Trabajo Fin de Máster

Cooperación en el Problema del Cartero Chino

Maria Gabriela Pérez Sierra

Máster en Técnicas Estadísticas

Curso 2025-2026

Propuesta de Trabajo Fin de Máster

Título en galego: Cooperación no Problema do Carteiro Chinés
Título en español: Cooperación en el Problema del Cartero Chino
English title: Cooperation in the Chinese Postman Problem
Modalidad: Modalidad A
Autor/a: Maria Gabriela Pérez Sierra, Universidade da Coruña
Director/a: Silvia María Lorenzo Freire
Tutor/a: No aplica, Universidade da Coruña
Breve resumen del trabajo: El objetivo del problema del cartero chino es recorrer todos los arcos de un grafo de tal forma que el coste asociado a dicho recorrido sea el menor posible y que el punto de partida sea también el punto de finalización de dicho recorrido. Tiene multitud de aplicaciones a situaciones reales: diseño de rutas de recogida de basura, diseño de rutas de autobuses, reparto de correo, trazado óptimo de un grafo mediante un plóter, etc. En este trabajo se propone hacer un estudio del problema del cartero chino y de los principales resultados de Teoría de Juegos relacionados con este problema. Además, en función de las preferencias y conocimientos del alumno, también se podrían contemplar otras opciones, como la creación de un paquete en R que permita resolver diferentes versiones de este problema.
Recomendaciones:
Otras observaciones:

Doña Silvia María Lorenzo Freire, Profesora Titular de la Universidad de A Coruña, informa que el Trabajo Fin de Máster titulado

Cooperación en el Problema del Cartero Chino

fue realizado bajo su dirección por doña Maria Gabriela Pérez Sierra para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, da su conformidad para su presentación y defensa ante un tribunal. Además, doña Silvia María Lorenzo Freire y doña Maria Gabriela Pérez Sierra

√ sí □ no

autorizan la publicación de la memoria en el repositorio de acceso público asociado al Máster en Técnicas Estadísticas.

En A Coruña, a 14 de julio de 2026.

LORENZO FREIRE SILVIA MARIA - 32680954R
Firmado digitalmente por LORENZO FREIRE SILVIA MARIA - 32680954R
Fecha: 2026.07.15 10:54:09 +02'00'

Firmado digitalmente por María Gabriela Pérez el día 14/07/2026.

La directora:
Doña Silvia María Lorenzo Freire

La autora:
Doña Maria Gabriela Pérez Sierra

Índice general

Resumen

0.1 Resumen en español

El Problema del Cartero Chino es uno de los modelos clásicos de los problemas de rutas por arcos. En su formulación tradicional, consiste en determinar una ruta cerrada de coste mínimo que recorra todas las aristas de una red al menos una vez. Sin embargo, en numerosos contextos reales, el interés no se limita a la resolución del problema para un único agente, sino que se extiende a situaciones en las que varios agentes comparten una misma red y pueden cooperar. En estos casos, además de minimizar el coste total, surge la necesidad de diseñar mecanismos de reparto que distribuyan de forma justa y estable los costes o ahorros obtenidos. Este trabajo estudia dicha extensión cooperativa del problema, situándola en la intersección entre la Teoría de Grafos, la optimización combinatoria y la Teoría de Juegos Cooperativos.

0.2 English abstract

The Chinese Postman Problem is one of the classical models in arc routing problems. In its traditional formulation, it consists of finding a minimum-cost closed route that traverses every edge of a network at least once. However, in many real-world settings, the interest does not lie only in solving the problem for a single agent, but also in analysing situations in which several agents share the same network and may cooperate. In such cases, besides minimizing the total system cost, an additional question arises: how to allocate costs or savings in a fair and stable way. This thesis studies this cooperative extension of the problem at the intersection of Graph Theory, combinatorial optimization and Cooperative Game Theory.

Capítulo 1

Introducción

1.1 Planteamiento

En el contexto actual, la optimización de recursos y la gestión eficiente de los medios disponibles se han convertido en elementos esenciales para mejorar la sostenibilidad y la competitividad de las organizaciones. La optimización no debe entenderse únicamente como una estrategia de ahorro, sino como un proceso orientado a estructurar de manera eficiente los recursos productivos y operativos con el fin de mejorar el funcionamiento global de un sistema.

En este sentido, la Investigación Operativa proporciona herramientas matemáticas para modelar y resolver problemas complejos de decisión. Esta disciplina actúa como un puente entre la realidad y la abstracción formal, empleando técnicas de análisis combinatorio, programación lineal y, de forma destacada, la Teoría de Grafos, para representar y resolver problemas logísticos como la planificación de rutas urbanas, la asignación de flotas o la cobertura de redes de servicio.

No obstante, la optimización individual presenta límites físicos, económicos y operativos. Por ello, la Investigación Operativa moderna incorpora también la colaboración como una herramienta estratégica de alto impacto. Desde la perspectiva de la Teoría de Juegos Cooperativos, es posible estudiar situaciones en las que varios agentes comparten infraestructuras o servicios, generando sinergias y obteniendo resultados potencialmente mejores que los alcanzables de forma aislada.

En este marco se sitúa el **Problema del Cartero Chino (CPP, *Chinese Postman Problem*)**, introducido por Kwan (1962) y considerado uno de los modelos fundamentales dentro de los problemas de rutas por arcos (Dror, 2000). Este problema clásico consiste en determinar una ruta cerrada de coste mínimo que permita recorrer todas las aristas de una red al menos una vez y regresar al punto de partida.

El presente Trabajo de Fin de Máster aborda esta problemática desde una doble perspectiva. En una primera fase, de carácter individual, se estudia la resolución algorítmica del problema para un único agente, aplicando herramientas de Teoría de Grafos para minimizar la distancia recorrida en redes con distintas topologías. En una segunda fase, de carácter cooperativo, se analiza la aplicación de la Teoría de Juegos para modelar escenarios en los que varios agentes colaboran sobre una misma red. En esta etapa, el interés no se limita a la formación de coaliciones, sino que se extiende al diseño y evaluación de mecanismos matemáticos que permitan un reparto justo y estable de los costes compartidos.

1.1.1 Alcance del Modelo

El análisis y la resolución algorítmica del Problema del Cartero Chino desarrollados en este documento se restringen exclusivamente al caso de **grafos no dirigidos**. Esta elección responde a dos motivos principales:

1. Permite una exposición más clara e intuitiva del modelo, favoreciendo una comprensión didáctica de sus fundamentos y de los algoritmos de resolución.

2. Garantiza que la complejidad computacional del problema siga siendo tratable en tiempo polinómico. En particular, el caso no dirigido admite procedimientos de resolución bien establecidos, entre ellos los basados en emparejamiento perfecto (Edmonds y Johnson, 1973). Al excluir variantes más generales del problema, se evita trasladar el análisis a contextos de mayor complejidad computacional.

1.1.2 Planteamiento del Problema

En este trabajo, la red de servicio se modela mediante un grafo $G = (V, E)$, donde los vértices V representan intersecciones o puntos de conexión y las aristas E representan calles o tramos que deben ser recorridos.

El objetivo fundamental consiste en determinar un recorrido cerrado de coste mínimo que cubra todas las aristas del sistema al menos una vez. Desde esta base, la investigación se organiza en dos fases:

- **Fase 1. Problema del Cartero Chino sin cooperación:** un único agente determina su ruta óptima de cobertura. Para ello, se busca un paseo cerrado de longitud mínima que recorra cada arista del grafo al menos una vez.

- **Fase 2. Extensión cooperativa del Problema del Cartero Chino:** cuando varias entidades comparten una misma red, el problema se analiza desde la Teoría de Juegos Cooperativos. El objetivo ya no es únicamente minimizar una ruta individual, sino garantizar además que el reparto de costes resultante sea estable y aceptable para todos los participantes.

1.2 Fundamentos Teóricos y Antecedentes Históricos

El origen de los problemas de rutas por arcos puede situarse en 1736, cuando Leonhard Euler resolvió el problema de los Siete Puentes de Königsberg. Su análisis mostró que la posibilidad de recorrer una red atravesando exactamente una vez cada conexión depende de la **paridad de los grados de sus vértices**. Este resultado no solo propició grandemente al nacimiento de la Teoría de Grafos, sino que sentó las bases conceptuales sobre las que, siglos más tarde, Kwan (1962) formalizaría el Problema del Cartero Chino.

1.2.1 Los Siete Puentes de Königsberg

Euler abordó este problema mediante un proceso de abstracción matemática que transformó una situación geográfica concreta en un modelo formal.

- **Nodos o vértices:** las cuatro masas de tierra separadas por el río Pregel se representaron como puntos de conexión.

- **Aristas:** los siete puentes se modelaron como conexiones entre dichos puntos.

Euler estudió el número de conexiones incidentes en cada vértice, observando que la posibilidad de construir un circuito cerrado depende de la paridad de los grados de los vértices.

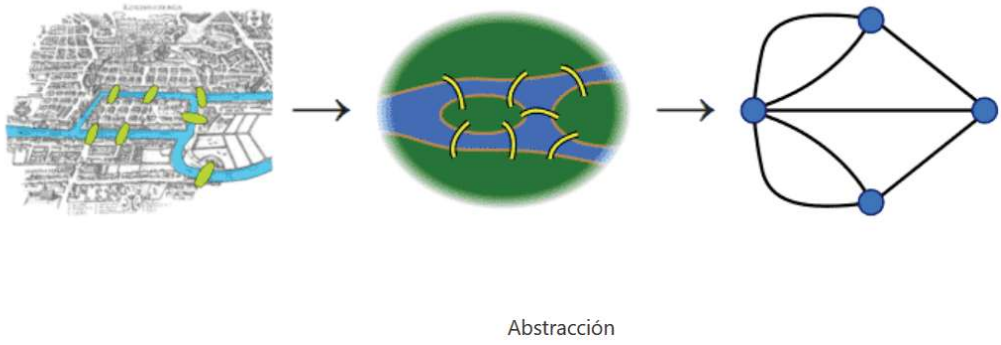


Figura 1.1: Proceso de abstracción de Euler. Fuente: adaptado de NetworkX (2024).

Así pues, a partir de esta representación, Euler observó que para que exista un recorrido cerrado que utilice exactamente una vez cada arista, cada vértice debe tener grado par. La razón es que cada vez que se entra en un vértice por una arista, debe existir otra arista distinta para salir de él, consumiendo las conexiones de dos en dos. Como en el caso de Königsberg todos los vértices tenían grado impar, concluyó que dicho recorrido era imposible bajo la configuración original.

Esta idea constituye el punto de partida de la teoría euleriana y, en consecuencia, de buena parte de los modelos de rutas por arcos estudiados en la actualidad.

1.2.2 Principales modelos de rutas

Los problemas de rutas pueden definirse, en términos generales, como problemas de optimización combinatoria en los que se busca determinar una secuencia de recorrido óptima sobre una red, normalmente minimizando distancia, tiempo o coste (Dror, 2000).

Año	**Hito histórico**	**Autor**	**Concepto clave**
1736	Puentes de Königsberg	Euler	Origen de la Teoría de Grafos y del análisis de recorridos sobre redes.
1959	Distribución de combustible	Dantzig y Ramser	Formulación inicial del Problema de Rutas de Vehículos (VRP).
1960–1962	Reparto postal	Mei-Ko Kwan	Formulación del Problema del Cartero Chino (CPP).
Actualidad	Redes y cooperación	Varios autores	Extensiones cooperativas, modelos multinivel y optimización colaborativa.

Cuadro 1.1: Cronología sintética de la optimización de rutas.

1.2.2.1 Modelos de rutas sobre nodos

En esta categoría, la demanda está situada en los vértices del grafo. El desplazamiento por las aristas actúa como medio para alcanzar los puntos donde se presta el servicio.

Problema del Viajante de Comercio (TSP, *Traveling Salesman Problem*). Es uno de los modelos clásicos de la optimización combinatoria (Bodin et al., 1983). Su objetivo consiste en

encontrar un recorrido de coste mínimo que visite cada nodo exactamente una vez y regrese al punto de partida.

Problema de Rutas de Vehículos (VRP, *Vehicle Routing Problem*). Puede entenderse como una generalización del TSP en la que interviene una flota de vehículos, cada uno con capacidad limitada Q (Bodin et al., 1983; Dror, 2000).

1.2.2.2 Modelos de rutas por arcos

Dentro de la taxonomía de los problemas de transporte, los problemas de rutas por arcos (*Arc Routing Problems*, ARP) se distinguen porque la demanda de servicio se encuentra en las conexiones de la red, y no en sus nodos (Eiselt et al., 1995; Corberán y Laporte, 2014).

Problema del Cartero Chino (CPP, *Chinese Postman Problem*). Introducido por Kwan (1962), es el modelo base de esta familia y constituye el objeto central de estudio de este trabajo. Su propósito es encontrar un recorrido cerrado de coste mínimo que atraviese todas las aristas de la red al menos una vez y regrese al punto de partida. Edmonds y Johnson (1973) demostraron que este problema puede resolverse en tiempo polinomial mediante técnicas de emparejamiento perfecto (*perfect matching*). Entre sus aplicaciones más conocidas se encuentran la recogida de residuos urbanos, la limpieza viaria o la inspección de redes (Eiselt et al., 1995; Dror, 2000).

Problema del Cartero Rural (RPP, *Rural Postman Problem*). En esta variante, la demanda no se encuentra en la totalidad de las aristas, sino en un subconjunto específico $E_R \subseteq E$. El objetivo es hallar un ciclo de coste mínimo que recorra al menos una vez cada arista de dicho subconjunto (Dror, 2000; Corberán y Laporte, 2014).

Problema de Rutas por Arcos con Capacidades (CARP, *Capacitated Arc Routing Problem*). En este modelo, cada arista $e \in E$ presenta una demanda asociada q_e y cada vehículo dispone de una capacidad máxima Q. El objetivo es minimizar el coste total de las rutas garantizando que la demanda atendida en cada recorrido no supere dicha capacidad (Dror, 2000; Eiselt et al., 1995):

Cuadro 1.2: Clasificación de los problemas de rutas

Demanda	Capacidad	Nombre del problema
NODOS	NO	Viajante de Comercio (TSP)
NODOS	SÍ	Problema de Rutas de Vehículos (VRP)
ARCOS	**NO**	**Problema del Cartero Chino (CPP)**
ARCOS	NO	Problema del Cartero Rural (RPP)
ARCOS	SÍ	Problema de Rutas con Capacidades (CARP)

1.3 Áreas de Aplicación y Versatilidad del Modelo

Las aplicaciones de este tipo de problemas podrian agruparse en dos grandes familias:

A. Logística motorizada e infraestructuras

- *Recogida de residuos urbanos:* El vehículo debe recorrer las calles de una zona para prestar el servicio de vaciado o recogida Eiselt et al. (1995). El objetivo del modelo es diseñar una ruta de coste mínimo que garantice la cobertura de todos los arcos (calles) con demanda,

minimizando el *dead-heading* o trayectos en vacío, es decir, aquellos tramos donde el camión transita por una calle que ya ha sido procesada.

- *Limpieza de nieve:* Las máquinas quitanieves deben cubrir la totalidad de la red viaria afectada bajo condiciones críticas de tiempo Dror (2000). El estudio de Dror destaca la complejidad de este problema al introducir restricciones de obligatoriedad de paso, sentidos de circulación y la necesidad de priorizar vías principales, transformando el problema básico en un modelo de rutas por arcos con jerarquías y restricciones de capacidad.
- *Inspección y mantenimiento de infraestructuras:* Vehículos dotados de sensores recorren carreteras o redes para detectar incidencias Corberán y Laporte (2000). El modelo del cartero chino se aplica aquí para minimizar la redundancia; el objetivo es encontrar el ciclo de coste mínimo que cubra toda la red, reduciendo tanto la duplicidad de tramos ya inspeccionados como los desplazamientos improductivos necesarios para cerrar la ruta.

B. Aplicaciones tecnológicas y abstractas

- *Robótica y sistemas autónomos:* Inspección de tuberías y alcantarillado mediante robots. El modelo garantiza la cobertura total de espacios confinados, optimizando la autonomía de la batería al minimizar el paso por tramos ya explorados.
- *Diseño de circuitos:* Optimización del trazado de conexiones en microchips y sistemas VLSI. El objetivo es reducir los movimientos improductivos del cabezal de grabado o soldadura mientras se conectan todos los componentes de la red.
- *Biología computacional:* Reconstrucción y ensamblado de secuencias de ADN. Los fragmentos genéticos se modelan como arcos de un grafo, donde la reconstrucción del genoma equivale a encontrar un camino euleriano que recorra toda la estructura.

Capítulo 2

Marco Teórico y Definiciones

2.1 Conceptos Básicos

Definición 2.1.1 (Grafo)**.** Un grafo es una estructura matemática representada por un conjunto finito de vértices y un conjunto de aristas que conectan pares de vértices.

Definición 2.1.2 (Grafo No Dirigido)**.** Un grafo no dirigido $G = (V, E)$ se compone de un conjunto de vértices V y aristas E representados como pares no ordenados $\{u, v\}$. La conexión entre dos vértices funciona en ambos sentidos, permitiendo transitar libremente en ambas direcciones.

Definición 2.1.3 (Dígrafo)**.** Un dígrafo $D = (V, A)$ se compone de un conjunto de vértices V y un conjunto de **arcos** A, representados como pares ordenados (u, v), donde:

- u es la **cola** u origen del arco.
- v es la **cabeza** o destino del arco.

Esto implica que el recorrido tiene sentido único: es posible ir de u a v, pero no regresar por el mismo arco.

Definición 2.1.4 (Grafo Mixto)**.** Estructura $M = (V, E, A)$ que combina los dos tipos de conexiones anteriores:

- V: conjunto finito de vértices.
- E: conjunto de **aristas no dirigidas** (doble sentido).
- A: conjunto de **arcos dirigidos** (sentido único).

El grafo mixto es el modelo más fiel a la topografía de una ciudad real, permitiendo representar tanto avenidas de doble sentido como calles con restricciones de circulación (Dror, 2000). No obstante, es también el modelo más complejo de resolver algorítmicamente (Corberán y Laporte, 2014) y, tal como se establece en la Sección 1.1.1, queda fuera del alcance de este trabajo , el cual se centra exclusivamente en grafos no dirigidos.

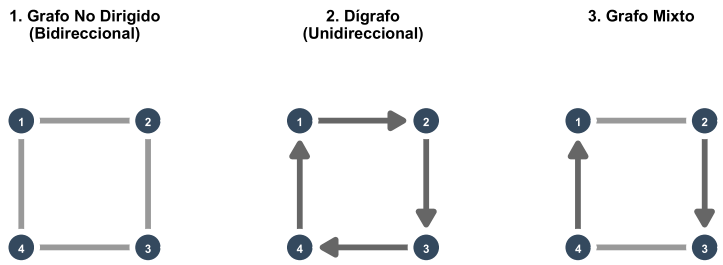


Figura 2.1: Clasificación de grafos según la naturaleza de sus aristas.

