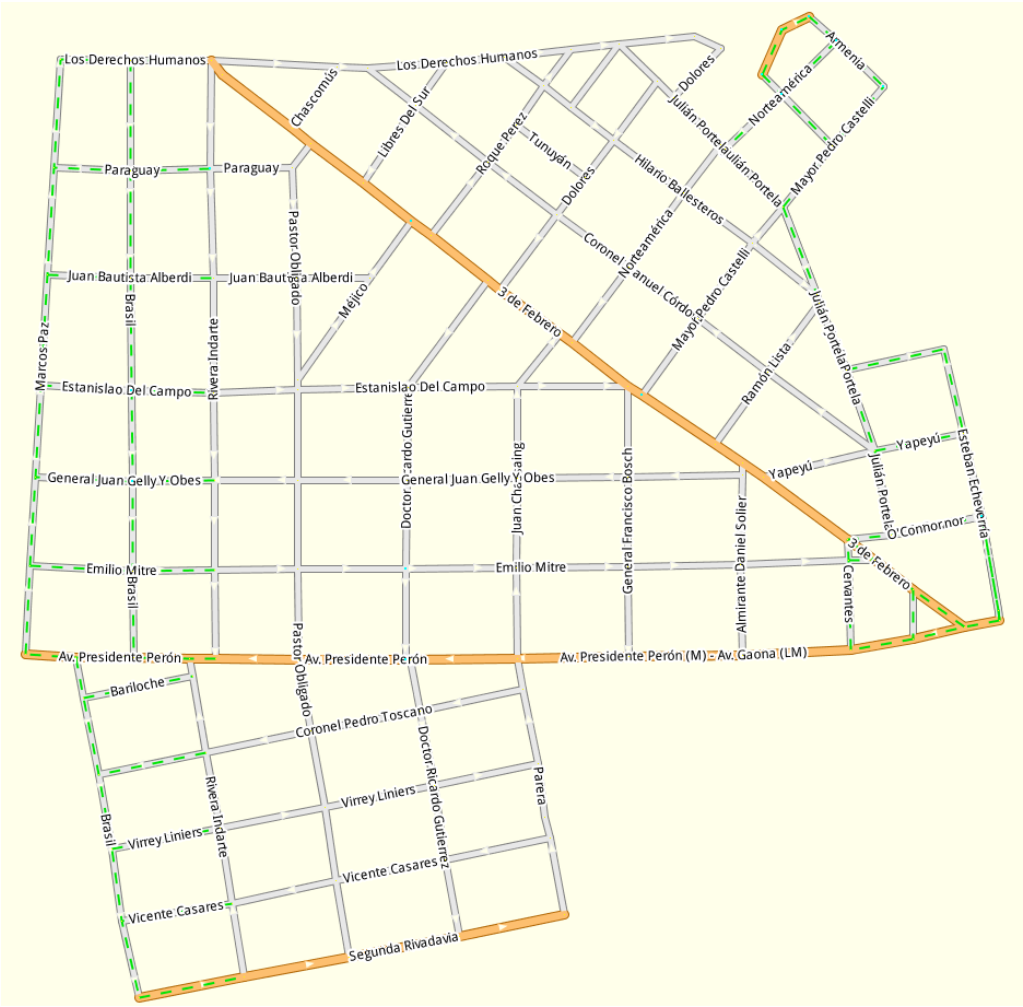


Fig. 3.11: Cuadras auxiliares (verde) alrededor de cuadro de la localidad de Villa Sarmiento

3.9. Construcción de la ruta

Una vez obtenido el resultado del modelo se construye la ruta a partir de los valores de las variables x_{ij}, s_v y t_v. Construimos un multigrafo dirigido a partir de los valores de x_{ij}, en donde hay x_{ij} aristas que salen del nodo i y llegan al nodo j. Es decir, $\#(i,j) = x_{i,j}$. Al construirlo de esta forma, se forma un multigrafo dirigido con un camino euleriano que se puede encontrar en tiempo polinomial, ya que la solución del modelo nos garantiza que

hay un camino que pasa por todas las aristas del multigrafo exactamente una vez. Las variables s_v y t_v se usan para indicar el nodo inicial y final del recorrido, respectivamente.

En la sección sobre detalles de implementación se verá que en realidad no se arma el grafo descripto, sino que se usa como ayuda visual para entender e implementar el algoritmo.

En la Figura 3.12 se ve un caso particular simple de cómo se construye el multidigrafo euleriano a partir de los valores de x_{ij}, s_v y t_v. Los valores de estas variables son obtenidos al encontrar una solución óptima del modelo.

$x_{1,2} = 1$
$x_{2,3} = 1$
$x_{4,5} = 1$
$x_{5,6} = 1$
$x_{8,7} = 1$
$x_{9,8} = 2$
$x_{4,1} = 1$
$x_{5,2} = 1$
$x_{3,6} = 1$
$x_{7,4} = 1$
$x_{8,5} = 1$
$x_{6,9} = 2$
$s_4 = 1$
$t_2 = 1$

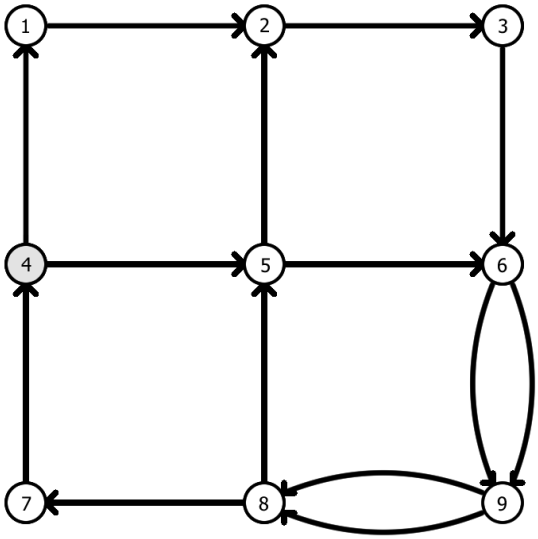


Fig. 3.12: Multigrafo dirigido euleriano que se obtiene a partir de una solución del modelo. Recorrido resultante: $[4, 1, 2, 3, 6, 9, 8, 7, 4, 5, 6, 9, 8, 5, 2]$

3.10. Rezonificación

Como se mencionó en la introducción, existe un objetivo adicional que consiste en mejorar la distribución en cuadros de cada zona para equilibrar los costos de recolección en cada cuadro, y reducir el tiempo total que dura la recolección. Para lograr este objetivo se realiza una búsqueda tabú con intercambio de manzanas entre los cuadros de una zona.

El proceso de rezonificación consiste en elegir dos cuadros adyacentes en la misma zona. De uno de los cuadros se toma una manzana en el límite entre ambos cuadros, y se la cambia al otro. Así se reduce el tamaño del primer cuadro y se aumenta el del segundo. Luego se calcula el costo del recorrido óptimo en los dos cuadros modificados. Si se mejoró la solución, se guarda el resultado, y se continúa iterando.

Decimos que una solución es mejor que otra, si el costo máximo entre los costos de

todos los cuadros de la zona es menor al de la otra solución. Decidimos usar este criterio debido a que el cuadro de mayor costo es el que realmente determina cuánto se tarda en realizar el recorrido de toda la zona, debido a que la recolección para todos los cuadros de una zona determinada se realiza al mismo tiempo.

Usando el algoritmo de búsqueda tabú se mantiene una lista de las manzanas recientemente intercambiadas para no volver a intercambiarlas en los próximos pasos del algoritmo. Así se evita estar moviendo una manzana particular de un cuadro a otro y de vuelta al primero.

Se determinan diferentes condiciones de parada para finalizar la búsqueda. Por ejemplo, se establece un tiempo máximo de ejecución, y un número máximo de iteraciones durante las cuales si la búsqueda no mejora, se reinicia el algoritmo.

Se optó por realizar un intercambio de manzanas en lugar de cuadras, debido a que se quiere mantener una forma agradable a la vista de los cuadros. Si usáramos cuadras, se podrían formar cuadros de formas muy difíciles de ver en papel, lo que desalentaría a los conductores de los camiones. Dada esa observación, se decidió utilizar manzanas para conservar en alguna medida la forma más "regular" que tienen los cuadros. Así, en el caso en que el recorrido sea interrumpido por alguna razón, el conductor tendrá una clara imagen del cuadro que le corresponde.

Un detalle importante a tener en cuenta es que en lugar de mover las manzanas de un cuadro a otro, pueden ser marcadas como cuadras auxiliares (Sección 3.8). De esta forma pueden ser usadas en el recorrido, pero no es necesario pasar por ellas.

En el próximo capítulo (Capítulo 4), se ve en detalle cómo se implementaron los diversos algoritmos explicados en esta sección.

4. DETALLES DE LA IMPLEMENTACIÓN

En este capítulo describiremos los algoritmos principales en mayor detalle. Para ello, podemos dividirlo en dos partes. En la primera veremos los algoritmos relacionados con la búsqueda tabú, y en la segunda, aquellos vinculados con el cálculo de la ruta óptima dentro de un cuadro.

Antes de pasar al detalle de cada algoritmo, veamos una lista de las tareas que realiza el programa:

1. Leer los datos del mapa de una zona determinada.
2. Construir la estructura de datos que representa la zona, como un conjunto de cuadros, donde cada cuadro es un conjunto de manzanas.
3. Mejorar la distribución de la zona realizando una búsqueda tabú.
   a) Cambiar una manzana de cuadro.
   b) Calcular el costo de los cuadros modificados.
   c) Analizar si se mejoró el costo de la distribución.
   d) Guardar la nueva distribución si se mejoró su costo.
4. Para cada cuadro que forma la distribución elegida, guardar la ruta óptima calculada.

La parte importante del algoritmo es la que se realiza en las tareas 2 y 3, y es lo que veremos en mayor detalle.

Durante la tarea 1 se leen los datos de un archivo *.osm* (datos XML obtenidos de OpenStreetMap). Los datos se procesan como un conjunto de esquinas y un conjunto de cuadras. Aunque las llamamos cuadras, siendo más estrictos, deberíamos llamarlos segmentos de cuadra, ya que una cuadra en el mapa de OpenStreetMap puede estar dividida en varios segmentos (Figura 2.3). A pesar de esto, seguiremos llamando *cuadras* a los segmentos de cuadra por simplicidad.

Los datos que se guardan de cada esquina son:

- Etiqueta: un número que identifica a la esquina.
- Coordenadas geográficas: valores de las coordenadas geográficas del punto en el mapa. Estos valores son importantes, pues serán utilizados para, por ejemplo, calcular la longitud de una cuadra.
- Semáforo: se guarda un valor booleano para indicar si en esa esquina se encuentra o no un semáforo.
- Inicial: otro valor booleano que indica si el recorrido puede comenzar en esa esquina.

Los datos de esta lista se obtienen directamente a partir del archivo *.osm*, a excepción del valor del campo "inicial". Este valor queda determinado por dos criterios: en primer lugar, sólo tomaremos como iniciales las esquinas que se encuentran al borde de un cuadro. En segundo lugar, de todas las esquinas que cumplen el primer criterio, solamente tomaremos aquellas que se encuentran más cerca del punto de partida. Agregamos estos criterios con la intención que se elija una esquina que esté en la cara del cuadro frente al punto de partida. De esta forma, el camión no debe pasar por cuadras del cuadro antes de realizar la recolección, por el solo hecho de llegar al punto de partida.

Para decidir las esquinas que se encuentran más cerca del punto de partida utilizamos la distancia euclideana entre los dos puntos. Usamos este método por su sencillez, pero en caso de que se quiera una estimación mucho más precisa, se podría utilizar el algoritmo de Dijkstra para calcular el costo de llegar desde el punto de partida a cada punto en el borde del cuadro. Luego, agregaríamos ese costo al modelo, que lo tendría en cuenta al encontrar el recorrido óptimo.

Para las cuadras se guardan los siguientes datos:

- Identificador: un identificador de la forma (i, j) donde i, j son las etiquetas de las esquinas que une la cuadra.
- Nombre: el nombre de la calle a la que pertenece la cuadra.
- Sentido: valor booleano que indica si la cuadra es de un solo sentido (valor True) o de doble sentido (valor False).
- Auxiliar: valor booleano que indica si es una cuadra auxiliar, es decir, por donde no es necesario realizar la recolección.

Una vez procesados los datos del mapa se pasa a la tarea 2. Por lo tanto, el siguiente paso es representar cada cuadro como un conjunto de manzanas, para facilitar la búsqueda tabú. En el Algoritmo 1 se ve cómo se procesa un cuadro para obtener las manzanas que lo componen.

Para entender el algoritmo debemos notar que cada cuadra debe pertenecer a dos manzanas, correspondientes a sus dos veredas. Por esa razón, vamos a decir que una cuadra esta compuesta por dos lados, que nombraremos lado derecho (o vereda derecha) y lado izquierdo (o vereda izquierda), en donde a cada lado le corresponde una manzana diferente (ver Figura 4.1).

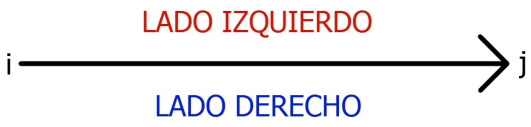


Fig. 4.1: Lado izquierdo y derecho de la cuadra (i, j)

En las primeras líneas del Algoritmo 1 (líneas 2 a 5) se procesan las cuadras cerradas como manzanas. Este paso lo hacemos aparte para que no interfiera en el resto del algoritmo. Consideramos a una cuadra cerrada como una manzana ya que la podemos mover de un cuadro a otro como cualquier otra manzana, siempre que se encuentre en el borde entre dos cuadros.

En las siguientes líneas se procesan las manzanas. Recordemos que cada cuadra forma parte de exactamente dos manzanas, una por cada uno de sus lados. Siguiendo esta idea, el algoritmo va marcando los lados de las cuadras que utilizamos a medida que formamos las manzanas. En cada iteración, elige una cuadra con al menos un lado sin marcar, y arma un ciclo que representará a la manzana correspondiente a ese lado de la cuadra. Cada vez que arma una manzana, marca los lados usados de las cuadras que forman el ciclo, para no volverlos a usar.

Para que este ciclo represente efectivamente a una manzana, se arma de determinada

Algoritmo 1 *procesarManzanas*

```
1: manzanas ← {}
2: para (i, j) ∈ cuadras hacer
3:    si (i, j) forma parte de una cuadra cerrada entonces
4:       agregar la cuadra cerrada a manzanas
5:       marcar como procesados ambos lados de la la cuadra cerrada
6: mientras queden cuadras sin marcar en alguno de sus dos lados hacer
7:    elegir (i, j) ∈ cuadras con al menos un lado sin marcar
8:    si no se marcó como usado el lado derecho de (i, j) entonces
9:       formar un ciclo partiendo de j y llegando a i, siempre eligiendo la cuadra más a
         la derecha entre aquellas cuadras cuyo lado a usar no está marcado
10:      marcar el lado usado de las cuadras del ciclo (derecha o izquierda)
11:      agregar el ciclo a manzanas
12:   si no se marcó como usado el lado izquierdo de (i, j) entonces
13:      formar un ciclo partiendo de j y llegando a i, siempre eligiendo la cuadra más a
         la izquierda entre aquellas cuadras cuyo lado a usar no está marcado
14:      marcar el lado usado de las cuadras del ciclo (derecha o izquierda)
15:      agregar el ciclo a manzanas
16: quitar la manzana más grande de manzanas, ya que representa el borde
17: devolver manzanas
```

manera. Si empezamos utilizando el lado derecho de la cuadra, siempre tomamos como siguiente cuadra en el ciclo aquella más a la derecha entre todas las cuadras incidentes en la ultima esquina del ciclo. En caso de tomar el lado izquierdo, elegimos siempre la cuadra más a la izquierda.

Para entender mejor este proceso, vemos la Figura 4.2. Tomamos como cuadra inicial el lado derecho de la cuadra marcada en rojo en el primer paso. En cada paso, elegimos la cuadra que forma el menor ángulo (a derecha) entre ella misma y la cuadra anterior en el ciclo. Se termina de formar la manzana cuando formamos un ciclo.

Para que cada cuadra pertenezca a exactamente dos manzanas debemos tomar la parte exterior del cuadro como una manzana más. Debido a esto, en la línea 16 quitamos la mayor de las manzanas (en términos de cantidad de cuadras) de la lista de manzanas, y la guardamos como el borde del cuadro (ver Figura 4.3). En general, tomar la mayor de las manzanas no siempre nos asegura que sea el borde, pero como en el caso de los cuadros que utilizamos (provenientes del partido de Morón) esto es así, no se implementó un método más complejo para decidir cuál de las manzanas es el borde del cuadro. En caso de ser necesario, se puede implementar otro método, como por ejemplo, elegir la manzana que incluya (como polígono) a todas las demás manzanas.