Definición 2.1.5 (Grafo Completo)**.** Un grafo completo, denotado K_n, es un grafo simple en el que cada par de vértices distintos es adyacente. El número total de aristas de un grafo completo con n vértices es:

$$|E| = \frac{n(n-1)}{2}.$$

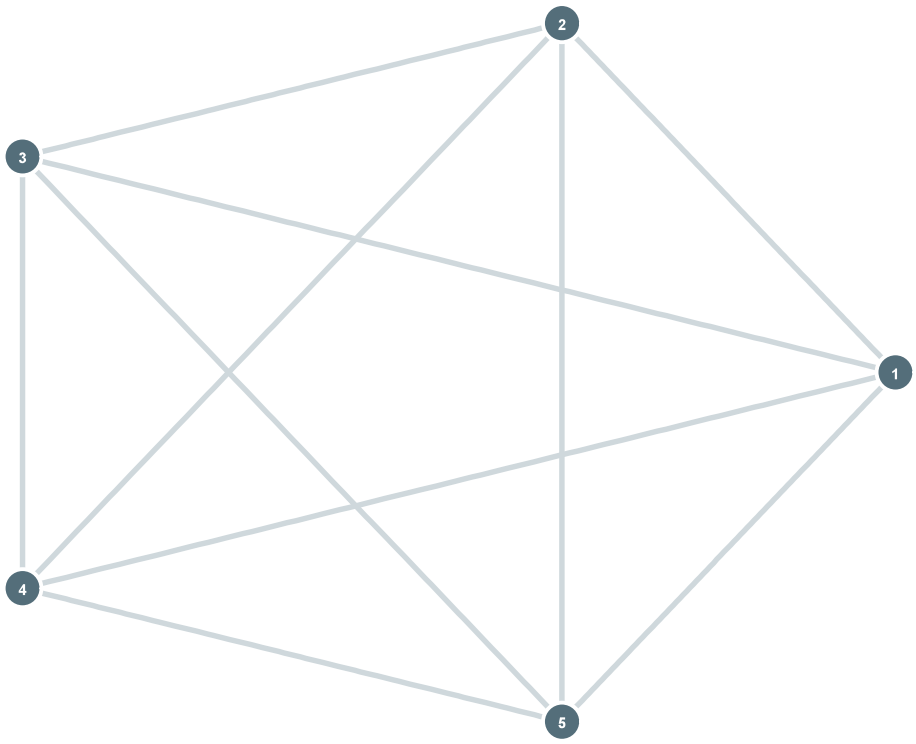


Figura 2.2: Grafo completo K_5 sobre cinco nodos. Cada par de vértices está conectado directamente por una arista, lo que garantiza que todas las combinaciones posibles de emparejamiento pueden ser evaluadas.

2.1.1 Grado de un vértice

El concepto de grado permite cuantificar la conectividad local de cada punto de la red. Siguiendo la terminología sistematizada por Fleischner (2000) en su estudio sobre grafos eulerianos, y de acuerdo con la notación estándar de Bondy y Murty (2008), esta definición presenta dos variantes fundamentales según la naturaleza del grafo.

Definición 2.1.6 (Grado en grafos no dirigidos)**.** Sea $G = (V, E)$ un grafo no dirigido. El **grado** de un vértice $v \in V$, denotado como $d_G(v)$, es el número de aristas incidentes en v.

Un vértice se clasifica según su paridad:

- **Vértice par:** si $d_G(v) \equiv 0 \pmod 2$.
- **Vértice impar:** si $d_G(v) \equiv 1 \pmod 2$.

La existencia de recorridos eulerianos está estrechamente relacionada con la paridad de los vértices, como se estudiará en detalle en secciones posteriores.

Ejemplo 2.1.1. Consideremos el grafo el grafo $G = (V, E)$ donde $V = 5$ y $E = 6$. El grado de cada vértice se obtiene contando el número de aristas incidentes en él.

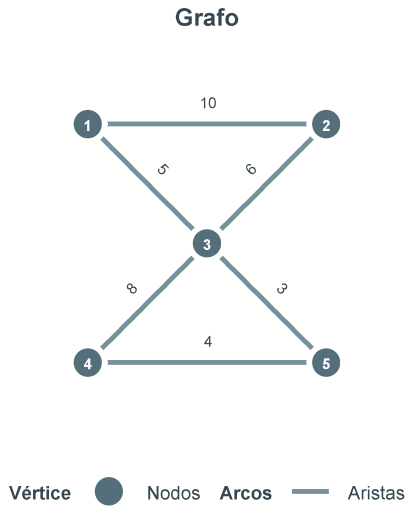
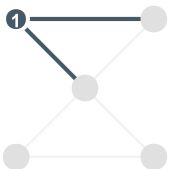
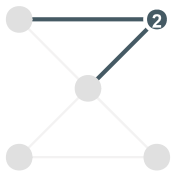


Figura 2.3: Fuente: Basado en Hamers et al. (1999).

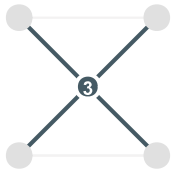
Nodo 1. Conecta con los nodos 2 y 3. Por tanto, $d(1) = 2$, luego es un vértice par.



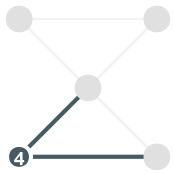
Nodo 2. Conecta con los nodos 1 y 3. Por tanto, $d(2) = 2$, luego es un vértice par.



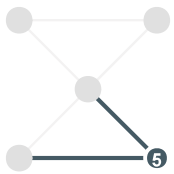
Nodo 3. Conecta con los nodos 1, 2, 4 y 5. Por tanto, $d(3) = 4$, luego es un vértice par.



Nodo 4. Conecta con los nodos 3 y 5. Por tanto, $d(4) = 2$, luego es un vértice par.



Nodo 5. Conecta con los nodos 3 y 4. Por tanto, $d(5) = 2$, luego es un vértice par.



En consecuencia, todos los vértices del grafo tienen grado par.

Aunque este trabajo se centra en grafos no dirigidos, conviene presentar también la noción de grado en grafos dirigidos para distinguir ambos contextos.

Definición 2.1.7 (Grado en grafos dirigidos)**.** En un grafo dirigido, se diferencia entre el flujo que entra y el que sale de cada vértice:

- **Grado de entrada** ($d^-(v)$): número de aristas dirigidas hacia v.
- **Grado de salida** ($d^+(v)$): número de aristas dirigidas desde v.

Un vértice se considera **equilibrado** si se cumple la condición:

$$d^-(v) = d^+(v).$$

Ejemplo 2.1.2. Para ilustrar la diferencia entre grado de entrada y de salida, analizamos el **Nodo 3** (el nodo central de la red):

- **Grado de entrada:** $d^-(3) = 2$, representado por las flechas naranjas, correspondientes al flujo recibido desde los nodos 1 y 2.
- **Grado de salida:** $d^+(3) = 2$, representado por las flechas azules, correspondientes al flujo enviado hacia los nodos 4 y 5.

Como se cumple la igualdad $d^-(3) = d^+(3) = 2$, el nodo 3 es un vértice **equilibrado**.

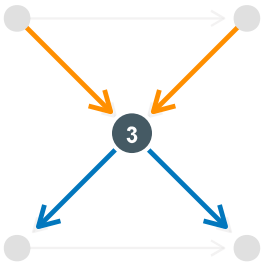


Figura 2.4: Grado de entrada y salida del Nodo 3 en un dígrafo.

Definición 2.1.8 (Puente)**.** Una arista $b \in E$ es un **puente** si su eliminación desconecta el grafo, es decir:

$$b \in B(G) \iff (V, E \setminus \{b\}) \text{ no es conexo.}$$

Para ilustrar el concepto de puente, se modifica la red añadiendo un vértice v_6 conectado únicamente al vértice v_4 mediante una nueva arista e_7. Esta arista constituye un callejón sin salida en la red, ya que v_6 no tiene ninguna otra conexión. En consecuencia, e_7 es un puente: su eliminación deja a v_6 aislado del resto de la red, tal como se aprecia en la Figura 2.5.

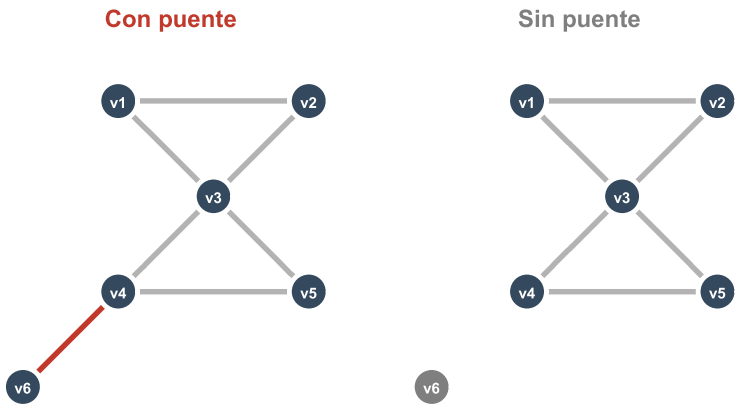


Figura 2.5: Ilustración del concepto de puente.

2.2 Tipos de Movimiento en el Grafo

Los distintos tipos de movimiento sobre un grafo pueden organizarse según dos criterios fundamentales: si el recorrido es cerrado o abierto, y si cubre la totalidad de las aristas del grafo o solo una parte de ellas.

2.2.1 Paseo

Definición 2.2.1 (Paseo)**.** Un paseo W es una secuencia finita de vértices que representa un recorrido continuo sobre un grafo no dirigido $G = (V, E)$, donde cada par de vértices consecutivos está conectado por una arista. En un paseo se permite la repetición tanto de vértices como de aristas:

$$W = (v_0, v_1, v_2, \dots, v_k).$$

El número k (la cantidad de aristas transitadas) se denomina **longitud** del paseo.

A continuación se ilustra un ejemplo de paseo sobre la red. A la izquierda, la red completa sin recorrido activo. A la derecha, el paseo $W = (1, 2, 3, 4, 3)$, donde el nodo 3 es visitado dos veces, lo cual es permitido en este tipo de movimiento. En un paseo no existe ninguna restricción que impida transitar más de una vez por el mismo vértice o por la misma arista.

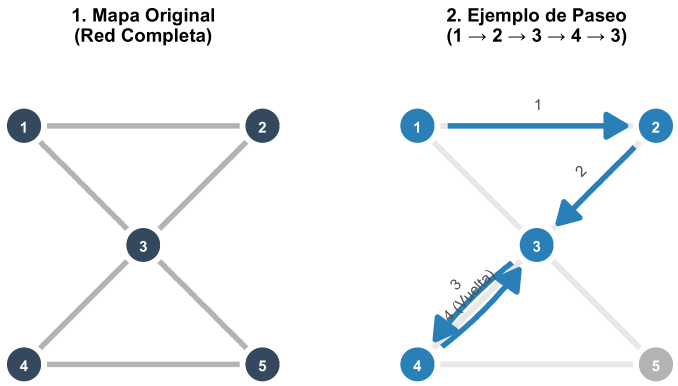


Figura 2.6: Ilustración de un paseo sobre un grafo de cinco nodos.

Definición 2.2.2 (Rastro)**.** Un rastro es un paseo en el que no se permite la repetición de aristas, aunque sí está permitido visitar el mismo vértice más de una vez.

Ejemplo 2.2.1. Consideremos la siguiente ruta sobre la red:

$$1 \to 2 \to 3 \to 4 \to 5 \to 3 \to 1.$$

El recorrido utiliza las siguientes aristas, cada una exactamente una vez:

1. Arista $\{1, 2\}$: desplazamiento del nodo 1 al nodo 2.
2. Arista $\{2, 3\}$: desplazamiento del nodo 2 al nodo 3.
3. Arista $\{3, 4\}$: desplazamiento del nodo 3 al nodo 4.
4. Arista $\{4, 5\}$: desplazamiento del nodo 4 al nodo 5.
5. Arista $\{3, 5\}$: desplazamiento del nodo 5 al nodo 3.
6. Arista $\{1, 3\}$: desplazamiento del nodo 3 al nodo 1.

Nótese que el nodo 3 es visitado dos veces, lo cual es válido en un rastro. Sin embargo, ninguna arista es recorrida más de una vez; dicho recorrido se ilustra en la siguiente figura.

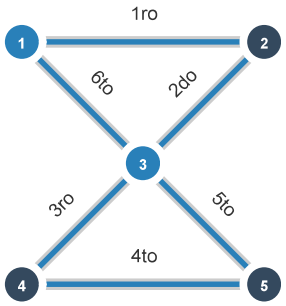


Figura 2.7: Rastro sobre un grafo de cinco nodos. El recorrido $1 \to 2 \to 3 \to 4 \to 5 \to 3 \to 1$ utiliza cada arista exactamente una vez, aunque algunos vértices, como el 1 y el 3, son visitados más de una vez.

Definición 2.2.3 (Ciclo)**.** Se denomina **ciclo** a un rastro cerrado en el que, además de no repetirse ninguna arista, tampoco se repite ningún vértice intermedio. Es decir, un recorrido que comienza y termina en el mismo vértice sin repetir aristas ni vértices intermedios.

Ejemplo 2.2.2. Consideremos el recorrido

$$1 \to 2 \to 3 \to 4 \to 5 \to 3 \to 1.$$

Este recorrido es un **rastro cerrado**, ya que no repite ninguna arista y comienza y termina en el mismo vértice. Sin embargo, el nodo 3 es visitado dos veces, por lo que **no es un ciclo** en sentido estricto, sino un **tour**.

Definición 2.2.4 (Tour)**.** Un tour es un rastro cerrado, es decir, un recorrido que comienza y termina en el mismo vértice ($v_0 = v_k$) sin repetir ninguna arista, pero que sí permite visitar vértices intermedios más de una vez.

La distinción entre ciclo y tour es relevante en el contexto del Problema del Cartero Chino: la solución buscada es un **tour** de coste mínimo que recorra todas las aristas de la red, no necesariamente un ciclo, ya que en general será necesario visitar algunas intersecciones más de una vez.

Definición 2.2.5 (Camino Euleriano)**.** Un camino euleriano es un rastro abierto que recorre todas y cada una de las aristas de un grafo exactamente una vez. En un grafo conexo no dirigido, existe si y solo si el grafo tiene exactamente dos vértices de grado impar (Bondy y Murty, 1976). El recorrido comienza en uno de dichos vértices y termina en el otro.

Definición 2.2.6 (Tour Euleriano)**.** Un tour euleriano es un tour que recorre todas y cada una de las aristas de un grafo exactamente una vez y regresa al vértice de partida. En un grafo conexo no dirigido, existe si y solo si todos sus vértices tienen grado par y es lo que denominamos como grafo euleriano.

La figura siguiente ilustra de forma comparativa los cuatro tipos de movimiento fundamentales sobre un mismo grafo de cinco nodos.

Figura 2.8: Comparativa de los tipos de movimiento sobre un grafo de cinco nodos.

La subfigura 1 muestra un **rastro**, un recorrido abierto que no repite aristas pero no tiene obligación de cubrir el grafo completo. La subfigura 2 muestra un **tour**, es decir, un rastro cerrado que regresa al vértice de origen. En la subfigura 3 corresponde a un **camino euleriano**, un rastro abierto que recorre todas las aristas exactamente una vez, comenzando y terminando en vértices distintos. Finalmente, la subfigura 4 muestra el **tour euleriano**, que combina las dos condiciones más exigentes: recorre todas las aristas exactamente una vez y regresa al vértice de origen.

El **Problema del Cartero Chino** consiste, precisamente, en determinar si existe un tour euleriano en la red y, en caso de que no exista, encontrar el recorrido cerrado de coste mínimo que cubra todas las aristas.

Definición 2.2.7 (Vértice aislado)**.** Un vértice $v \in V(G)$ se denomina **aislado** si su grado es cero, es decir, $d(v) = 0$. Esto significa que no existe ninguna arista incidente en v.

Definición 2.2.8 (Componente conexa)**.** Un subgrafo G' de un grafo G se denomina **componente conexa** de G si G' es un subgrafo conexo maximal de G respecto de la inclusión.

Definición 2.2.9 (Grafo conexo)**.** Un grafo G es **conexo** si para cualquier par de vértices $u, v \in V$ existe al menos un camino que los une, es decir, una secuencia de aristas que permite desplazarse desde u hasta v.

Ejemplo 2.2.3. La figura siguiente ilustra la diferencia entre un grafo conexo y uno disconexo. En el panel izquierdo, el grafo es **conexo** ya que existe al menos un camino entre cualquier par de nodos, aunque no haya una arista directa entre ellos. En el panel derecho, al eliminar las aristas $(3, 4)$, $(3, 5)$ y $(4, 5)$, los nodos 4 y 5 quedan aislados y la red se divide en tres componentes: $\{1, 2, 3\}$, $\{4\}$ y $\{5\}$, dando lugar a un grafo **disconexo**.

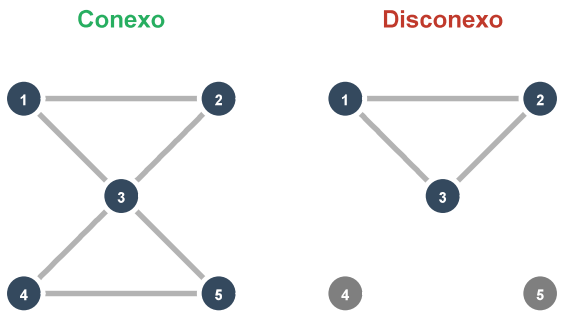


Figura 2.9: Comparación entre un grafo conexo y uno disconexo.

Definición 2.2.10 (Grafo débilmente euleriano)**.** Un grafo conexo se denomina **débilmente euleriano** si, al eliminar todos sus puentes, cada componente conexa resultante es un grafo euleriano o un vértice aislado.

Para identificar los puentes de una red se aplica el criterio de conectividad: una arista es un puente si su eliminación aumenta el número de componentes conexas del grafo. En la práctica algorítmica, este test se ejecuta sistemáticamente sobre todas las aristas de la red, de forma análoga a la lógica de la regla del puente del algoritmo de Fleury descrita en la Sección 3.2.6.

Definición 2.2.11 (Punto de Corte)**.** Sea G un grafo conexo. Un vértice $v \in V$ se denomina **punto de corte** si su eliminación aumenta el número de componentes conexas del grafo. Es decir, el grafo resultante $G - \{v\}$ es disconexo. Al eliminar el vértice, desaparecen obligatoriamente todas las aristas incidentes en él.

Ejemplo 2.2.4. En el grafo de la izquierda, si se elimina el vértice 3, la red se fragmenta en dos componentes aisladas, tal como se muestra en el grafo de la derecha. Esto indica que el vértice 3 es el único nexo de unión entre dos partes de la red y, por tanto, constituye un punto de corte.

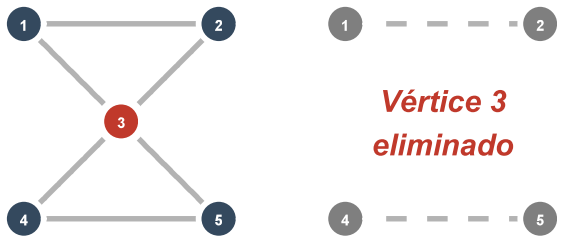


Figura 2.10: Ilustración de un punto de corte.

Definición 2.2.12 (Grafo ponderado)**.** Un **grafo ponderado** es una tupla $G = (V, E, w)$ donde $w : E \to \mathbb{R}^+$ es una función que asigna un peso a cada arista del conjunto E. Dicho peso puede representar un coste, una distancia o un tiempo de desplazamiento.