Como última observación sobre el Algoritmo 1, debemos notar que aunque en general no importa si nombramos a una cuadra como (i, j) o (j, i) (ya que es un par no ordenado), esto sí afecta la decisión de cuál de los dos lados es el derecho y cuál el izquierdo (ver Figura 4.4). Esta observación es importante porque implica que, aunque esté siguiendo, por ejemplo, siempre la cuadra más a la derecha, esto no significa que use la parte derecha

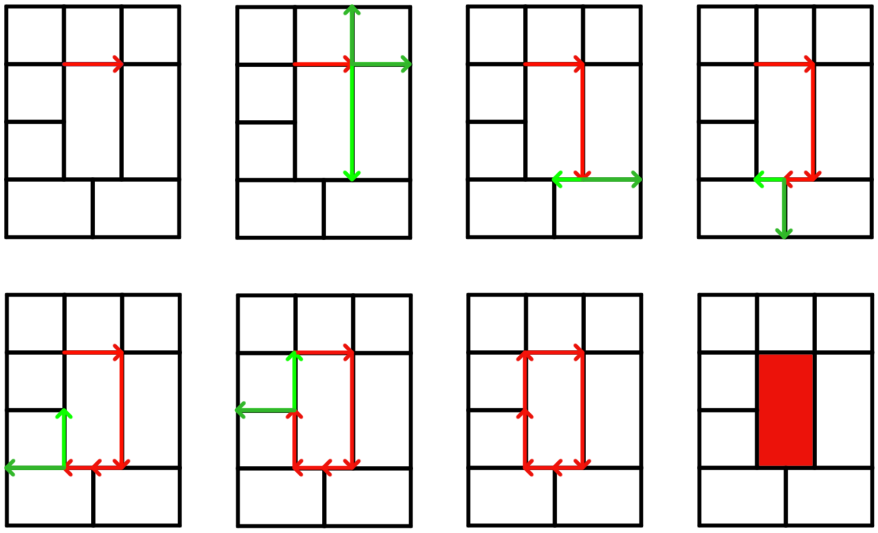


Fig. 4.2: Ciclo que forma el algoritmo 1 a partir del lado derecho de una cuadra

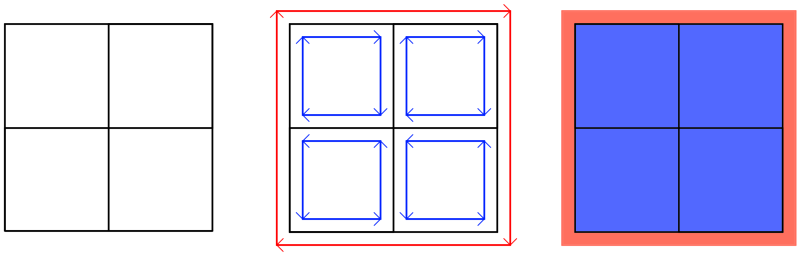


Fig. 4.3: Detección de manzanas (azul) y borde (rojo) de un cuadro compuesto por cuatro manzanas

de la cuadra (se puede usar la izquierda, dependiendo de cómo fue nombrada).

Una vez que cada cuadro fue procesado como un conjunto de manzanas, podemos empezar el proceso para mejorar los cuadros iniciales mediante una búsqueda tabú. Este proceso se lleva a cabo cambiando una manzana de un cuadro a otro, para luego evaluar si se mejoró el costo de la distribución.

El Algoritmo 2 se encarga de, dada una distribución D, intentar mejorarla mediante una metaheurística de búsqueda tabú. Las variables a considerar son:

- D la distribución inicial, y que se irá modificando a lo largo de la ejecución de la búsqueda tabú.
- D' representa la mejor distribución encontrada hasta un momento determinado de la ejecución del programa.
- *listatabú* es la lista de las manzanas elegidas recientemente para el intercambio de un cuadro a otro, y que se evitarán volver a elegir en el futuro próximo.
- $maxTam$ es el tamaño máximo de *listatabú*.
- *manzana* representa la manzana que se elije para mover de un cuadro a otro en cada iteración de la búsqueda tabú.

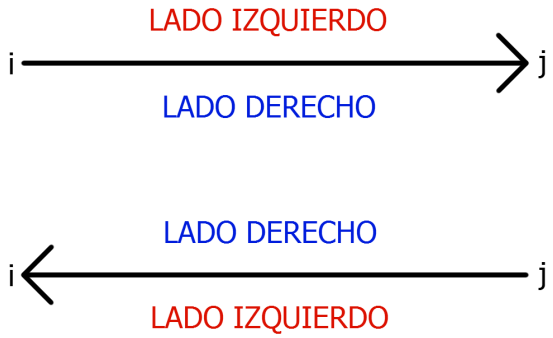


Fig. 4.4: La asignación de lado derecho e izquierdo es diferente si la cuadra se nombra (i, j) o (j, i)

- un tiempo límite para ejecución del algoritmo.

Algoritmo 2 *mejorarDivision*

Entrada: una distribución inicial D de la zona, un tamaño máximo $tamMax$ de la lista tabú, un tiempo límite para correr el algoritmo
Salida: una distribución D' cuya ruta máxima sea menor o igual en costo a la de D

```
1:  D' ← D
2:  listatabú ← ∅
3:  mientras no se cumpla con el tiempo limite hacer
4:      elegir una manzana que comparta al menos una cuadra entre dos cuadros en D y,
        de ser posible, que no se encuentre en listatabú
5:      modificar D moviendo manzana al cuadro vecino con el que comparte la cuadra
6:      si |listatabú| > maxTam entonces
7:          quitar el elemento más antiguo de listatabú
8:      agregar manzana a listatabú
9:      si D es válida entonces
10:         calcular las rutas óptimas de la distribución D modificada
11:         si se redujo el costo de la ruta máxima de la zona entonces
12:             D' ← D
13: devolver D'
```

El algoritmo de búsqueda tabú en sí es bastante estándar. Hay detalles importantes que no se ven en este pseudo-código, cómo por ejemplo, cómo se elige la manzana para realizar el cambio, o qué significa que una distribución es válida.

Veamos primero cómo se elige la manzana a intercambiar. En primer lugar, no cualquier manzana es válida para el intercambio, sino que debe ser una que se encuentre en el borde de al menos dos cuadros. Es decir, que comparta al menos una cuadra entre dos cuadros. Así al moverla de un cuadro a otro, se mantienen conexos los grafos que representan a los cuadros. Otro detalle a tener en cuenta es que se evitan manzanas que estén en la lista tabú, a menos que todas las manzanas válidas se encuentren en dicha lista.

Por otro lado, decimos que una distribución es válida si cada cuadro forma un grafo fuertemente conexo, es decir, se puede llegar de un punto cualquiera a todos los demás del cuadro. También se establece un número máximo y mínimo de manzanas que puede tener un cuadro. Estos números se calculan dinámicamente para cada distribución, de acuerdo

con la cantidad de manzanas que tienen los cuadros inicialmente. Por ejemplo, se establece que el número de manzanas de un cuadro no puede reducirse más del 50 % ni aumentar a más del 150 %. Este chequeo, anterior al cálculo del costo de la distribución, evita el procesamiento innecesario al descartar distribuciones que no son válidas o que tienen un cuadro excesivamente grande o excesivamente pequeño.

Pasaremos ahora a la segunda mitad de la sección, en la que veremos cómo se calcula el costo de una distribución en cuadros.

Para poder calcular el costo de una distribución es necesario calcular el recorrido óptimo para cada uno de sus cuadros (junto con su costo). Esa es la tarea del Algoritmo 3. Dado un grafo que representa el mapa de un cuadro, lo expande y resuelve el modelo (visto en la Sección 3) sobre el grafo expandido. La resolución del modelo devuelve la cantidad de veces que se pasa por cada arco, junto con el nodo inicial s y el final t del camino. Este resultado puede verse también como un multigrafo dirigido, en donde se construye un arco por cada vez que la solución pasa por una determinada cuadra. Hay que notar que este multidigrafo es euleriano (ya que está formado sólo por los arcos que forman el camino, y el recorrido pasa exactamente una vez por cada arco del multidigrafo).