Para ilustrar este concepto, se considera el grafo con estructura de lazo utilizado a lo largo de este trabajo, representado en la Figura 2.11. Las ponderaciones corresponden a los costes de desplazamiento asignados a cada arista.

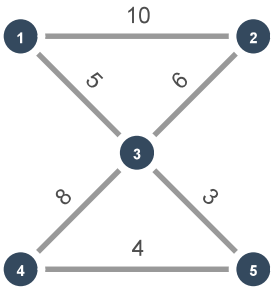


Figura 2.11: Grafo ponderado con estructura de lazo. Los pesos de las aristas representan los costes de desplazamiento entre intersecciones.

2.2.2 El Lema del Apretón de Manos

Es fundamental analizar la estructura del grafo, y en particular la paridad del grado de sus vértices. Este análisis se apoya en el **Lema del Apretón de Manos**, el cual establece que el número de vértices de grado impar en cualquier grafo es siempre par.

Este resultado fue demostrado originalmente por Leonhard Euler en 1736 y constituye el primer teorema de la historia de la Teoría de Grafos [véase (Biggs et al., 1986)]. El término *apretón de manos* es una denominación didáctica que ilustra la relación entre vértices y aristas mediante una analogía: en una reunión, cada vez que dos personas se dan la mano se contabilizan dos manos

estrechadas; del mismo modo, cada arista en un grafo aporta exactamente dos unidades a la suma total de los grados.

Teorema 2.2.1 (Lema del Apretón de Manos)**.** Para todo grafo no dirigido $G = (V, E)$, la suma de los grados de todos sus vértices es igual al doble del número de aristas:

$$\sum_{v \in V(G)} d(v) = 2 \cdot |E(G)|.$$

Cada arista $e = \{u, v\} \in E$ contribuye exactamente una unidad al grado de u y una unidad al grado de v. Por tanto, al sumar los grados de todos los vértices, cada arista se contabiliza exactamente dos veces, lo que prueba la igualdad.

De este resultado se derivan tres consecuencias fundamentales para el análisis de rutas:

1. **Resultado par.** Dado que $2 \cdot |E|$ es necesariamente un número par, la suma de todos los grados también debe serlo.
2. **Paridad de los vértices impares.** Los vértices de grado impar siempre aparecen en número par. No puede existir un grafo con un número impar de vértices impares.
3. **Base para el algoritmo de resolución.** Esta propiedad permite, en fases posteriores, aplicar algoritmos de emparejamiento (*matching*) para convertir un grafo no euleriano en uno euleriano, duplicando los caminos mínimos entre los pares de vértices impares (Edmonds y Johnson, 1973).

Desde una perspectiva operativa, el Lema del Apretón de Manos aporta las siguientes garantías sobre la estructura de la red:

1. **Integridad de la red.** Dado que cada arista tiene obligatoriamente dos extremos, el lema asegura que no existan tramos con un extremo desconectado. Toda entrada a la red debe tener una salida correspondiente.
2. **Equilibrio de flujo.** En un contexto como la recogida de residuos, el vehículo opera bajo un principio de conservación de flujo. Si una intersección tiene grado impar, el equilibrio se rompe: el vehículo necesitará repetir algún tramo para compensar la arista sobrante (Eiselt et al., 1995).
3. **Conectividad entre vértices problemáticos.** El lema garantiza que los vértices impares estén siempre vinculados entre sí de forma que el algoritmo de optimización pueda trazar caminos entre ellos para equilibrar la red (Edmonds y Johnson, 1973).

Capítulo 3

El Problema del Cartero Chino

3.1 Planteamiento

Consideremos un agente de servicio —ya sea un cartero, un vehículo de recolección de residuos o una máquina de limpieza— que debe operar sobre una red vial modelizada mediante un grafo conexo no dirigido $G = (V, E)$. El problema consiste en determinar un paseo cerrado de servicio que satisfaga las siguientes condiciones operativas:

1. **Cobertura total.** El agente debe recorrer todas las aristas del grafo al menos una vez.
2. **Recorrido Cíclico.** El recorrido debe comenzar y terminar en un mismo vértice $v_0 \in V$, que representa el depósito, garaje u oficina central.
3. **Optimalidad.** Entre todos los recorridos factibles, se busca aquel que minimiza el coste total de desplazamiento.

Este problema da lugar al **Problema del Cartero Chino no dirigido**. El desafío reside en minimizar la distancia total recorrida, lo cual se traduce en identificar qué aristas o caminos es necesario repetir para obtener un paseo cerrado, procurando que dichas repeticiones tengan coste mínimo.

Definición 3.1.1 (Problema del Cartero Chino no dirigido)**.** Sea $G = (V, E)$ un grafo conexo no dirigido y sea $c : E \to \mathbb{R}_+$ una función de costes que asigna a cada arista $e \in E$ un coste $c(e)$, donde $\mathbb{R}_+$ denota el conjunto de los números reales no negativos ($\mathbb{R}_+ = \{x \in \mathbb{R} : x \geq 0\}$). El Problema del Cartero Chino no dirigido, denotado por el par (G, c), consiste en encontrar un paseo cerrado de coste mínimo que recorra cada arista de E al menos una vez.

Si se fija un vértice $v_0 \in V$, denominado depósito, el paseo debe comenzar y terminar en v_0. Cabe destacar que, al tratarse de un grafo no dirigido, la elección de v_0 no altera el coste óptimo del problema, ya que cualquier circuito euleriano puede iniciarse desde un vértice arbitrario.

Para formalizar matemáticamente este problema, tal como exponen Eiselt et al. (1995), se pueden plantear dos formulaciones equivalentes. La primera modela directamente el número total de veces que se recorre cada arista, mientras que la segunda modela únicamente las copias adicionales que es necesario añadir al grafo original para eulerizarlo.

Formulación 1: Modelo directo por multiplicidades

Para cada arista $e \in E$, sea $x_e \in \mathbb{Z}_+$ el número de veces que dicha arista es recorrida en la solución. Denotemos por $\delta(v)$ el conjunto de aristas incidentes en el vértice $v \in V$. Adicionalmente, se

introduce una variable auxiliar $z_v \in \mathbb{Z}_+$ para cada vértice $v \in V$, que representa la mitad del grado de v en el multigrafo resultante, garantizando así la propiedad de paridad. El problema puede formularse como:

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e)\, x_e \\ \text{s.a.} \quad & x_e \geq 1, && \forall e \in E, \\ & \sum_{e \in \delta(v)} x_e = 2z_v, && \forall v \in V, \\ & x_e \in \mathbb{Z}_+, && \forall e \in E, \\ & z_v \in \mathbb{Z}_+, && \forall v \in V. \end{aligned}$$

La restricción $x_e \geq 1$ garantiza la cobertura total (cada arista se recorre al menos una vez). Por su parte, la condición $\sum_{e \in \delta(v)} x_e = 2z_v$ impone que la suma del número de veces que se atraviesan las aristas incidentes en cada vértice v sea un número par (concretamente, el doble del valor entero z_v). Esto asegura la **eulerianidad** de la red y, por consiguiente, la existencia de un paseo cerrado que recorra todas las aristas.

Formulación 2: Modelo de augmentación

Sea $T = \{v \in V : d_G(v) \text{ es impar}\}$ el conjunto de vértices de grado impar del grafo original. Para cada arista $e \in E$, sea y_e el número de copias adicionales de e que se añaden al grafo (es decir, $y_e = x_e - 1$). Denotemos por $\delta(S)$ al conjunto de aristas con un extremo en S y el otro en $V \setminus S$.

Una formulación equivalente del problema es:

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e)\, y_e \\ \text{s.a.} \quad & \sum_{e \in \delta(S)} y_e \geq 1, && \forall S \subset V \text{ tal que } |S \cap T| \text{ es impar}, \\ & y_e \in \mathbb{Z}_+, && \forall e \in E. \end{aligned}$$

Esta formulación minimiza exclusivamente el coste de las aristas repetidas. Dado que las aristas originales del grafo G tienen un coste fijo inevitable, el coste total de una solución viene dado por la suma del coste original y el coste de augmentación:

$$\text{Coste Total} = \sum_{e \in E} c(e) + \sum_{e \in E} c(e)\, y_e.$$

Cuando el grafo G es euleriano ($T = \emptyset$), la solución óptima consiste en recorrer cada arista exactamente una vez ($y_e = 0$). En caso contrario, será necesario duplicar algunos caminos para eulerizar la red, lo que constituye el núcleo computacional del problema (Edmonds y Johnson, 1973; Eiselt et al., 1995).

Teorema 3.1.1. Las Formulaciones 1 y 2 del CPP no dirigido son equivalentes. Si $(x_e^*)_{e \in E}$ es solución óptima de la Formulación 1, entonces $y_e^* = x_e^* - 1$ es solución óptima de la Formulación 2, y viceversa.

Teorema 3.1.2. El poliedro definido por las restricciones de la Formulación 2

$$P = \left\{ y \in \mathbb{R}_+^{|E|} : \sum_{e \in \delta(S)} y_e \geq 1, \quad \forall\, S \subset V \text{ tal que } |S \cap T| \text{ es impar} \right\}$$

coincide con la envolvente convexa de las soluciones enteras factibles del problema. En consecuencia, la relajación lineal de la Formulación 2 tiene siempre solución óptima entera.

3.2 Metodología y Algoritmos

3.2.1 Problema del Camino más Corto

Dado un grafo $G = (V, E)$ con pesos $w(e) \geq 0$ para cada arista $e \in E$, y dos vértices distinguidos $s, t \in V$ (origen y destino), el **Problema del Camino Más Corto** consiste en encontrar un camino P desde s hasta t que minimice la suma de los pesos de sus aristas:

$$\min_{P \in \mathcal{P}_{s,t}} \sum_{e \in P} w(e),$$

donde $\mathcal{P}_{s,t}$ denota el conjunto de todos los caminos posibles entre s y t en G.

De forma equivalente, el problema puede formularse como un problema de flujo en redes (Taha, 2017). Se define la variable $x_{ij} \geq 0$, $x_{ij} \in \mathbb{Z}$, como el flujo enviado por el arco (i, j). La formulación es la siguiente:

$$\min \sum_{i,j \in V} c_{ij} x_{ij}$$

sujeto a las restricciones de flujo:

$$\sum_j x_{ij} - \sum_l x_{li} = \begin{cases} 1 & \text{si } i = s \quad \text{(origen)}, \\ 0 & \text{si } i \neq s, t \quad \text{(transbordo)}, \\ -1 & \text{si } i = t \quad \text{(destino)}. \end{cases}$$

Estas restricciones garantizan el equilibrio de flujo en la red: todo lo que entra en una intersección debe salir obligatoriamente hacia el siguiente punto. El nodo origen se inicializa con un valor $+1$ para iniciar el movimiento, los nodos intermedios mantienen un valor 0 que asegura el tránsito continuo, y el nodo destino se marca con -1.

3.2.2 Algoritmo de Dijkstra

Existen varios algoritmos para calcular la distancia mínima entre dos nodos. En este trabajo se utilizará el **algoritmo de Dijkstra** (Dijkstra, 1959), cuyo objetivo es encontrar el camino de menor coste desde un vértice de origen hasta todos los demás vértices del grafo. Este algoritmo exige que todos los pesos de las aristas sean no negativos.

Procedimiento paso a paso

El algoritmo funciona de manera iterativa siguiendo cuatro etapas :

1. **Inicialización.** Se asigna al nodo de origen una distancia de 0 y al resto de los nodos una distancia de ∞, dado que inicialmente se desconoce cómo llegar a ellos.
2. **Exploración.** Desde el nodo actual, se examinan todos sus vecinos que aún no han sido visitados.
3. **Actualización (relajación).** Se calcula la distancia acumulada hasta cada vecino. Si el nuevo valor es menor que el almacenado previamente, se actualiza con el valor más bajo.
4. **Selección.** Se elige como siguiente nodo aquel con la distancia acumulada más pequeña y se repite el proceso hasta haber procesado todos los nodos necesarios.

Ejemplo 3.2.1. Consideremos la red ponderada de la figura siguiente. El objetivo es determinar el camino de coste mínimo entre el nodo 1, que actúa como origen, y el nodo 2, que actúa como destino. Los pesos asociados a las aristas representan el coste de desplazamiento entre nodos adyacentes.

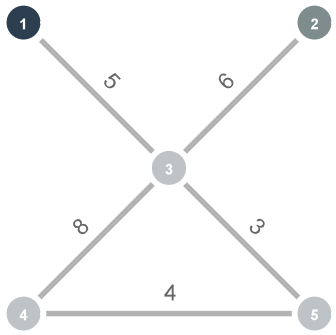


Figura 3.1: Red ponderada utilizada para ilustrar el algoritmo de Dijkstra.

A continuación se muestra la evolución de las distancias mínimas calculadas por el algoritmo. En las tablas se utiliza la siguiente notación:

- [**Número**]: nodo con distancia definitiva (fijado).
- **Negrita**: vecinos del nodo actual bajo evaluación.
- ∞: distancia desconocida.

Paso 0. Inicialización. Nos situamos en el nodo de partida (nodo 1). Su distancia inicial es 0, ya que la distancia de un nodo hacia sí mismo es nula, y el resto de distancias son desconocidas (∞).

Nodo	1	2	3	4	5
Distancia	[**0**]	∞	∞	∞	∞

Cuadro 3.1: Paso 0: inicialización del algoritmo de Dijkstra.

Paso 1. Exploración desde el nodo 1. El único vecino del nodo 1 es el nodo 3. Se calcula su distancia ($0 + 5 = 5$). Al ser la única opción disponible, se fija el nodo 3.

Nodo	1	2	3	4	5
Distancia	[0]	∞	[**5**]	∞	∞

Cuadro 3.2: Paso 1: fijación del nodo 3.

Paso 2. Exploración desde el nodo 3. Desde el nodo 3 se evalúan sus vecinos: nodo 2 ($5+6=11$), nodo 4 ($5+8=13$) y nodo 5 ($5+3=8$). El valor más pequeño es 8, por lo que se fija el nodo 5.

Nodo	1	2	3	4	5
Distancia	[0]	**11**	[5]	**13**	[**8**]

Cuadro 3.3: Paso 2: fijación del nodo 5.

Paso 3. Exploración desde el nodo 5. Desde el nodo 5 se evalúa su único vecino pendiente: el nodo 4. La nueva distancia es $8+4=12$, que mejora el valor anterior de 13, por lo que se actualiza. Comparando los nodos pendientes (nodo 2 con distancia 11 y nodo 4 con distancia 12), se fija el nodo 2 por ser el de menor coste.

Nodo	1	2	3	4	5
Distancia	[0]	[**11**]	[5]	**12**	[8]

Cuadro 3.4: Paso 3: fijación del nodo 2.

Paso 4. Finalización. Al fijar el nodo 2, que es el destino, el algoritmo concluye. La distancia mínima desde el nodo 1 hasta el nodo 2 es **11**. El nodo 4 queda con su distancia mínima de 12 para futuras consultas.

Nodo	1	2	3	4	5
Distancia	[0]	[11]	[5]	12	[8]

Cuadro 3.5: Paso 4: finalización del algoritmo.

La tabla siguiente resume la evolución completa del algoritmo de Dijkstra sobre esta red:

Paso	**Nodo fijado**	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$
0	Inicio (1)	[**0**]	∞	∞	∞	∞
1	1	[0]	∞	[**5**]	∞	∞
2	3	[0]	11	[5]	13	[**8**]
3	5	[0]	[**11**]	[5]	12	[8]
4	2 (Fin)	[0]	[11]	[5]	12	[8]

Cuadro 3.6: Evolución completa del algoritmo de Dijkstra.

3.2.3 Teoría de Emparejamiento Perfecto

Siguiendo a Lovász y Plummer (1986), un **emparejamiento** M es un subconjunto de aristas $M \subseteq E(G)$ tal que ningún vértice es incidente a más de una arista en M. Se denomina **emparejamiento**

perfecto si todos los vértices del grafo están cubiertos, es decir, cada vértice incide exactamente en una arista de M.

En términos intuitivos, un emparejamiento perfecto consiste en formar parejas de vértices de modo que cada vértice tenga exactamente una pareja y ninguno quede sin emparejar. En el contexto del Problema del Cartero Chino, dado un grafo completo auxiliar $G' = (V', E')$ cuyos vértices son los nodos de grado impar de la red original, se busca el emparejamiento perfecto de peso mínimo:

$$\min \sum_{e \in M} w(e).$$

El problema de encontrar el emparejamiento perfecto de peso mínimo puede formularse como un modelo de optimización lineal entera (Lovász y Plummer, 1986):

$$\min \sum_{i=1}^{n} \sum_{j=i+1}^{n} c_{ij} x_{ij}$$

sujeto a:

1. **Restricción de emparejamiento perfecto.** Cada nodo i debe estar emparejado exactamente con otro nodo j:

$$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{n} x_{ij} = 1, \quad \forall\, i \in \{1, \dots, n\}.$$

2. **Restricción de integridad.** Las variables son binarias:

$$x_{ij} \in \{0, 1\},$$

donde c_{ij} denota el coste de la arista que une los nodos i y j, y x_{ij} toma el valor 1 si dichos nodos se emparejan y 0 en caso contrario.

3.2.4 Algoritmo de Edmonds (Blossom Algorithm)

Es fundamental distinguir entre el concepto matemático y su resolución computacional. La teoría de emparejamiento establece qué debe conseguirse: un subgrafo donde cada vértice de grado impar tenga exactamente una pareja. Sin embargo, la teoría por sí sola no indica cómo encontrar la combinación de menor coste entre las múltiples opciones posibles.

Para resolver este problema, se utiliza el algoritmo de Edmonds (Edmonds 1965), también conocido como *Blossom Algorithm*. Este algoritmo es el motor computacional que aplica la teoría de emparejamiento sobre grafos generales, superando la dificultad que presentan los ciclos de longitud impar en grafos generales.

Sea $G_{imp} = (V_{imp}, E_{imp})$ el grafo completo auxiliar, donde $V_{imp} \subseteq V(G)$ corresponde a los nodos de grado impar del grafo original G y $E_{imp} = E(G_{imp})$ es el conjunto de aristas del grafo completo inducido sobre V_{imp}. El algoritmo busca un subconjunto de aristas $M \subseteq E_{imp}$ tal que:

1. **Emparejamiento perfecto.** Todo vértice $v \in V_{imp}$ incide exactamente en una arista de M.

2. **Optimización de coste.** Se minimiza la función objetivo:

$$Z = \min \sum_{e \in M} w(e),$$

donde $w(e)$ representa el peso de la arista e, obtenido previamente mediante el algoritmo de Dijkstra (Dijkstra, 1959).

3.2.4.1 El concepto de flor y contracción

La innovación central del algoritmo de Edmonds consiste en un mecanismo para evitar que el procedimiento quede bloqueado al encontrar un ciclo de longitud impar, denominado **flor** (*blossom*). En un conjunto con un número impar de nodos es imposible que todos queden emparejados internamente, pues siempre sobra uno. La solución se articula en dos fases:

1. **Contracción (*shrinking*).** Cuando el algoritmo detecta una flor, agrupa todos sus nodos en un único **supervértice**. Al reducir el ciclo impar a un solo nodo, el grafo se simplifica y el emparejamiento puede continuar sin inconsistencias.
2. **Expansión (*lifting*).** Una vez emparejado el resto del grafo, el supervértice se expande. Como uno de los nodos de la flor habrá quedado emparejado con un nodo externo, el número de nodos restantes dentro de la flor será par, permitiendo su emparejamiento interno.