En el caso de que la solución del modelo contenga algún subciclo, los intenta unir al camino principal. De ser posible dicha unión, devuelve el recorrido obtenido. Pero en caso de que no pueda unir al menos uno de los subciclos, agrega las restricciones para evitarlos (Sección 3.4) y vuelve a resolver el modelo (con las nuevas restricciones). Este ciclo continúa hasta que se encuentre una solución sin subciclos (o hasta que el modelo que se intenta resolver no tenga solución, en tal caso se devuelve una solución de costo infinito).

Las variables a considerar por el algoritmo son:

- G: el grafo que representa el mapa del cuadro.
- G': grafo resultante de expandir las esquinas de G.
- G'': grafo que cumple con las restricciones del modelo. Puede no ser conexo (ya que puede tener subciclos).
- s: el nodo inicial del camino principal.
- t: el nodo final del camino principal.
- *caminoPrincipal*: el camino que se forma a partir de G''.
- *subciclos*: lista de los subciclos encontrados. Si es vacía quiere decir que no se encontraron subciclos.

Algoritmo 3 *resolverModelo*

Entrada: un grafo G que representa el mapa de un cuadro
Salida: un camino a través de G que pasa por todas las cuadras

```
 1: G' ← expandir esquinas de G
 2: crear las restricciones del modelo a partir de G'
 3: (G'', s, t) ← resolver el modelo
 4: si se encontró una solución entonces
 5:    (caminoPrincipal, subciclos) ← encontrarSubCiclos(G'', s, t)
 6:    si subciclos ≠ ∅ entonces
 7:       subciclosCopia ← copiar(subciclos)
 8:       mientras haya cambios en caminoPrincipal hacer
 9:          para todo subciclo ∈ subciclosCopia hacer
10:             si unirSubCiclos(caminoPrincipal, subciclo, G'') entonces
11:                eliminar subciclo de subciclosCopia
12:       si subcliclosCopia ≠ ∅ entonces
13:          para subciclo ∈ subciclos hacer
14:             agregar al modelo las restricciones para evitar subcliclo
15:          volver a la línea 2
16:       si no
17:          devolver caminoPrincipal
18:    si no
19:       devolver caminoPrincipal
20: si no
21:    devolver no existe solución
```

El Algoritmo 4, que se encarga de encontrar los subciclos, parte del nodo inicial y "camina" (Algoritmo 5) por los arcos sin usar hasta llegar al nodo final. A medida que va usando cada arco lo quita de la lista $sinUsar$. Si al finalizar el camino no quedan arcos sin usar, se debe a que no se formó ningún subciclo. En cambio si quedan arcos sin usar se debe a que se formaron uno o más subciclos.

Para procesar cada uno de los subciclos se sigue un procedimiento similar. En primer lugar se elige un vértice u de forma tal que haya al menos una arista incidente sobre el en los arcos sin usar. Este vértice será el punto de partida para empezar a formar un subciclo. Luego, partiendo de u, "camina" por los arcos sin usar hasta llegar de nuevo al vértice u, formando un ciclo. Al igual que antes, debe quitar los arcos usados de la lista $sinUsar$. De esta forma, repite este proceso hasta que no queden arcos sin usar.

Así se obtienen el camino principal y los subciclos a partir de la solución provista por el modelo. A continuación veamos un par de detalles importantes presentes en el algoritmo que realiza la "caminata".

El Algoritmo 5, que se encarga de formar los caminos, es bastante simple. Lo único a tener en cuenta es que para que no se formen subciclos innecesarios, se deben priorizar las aristas que no son de corte en la elección de la próxima arista que formará el camino. Este algoritmo es muy similar al que encuentra un camino euleriano en un grafo euleriano.

Algoritmo 4 *encontrarSubCiclos*

Entrada: un multi-digrafo $G = (V, A)$, un vértice inicial s, un vértice final t
Salida: un camino en G que empieza en s y termina en t, y un conjunto de los subciclos que se forman

$sinUsar \leftarrow copiar(A)$
$G' \leftarrow$ subgrafo de G inducido por $sinUsar$
$caminoPrincipal \leftarrow caminar(G', sinUsar, s, t)$
$sinUsar \leftarrow sinUsar - caminoPrincipal$
$subCiclos \leftarrow []$
mientras $sinUsar$ no es vacío **hacer**
  $G' \leftarrow$ subgrafo de G inducido por $sinUsar$
  $u \leftarrow$ elegir un vértice cualquiera que tenga una arista en $sinUsar$ incidente a él
  $subCicloNuevo \leftarrow caminar(G, sinUsar, u, u)$
  agregar $subCicloNuevo$ a $subCiclos$
  $sinUsar \leftarrow sinUsar - subCicloNuevo$
devolver $(caminoPrincipal, subCiclos)$

Nota: el símbolo $-$ representa la resta de conjuntos o listas

Algoritmo 5 *caminar*

Entrada: un multi-digrafo $G = (V, A)$, un vértice inicial s, un vértice final t
Salida: el camino más largo en G que empieza en s y termina en t

$camino \leftarrow []$
$u \leftarrow s$
mientras haya vecinos de u en G' **hacer**
  $u' \leftarrow$ cualquier vértice en la lista de vecinos de u en G'
  para todo vértice w en la lista de vecinos de u en G' **hacer**
    si (u, w) no es una arista de corte **entonces**
      $u' \leftarrow w$
      si $u' \neq t$ **entonces**
        Terminar ciclo
  Agregar (u, u') a $camino$
  $u \leftarrow u'$
devolver $camino$

Por último, veamos el Algoritmo 6 que se encarga de unir los subciclos al camino principal. Según el lugar en que uniremos el subciclo al camino principal, podemos dividirlo en tres casos:

- Caso A: cuando se une el subciclo en un punto intermedio del camino (no afecta al vértice inicial s ni al vértice final t)
- Caso B: cuando se une el subciclo al inicio del camino (cambia el vértice inicial s)
- Caso C: cuando se une el subciclo al final del camino (cambia el vértice final t)

En la Figura 4.5 se ven los tres casos, antes y después de unir el subciclo. Las líneas azules representan el camino principal, mientras que las rojas al subciclo (las líneas punteadas rojas que unen a los dos vértices se utilizan para indicar que se forma un ciclo, a diferencia del azul, que es un camino). Las líneas violetas se usan para los arcos que se agregan para unir el subciclo. Notar que la eliminación o adición de arcos al camino siempre es sobre arcos auxiliares, cuyo costo es nulo, y por lo tanto no afecta el costo del recorrido (tomando como costo del recorrido el del camino principal más el costo del los subciclos).

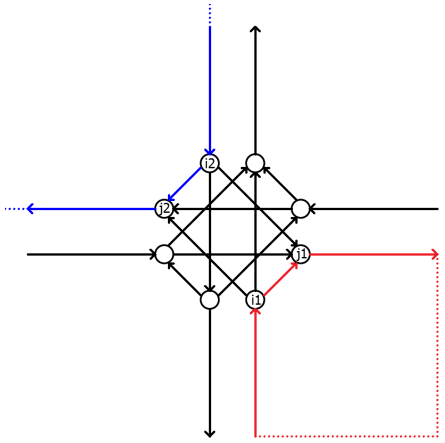
Algoritmo 6 *unirSubCiclos*

Entrada: un grafo G, un *camino* en G que empieza en s y termina en t, y un *subciclo* en G que se desea unir al *camino*

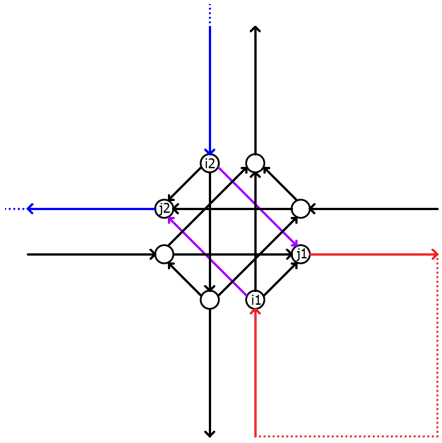
Salida: un valor booleano que indica si se pudo unir el *subciclo* a *camino*. En caso afirmativo, se modifica *camino* (incluyendo el subtour).

para $(i_1, j_1) \in subtour$ **hacer**
  para $(i_2, j_2) \in camino$ **hacer**
    si (i_1, j_1) y (i_2, j_2) son aristas auxiliares en G **entonces**
      $k \leftarrow$ índice de (i_2, j_2) en *camino*
      $l \leftarrow$ índice de (i_1, j_1) en *subtour*
      $camino \leftarrow camino[0, ..., k-1] + (i_2, j_1) + subtour[l+1, ...l-1] + (i_1, j_2)$
        $+ camino[k+1, ..., |camino|]$
      devolver True
$t \leftarrow$ vértice final de *camino*
para $(i, j) \in subtour$ **hacer**
  si (t, j) y (i, j) son aristas auxiliares en G **entonces**
    $l \leftarrow$ índice de (i, j) en *subtour*
    $camino \leftarrow camino + (t, j) + subtour[l+1, ...l-1]$
    devolver True
$s \leftarrow$ vértice inicial de *camino*
para $(i, j) \in subtour$ **hacer**
  si (i, s) y (i, j) son aristas auxiliares en G **entonces**
    $l \leftarrow$ índice de (i, j) en *subtour*
    $camino \leftarrow subtour[l+1, ...l-1] + (i, s) + camino$
    devolver True

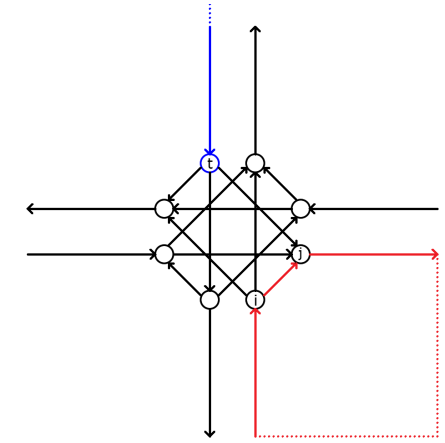
Notas: el símbolo + representa la concatenación de listas.
La operación sobre listas [k,...l] representa la sublista desde el índice k hasta el indice l.



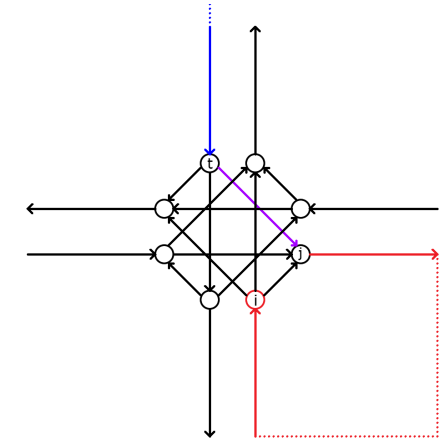
(a) Caso A antes de la unión.
Recorrido principal:
$[s, ..., i2, j2, ..., t]$
Subciclo: $[i1, j1, ..., i1]$



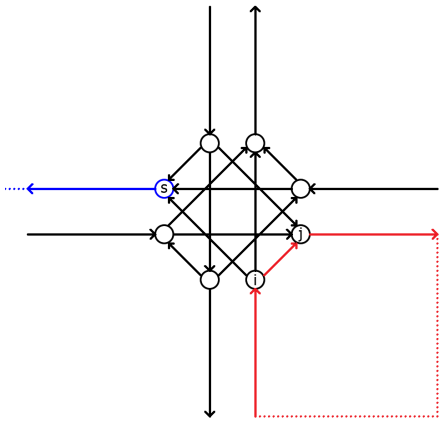
(b) Caso A después de la unión.
Recorrido resultante:
$[s, ..., i2, j1, ..., i1, j2, ..., t]$



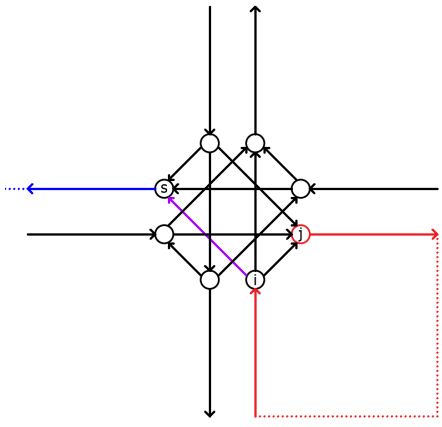
(c) Caso B antes de la unión.
Recorrido principal: $[s, ..., t]$
Subciclo: $[i, j, ..., i]$



(d) Caso B después de la unión.
Recorrido resultante: $[s, ..., t, j, ..., i]$



(e) Caso C antes de la unión.
Recorrido principal: $[s, ..., t]$
Subciclo: $[i, j, ..., i]$



(f) Caso C después de la unión.
Recorrido resultante: $[j, ..., i, s, ..., t]$

Fig. 4.5: Antes y después de la unión de un subciclo (rojo) al camino principal (azul) para cada uno de los casos contemplados en el algortimo 6

5. RESULTADOS

En esta sección veremos los resultados obtenidos a partir de las experiencias realizadas para mostrar la efectividad de los algoritmos implementados. Para todas las pruebas se fijó como tiempo máximo para la ejecución de la resolución del modelo en 600 segundos (10 minutos), a menos que se aclare lo contrario. Este valor fue elegido debido a que la mayoría de los cuadros son resueltos de manera óptima en ese tiempo, y en los casos en que no, la diferencia con el óptimo es muy pequeña.

Las experiencias fueron realizadas en una PC con procesador AMD Phenon II x4 945 de 3Ghz de velocidad de procesamiento y 4Gb de memoria RAM, corriendo con el sistema operativo Linux Ubuntu 12.04 de 32bits. Para la resolución del modelo de programación lineal se usó el solver no comercial SCIP.

5.1. Comparación entre el uso y el no uso del algoritmo de unión de subciclos

En la Tabla 5.1 presentamos los resultados obtenidos en las experiencias realizadas para probar la efectividad del algoritmo de unión de subtours.

La primer columna indica sobre qué cuadro fue realizada la experiencia. Elegimos Castelar para realizar las pruebas debido a que los cuadros de esta zona presentan una gran diversidad entre sí (cuadros más grandes que otros, cuadros compuestos en su mayoría por cuadras de mano única, mientras que otros están compuestos en su mayoría por cuadras de doble sentido, cuadros compuestos por manzanas de formas regulares como rectángulos, pero también otras de formas más irregulares).

La segunda y tercer columna de la tabla indican el tiempo de ejecución y el número de iteraciones que le tomó al programa para resolver el respectivo cuadro, sin implementar el algoritmo de unión de subtours. Las últimas dos columnas corresponden a los valores utilizando el algoritmo que une los subtours.

Cuadro	Sin unir subtours		Con el algoritmo de unión de subtours	
	Iteraciones	Tiempo	Iteraciones	Tiempo
Castelar1	2	5,2 seg	1	2,4 seg
Castelar2	5	19 seg	1	3 seg
Castelar3	59	9,7 hs	1	10 min
Castelar4	6	52,4 seg	1	5,6 seg
Castelar5	2	5,7 seg	1	3 seg
Castelar6	22	3,7 hs	1	10 min
Castelar7	35	5,9 hs	2	20 min
Castelar8	24	3,7 hs	1	17,9 seg

Tab. 5.1: Resultados obtenidos en las experiencias realizadas sobre la zona de Castelar para probar la efectividad del algoritmo de unión de subtours.

5.2. Comparación entre el recorrido óptimo y el usado por los choferes

En la Tabla 5.2 se muestra la información que nos proporcionó la Municipalidad de Morón al comienzo del proyecto. Estos datos fueron obtenidos a partir de dispositivos GPS que fueron ubicados en los camiones de recolección.

Los recorridos mostrados en la tabla corresponden a los efectuados el día de la fecha especificada, construidos por el conductor del camión en el momento de la recolección, sin el uso de guías u otra herramienta más que una gráfica del cuadro correspondiente.

Una aclaración importante que se debe mencionar, es que los cuadros se iban modificando a medida que se realizaban las pruebas. El personal de la municipalidad ajustaba los cuadros para equilibrar el tiempo de recolección y tratar de reducir el número de cuadras sin recolectar. Por esta razón, cuadros que en principio tienen el mismo identificador (por ejemplo, *palomar*4) pero distinta fecha no son necesariamente iguales, lo que da lugar a que en las tablas haya diferencias entre los valores de los datos entre cuadros de igual identificador.

En los próximos experimentos compararemos estos datos con los obtenidos por el programa, para luego determinar las ventajas en la utilización del recorrido óptimo.