Durante la fase de expansión, el nodo de la flor adyacente al exterior se empareja con el nodo externo correspondiente. Los nodos restantes dentro de la flor, cuyo número es siempre par al haber extraído uno, quedan emparejados entre sí. Esta propiedad garantiza que la expansión siempre produce un emparejamiento válido.

En cuanto a su eficiencia computacional, la complejidad del algoritmo original es $\mathcal{O}(|V|^2|E|)$ (Edmonds 1965), reducida a $\mathcal{O}(|V|^3)$ por Lawler (1976) mediante el uso de estructuras de datos más eficientes, y posteriormente a $\mathcal{O}(|V|(|E| + |V| \log |V|))$ por Gabow (1990).

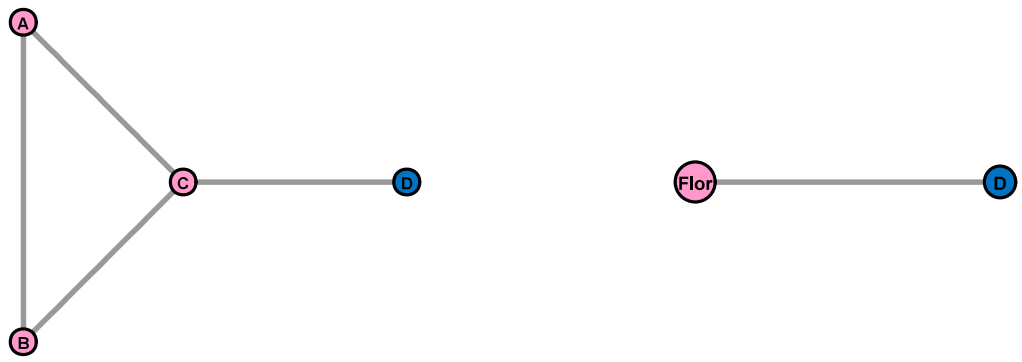


Figura 3.2: Proceso de contracción en el algoritmo de Edmonds. En la subfigura izquierdo se muestra la flor detectada (nodos A, B y C, en rosa) junto con el nodo externo D. En la subfigura del lado derecho, los tres nodos del ciclo impar han sido contraídos en un único supervértice, simplificando el grafo para continuar el emparejamiento.

3.2.4.2 Procedimiento general: Eulerianización

Para aplicar la teoría del emparejamiento al Problema del Cartero Chino, el modelo sigue una secuencia lógica conocida como proceso de **eulerianización**:

1. **Identificación de nodos impares.** Se calcula el grado $d(v)$ de cada vértice y se construye el conjunto V_{imp} formado por aquellos nodos cuyo grado es impar. Por el Lema del Apretón de Manos, $|V_{imp}|$ es siempre un número par.
2. **Cálculo de costes mínimos.** Se aplica el algoritmo de Dijkstra (Dijkstra, 1959) entre cada par de nodos de V_{imp}, obteniendo una matriz completa de distancias mínimas.

3. **Construcción del grafo auxiliar.** Con los nodos de V_{imp} y las distancias calculadas, se construye un grafo completo auxiliar $K_{|V_{imp}|}$, cuya definición se recoge a continuación.

4. **Emparejamiento perfecto mínimo.** Sobre el grafo auxiliar $K_{|V_{imp}|}$, se aplica el algoritmo de Edmonds (Edmonds y Johnson, 1973) para obtener el emparejamiento perfecto de coste mínimo M^*.

5. **Duplicación de aristas y eulerización.** Para cada par de nodos emparejados en M^*, se añaden al grafo original G las aristas del camino mínimo que los une, obtenido en el paso 2. El grafo resultante G^* tiene todos sus vértices de grado par y es, por tanto, euleriano.

6. **Obtención del tour euleriano.** Sobre el grafo eulerizado G^* se aplica un algoritmo de búsqueda de circuito euleriano, como el algoritmo de Hierholzer o el de Fleury, para determinar el tour de coste mínimo.

El coste total del tour óptimo se obtiene como la suma del coste de todas las aristas originales más el coste del emparejamiento perfecto mínimo:

$$\text{Coste}_{CPP} = \sum_{e \in E} c(e) + \sum_{e \in M^*} w(e).$$

Esta expresión refleja que el coste adicional respecto al recorrido ideal es exactamente el necesario para eulerizar la red.

3.2.5 Algoritmo de Hierholzer

Sea $G = (V, E)$ un grafo conexo donde todos sus vértices tienen grado par. El algoritmo de Hierholzer (Fleischner, 1990) se basa en el teorema que establece que un grafo euleriano puede descomponerse en una unión de ciclos disjuntos en sus aristas.

3.2.5.1 Formalización del algoritmo

1. **Construcción del ciclo inicial.** Se selecciona un vértice de inicio $v_0 \in V$ y se construye un paseo W recorriendo aristas no visitadas hasta regresar a v_0. El resultado es un ciclo C_1:

$$C_1 = (v_0, v_1, \dots, v_k, v_0).$$

2. **Identificación de vértices con grado residual.** Si las aristas de C_1 no cubren la totalidad de E, es decir, si $E(C_1) \subset E$, se selecciona un vértice $v_i \in C_1$ que tenga aristas incidentes aún no recorridas.

3. **Generación de subciclos.** Desde v_i, se repite el proceso de exploración para encontrar un nuevo ciclo C' en el grafo residual $G' = (V, E \setminus E(W))$.

4. **Operación de inserción (*splicing*).** El nuevo ciclo C' se integra en el paseo W en la posición del vértice v_i. Si $W = (x, \dots, v_i, \dots, z)$, la nueva secuencia es:

$$W_{\text{nuevo}} = \left(x, \dots, v_i, \underbrace{v_i, u_1, \dots, u_m, v_i}_{\text{ciclo nuevo } C'}, \dots, z \right).$$

5. **Condición de parada.** El proceso se repite de forma iterativa hasta que el conjunto de aristas visitadas coincida con el total de aristas del grafo:

$$E(W) = E.$$

Ejemplo 3.2.2. Consideremos el grafo euleriano trabajado a lo largo de este capítulo. Aplicando el algoritmo de Hierholzer:

1. Partimos del nodo 1 y construimos el ciclo inicial $C_1 = (1, 2, 3, 1)$, recorriendo las aristas disponibles hasta regresar al origen.
2. Observamos que el nodo 3 tiene aristas no recorridas. Construimos el subciclo $C' = (3, 4, 5, 3)$.
3. Insertamos C' en C_1 en la posición del nodo 3, obteniendo el tour euleriano completo:

$$W = (1, 2, 3, 4, 5, 3, 1).$$

Este recorrido cubre todas las aristas del grafo exactamente una vez y regresa al vértice de origen, constituyendo el tour euleriano óptimo.

3.2.6 Algoritmo de Fleury

El algoritmo de Fleury es uno de los procedimientos clásicos para encontrar un circuito o trayecto euleriano en un grafo que cumple las condiciones de paridad. Su principio fundamental es el siguiente:

No cruzar un puente a menos que sea estrictamente necesario.

Sea G un grafo euleriano. El algoritmo construye una secuencia de aristas $(e_1, e_2, \dots, e_m)$ siguiendo estas reglas:

1. **Inicio.** Si todos los vértices tienen grado par, el recorrido puede iniciarse en cualquier vértice v_0 y constituirá un circuito euleriano. Si existen exactamente dos vértices de grado impar, el recorrido debe iniciarse obligatoriamente en uno de ellos, y terminará en el otro.
2. **Selección.** En cada paso i, se elige una arista e_i del grafo actual G_i tal que:
   - e_i sea incidente al vértice actual v_{i-1},
   - e_i no sea un puente del grafo restante G_i, salvo que no existan otras opciones disponibles.
3. **Eliminación.** Tras recorrer e_i, se avanza al siguiente vértice v_i y se elimina la arista e_i del grafo de trabajo.

Todos los vértices de grado par. El agente puede comenzar en cualquier vértice, por ejemplo el depósito, y tiene garantizado que podrá regresar al mismo punto tras recorrer todas las aristas. Este caso corresponde a un **circuito euleriano**.

Ejemplo 3.2.3. Consideremos un grafo de cuatro nodos $\{A, B, C, D\}$. El nodo en rojo indica la posición actual del agente y las aristas en verde las ya recorridas. El recorrido $A \to B \to C \to D \to A$ cubre todas las aristas exactamente una vez con un coste total de 14.

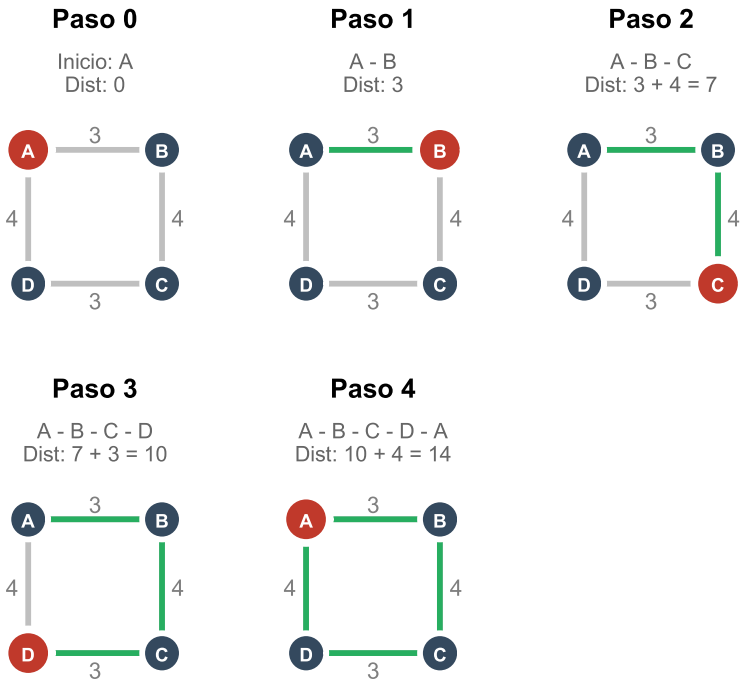


Figura 3.3: Aplicación del algoritmo de Fleury para la construcción de un circuito euleriano sobre un grafo de cuatro nodos.

Exactamente dos vértices de grado impar. Euler demostró que el recorrido solo es posible si se inicia en uno de los dos vértices de grado impar. En este caso, el recorrido terminará obligatoriamente en el otro vértice impar, dando lugar a un **camino euleriano**.

Ejemplo 3.2.4. Consideremos un grafo de cuatro nodos $\{A, B, C, D\}$ con aristas A–B (peso 3), B–C (peso 4) y C–D (peso 5). Los nodos A y D tienen grado impar, por lo que el recorrido debe iniciarse obligatoriamente en uno de ellos.

Partiendo del nodo A, el camino euleriano obtenido es:

$$A \to B \to C \to D,$$

con un coste total de $3 + 4 + 5 = 12$. El recorrido termina en el nodo D, el otro vértice de grado impar, confirmando el resultado teórico.

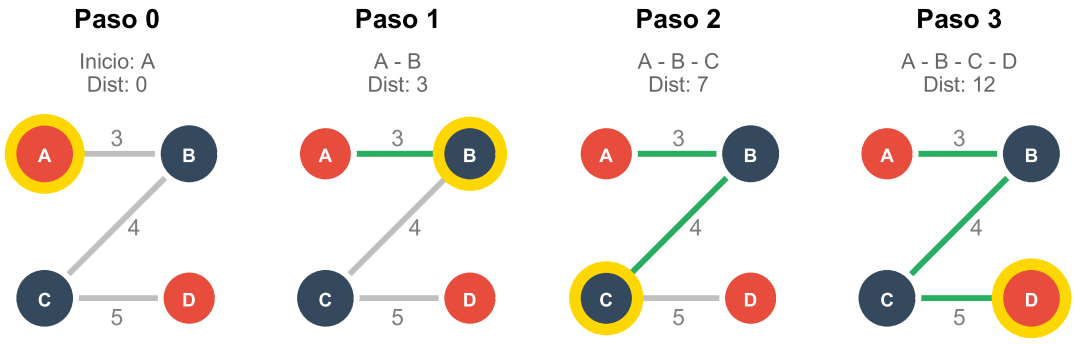


Figura 3.4: Aplicación del algoritmo de Fleury para la construcción de un camino euleriano sobre un grafo con exactamente dos vértices de grado impar. Los nodos en rojo indican los vértices impares (A y D). El borde dorado señala la posición actual del agente.

3.3 Implementación de los Algoritmos de Solución al CPP

En esta sección se ilustra de forma manual y paso a paso la resolución del Problema del Cartero Chino sobre dos casos concretos. El objetivo es mostrar cómo se aplican secuencialmente los algoritmos descritos anteriormente: identificación de vértices impares, cálculo de caminos mínimos, emparejamiento perfecto, duplicación de aristas y obtención del tour euleriano.

3.3.1 Caso 1: Grafo Euleriano

Para ilustrar el procedimiento de resolución, se considera un grafo de ejemplo adaptado de los casos clásicos presentes en la literatura sobre el Problema del Cartero Chino. El escenario se contextualiza como sigue: un camión de recogida de residuos debe recorrer todas las calles de un vecindario y regresar al punto de partida. El vehículo inicia su recorrido en la **Intersección 1**, que actúa como punto de origen y regreso, y debe cubrir la totalidad de las aristas de la red.

Dado que todos los vértices del grafo tienen grado par ($d(1) = d(2) = d(4) = d(5) = 2$ y $d(3) = 4$), se garantiza la existencia de un circuito euleriano que comenzará y terminará en la Intersección 1.

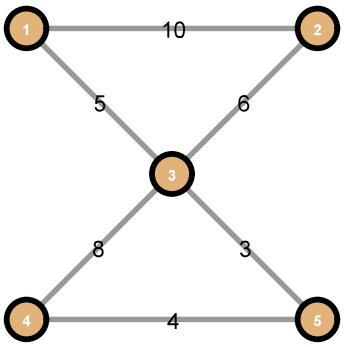


Figura 3.5: Red de recogida de residuos del Caso 1. Los valores sobre las aristas indican el coste de desplazamiento entre intersecciones.

Aplicación del Algoritmo de Fleury paso a paso

Iniciamos el recorrido en el nodo 1, que actúa como punto de partida y de regreso (Intersección 1/parque de vehículos).

Paso 1. Salida desde la base.

- **Posición actual:** nodo 1.
- **Opciones disponibles:** aristas $\{1, 2\}$ y $\{1, 3\}$.
- **Decisión:** se selecciona la arista $\{1, 2\}$.
- **Justificación:** ninguna de las dos aristas constituye un puente en este punto, por lo que cualquiera de ellas puede elegirse sin comprometer la conectividad del grafo restante.

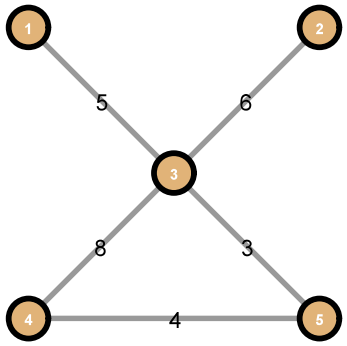


Figura 3.6: Estado de la red tras el Paso 1 del algoritmo de Fleury. La arista $\{1, 2\}$ ha sido recorrida y eliminada del grafo de trabajo.

Estado de la ruta: $1 \to 2$. **Distancia acumulada:** 10 unidades.

Paso 2. Movimiento hacia el nodo central.

- **Posición actual:** nodo 2.
- **Opciones disponibles:** solo la arista $\{2, 3\}$.
- **Decisión:** se toma la arista $\{2, 3\}$.
- **Justificación:** es la única opción disponible para continuar el recorrido desde el nodo 2.

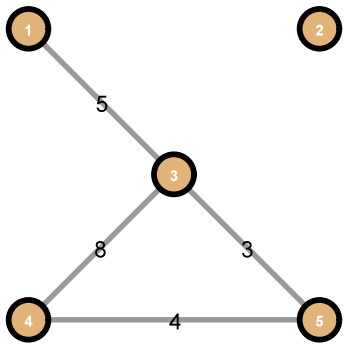


Figura 3.7: Estado de la red tras el Paso 2 del algoritmo de Fleury. La arista $\{2, 3\}$ ha sido recorrida y eliminada del grafo de trabajo.

Estado de la ruta: $1 \to 2 \to 3$. **Distancia acumulada:** $10 + 6 = 16$ unidades.

Paso 3. La regla del puente.

- **Posición actual:** nodo 3.
- **Opciones disponibles:** aristas $\{3, 1\}$, $\{3, 4\}$ y $\{3, 5\}$.

- **Decisión:** no puede elegirse la arista $\{3, 1\}$ en este momento.
- **Justificación:** si se eligiera $\{3, 1\}$ ahora, esta arista actuaría como un puente que dejaría los nodos 4 y 5 inaccesibles, imposibilitando el cierre del circuito. Por tanto, se selecciona la arista $\{3, 5\}$.

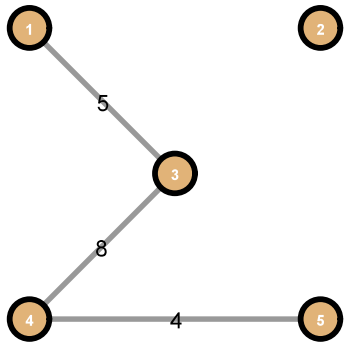


Figura 3.8: Estado de la red tras el Paso 3. La arista $\{3, 5\}$ ha sido recorrida y eliminada. La arista $\{3, 1\}$ no puede elegirse aún por constituir un puente.

Estado de la ruta: $1 \to 2 \to 3 \to 5$. **Distancia acumulada:** $16 + 3 = 19$ unidades.

Paso 4. Avance hacia el nodo 4.

- **Posición actual:** nodo 5.
- **Decisión:** se toma la arista $\{5, 4\}$.
- **Distancia del tramo:** 4 unidades.

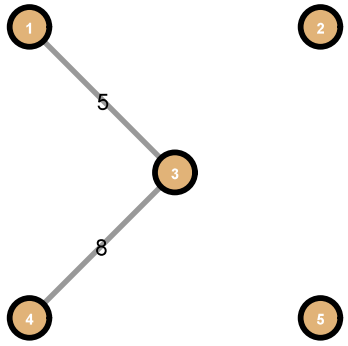


Figura 3.9: Estado de la red tras el Paso 4. La arista $\{5, 4\}$ ha sido recorrida y eliminada del grafo de trabajo.

Estado de la ruta: $1 \to 2 \to 3 \to 5 \to 4$. **Distancia acumulada:** $19 + 4 = 23$ unidades.

Paso 5. Regreso al nodo central.

- **Posición actual:** nodo 4.
- **Decisión:** se toma la arista $\{4, 3\}$.
- **Justificación:** cerramos el triángulo inferior regresando al nodo central.
- **Distancia del tramo:** 8 unidades.