Prueba	Recorrido efectuado por el chofer del camión de recolección					
	Cuadro	Fecha	Distancia recorrida	Cuadras faltantes	Tiempo de recolección	% de cuadro completado
Prueba 1	castelar4	26/04/2013	46 km	60	4:10hs	88,5 %
Prueba 2	castelar7	19/04/2013	43,1 km	6	3:40hs	98,6 %
Prueba 3	haedo2	23/04/2013	17,5 km	20	2:26hs	89,7 %
Prueba 4	palomar1	10/04/2013	25 km	46	3:38hs	84,5 %
Prueba 5	palomar3	10/04/2013	24,5 km	35	4:10hs	87,5 %
Prueba 6	palomar4	24/04/2013	31,2 km	17	3:12hs	94,8 %
Prueba 7	palomar4	27/03/2013	35,2 km	35	4:46hs	91,0 %
Prueba 8	villaSarmiento2	06/04/2013	24,6 km	10	3:09hs	96,1 %
Prueba 9	villaSarmiento2	13/04/2013	23,7 km	9	2:44hs	96,3 %
Prueba 10	villaSarmiento2	15/04/2013	23,2 km	17	3:13hs	93,2 %
Prueba 11	villaSarmiento2	23/04/2013	26,5 km	5	3:07hs	98,1 %

Tab. 5.2: Datos del GPS proporcionados por la municipalidad para diferentes cuadros.

En la Tabla 5.3 se ven los resultados obtenidos en el cálculo del recorrido óptimo para los mismos cuadros que en la Tabla 5.2. La idea es luego comparar la mejora obtenida al utilizar el recorrido calculado.

La última columna (cuadras faltantes) sólo se muestra por claridad, ya que los recorridos calculados pasan por todas las cuadras del cuadro.

Prueba	Recorrido calculado					
	Cuadro	Tiempo de Ejecución	Gap	Iteraciones	Costo	Cuadras faltantes
Prueba 1	castelar4	600 seg	0.008	1	50,45 km	0
Prueba 2	castelar7	646 seg	0.003	2	40,45 km	0
Prueba 3	haedo2	0,3 seg	0	1	18,31 km	0
Prueba 4	palomar1	30 seg	0	1	33,51 km	0
Prueba 5	palomar3	512 seg	0	1	26,90 km	0
Prueba 6	palomar4	1200 seg	0.003	2	30,53 km	0
Prueba 7	palomar4	1200 seg	0.003	2	30,53 km	0
Prueba 8	villaSarmiento2	1 seg	0	1	24,17 km	0
Prueba 9	villaSarmiento2	1 seg	0	1	24,25 km	0
Prueba 10	villaSarmiento2	1 seg	0	1	25,08 km	0
Prueba 11	villaSarmiento2	1 seg	0	1	25,13 km	0

Tab. 5.3: Resultados obtenidos al calcular el recorrido utilizando el programa implementado, para los cuadros de la Tabla 5.2.

La Tabla 5.4 muestra la diferencia entre el recorrido efectuado por el chofer y el recorrido calculado por el programa. En esencia, es la unión de las Tablas 5.2 y 5.3, omitiendo los datos no interesantes para la comparación.

Prueba	Recorrido (chofer)		Recorrido (calculado)	
	Distancia	Cuadras faltantes	Distancia	Cuadras faltantes
Prueba 1	46km	60	50,45km	0
Prueba 2	43,1km	6	40,45km	0
Prueba 3	17,5km	20	18,31km	0
Prueba 4	25km	46	33,51km	0
Prueba 5	24,5km	35	26,90km	0
Prueba 6	31,2km	17	30,53km	0
Prueba 7	35,2km	35	30,53km	0
Prueba 8	24,6km	10	24,17km	0
Prueba 9	23,7km	9	24,25km	0
Prueba 10	23,2km	17	25,08km	0
Prueba 11	26,5km	5	25,13km	0

Tab. 5.4: Comparación entre el recorrido efectuado por el chofer y el recorrido calculado por el programa.

5.3. Comparación entre el el uso y el no uso del algoritmo de rezonificación

En esta sección, llamaremos *distribución inicial* a la distribución en cuadros provista por la municipalidad y que será la solución inicial sobre la que se aplicará la búsqueda tabú. Llamaremos *distribución mejorada* al resultado devuelto por la búsqueda tabú. Recordemos también que llamamos *costo* de un cuadro a la longitud de su recorrido óptimo.

En los experimentos realizados a continuación se bajó el tiempo máximo para resolver el modelo a 120 segundos, debido a que al realizar la búsqueda tabú se necesita que cada iteración dure lo menos posible. Con valores menores a los 120 segundos, algunos cuadros producen soluciones muy alejadas del óptimo, mientras que con este valor, la diferencia con el óptimo se mantiene baja, permitiendo el posterior análisis.

En la Tabla 5.5 presentamos los resultados obtenidos al aplicar la metaheurística de búsqueda tabú a cada una de las zonas del partido, con el objetivo de mejorar la distribución de los cuadros. Para cada una una de las zonas, se muestra cuánto tiempo se

dejó corriendo el programa y cuántas iteraciones de la búsqueda tabú se llevaron a cabo en ese tiempo, junto con los costos de cada cuadro antes y después de correr el algoritmo. Recordar que la distribución mejorada hace que el costo máximo de los cuadros sea menor o igual que en la distribución inicial.

Debido al gran tamaño de la zona de Castelar, ésta se dividió en dos: Castelar Norte y Castelar Sur.

Zona	Tiempo de ejecución	Iteraciones	Costo distribución inicial Costo distribución mejorada				
			Cuadro 1	Cuadro 2	Cuadro 3	Cuadro 4	Cuadro 5
Morón	10,6 hs	25	37,4 km	67,0 km	60,5 km	88,2 km	42,9 km
			37,4 km	67,0 km	60,5 km	88,2 km	42,9 km
Palomar	10,5 hs	84	30,5 km	33,2 km	33,5 km	32,2 km	-
			32,9 km	32,7 km	32,8 km	33,0 km	-
Haedo	4,7 hs	121	19,2 km	45,4 km	37,6 km	41,3 km	-
			19,2 km	45,2 km	37,8 km	41,3 km	-
Castelar Norte	7,1 hs	64	39,8 km	35,3 km	37,9 km	30 km	-
			35,8 km	39,3 km	39,5 km	32,1 km	-
Castelar Sur	6,3 hs	42	38,3 km	38,6 km	36,5 km	38,4 km	-
			38,1 km	37,9 km	38,2 km	38,1 km	-
Villa Sarmiento	6,0 hs	1261	23,9 km	28,6 km	22,9 km	-	-
			24,9 km	24,7 km	25,0 km	-	-

Tab. 5.5: Resultados de la re-zonificación. Para cada zona, la fila superior indica el costo en la distribución inicial de cada cuadro y la inferior su costo en la distribución mejorada. Los gaps obtenidos no superan el 0.05

La Tabla 5.6 reúne los datos de la Tabla 5.5 de mayor interés a la hora de comprobar las ventajas en el uso del algoritmo de búsqueda tabú. Se muestra, para cada una de las zonas, el costo máximo entre los costos de todos los cuadros de la distribución inicial y de la distribución mejorada. La última columna indica cuánto se redujeron estos costos al aplicar el algoritmo.

Zona	Máximo de la distribución inicial	Máximo de la distribución mejorada	Porcentaje de mejora
Morón	88,2 km	88,2 km	0 %
Palomar	33,5 km	33 km	1,53 %
Haedo	45,4 km	45,2 km	0,53 %
Castelar Norte	39,8 km	39,5 km	0,75 %
Castelar Sur	38,6 km	38,2 km	1,18 %
Villa Sarmiento	28,6 km	25 km	12,62 %

Tab. 5.6: Porcentajes de mejora al aplicar el algoritmo de re-zonificación

6. CONCLUSIONES

A continuación analizaremos los resultados expuestos en la sección anterior, y, por último, daré una valoración personal del trabajo realizado.

En primer lugar analicemos los resultados mostrados en la Tabla 5.1. En esta tabla se compara el uso y el no uso del algoritmo que se encarga de unir los subtours encontrados en la solución del modelo. En los peores casos de mejora, como son Castelar1 y Castelar5, se redujo el número de iteraciones de 2 a 1, y el tiempo en que el algoritmo calcula el recorrido a aproximadamente la mitad. Para Castelar3 se pasó de 59 iteraciones y casi 10hs de ejecución, a una sola iteración que duro 10min utilizando el algoritmo de unión de subtours. La diferencia de tiempo es más que considerable. En varios de los casos se pasa de varias horas de ejecución a algunos minutos y hasta a segundos (como es el caso de Castelar8).

A raíz de estos resultados, quedamos conformes con el comportamiento del algoritmo de unión de subtours. Creemos que debido a este algoritmo es que se volvió posible en la práctica la implementación la búsqueda tabú para mejorar la distribución de los cuadros.