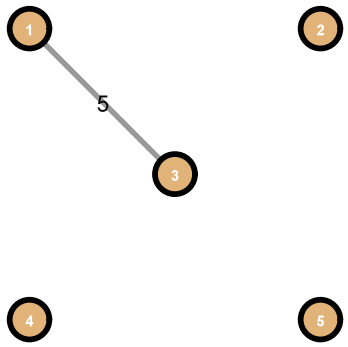


Figura 3.10: Estado de la red tras el Paso 5. La arista $\{4, 3\}$ ha sido recorrida y eliminada. Solo queda la arista $\{3, 1\}$ para cerrar el circuito.

Estado de la ruta: $1 \to 2 \to 3 \to 5 \to 4 \to 3$. **Distancia acumulada:** $23 + 8 = 31$ unidades.

Paso 6. Retorno a la base.

- **Posición actual:** nodo 3.
- **Decisión:** se toma la arista $\{3, 1\}$.
- **Justificación:** todas las demás aristas han sido recorridas, por lo que $\{3, 1\}$ deja de ser un puente y se convierte en el camino de regreso al origen.
- **Distancia del tramo:** 5 unidades.

Ruta final: $1 \to 2 \to 3 \to 5 \to 4 \to 3 \to 1$. **Distancia total:** $31 + 5 = 36$ unidades.

Análisis de resultados: Caso 1

1. **Cobertura.** Se han recorrido las seis aristas del grafo ($|E| = 6$), sin omitir ninguna.
2. **Repetición.** No se ha repetido ninguna arista, lo que confirma que el recorrido constituye un circuito euleriano.
3. **Coste total.** La distancia recorrida es la suma de los pesos de todas las aristas:

$$10 + 6 + 3 + 4 + 8 + 5 = 36 \text{ unidades.}$$

El algoritmo de Fleury ha permitido obtener la ruta óptima de cobertura total. En grafos donde todos los vértices son de grado par, el coste real del circuito euleriano coincide con el coste mínimo teórico del grafo, dado que no es necesario repetir ninguna arista.

3.3.2 Caso 2: Grafo no euleriano

En este segundo escenario, dos calles del vecindario se encuentran temporalmente fuera de servicio y no pueden ser utilizadas por el vehículo de recogida. En particular, las conexiones $\{1, 2\}$ y $\{4, 5\}$ quedan inhabilitadas debido a trabajos de reestructuración urbana o a una incidencia en la red de tuberías subterráneas.

Como consecuencia, la red operativa disponible se reduce al subgrafo formado por las aristas $\{1, 3\}$, $\{2, 3\}$, $\{3, 4\}$ y $\{3, 5\}$. El camión debe iniciar obligatoriamente su recorrido en la **Intersección 1 (Parque de Vehículos)**, espacio destinado por el ayuntamiento al resguardo de la flota municipal, y finalizar su turno en ese mismo punto.

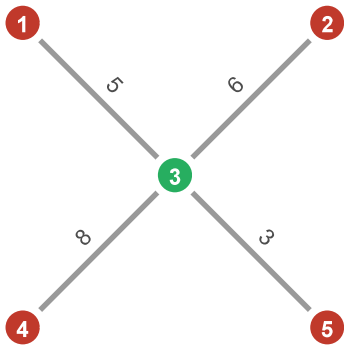


Figura 3.11: Red operativa del Caso 2 tras la inhabilitación temporal de las calles $\{1, 2\}$ y $\{4, 5\}$.

Al evaluar los grados de los vértices de esta red, nos encontramos con un escenario mixto:

- **Vértice par:** el nodo central tiene $d(3) = 4$.
- **Vértices impares:** los nodos periféricos tienen $d(1) = d(2) = d(4) = d(5) = 1$.

Dado que existen vértices de grado impar, el Teorema de Euler establece que es matemáticamente imposible trazar un circuito euleriano. Es decir, el camión no podrá recorrer todas las calles exactamente una vez y regresar a la Intersección 1 sin verse obligado a repetir algún tramo.

Para resolver esta ineficiencia, es necesario aplicar un proceso de **eulerianización** mediante el **algoritmo de Edmonds**, que identificará qué calles debe repetir el vehículo para minimizar el sobrecoste de kilometraje.

Paso 1. Identificación de los nodos impares.

Tras analizar los grados de la red, el conjunto de nodos problemáticos es:

$$V_{\text{imp}} = \{1, 2, 4, 5\}.$$

Estos son los vértices que deben emparejarse para eulerianizar la red y garantizar que el vehículo no quede bloqueado en ningún punto.

Paso 2. Cálculo de distancias mínimas mediante Dijkstra.

Para construir la matriz de distancias entre los nodos impares, aplicamos el algoritmo de Dijkstra. A modo de ejemplo, calculamos los caminos más cortos partiendo desde el **nodo 1**. Se utiliza la siguiente notación:

- [Número]: nodo con distancia definitiva (fijado).
- **Negrita**: vecinos del nodo actual bajo evaluación.
- ∞: distancia desconocida.

Paso 0. Inicialización.El algoritmo se sitúa en el nodo 1 con distancia inicial 0. Lo demás nodos se marcan con infinito.

Nodo	1	2	3	4	5
Distancia	[**0**]	∞	∞	∞	∞

Cuadro 3.7: Paso 0: inicialización desde el nodo 1.

Paso 1. Exploración desde el nodo 1. El nodo 1 es un fondo de saco cuyo único vecino es el nodo 3 (distancia 5). Al no existir otra opción, se fija el nodo 3.

Nodo	1	2	3	4	5
Distancia	[0]	∞	[**5**]	∞	∞

Cuadro 3.8: Paso 1: fijación del nodo 3.

Paso 2. Exploración desde el nodo 3. Desde el nodo 3 (distancia acumulada 5) se evalúan sus vecinos restantes:

- hacia el nodo 2: $5 + 6 = 11$,

- hacia el nodo 4: $5 + 8 = 13$,
- hacia el nodo 5: $5 + 3 = 8$.

El valor mínimo es 8, correspondiente al nodo 5, que queda fijado.

Nodo	1	2	3	4	5
Distancia	[0]	**11**	[5]	**13**	[**8**]

Cuadro 3.9: Paso 2: fijación del nodo 5.

Paso 3. Exploración desde el nodo 5. El nodo 5 es un fondo de saco cuyo único vecino es el nodo 3, ya fijado. No se descubren nuevos caminos. Comparando los nodos pendientes, el nodo 2 (distancia 11) es menor que el nodo 4 (distancia 13), por lo que se fija el nodo 2.

Nodo	1	2	3	4	5
Distancia	[0]	[**11**]	[5]	13	[8]

Cuadro 3.10: Paso 3: fijación del nodo 2.

Paso 4. Finalización. Solo queda el nodo 4 pendiente. Al no existir más caminos posibles en la red, queda fijado con distancia 13.

Nodo	1	2	3	4	5
Distancia	[0]	[11]	[5]	[**13**]	[8]

Cuadro 3.11: Paso 4: fijación del nodo 4.

Tabla resumen: traza de Dijkstra desde el nodo 1.

Paso	**Nodo fijado**	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$
0	Inicio (1)	[**0**]	∞	∞	∞	∞
1	1	[0]	∞	[**5**]	∞	∞
2	3	[0]	11	[5]	13	[**8**]
3	5	[0]	[**11**]	[5]	13	[8]
4	2	[0]	[11]	[5]	[**13**]	[8]

Cuadro 3.12: Traza completa del algoritmo de Dijkstra desde el nodo 1. Caso 2.

Trazas desde los nodos 2, 4 y 5.

Aplicando el mismo procedimiento descrito anteriormente, se obtienen las trazas del algoritmo de Dijkstra desde los nodos impares restantes. Por razones de concisión, se presentan directamente las tablas resumen de cada ejecución.

Desde el nodo 2:

Paso	**Nodo fijado**	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$
0	Inicio (2)	∞	[**0**]	∞	∞	∞
1	2	∞	[0]	[**6**]	∞	∞
2	3	11	[0]	[6]	14	[**9**]
3	5	[**11**]	[0]	[6]	14	[9]
4	1	[11]	[0]	[6]	[**14**]	[9]

Cuadro 3.13: Traza completa del algoritmo de Dijkstra desde el nodo 2. Caso 2.

Desde el nodo 4:

Paso	**Nodo fijado**	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$
0	Inicio (4)	∞	∞	∞	[**0**]	∞
1	4	∞	∞	[**8**]	[0]	∞
2	3	13	14	[8]	[0]	[**11**]
3	5	[**13**]	14	[8]	[0]	[11]
4	1	[13]	[**14**]	[8]	[0]	[11]

Cuadro 3.14: Traza completa del algoritmo de Dijkstra desde el nodo 4. Caso 2.

Desde el nodo 5:

Paso	**Nodo fijado**	$d(1)$	$d(2)$	$d(3)$	$d(4)$	$d(5)$
0	Inicio (5)	∞	∞	∞	∞	[**0**]
1	5	∞	∞	[**3**]	∞	[0]
2	3	[**8**]	9	[3]	11	[0]
3	1	[8]	[**9**]	[3]	11	[0]
4	2	[8]	[9]	[3]	[**11**]	[0]

Cuadro 3.15: Traza completa del algoritmo de Dijkstra desde el nodo 5. Caso 2.

Construcción del grafo auxiliar K_4.

Una vez calculadas todas las distancias mínimas mediante el algoritmo de Dijkstra, se aísla el conjunto de nodos problemáticos $V_{\text{imp}} = \{1, 2, 4, 5\}$ y se construye un **grafo auxiliar completo** K_4, en el que cada nodo está conectado con todos los demás. Los pesos de las aristas de este grafo auxiliar no son distancias arbitrarias, sino que corresponden exactamente a los valores obtenidos en la matriz de distancias mínimas calculada en el paso anterior.

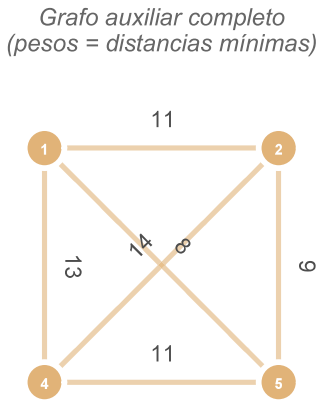


Figura 3.12: Grafo auxiliar completo K_4 construido sobre los nodos impares $V_{\text{imp}} = \{1, 2, 4, 5\}$. Los pesos de las aristas corresponden a las distancias mínimas obtenidas mediante el algoritmo de Dijkstra.

Análisis de combinaciones: emparejamiento perfecto

A partir del grafo auxiliar K_4, se evalúan las tres combinaciones posibles para emparejar los cuatro nodos impares. El objetivo es encontrar el emparejamiento perfecto de peso mínimo.

Combinación	Parejas	Distancias	Coste extra
Opción A	$(1,2)$ y $(4,5)$	$11 + 11$	22
Opción B	$(1,5)$ y $(2,4)$	$8 + 14$	22
Opción C	$(1,4)$ y $(2,5)$	$13 + 9$	22

Cuadro 3.16: Evaluación de las tres combinaciones de emparejamiento perfecto sobre el grafo auxiliar K_4.

En este caso particular, las tres combinaciones producen exactamente el mismo coste extra de $Z = 22$ unidades. Este resultado no es habitual en la práctica y refleja una simetría estructural de la red reducida. Desde el punto de vista del algoritmo de Edmonds, cualquiera de las tres opciones es igualmente óptima. En la implementación, se seleccionará la **Opción A**, formada por las parejas $(1, 2)$ y $(4, 5)$, por seguir el orden natural de los nodos.

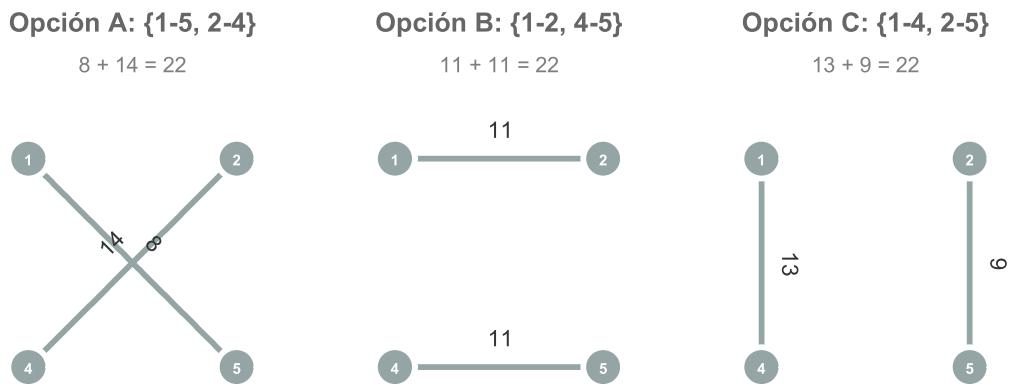


Figura 3.13: Comparación de las tres opciones de emparejamiento perfecto sobre el grafo auxiliar K_4.

Tal como se observa en la figura, nos encontramos ante un caso de empate: las tres combinaciones posibles producen exactamente el mismo coste extra de $Z = 22$ unidades. Por convención, se selecciona la **Opción A**, formada por las parejas $\{1, 5\}$ y $\{2, 4\}$.

En este caso de estudio, el grafo auxiliar K_4 contiene únicamente cuatro nodos impares y no presenta ciclos de longitud impar. Por tanto, el algoritmo de Edmonds opera en el **Nivel 2** de la estrategia multicapa, evaluando directamente las tres combinaciones posibles sin necesidad de aplicar la técnica de contracción de flores. El resultado es además especialmente llamativo: las tres opciones producen el mismo coste extra de $Z = 22$ unidades, lo que refleja una simetría estructural de la red reducida.

Duplicación de aristas

Una vez seleccionado el emparejamiento óptimo, se incorporan al grafo original las aristas duplicadas correspondientes. Para la Opción A, los caminos mínimos que conectan las parejas son:

- pareja $\{1, 5\}$: el camino mínimo es $1 \to 3 \to 5$, con coste 8.
- pareja $\{2, 4\}$: el camino mínimo es $2 \to 3 \to 4$, con coste 14.

Ambos caminos se duplican en la red original, añadiendo una copia de cada arista recorrida. El resultado es un **multigrafo** en el que cada tramo duplicado dispone de su respectiva vía de retorno ($x_e = 2$). Tras esta operación, todos los vértices de la red pasan a tener grado par, garantizando así la existencia de un tour euleriano.

La siguiente figura muestra el resultado del proceso de eulerianización. En la subfigura del lado izquierdo se presenta la red original, donde los nodos 1, 2, 4 y 5 tienen grado impar. En la sugfigura del lado derecho, tras duplicar los caminos $1 \rightarrow 3 \rightarrow 5$ y $2 \rightarrow 3 \rightarrow 4$, todos los vértices pasan a tener grado par. Las aristas naranjas representan los tramos duplicados.

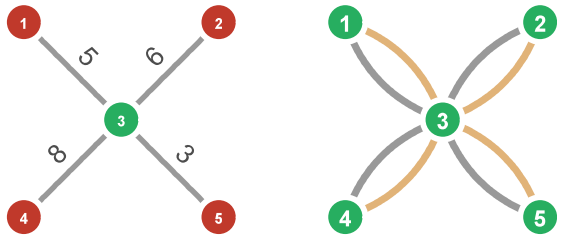


Figura 3.14: Resultado del proceso de eulerianización. A la derecha, el multigrafo resultante tras la duplicación de las aristas correspondientes al emparejamiento óptimo.

Una vez verificado que todos los vértices del multigrafo tienen grado par, la red admite un tour euleriano. A continuación se aplica el algoritmo de Fleury para construir dicho recorrido paso a paso.

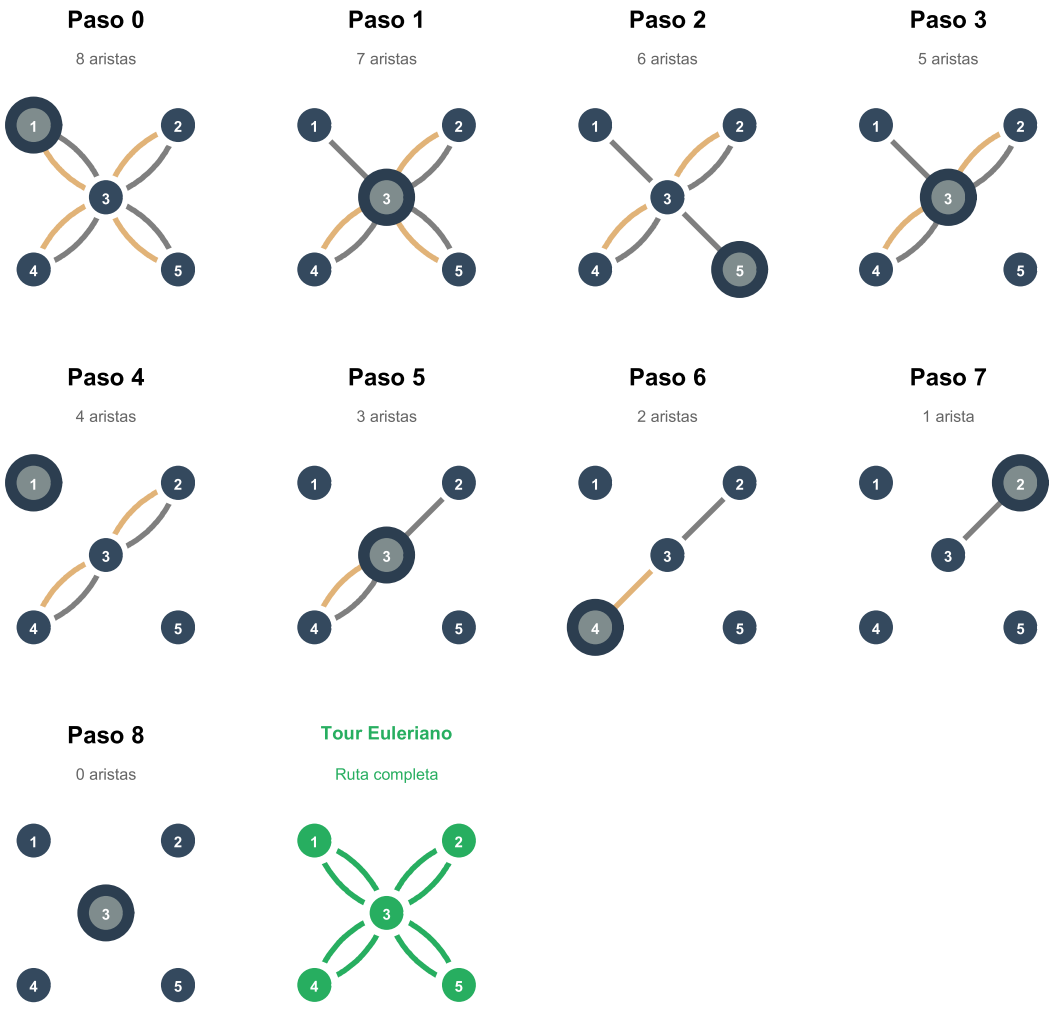


Figura 3.15: Aplicación del algoritmo de Fleury sobre el multigrafo eulerianizado del Caso 2. Las aristas naranjas corresponden a los tramos originales de la red y las grises a los tramos duplicados por el proceso de eulerianización. El panel final muestra el tour euleriano completo.