En la Tabla 5.2 podemos ver que en algunos casos la cantidad de cuadras sin recolectar es bastante considerable (por ejemplo en la primer prueba, no se pasó por 60 cuadras). Al avanzar el proyecto, los cuadros fueron cambiando, mientras que los conductores ganaban experiencia en su "construcción mental" de los recorridos. De esta forma, se fueron mejorando de a poco estos valores. Sin embargo, seguían dejando sin pasar algunas cuadras. Por eso, consideramos que las pruebas en las que no se pasan por más de 30 cuadras están bastante alejadas de la realidad que se vivió a finales del proyecto. Consideramos que las demás pruebas sí se asemejan más a la mejora real obtenida. Sin embargo, no se pudo establecer un arreglo con la municipalidad para comparar los recorridos en las últimas instancias del proyecto, por lo que usaremos los datos que tenemos para realizar la comparación.

La Tabla 5.3 muestra los recorridos calculados para los mismos cuadros que en la Tabla 5.2. Podemos ver los que consideramos tiempos de resolución bastante aceptables para nuestro problema en particular (no mas de 20min por cuadro). Aceptamos estos tiempos como buenos debido a que en nuestro problema no se requiere respuesta inmediata ante un cambio en el cuadro. Una vez establecido el recorrido no se cambia a menos que haya un cambio en el mapa (lo cual es poco común). Para estos tiempos de ejecución se obtuvieron gaps muy pequeños, por lo que estas soluciones son perfectamente aceptables en la práctica.

Pasemos a comparar ambos recorridos, el construido por el chofer en el momento de la recolección y el óptimo calculado por el programa. En la Tabla 5.4 podemos ver como en varios de los casos el recorrido óptimo es "más costoso" que el del chofer. Esto se debe a que se recorren muchas más cuadras. Sin embargo en algunos cuadros, en donde la cantidad de cuadras sin recolectar es menor, el recorrido óptimo lo completa recorriendo una distancia menor. Esto se puede ver en las pruebas 2, 6, 7, 8, y 11. En la mayoría de las demás pruebas, sin embargo, aunque el costo del recorrido "óptimo" es algo mayor,

esa diferencia es bastante baja. Por ejemplo, en las pruebas 3, 5, 9, y 10, esta diferencia no supera los 2km, mientras que se recorren todas las cuadras.

Remarcamos estas observaciones para llegar a la siguiente conclusión: si el camión realizaba la recolección en x horas, sin pasar por algunas cuadras, si le damos un recorrido de distancia similar que pase por todas las cuadras, el objetivo del trabajo está cumplido, ya que podrá realizar ese nuevo recorrido en aproximadamente x horas, sin pasarse de la franja horaria permitida.

Debido a este análisis y a las respuestas positivas por parte de la municipalidad, podemos concluir que efectivamente se cumplió de forma satisfactoria con el objetivo del proyecto. Sin embargo, todavía queda por analizar las mejoras obtenidas en las experiencias realizadas aplicando la búsqueda tabú para mejorar la distribución de los cuadros, que es lo que veremos a continuación.

En este aspecto no obtuvimos tan buenos resultados. En la Tabla 5.5 podemos ver que, a pesar de los altos tiempos de ejecución, en general el número de iteraciones es muy bajo. Tomando de ejemplo la zona de Morón, en aproximadamente 10 horas corriendo el programa, la búsqueda tabú solamente intercambió de lugar 25 manzanas. Las demás zonas muestran valores similares, a excepción de Villa Sarmiento, en donde se ve un comportamiento más deseable: 1261 iteraciones en 6 horas. Esto se debe a que Villa Sarmiento es la más pequeña de las localidades.

Al analizar la mejora obtenida al aplicar el algoritmo de búsqueda tabú (Tabla 5.6), tampoco se ve una ganancia significativa. Después dejar correr el programa durante horas, no se ve una mejora mucho mayor al 1,5 %. De nuevo la excepción es Villa Sarmiento, en el que el máximo de la distribución se redujo un 12,62 %.

Una posible hipótesis para explicar los resultados no tan alentadores obtenidos en la rezonificación puede ser que la zonificación inicial proporcionada por la municipalidad es razonablemente buena, teniendo en cuenta los ajustes que realizaron sobre la forma de los cuadros durante los primeros meses del proyecto de recolección de residuos reciclables. Como observación final, nos gustaría comentar que actualmente la municipalidad de Morón se encuentra usando los recorridos que les proporcionamos para realizar la recolección.

7. APÉNDICE

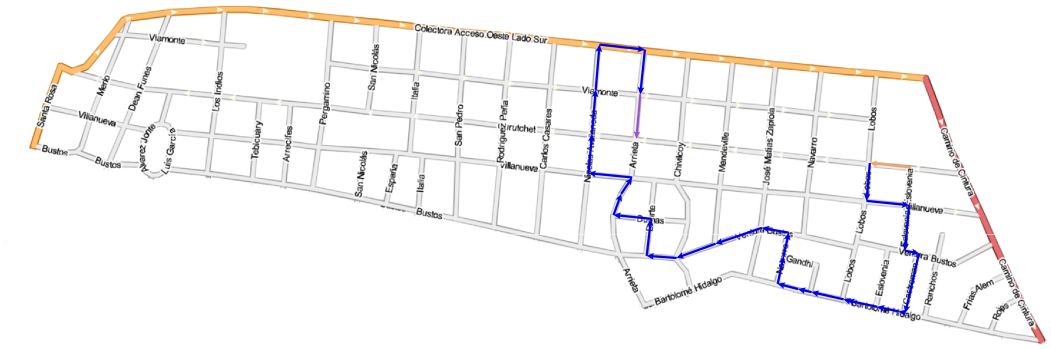
7.1. Ejemplo de la ruta calculada para el cuadro de Castelar 4

Empezar en Frias Alem y Camino de Cintura
Doblar a la izquierda en Bartolomé Hidalgo
Doblar a la izquierda en Rojas
Doblar a la derecha en Camino de Cintura

Doblar a la derecha en Bartolomé Hidalgo
Doblar a la derecha en Ranchos
Doblar a la izquierda en Ventura Bustos
Doblar a la derecha en Mendeville
Doblar a la derecha en Villanueva
Doblar a la izquierda en José Matías Zapiola
Doblar a la derecha en Viamonte
Doblar a la izquierda en Lobos
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Camino de Cintura

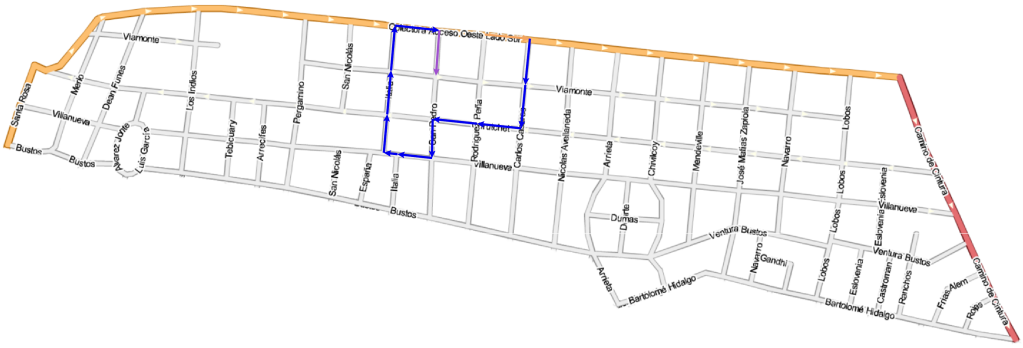
Doblar a la derecha en Ventura Bustos
Doblar a la derecha en Ranchos y hacer esa cuadra cerrada
Salir por Ranchos, y doblar a la derecha en Bartolomé Hidalgo
Doblar a la derecha en Eslovenia
Doblar a la izquierda en Ventura Bustos
Doblar a la derecha en Lobos
Doblar a la izquierda en Villanueva
Doblar a la derecha en Navarro
Doblar a la izquierda en Curutchet
Doblar a la derecha en Chivilcoy
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Camino de Cintura
Doblar a la derecha en Curutchet

Doblar a la izquierda en Lobos
Doblar a la izquierda en Villanueva
Doblar a la derecha en Eslovenia
Doblar a la izquierda en Ventura Bustos
Doblar a la derecha en Castroman
Doblar a la derecha en Bartolomé Hidalgo
Doblar a la derecha en Navarro
Doblar a la izquierda en Ventura Bustos
Doblar a la derecha en Duarte
Doblar a la izquierda en Dumas
Doblar a la derecha en Arrieta
Doblar a la izquierda en Villanueva
Doblar a la derecha en Nicolas Avellaneda
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Arrieta

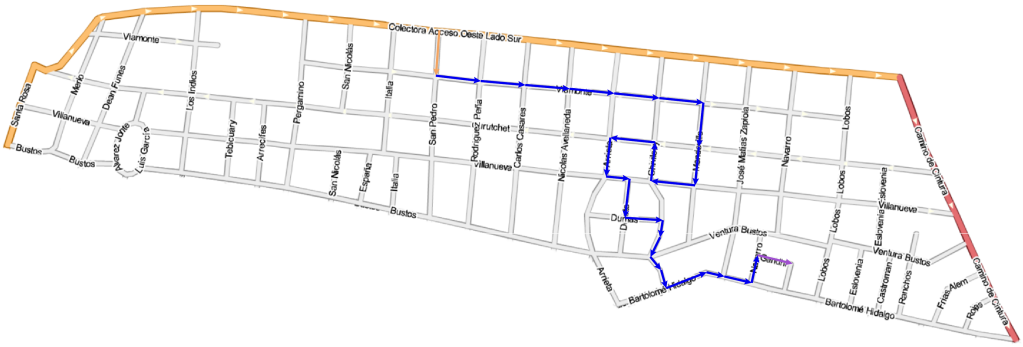


Doblar a la derecha en Curutchet
Doblar a la izquierda en Carlos Casares
Doblar a la izquierda en Bustos
Doblar a la izquierda en Nicolas Avellaneda
Doblar a la izquierda en Villanueva
Doblar a la izquierda en San Pedro
Doblar a la izquierda en Bustos
Doblar a la izquierda en Rodriguez Peña
Doblar a la derecha en Colectora Acceso Oeste Lado Sur

Doblar a la derecha en Carlos Casares
Doblar a la derecha en Curutchet
Doblar a la izquierda en San Pedro
Doblar a la derecha en Villanueva
Doblar a la derecha en Italia
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en San Pedro

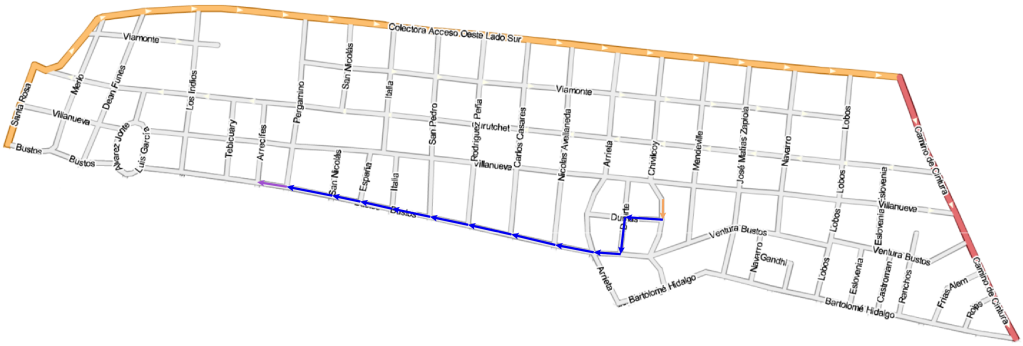


Doblar a la izquierda en Viamonte
Doblar a la derecha en Mendeville
Doblar a la derecha en Villanueva
Doblar a la derecha en Chivilcoy
Doblar a la izquierda en Curutchet
Doblar a la izquierda en Arrieta
Doblar a la izquierda en Villanueva
Doblar a la derecha en Duarte
Doblar a la izquierda en Dumas
Doblar a la derecha en Chivilcoy
Doblar a la izquierda en Bartolomé Hidalgo
Doblar a la izquierda en Navarro
Doblar a la derecha en Gandhi



Doblar a la derecha en Luther King
Doblar a la izquierda en Bartolomé Hidalgo
Doblar a la izquierda en Lobos
Doblar a la izquierda en Ventura Bustos
Doblar a la derecha en Navarro
Doblar a la izquierda en Villanueva
Doblar a la izquierda en José Matías Zapiola
Doblar a la derecha en Bartolomé Hidalgo
Doblar a la derecha en Arrieta
Doblar a la derecha en Dumas
Doblar a la izquierda en Duarte
Doblar a la derecha en Villanueva
Doblar a la derecha en Chivilcoy

Doblar a la derecha en Dumas
Doblar a la izquierda en Duarte
Doblar a la derecha en Bustos



Seguir por Bustos
Doblar a la derecha en Merlo
Doblar a la izquierda en Villanueva
Doblar a la derecha en Santa Rosa
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Dean Funes
Doblar a la derecha en Curutchet

Doblar a la derecha en Merlo
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Los Indios
Doblar a la derecha en Bustos
Doblar a la derecha en Luis García
Doblar a la derecha en Alvarez Jonte
Doblar a la izquierda en Curutchet

Doblar a la izquierda en Dean Funes
Doblar a la izquierda en Villanueva
Doblar a la derecha en Tebicuary
Doblar a la izquierda en Bustos
Doblar a la izquierda en Arrecifes
Doblar a la izquierda en Curutchet

Doblar a la izquierda en Tebicuary
Doblar a la izquierda en Villanueva
Doblar a la derecha en Pergamino
Doblar a la izquierda en Bustos
Doblar a la izquierda en San Nicolás
Doblar a la derecha en Villanueva
Doblar a la derecha en España
Doblar a la izquierda en Bustos
Doblar a la izquierda en Italia

Doblar a la izquierda en Villanueva
Doblar a la derecha en Pergamino
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en San Nicolás
Doblar a la derecha en Curutchet
Doblar a la izquierda en Dean Funes
Doblar a la izquierda en Villanueva
Doblar a la derecha en Alvarez Jonte
Doblar a la derecha en Bustos

Doblar a la derecha en Dean Funes
Doblar a la izquierda en Villanueva
Doblar a la derecha en Santa Rosa
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Seguir por Curutchet
Doblar a la derecha en Merlo
Doblar a la derecha en Bustos

Doblar a la derecha en Santa Rosa
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Viamonte
Seguir hasta la cuadra cerrada
Hacer la cuadra cerrada y al salir doblar a la derecha en Los Indios
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Pergamino
Doblar a la izquierda en Viamonte
Doblar a la derecha en San Pedro
Doblar a la derecha en Curutchet

Doblar a la derecha en San Nicolás
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Mendeville
Doblar a la izquierda en Viamonte
Doblar a la izquierda en José Matías Zapiola
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Navarro
Doblar a la izquierda en Viamonte
Doblar a la derecha en Lobos
Doblar a la derecha en Curutchet

Doblar a la derecha en Navarro
Doblar a la derecha en Viamonte
Doblar a la izquierda en Lobos
Doblar a la derecha en Colectora Acceso Oeste Lado Sur
Doblar a la derecha en Camino de Cintura
Doblar a la derecha en Curutchet
Doblar a la izquierda en Eslovenia
Doblar a la izquierda en Villanueva
Doblar a la derecha en Camino de Cintura
Finalizar el recorrido en el cruce con Rojas

Bibliografía

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

[2] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation, Cowles Commission Monograph*, 13:339–347, 1951.

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

[4] J. Edmonds and E. L. Johnson. Matching, Euler tours and the chinese postman. *Mathematical Programming*, 5:88–124, 1973.

[5] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4:61–75, 1956.

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

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

[8] A. Hedges. http://andrew.hedges.name/experiments/haversine/.

[9] H. C. Kappauf and G. J. Koehler. The mixed postman problem. *Discrete Applied Mathematics*, 1:89–103, 1976.

[10] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Mathematics*, 20:191–194, 1979.

[11] J. E. Mitchell. *Branch-and-Cut Algorithms for Combinatorial Optimization Problems*. Oxford University Press, 2002.

[12] I. H. Osman and J. P. Kelly. *Meta-Heuristics: Theory & Applications*. Kluwer Academic Publishers, 1996.

[13] C. H. Papadimitriou. On the complexity of edge traversing. *Journal of the ACM*, 23:544–554, 1976.

[14] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[15] T. K. Ralphs. On the mixed chinese postman problem. *Operations Research*, 14:123–127, 1993.

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

[17] A. Schrijver. *Theory of linear and integer programming*. Wiley-lnterscience Series in Discrete Mathematics and Optimization, 1986.

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