La tabla siguiente resume los costes del proceso de resolución:

Cuadro 3.17: Resumen del Problema del Cartero Chino. Caso 2.

Concepto	Valor
Coste original del grafo	$5 + 6 + 8 + 3 = 22$
Aristas duplicadas	$1 \to 3 \to 5$ y $2 \to 3 \to 4$
Coste adicional (Edmonds)	$8 + 14 = 22$
Coste total	$\mathbf{22 + 22 = 44}$

El coste total de la ruta óptima asciende a 44 unidades, de las cuales 22 corresponden al recorrido de la red original y otras 22 al sobrecoste inevitable derivado de la necesidad de repetir ciertos tramos para eulerianizar la red. Este sobrecoste es el mínimo posible, garantizado por el emparejamiento perfecto de peso mínimo obtenido mediante el algoritmo de Edmonds.

Capítulo 4

Arquitectura Computacional

4.1 Implementación Computacional

Resolver el Problema del Cartero Chino de forma manual resulta viable únicamente para redes pequeñas. En el caso de redes complejas, se requiere una arquitectura algorítmica automatizada que permita reducir el tiempo de respuesta. Con este objetivo, se ha desarrollado el paquete denominado `CoopPostman` utilizando el lenguaje de programación R.

El funcionamiento del paquete se desarrolla en torno a una secuencia de fases bien diferenciadas, cada una de las cuales encapsula una parte del proceso de resolución del Problema del Cartero Chino. La implementación sigue el esquema de eulerianización descrito en el Capítulo 4: diagnóstico de la red, cálculo de caminos mínimos, emparejamiento perfecto, duplicación de aristas y obtención del tour euleriano.

4.1.1 Fase 1: Diagnóstico de la red

El primer paso consiste en analizar la topología de la red para determinar si el grafo es euleriano o requiere un proceso de eulerianización previo. La función calcula el grado de cada vértice mediante `degree()` de la librería `igraph` y filtra aquellos cuyo resto al dividir entre 2 es distinto de cero. Si el resultado es un vector vacío, el grafo es euleriano y no requiere reparación.

4.1.2 Función `odd_vertices()`

La función `odd_vertices()` tiene como objetivo identificar los vértices de grado impar del grafo. Recibe como *input* el grafo original $G = (V, E)$ y devuelve como *output* la lista de vértices de grado impar, es decir, el conjunto $V_{\text{imp}} = \{v \in V : d(v) \equiv 1 \pmod 2\}$.

```
odd_vertices <- function(grafo) {
  # Calculamos el grado de cada nodo
  grados <- igraph::degree(grafo, mode = "all")
  # Filtramos los nodos cuyo grado es impar
  nodos_impares <- names(grados[(grados %% 2) == 1])
  return(nodos_impares)
}
```

4.2 Fase 2: Determinación de costes de conexión

La función `get_dijkstra_matrix()` tiene como objetivo construir la matriz de distancias mínimas entre todos los pares de vértices impares. Recibe como *input* el grafo original y la lista de vértices impares, y devuelve como *output* una matriz simétrica $D \in \mathbb{R}^{n \times n}$, donde cada entrada d_{ij} representa la distancia mínima entre los vértices impares i y j:

$$D = \begin{pmatrix} 0 & d_{12} & \cdots & d_{1n} \\ d_{21} & 0 & \cdots & d_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n1} & d_{n2} & \cdots & 0 \end{pmatrix}$$

La función utiliza internamente el algoritmo de Dijkstra, implementado a través de la función `distances()` de la librería `igraph`, para encontrar la ruta de coste mínimo entre cada par de vértices impares, incluso cuando estos no están directamente conectados por una arista.

Observación 4.2.1. La matriz D es simétrica, es decir, $d_{ij} = d_{ji}$ para todo par $i, j \in V_{\text{imp}}$, dado que el grafo es no dirigido y el coste de desplazamiento entre dos vértices es independiente del sentido del recorrido.

```
get_dijkstra_matrix <- function(grafo, nodos_impares) {
  matriz_distancias <- igraph::distances(
    graph = grafo,
    v     = nodos_impares,
    to    = nodos_impares,
    mode  = "all"
  )
  return(matriz_distancias)
}
```

La función utiliza internamente el algoritmo de Dijkstra para encontrar la ruta de coste mínimo entre cada par de vértices impares, incluso cuando estos no están directamente conectados por una arista. El resultado es una matriz cuadrada $n \times n$ donde cada entrada d_{ij} representa la distancia mínima entre los vértices impares i y j.

4.3 Fase 3: Análisis combinatorio del sistema

En esta fase del algoritmo el objetivo es encontrar el **emparejamiento perfecto de peso mínimo** entre los vértices impares, es decir, la forma de agruparlos en parejas tal que la suma de las distancias entre cada pareja sea mínima.

4.3.1 Función `get_optimal_matching()`

La función `get_optimal_matching()` determina el emparejamiento óptimo entre los vértices impares. Recibe como *input* la matriz de distancias mínimas generada en la Fase 2 y devuelve como *output* la lista de parejas óptimas, indicando qué vértices impares deben conectarse mediante la duplicación de sus caminos mínimos.

Sea $n = |V_{\text{imp}}|$ el número de vértices impares. El número total de emparejamientos perfectos posibles viene dado por (Lovász y Plummer 1986):

$$\mathcal{M}(n) = (n-1)!! = (n-1)(n-3)(n-5)\cdots(1)$$

Esta función crece de forma exponencial, tal como se muestra en la Tabla~4.1.

Nodos impares (n)	**Combinaciones $\mathcal{M}(n)$**
2	1
4	3
6	15
8	105
10	945
12	10.395
20	654.729.075

Cuadro 4.1: Crecimiento del espacio de búsqueda según el número de vértices impares.

Este crecimiento hace inviable la enumeración exhaustiva para valores grandes de n, lo que justifica el uso de un algoritmo especializado que garantice el óptimo global en tiempo polinomial.

La función implementa una estrategia de dos niveles:

Nivel 1. Resolución inmediata ($n = 2$). Cuando solo existen dos vértices de grado impar, no es necesaria la optimización combinatoria. El algoritmo extrae directamente el coste de la conexión única desde la matriz de distancias.

Nivel 2. Algoritmo de Blossom para $n \geq 4$. Para cuatro o más vértices impares, la función delega la resolución en el **algoritmo de Blossom de Edmonds**, implementado en la función `max_weight_matching` de la librería `networkx` (Hagberg et al. 2008), accesible desde R mediante el paquete `reticulate` (Ushey et al. 2023).

Dado que dicha función resuelve un problema de maximización de pesos, se aplica la transformación estándar:

$$w'_{ij} = M - d_{ij}$$

donde d_{ij} es la distancia mínima entre los vértices impares i y j, y M es una constante estrictamente mayor que todas las distancias de la matriz. Esta transformación garantiza que el emparejamiento de máximo peso en el grafo transformado coincida con el emparejamiento perfecto de mínimo coste en el grafo original.

Nivel	**Método**	**Rango**	**Garantía**
1	Resolución directa	$n = 2$	Óptimo global
2	Blossom (Edmonds)	$n \geq 4$	Óptimo global

Cuadro 4.2: Estrategia de resolución del emparejamiento perfecto de mínimo coste.

4.4 Fase 4: Reparación de la red

En esta fase se procede a añadir las aristas duplicadas que el emparejamiento óptimo ha determinado como necesarias para convertir el grafo en euleriano. La operación central de esta fase es la duplicación de los caminos mínimos entre cada pareja de vértices impares, utilizando funciones de la librería `igraph`.

La función `duplicate_shortest_path()` tiene como objetivo duplicar las aristas del camino mínimo entre cada pareja de vértices impares, incorporándolas al grafo original. Recibe como *input* el grafo actual y los identificadores de los dos nodos de la pareja a conectar (`from` y `to`), y devuelve como *output* el grafo actualizado con las aristas duplicadas incorporadas, listo para la siguiente iteración o para el trazado del circuito euleriano.

La función se apoya en tres operaciones internas de `igraph` :

1. **`igraph::shortest_paths()`.** Calcula el camino mínimo entre los dos nodos de la pareja. El argumento `output = "epath"` indica que se devuelvan las aristas que forman dicho camino, lo que permite identificar exactamente qué tramos deben duplicarse.
2. **`igraph::ends()`.** Extrae los pares de vértices extremos de cada arista del camino mínimo, convirtiéndolos en el formato vectorial que requiere la función de adición de aristas.
3. **`igraph::add_edges()`.** Incorpora las nuevas aristas al grafo, conservando el mismo peso (`weight`) que tenían las aristas originales. Las aristas duplicadas se marcan con el atributo `color = "red"` para su identificación visual.

Las aristas duplicadas no son aristas físicamente nuevas, sino una representación matemática de la obligación del agente de recorrer ese tramo más de una vez. En el contexto de la recogida de residuos, esto significa que el camión deberá transitar por esa calle en dos ocasiones, siendo este el sobrecoste mínimo inevitable que el emparejamiento perfecto ha garantizado.

```r
duplicate_shortest_path <- function(grafo, from, to) {


  ruta <- igraph::shortest_paths(
    grafo,
    from   = from,
    to     = to,
    output = "epath"
  )
  aristas_camino <- ruta$epath[[1]]


  grafo_modificado <- igraph::add_edges(
    grafo,
    edges = as.vector(t(igraph::ends(grafo, aristas_camino))),
    attr  = list(
      weight = igraph::edge_attr(grafo, "weight", aristas_camino),
      color  = "red"
    )
  )
  return(grafo_modificado)
}
```

Tras aplicar esta función de forma iterativa para cada pareja del emparejamiento óptimo, todos los vértices del grafo tendrán grado par y la red será euleriana.

4.5 Fase 5: El circuito euleriano

Con el grafo ya eulerianizado, todos sus vértices tienen grado par y se garantiza la existencia de un circuito euleriano. En esta fase se procede a trazar dicho recorrido óptimo mediante el algoritmo de Fleury.

La función `fleury_path()` tiene como objetivo construir la secuencia óptima de visita de todas las aristas del grafo eulerianizado, formando un circuito cerrado que comienza y termina en el mismo vértice. Recibe como *input* el grafo eulerianizado resultante de la Fase 4, con todos sus vértices de grado par, y devuelve como *output* un vector de caracteres con la secuencia ordenada de vértices que forma el circuito euleriano óptimo, por ejemplo `c("1","3","2","3","4","3","5","3","1")`.

La función implementa el **algoritmo de Fleury** siguiendo tres principios fundamentales:

1. **Eliminación progresiva.** Cada arista recorrida se elimina del grafo de trabajo, de modo que el algoritmo nunca repite un tramo ya visitado.
2. **Regla del puente.** Antes de seleccionar la siguiente arista, la función verifica si dicha arista es un puente del grafo residual mediante `igraph::count_components()` (Csardi y Nepusz, 2023). Solo se cruza un puente cuando no existe ninguna otra opción disponible.
3. **Garantía de completitud.** El proceso continúa hasta que el conjunto de aristas visitadas coincide con el total de aristas del grafo eulerianizado.

La detección de puentes mediante `count_components()` es más robusta que otras implementaciones basadas en `is_bridge()`, ya que funciona correctamente en multigrafos con aristas paralelas, como el grafo eulerianizado resultante de la Fase 4.

```
fleury_path <- function(grafo) {

  g_temp      <- grafo
  nodo_actual <- igraph::V(g_temp)$name[1]
  ruta        <- c(nodo_actual)

  while (igraph::ecount(g_temp) > 0) {

    vecinos   <- names(igraph::neighbors(g_temp, nodo_actual))
    siguiente <- NULL

    if (length(vecinos) == 1) {

      siguiente <- vecinos[1]
    } else {

      for (candidato in vecinos) {
        n_antes  <- igraph::count_components(g_temp)
        g_test   <- igraph::delete_edges(
          g_temp,
          paste0(nodo_actual, "|", candidato)
        )
        n_despues <- igraph::count_components(g_test)

        if (n_despues <= n_antes) {
          siguiente <- candidato
          break
        }
      }
```

```
      # Si todas las aristas son puentes, tomamos la primera disponible
      if (is.null(siguiente)) siguiente <- vecinos[1]
    }

    ruta    <- c(ruta, siguiente)
    g_temp  <- igraph::delete_edges(
      g_temp,
      paste0(nodo_actual, "|", siguiente)
    )
    nodo_actual <- siguiente
  }

  return(ruta)
}
```

Función auxiliar `route_cost()`

La función `route_cost()` tiene como objetivo calcular el coste total de un circuito euleriano recorriendo las aristas del multigrafo en el orden exacto indicado por la ruta, evitando contabilizar dos veces las aristas paralelas (Csardi y Nepusz, 2023). Recibe como *input* el multigrafo eulerianizado y el vector de la ruta devuelto por `fleury_path()`, y devuelve como *output* un valor numérico con el coste total del circuito en las unidades de los pesos del grafo (metros, minutos, etc.).

La función `route_cost()` es invocada internamente por `solve_cpp()` para calcular el componente `$coste_total` del resultado. Su diseño basado en eliminación progresiva de aristas es especialmente relevante en multigrafos con tramos duplicados, donde una suma directa de pesos produciría un resultado incorrecto al no distinguir entre aristas paralelas.

```
route_cost <- function(grafo, ruta) {

  if (length(ruta) < 2) return(0)

  g_temp      <- grafo
  coste_total <- 0

  for (i in seq_len(length(ruta) - 1)) {
    u <- ruta[i]
    v <- ruta[i + 1]

    # Identificamos todas las aristas incidentes en el nodo actual
    candidatas <- igraph::incident(g_temp, u, mode = "all")
    extremos   <- igraph::ends(g_temp, candidatas, names = TRUE)

    # Seleccionamos la primera arista disponible entre u y v
    idx <- which(
      (extremos[, 1] == u & extremos[, 2] == v) |
      (extremos[, 1] == v & extremos[, 2] == u)
    )[1]

    arista_id   <- candidatas[idx]
    coste_total <- coste_total + igraph::E(g_temp)[arista_id]$weight

    # Eliminamos la arista usada para no contabilizarla dos veces
    g_temp <- igraph::delete_edges(g_temp, arista_id)
  }
```

```
  return(coste_total)
}
```

4.6 Función `solve_cpp()`

Todas las funciones anteriores están integradas en la función envolvente `solve_cpp()`, que gestiona automáticamente el flujo completo de resolución del Problema del Cartero Chino. Para el cálculo preciso del coste total del circuito, esta función utiliza internamente una función auxiliar denominada `route_cost()`.

La necesidad de esta función auxiliar surge de una particularidad de los multigrafos: cuando entre dos vértices existen varias aristas paralelas, una simple suma de pesos no permite distinguir cuál de ellas se recorre en cada paso del circuito. La función `route_cost()` resuelve este problema eliminando progresivamente cada arista a medida que es recorrida, garantizando que el coste se calcule con precisión incluso en presencia de tramos duplicados.

Adicionalmente, `solve_cpp()` incorpora un **mecanismo de cortocircuito** que evalúa en primer lugar si el grafo ya es euleriano. Si todos los vértices tienen grado par, se omiten las fases de diagnóstico, optimización y reparación, y se procede directamente al trazado del circuito. En caso contrario, se ejecuta el protocolo completo de **eulerización**.

La función `solve_cpp()` actúa como el punto de entrada principal del paquete. Recibe como *input* el grafo original $G = (V(G), E(G))$ con el atributo `weight` definido en cada arista, y devuelve como *output* una lista estructurada con los siguientes componentes:

- `$grafo`: el grafo eulerianizado resultante.
- `$ruta`: la secuencia ordenada de vértices que compone el circuito óptimo.
- `$ruta_texto`: la representación textual de la ruta.
- `$vertices_impares`: el conjunto de vértices de grado impar identificados en la fase de diagnóstico.
- `$matching`: el emparejamiento perfecto de peso mínimo aplicado.
- `$coste_original`: el coste asociado al recorrido de las aristas originales.
- `$coste_reparacion`: el sobrecoste inducido por la duplicación de aristas.
- `$coste_total`: la suma final de ambos costes.

```
solve_cpp <- function(grafo) {

  # Coste original del grafo
  coste_original <- sum(igraph::E(grafo)$weight)

  # FASE 1: Identificación de vértices impares
  impares        <- odd_vertices(grafo)
  grafo_reparado <- grafo
  parejas        <- NULL
  coste_reparacion <- 0

  if (length(impares) == 0) {
    message("El grafo es euleriano. No requiere duplicación.")
  } else {
    message(sprintf(
```

```
      "Se han detectado %d vértices impares. Iniciando optimización...",
      length(impares)
    ))

    # FASE 2: Matriz de distancias mínimas
    matriz <- get_dijkstra_matrix(grafo, impares)

    # FASE 3: Emparejamiento perfecto de peso mínimo
    parejas <- get_optimal_matching(matriz)

    # FASE 4: Duplicación de caminos mínimos
    for (i in 1:nrow(parejas)) {
      grafo_reparado <- duplicate_shortest_path(
        grafo_reparado,
        parejas$n1[i],
        parejas$n2[i]
      )
    }

    coste_reparacion <- sum(igraph::E(grafo_reparado)$weight) - coste_original
  }

  # FASE 5: Circuito euleriano
  ruta_nodos <- fleury_path(grafo_reparado)

  # Coste total
  coste_total <- coste_original + coste_reparacion

  # Resultado
  return(list(
    grafo            = grafo_reparado,
    ruta             = ruta_nodos,
    ruta_texto       = paste(ruta_nodos, collapse = " -> "),
    vertices_impares = impares,
    matching         = parejas,
    coste_original   = coste_original,
    coste_reparacion = coste_reparacion,
    coste_total      = coste_total
  ))
}
```

4.6.1 Flujo de ejecución

El flujo de ejecución sigue la secuencia lógica siguiente:

1. **Diagnóstico.** `odd_vertices()` identifica los vértices de grado impar. Si no hay ninguno, se salta al paso 5.
2. **Costes.** `get_dijkstra_matrix()` construye la matriz de distancias mínimas entre los vértices impares (Dijkstra, 1959).
3. **Emparejamiento.** `get_optimal_matching()` determina las parejas óptimas aplicando la estrategia multicapa (Lovász y Plummer, 1986).
4. **Reparación.** `duplicate_shortest_path()` se ejecuta de forma iterativa para cada pareja, incorporando las aristas duplicadas al grafo.

5. **Circuito.** `fleury_path()` construye el circuito euleriano óptimo sobre el grafo ya reparado. El coste total se calcula mediante `route_cost()`.

El mecanismo de cortocircuito tiene una relevancia práctica directa: en redes donde todos los vértices son de grado par, como el Caso 1 desarrollado en este trabajo, el paquete detecta automáticamente que no es necesario ningún proceso de eulerianización y procede directamente al trazado del circuito, reduciendo el tiempo de cómputo al mínimo posible.

La figura siguiente sintetiza el protocolo completo de resolución implementado en `solve_cpp()`.

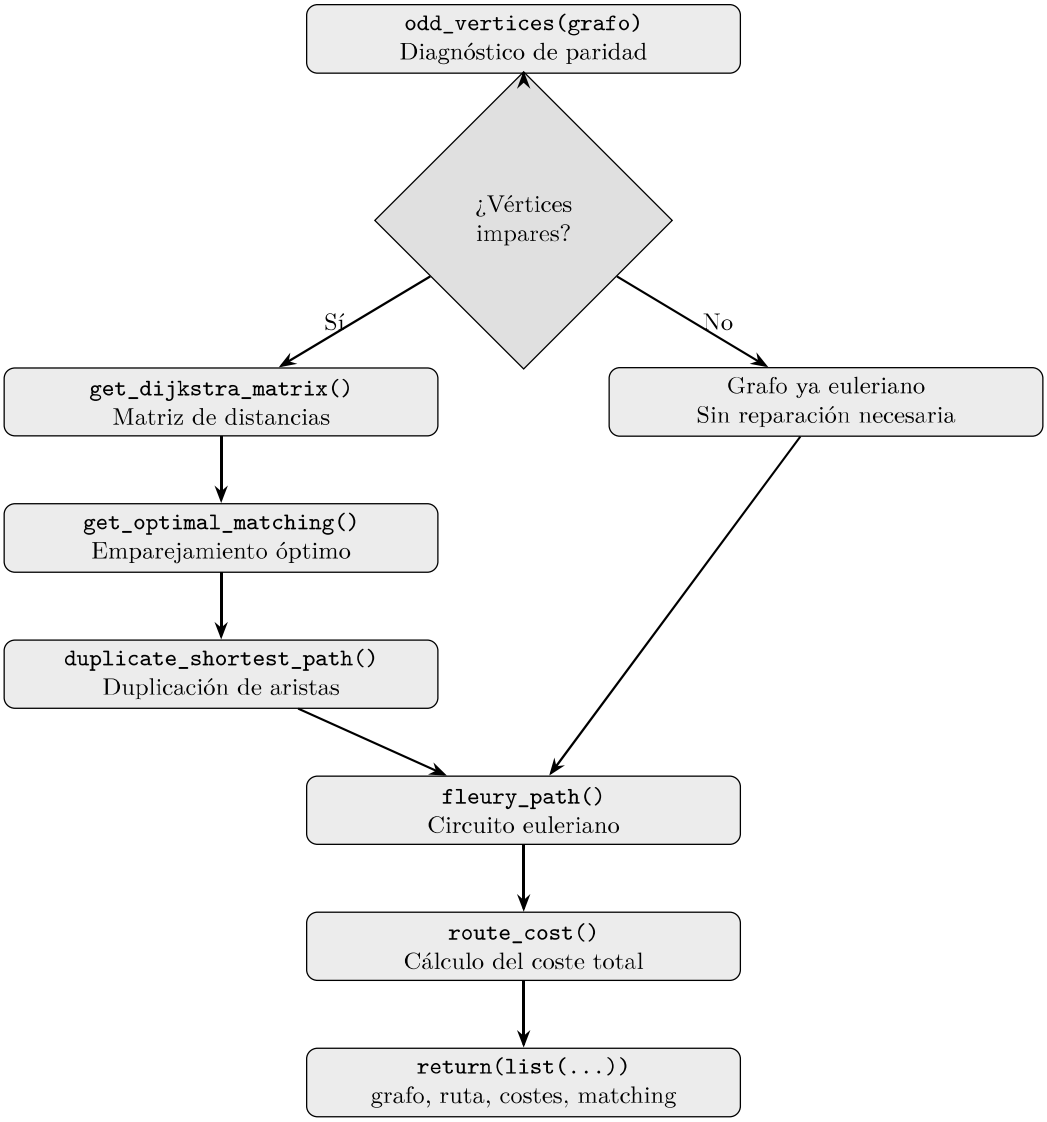


Figura 4.1: Diagrama de flujo del algoritmo `solve_cpp()`.

4.7 Caso de estudio genérico

Para validar el funcionamiento del paquete `CoopPostman`, se aplica la función `solve_cpp()` sobre el grafo de la red de recogida de residuos del Caso 2, definido anteriormente.

```
solve_cpp(g_cartero)
```

```
## $grafo
## IGRAPH 2356191 UNW- 5 8 --
## + attr: name (v/c), grado (v/n), paridad (v/c), weight (e/n), color
## | (e/c)
## + edges from 2356191 (vertex names):
## [1] 1--3 2--3 3--4 3--5 1--3 3--4 2--3 3--5
##
## $ruta
## [1] "1" "3" "2" "3" "4" "3" "5" "3" "1"
##
## $ruta_texto
## [1] "1 -> 3 -> 2 -> 3 -> 4 -> 3 -> 5 -> 3 -> 1"
##
## $vertices_impares
## [1] "1" "2" "4" "5"
##
## $matching
##   n1 n2 coste
## 1  1  4    13
## 2  2  5     9
##
## $coste_original
## [1] 22
##
## $coste_reparacion
## [1] 22
##
## $coste_total
## [1] 44
```

La ejecución de `solve_cpp(g_cartero)` devuelve un objeto con los siguientes componentes:

Componente `$grafo`. Muestra que la red resultante contiene 8 aristas en total: las 4 originales más las 4 duplicadas por el proceso de eulerianización. En concreto, los tramos $\{1,3\}$, $\{2,3\}$, $\{3,4\}$ y $\{3,5\}$ aparecen dos veces, marcados con el atributo `color = "red"` para su identificación visual.

Componente `$ruta`. Devuelve la secuencia de vértices visitados:

$$1 \to 3 \to 2 \to 3 \to 4 \to 3 \to 5 \to 3 \to 1.$$

Componente `$ruta_texto`. Expresa la misma secuencia en formato legible:

`"1 -> 3 -> 2 -> 3 -> 4 -> 3 -> 5 -> 3 -> 1"`

Componente `$vertices_impares`. Confirma que los vértices de grado impar detectados en la red original son:

$$V_{\text{imp}} = \{1, 2, 4, 5\}.$$

Componente `$matching`. Muestra las parejas seleccionadas por el algoritmo de emparejamiento perfecto de peso mínimo:

Pareja	Nodos	Coste (m)
1	$(1, 2)$	11
2	$(4, 5)$	11

Cuadro 4.3: Emparejamiento perfecto de peso mínimo obtenido para el Caso 2.

Como se observa, el algoritmo seleccionó las parejas $(1, 2)$ y $(4, 5)$, cada una con un coste de 11 unidades. Esto coincide con el empate técnico identificado en el análisis manual, donde las tres combinaciones posibles producían el mismo coste total de 22 unidades.

Componente `$coste_original`. El coste de recorrer todas las aristas de la red original es:

$$5 + 6 + 8 + 3 = 22 \text{ unidades.}$$

Componente `$coste_reparacion`. El coste adicional introducido por la duplicación de aristas es:

$$11 + 11 = 22 \text{ unidades.}$$

Componente `$coste_total`. El coste total del circuito euleriano óptimo es:

$$22 + 22 = 44 \text{ unidades.}$$

Este resultado coincide exactamente con el obtenido en la resolución manual del Caso 2, lo que valida el correcto funcionamiento del paquete `CoopPostman`.

El vértice 3 aparece cuatro veces en la secuencia de la ruta, lo cual es coherente con su grado $d(3) = 8$ en el multigrafo eulerianizado. Este nodo actúa como nexo central de la red y debe ser visitado tantas veces como aristas incidan en él.

4.8 Caso de Estudio Real: Ronda de Outeiro, A Coruña.

La Ronda de Outeiro constituye una de las arterias principales de A Coruña, con un tránsito diario de miles de vehículos y una alta densidad de actividad urbana. Su proximidad al colegio San Francisco Javier la convierte en una zona de especial sensibilidad desde el punto de vista de la salubridad y la calidad del espacio público.

Desde 2020, la ciudad de A Coruña atravesó una situación de precariedad en el servicio de recogida de residuos urbanos, derivada de la ausencia de contrato estable por resolución judicial. Esta circunstancia generó acumulaciones de residuos en vías públicas de especial relevancia, afectando de forma visible al entorno del colegio San Francisco Javier y comprometiendo las condiciones de higiene y movilidad peatonal de la zona.

Este escenario ilustra de forma muy directa la importancia de una planificación eficiente del servicio de recogida. Una ruta mal diseñada no solo supone un mayor coste operativo, sino que puede dejar tramos sin cobertura o generar repeticiones innecesarias que incrementan el tiempo de servicio. En zonas próximas a centros educativos, estas ineficiencias tienen un impacto directo en la calidad del entorno escolar.

El Problema del Cartero Chino proporciona un marco matemático preciso para abordar este tipo de situaciones. Dado un grafo $G = (V, E)$ que representa la red viaria de la zona, el objetivo es determinar una ruta cerrada de coste mínimo que garantice la cobertura completa de todas las calles, partiendo y regresando al punto de salida del vehículo de recogida.

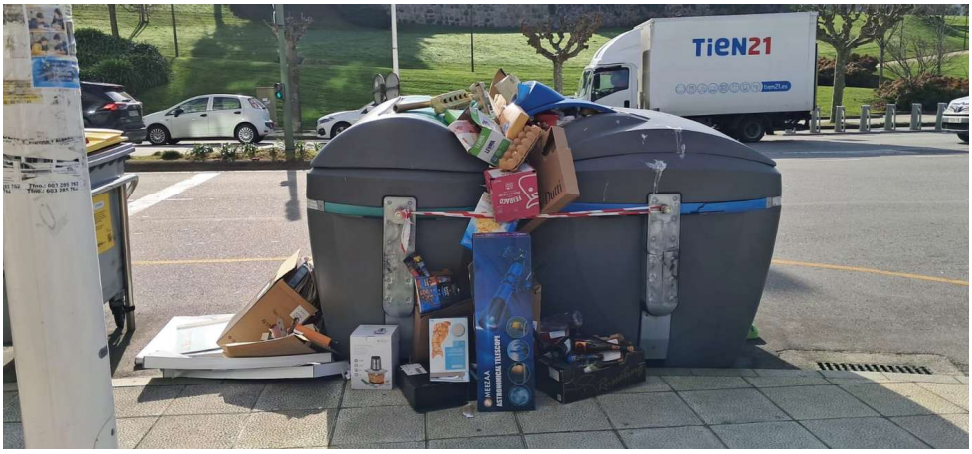


Figura 4.2: Acumulación de residuos en la Ronda de Outeiro, A Coruña, en las inmediaciones del colegio San Francisco Javier. Fuente: Noticias Coruña (2024).

Figura 4.3: Vehículo de recogida de residuos urbanos en una calle de A Coruña. Fuente: 123RF Stock Photo (2024).

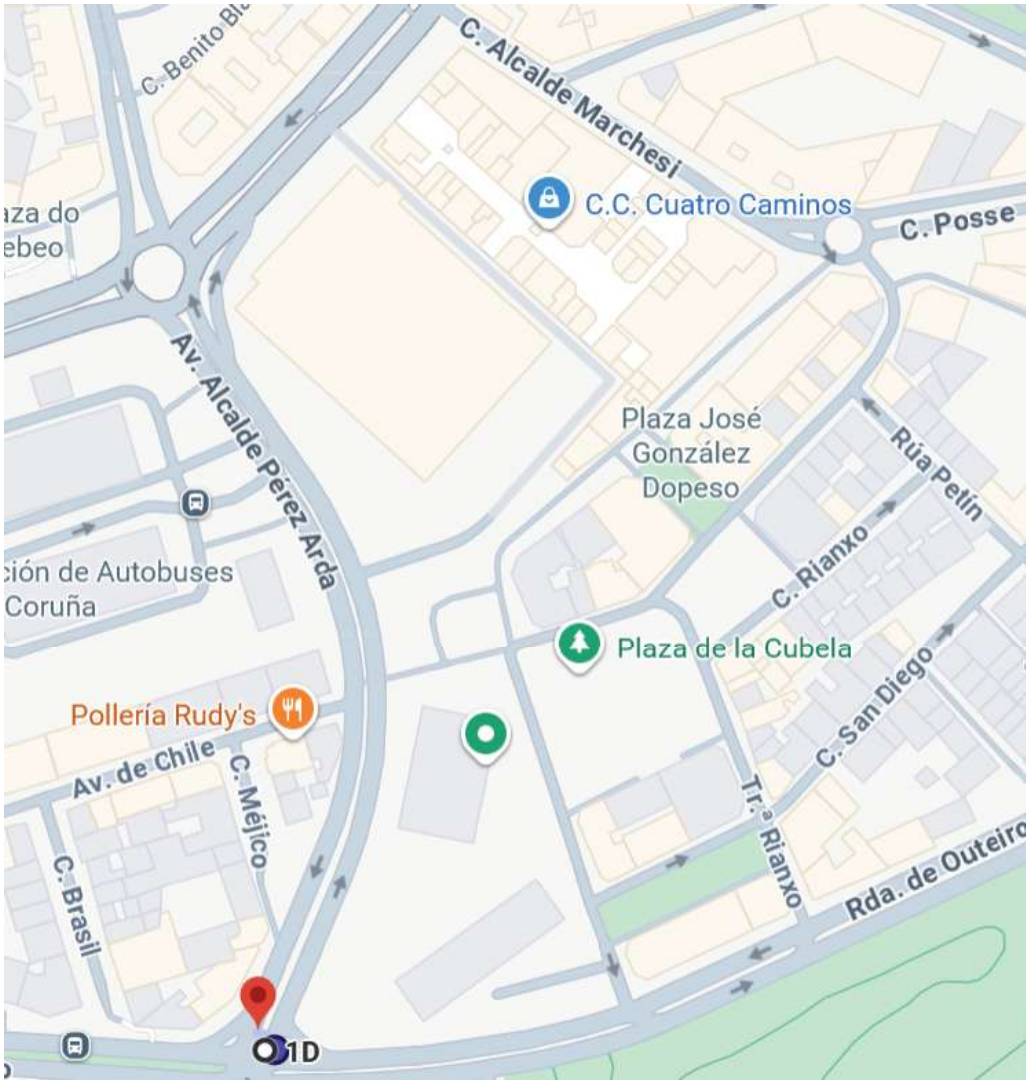


Figura 4.4: Zona de estudio correspondiente a la Ronda de Outeiro y calles adyacentes, A Coruña. Fuente: elaboración propia a partir de Google Maps (2024).

4.9 Definición de la red

La red de recogida de la zona de estudio se modela formalmente como un grafo no dirigido $G = (V, E)$ compuesto por 8 vértices y 9 aristas. En esta representación, cada vértice representa una intersección real de la zona de la Ronda de Outeiro y cada arista corresponde a un tramo de vía pública que debe ser recorrido íntegramente por el vehículo de recogida para prestar el servicio.

Los pesos asignados a las aristas representan la distancia aproximada en metros entre las intersecciones correspondientes, habiendo sido estimadas a partir de cartografía real mediante herramientas de geolocalización. Esta modelización permite transformar la red urbana en un sistema manejable para los algoritmos de optimización descritos en los capítulos previos.

4.9.1 Tabla de vértices

Código	Intersección
V1	Rda. de Outeiro / Pérez Ardá (Depósito)
V2	Rda. de Outeiro / Roberto Tojeiro Díaz
V3	Calle San Diego / Roberto Tojeiro Díaz
V4	Calle San Diego / Travesía de Rianxo
V5	Calle Rianxo
V6	Calle Petín
V7	Calle Caballeros
V8	Avda. Alcalde Pérez Ardá (Estación)

Cuadro 4.4: Vértices del grafo correspondientes a las intersecciones reales de la zona de estudio. Fuente: elaboración propia a partir de Google Maps (2024).

4.9.2 Tabla de aristas

Arista	Desde	Hasta	Distancia (m)
E1	V1	V2	180
E2	V2	V3	70
E3	V3	V4	120
E4	V4	V5	50
E5	V5	V6	100
E6	V5	V8	280
E7	V6	V7	220
E8	V7	V8	300
E9	V8	V1	150

Cuadro 4.5: Aristas del grafo con sus distancias aproximadas en metros. Fuente: elaboración propia a partir de Google Maps (2024).

4.9.3 Análisis de paridad

Vértice	Intersección	Grado	Paridad
V1	Rda. Outeiro / Pérez Ardá	2	Par
V2	Rda. Outeiro / R. Tojeiro Díaz	2	Par
V3	San Diego / R. Tojeiro Díaz	2	Par
V4	San Diego / Travesía de Rianxo	2	Par
V5	Calle Rianxo	3	Impar
V6	Calle Petín	2	Par
V7	Calle Caballeros	2	Par
V8	Avda. Pérez Ardá (Estación)	3	Impar

Cuadro 4.6: Análisis de paridad de los vértices de la red.

El análisis de paridad revela que la red presenta exactamente dos vértices de grado impar:

$$V_{\text{imp}} = \{V5, V8\}.$$

Por el Lema del Apretón de Manos, el número de vértices impares es siempre par, lo que en este caso se cumple con $|V_{\text{imp}}| = 2$. Al existir exactamente dos vértices de grado impar, la red no es euleriana y requiere un proceso de eulerianización: será necesario duplicar el camino mínimo entre V5 y V8 para transformar la red en un grafo euleriano sobre el que trazar el circuito óptimo de recogida.

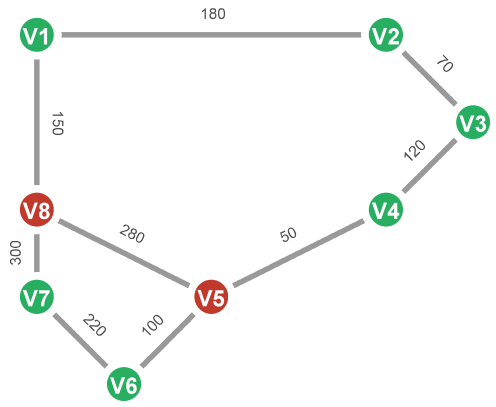


Figura 4.5: Red de recogida de residuos modelada sobre la zona de la Ronda de Outeiro. Los vértices V5 y V8, en rojo, tienen grado impar. Los pesos de las aristas representan la distancia aproximada en metros entre intersecciones.

Por tanto, la red no admite un circuito euleriano en su estado actual. Para cumplir la condición operativa de que el vehículo regrese al depósito (V1), será necesario duplicar el camino de coste mínimo entre V5 y V8, eulerianizando la red mediante el procedimiento descrito en el Capítulo 4. El coste de esta duplicación constituye el sobrecoste mínimo inevitable del recorrido, y su cálculo es precisamente el objeto de la siguiente sección.

```
solucion <- solve_cpp(g_real)

solucion$coste_original
```

```
## [1] 1470
```

```
solucion$coste_reparacion
```

```
## [1] 280
```

```
solucion$coste_total
```

```
## [1] 1750
```

```
solucion$ruta_texto
```

```
## [1] "V1 -> V2 -> V3 -> V4 -> V5 -> V6 -> V7 -> V8 -> V5 -> V8 -> V1"
```

```
solucion$vertices_impares
```

```
## [1] "V5" "V8"
```

4.9.4 Resultados computacionales

La ejecución de la función `solve_cpp()` sobre la red real de la Ronda de Outeiro devuelve un resultado consistente con la teoría del Problema del Cartero Chino.

En primer lugar, el algoritmo identifica correctamente los vértices impares de la red:

$$V_{\text{imp}} = \{V5, V8\}.$$

Dado que la red presenta exactamente dos vértices de grado impar, no existe un circuito euleriano sobre el grafo original. Para garantizar el regreso al depósito situado en el vértice $V1$, el algoritmo aplica el proceso de eulerianización y duplica el camino mínimo entre los vértices impares $V5$ y $V8$, obtenido mediante el algoritmo de Dijkstra.

El coste original de recorrer todas las aristas de la red es:

$$\sum_{e \in E} c(e) = 1470 \text{ m},$$

mientras que el coste adicional introducido por la eulerianización es:

$$c(V5, V8) = 280 \text{ m}.$$

Por tanto, el coste total del recorrido óptimo asciende a:

$$\text{Coste}_{CPP} = 1470 + 280 = 1750 \text{ m}.$$

La ruta final obtenida por el algoritmo es:

$$V1 \to V2 \to V3 \to V4 \to V5 \to V6 \to V7 \to V8 \to V5 \to V8 \to V1.$$

El recorrido resultante cubre todas las aristas de la red y regresa al punto de partida, constituyendo la solución óptima del Problema del Cartero Chino para el caso de estudio considerado. El tramo $V5 \to V8$ aparece recorrido dos veces en la secuencia, siendo este el único sobrecoste inevitable derivado de la estructura topológica de la red.

La figura siguiente representa gráficamente la ruta óptima obtenida por el paquete sobre la red real estudiada. El recorrido parte del depósito situado en $V1$, cubre la totalidad de las calles del área considerada y regresa al punto de origen. La numeración sobre las aristas indica el orden de visita de cada tramo.

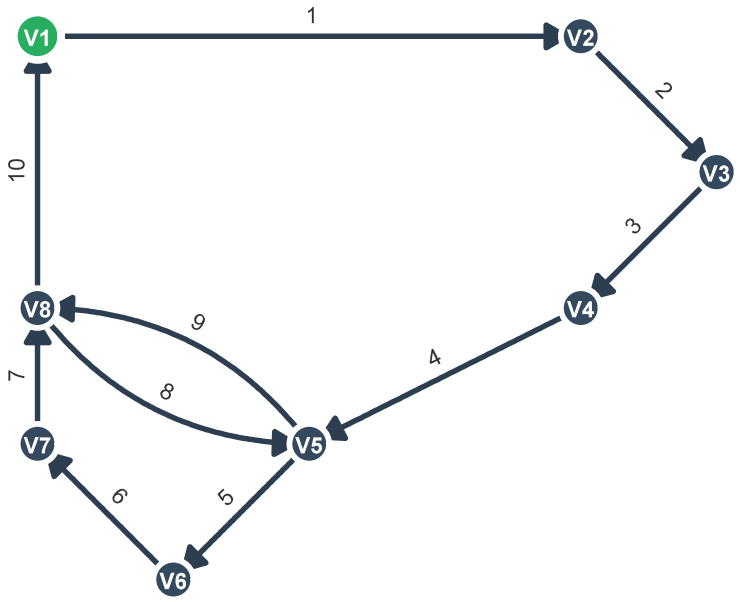


Figura 4.6: Ruta óptima obtenida para el caso real de la Ronda de Outeiro. Las aristas grises representan la red original y las flechas azules indican el recorrido óptimo del vehículo, numerado en orden de visita. El nodo V1, resaltado en verde, representa el depósito de salida y retorno.

Se observa que el recorrido repite el tramo entre $V5$ y $V8$, lo cual es coherente con el proceso de eulerianización previamente descrito. Esta duplicación introduce un sobrecoste mínimo de 280 metros, pero permite convertir la red en euleriana y garantizar el retorno al depósito.

Desde el punto de vista de la gestión urbana, este resultado tiene una lectura directa: el vehículo de recogida debe recorrer un total de **1.750 metros** para garantizar la cobertura completa de la zona, partiendo y regresando al depósito situado en la intersección V1 (Ronda de Outeiro / Pérez Ardá).

Concepto	**Valor**
Distancia total de la red	1.470 metros
Vértices de grado impar	V5, V8
Tramo duplicado	E6: V5–V8 (280 metros)
Sobrecoste de eulerización	280 metros
Distancia total de la ruta	1.750 metros
Porcentaje de sobrecoste	19,05%

Cuadro 4.7: Resumen de resultados del caso real. Ronda de Outeiro, A Coruña.

El sobrecoste de eulerización, correspondiente a la duplicación del tramo E6 entre V5 y V8, supone un **19,05%** sobre la distancia base de la red. Este sobrecoste es el mínimo posible dado que:

- E6 es el camino más corto entre los únicos dos vértices de grado impar de la red (V5 y V8),
- con una distancia de 280 metros frente a la alternativa de 620 metros por V6 y V7.

En esta instancia real, la red presenta únicamente **dos vértices de grado impar** (V5 y V8), por lo que la fase de emparejamiento perfecto se reduce a una única pareja posible. Esta simplicidad no resta validez al caso de estudio, sino que permite interpretar de manera transparente el proceso de eulerización sobre una topografía urbana real.

Los aspectos combinatorios más complejos del emparejamiento, incluyendo el papel del algoritmo de Blossom de Edmonds y la transformación afín de pesos, se ilustran en los casos didácticos previos, donde la presencia de múltiples vértices impares hace no trivial la elección del emparejamiento óptimo.

Capítulo 5

Juegos cooperativos

5.1 Origen histórico de la Teoría de Juegos

La Teoría de Juegos es una rama de las matemáticas que estudia la toma de decisiones estratégicas entre agentes racionales. Su formalización moderna surge en 1944 con la publicación de *Theory of Games and Economic Behavior* por von Neumann y Morgenstern (1944).

Sin embargo, fue durante la Guerra Fría (1947–1991) cuando esta disciplina alcanzó su máxima relevancia práctica. En el contexto del enfrentamiento entre Estados Unidos y la Unión Soviética, matemáticos como John Nash desarrollaron modelos para analizar escenarios de conflicto estratégico. El objetivo era encontrar estrategias donde ninguna de las partes quedara en una situación catastrófica, lo que se conoce como **equilibrio de Nash**.

Si bien los primeros modelos se centraban en el conflicto, entendido como juegos de suma cero donde lo que gana un agente lo pierde el otro, la Teoría de Juegos evolucionó hacia el estudio de la cooperación. Se descubrió que, en muchos escenarios reales, los agentes pueden obtener mejores resultados trabajando juntos que compitiendo entre sí (Owen, 1995).

La Teoría de Juegos se divide en dos grandes ramas según la posibilidad de establecer acuerdos vinculantes entre los agentes (Peleg y Sudhölter, 2007):

Tipo de juego	**Características**	**Ejemplo**
No cooperativo	Cada jugador busca maximizar su propio beneficio sin acuerdos vinculantes.	Dos empresas compitiendo por el mismo mercado.
Cooperativo	Los jugadores pueden formar coaliciones y firmar acuerdos para repartir beneficios conjuntos.	Varios municipios compartiendo un servicio de recogida de residuos.

Cuadro 5.1: Comparación entre juegos cooperativos y no cooperativos. Fuente: adaptado de Peleg y Sudhölter (2007).

Definición 5.1.1 (Juego cooperativo)**.** Un juego cooperativo es un par (N, v) donde:

$N = \{1, 2, \dots, n\}$ es el conjunto finito de **jugadores**.

$v : 2^N \to \mathbb{R}$ es la **función característica**, con $v(\emptyset) = 0$.

$v(S)$ asigna un valor a cada coalición $S \subseteq N$, representando lo que ese grupo puede garantizarse por sí solo.

Cuando la función característica representa costes en lugar de ganancias, el par (N, c) se denomina **juego de costes**.

Dado que en el problema del cartero chino (CPP) el objetivo es minimizar distancias, el modelo cooperativo asociado se formulará a partir de ahora precisamente como un juego de costes.

Definición 5.1.2 (Núcleo)**.** El núcleo de un juego de costes (N, c) es el conjunto de vectores de reparto $x = (x_1, \dots, x_n)$ que satisfacen las dos condiciones siguientes:

Eficiencia: el coste total de la gran coalición se reparte exactamente entre todos los jugadores:

$$\sum_{i \in N} x_i = c(N).$$

Estabilidad coalicional: la suma de los repartos no supera el coste asociado a la coalición:

$$\sum_{i \in S} x_i \leq c(S) \quad \forall\, S \subseteq N.$$

Propiedad	**Significado**
Eficiencia	Se reparte exactamente el coste total de la gran coalición.
Estabilidad	Ningún subgrupo obtendría un coste menor separándose.

Cuadro 5.2: Condiciones que debe satisfacer un reparto perteneciente al núcleo. Fuente: adaptado de Gillies (1959).

Un reparto pertenece al núcleo si y solo si es eficiente, es decir, reparte exactamente el coste total $c(N)$, y coalicionalmente estable, es decir, ningún subgrupo tiene incentivos para separarse de la gran coalición.

5.1.1 Propiedades de los juegos cooperativos

Definición 5.1.3 (Juego aditivo)**.** Un juego (N, v) es **aditivo** si:

$$v(S \cup T) = v(S) + v(T)$$

para todas las coaliciones disjuntas $S, T \subseteq N$, es decir, $S \cap T = \emptyset$.

En este caso, cooperar no genera ni ahorros ni pérdidas. El valor de la unión es exactamente igual a la suma de los valores individuales.

Definición 5.1.4 (Juego subaditivo)**.** Un juego de costes (N, c) es **subaditivo** (Peleg y Sudhölter, 2007) si:

$$c(S \cup T) \leq c(S) + c(T)$$

para todas las coaliciones disjuntas $S, T \subseteq N$.

La subaditividad implica que el *coste* de la unión de dos grupos es siempre menor o igual a la suma de sus costes si operasen por separado. En otras palabras, la cooperación genera ahorros o, en el peor de los casos, mantiene los costes iguales, pero nunca los penaliza.

Definición 5.1.5 (Juego monótono)**.** Un juego (N, v) es **monótono** si:

$$S \subseteq T \implies v(S) \leq v(T).$$

En la sección correspondiente al Juego del Cartero Chino se comprobará que existe una monotonía clara en el coste total del recorrido: a medida que aumenta el número de aristas requeridas, el coste total del tour mínimo nunca disminuye. Sin embargo, el interés de la cooperación en este tipo de juegos no reside en el coste total, sino en cómo se distribuyen los costes no separables.

Definición 5.1.6 (Juego cóncavo en costes)**.** Un juego de costes (N, c) es **cóncavo** si:

$$c(S \cup \{i\}) - c(S) \leq c(T \cup \{i\}) - c(T)$$

para todo $S \subseteq T \subseteq N \setminus \{i\}$.

Un juego de costes es cóncavo cuando presenta contribuciones marginales decrecientes: el coste adicional de incorporar a un nuevo jugador es menor cuanto más grande sea la coalición a la que se une.

Definición 5.1.7 (Juego equilibrado)**.** Un juego (N, v) es **equilibrado** si su núcleo es no vacío:

$$\text{Core}(N, v) \neq \emptyset.$$

Esto garantiza la existencia de al menos un reparto estable, es decir, una asignación que ninguna coalición tiene incentivos para rechazar.

Definición 5.1.8 (Juego totalmente equilibrado)**.** Un juego (N, v) es **totalmente equilibrado** si todo subjuego tiene núcleo no vacío:

$$\text{Core}(S, v|_S) \neq \emptyset \quad \forall\, S \subseteq N.$$

5.1.2 Método de Reparto de Shapley

Definición 5.1.9 (Valor de Shapley)**.** El **valor de Shapley** asigna a cada jugador i una contribución basada en el promedio de sus aportaciones marginales considerando todos los órdenes posibles de incorporación a la coalición:

$$\varphi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{n!} \cdot [v(S \cup \{i\}) - v(S)].$$

El valor de Shapley satisface las siguientes propiedades:

Propiedad	**Significado**
Eficiencia	$\sum_{i \in N} \varphi_i = v(N)$
Simetría	Jugadores con igual contribución reciben igual pago
Jugador nulo	Si un jugador no aporta valor, recibe cero
Aditividad	$\varphi(v + w) = \varphi(v) + \varphi(w)$

Cuadro 5.3: Propiedades axiomáticas del valor de Shapley. Fuente: adaptado de Shapley (1953).

5.2 El Juego del Cartero Chino

Definición 5.2.1 (Juego del Cartero Chino)**.** Sea $G = (V, E, w)$ un grafo conexo ponderado. El **Juego del Cartero Chino** se define como el par (N, c), donde N es el conjunto de jugadores,

identificado con el conjunto de aristas E, de modo que $N = \{1, 2, \dots, |E|\}$, y $c : 2^N \to \mathbb{R}$ es la función característica que asigna a cada coalición $S \subseteq N$ su coste no separable mínimo:

$$c(\emptyset) = 0,$$

$$c(S) = \text{coste del } S\text{-tour mínimo} - \sum_{e \in S} w(e) \quad \text{para } S \neq \emptyset.$$

El término $\sum_{e \in S} w(e)$ representa los costes **separables** de la coalición, es decir, el coste directo de recorrer cada arista perteneciente a S. Al restarlos, $c(S)$ captura exclusivamente los **costes no separables**: el sobrecoste generado por la necesidad de transitar aristas adicionales, no pertenecientes a S, para poder completar un recorrido cerrado que cubra todas las aristas de la coalición y regrese al depósito v_0.

Ejemplo 5.2.1. La Figura 5.1 ilustra la red sobre la que se define el Juego del Cartero Chino. Cada arista representa un jugador del juego cooperativo, de modo que el conjunto de jugadores es $N = \{e_1, e_2, e_3, e_4, e_5, e_6\}$. El vértice central v_3, resaltado en naranja, actúa como depósito u oficina de correos desde el que parte y al que regresa el cartero.

La red se divide en dos alas simétricas. El **ala superior** está formada por las aristas e_1, e_2 y e_3: la arista e_1 conecta v_1 y v_2 con peso $w_1 = 10$; la arista e_2 conecta v_1 y v_3 con peso $w_2 = 5$; y la arista e_3 conecta v_2 y v_3 con peso $w_3 = 6$. El **ala inferior** está formada por las aristas e_4, e_5 y e_6: la arista e_4 conecta v_3 y v_4 con peso $w_4 = 8$; la arista e_5 conecta v_4 y v_5 con peso $w_5 = 4$; y la arista e_6 conecta v_3 y v_5 con peso $w_6 = 3$.

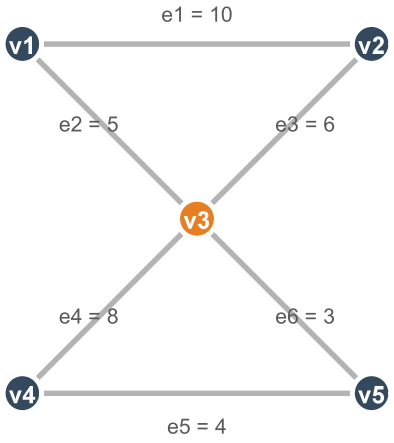


Figura 5.1: Grafo Juego del Cartero Chino.

Definición 5.2.2 (S-tour)**.** Sea $G = (V, E, w)$ un grafo conexo no dirigido y sea $v_0 \in V$ un vértice fijo. Para cualquier coalición $S \subseteq N$, un ***S*-tour** con respecto a v_0 es un recorrido cerrado:

$$v_0, e_1, v_1, \dots, v_{k-1}, e_k, v_0$$

en G, tal que $S \subseteq \{e_j \mid j \in \{1, \dots, k\}\}$, es decir, todas las aristas de la coalición S son visitadas al menos una vez durante el recorrido.

El agente comienza y finaliza en el vértice v_0 y visita cada arista perteneciente a la coalición S al menos una vez. Las aristas no pertenecientes a S pueden recorrerse si son necesarias para conectar el recorrido, pero su coste se contabiliza como sobrecoste no separable.

Ejemplo 5.2.2. Consideremos la coalición $S = \{e_4, e_5\}$, formada por las aristas que conectan v_3–v_4 y v_4–v_5 respectivamente. El S-tour de coste mínimo con respecto al depósito $v_0 = v_3$ debe partir de v_3, visitar ambas aristas y regresar a v_3.

La figura siguiente muestra dicho recorrido. Las aristas en rojo forman el recorrido cerrado óptimo desde el depósito v_3: el cartero recorre e_4, e_5 y e_6 para cubrir la coalición y regresar al origen.

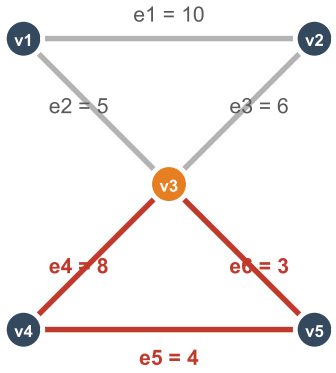


Figura 5.2: S-tour de coste mínimo para la coalición $S = \{e_4, e_5\}$. Fuente: adaptado de Hamers et al. (1999).

En este escenario los jugadores e_4 y e_5 forman una coalición. La misión del cartero es salir del depósito $v_0 = v_3$, recorrer ambas aristas de la coalición al menos una vez y regresar a v_3.

Las aristas de la coalición son $e_4 = (v_3, v_4)$ con peso $w_4 = 8$ y $e_5 = (v_4, v_5)$ con peso $w_5 = 4$. Se consideran tres rutas posibles:

Ruta 1. Cerrar el circuito usando e_6:

$$v_3 \xrightarrow{e_4} v_4 \xrightarrow{e_5} v_5 \xrightarrow{e_6} v_3 \qquad \text{Coste} = 8 + 4 + 3 = 15.$$

Ruta 2. Duplicar e_4 y e_5:

$$v_3 \xrightarrow{e_4} v_4 \xrightarrow{e_5} v_5 \xrightarrow{e_5} v_4 \xrightarrow{e_4} v_3 \qquad \text{Coste} = 8 + 4 + 4 + 8 = 24.$$

Ruta 3. Empezar por e_6 y cerrar con e_4:

$$v_3 \xrightarrow{e_6} v_5 \xrightarrow{e_5} v_4 \xrightarrow{e_4} v_3 \qquad \text{Coste} = 3 + 4 + 8 = 15.$$

Las tres rutas son S-tours válidos para la coalición $S = \{e_4, e_5\}$. El **S-tour mínimo** tiene coste 15 y corresponde a las Rutas 1 y 3. El coste no separable de la coalición es por tanto:

$$c(\{e_4, e_5\}) = 15 - (w_4 + w_5) = 15 - (8 + 4) = 3.$$

El valor $c(\{e_4, e_5\}) = 3$ representa exactamente el coste de la arista e_6, que el cartero debe recorrer para poder cerrar el circuito sin repetir aristas. Este sobrecoste es el que la Teoría de Juegos Cooperativos distribuirá de forma justa entre los jugadores de la coalición.

5.2.1 Estructura y propiedades de los grafos

En esta sección se realiza un análisis topológico de los grafos considerados con el objetivo de determinar la aplicabilidad de la regla γ. Para ello, se examinan las propiedades estructurales clave (existencia de puentes, eulerianidad y el recorrido ciclico de las componentes) que condicionan tanto la validez del reparto como la estabilidad del núcleo del juego cooperativo asociado.

Un grafo conexo $G = (V, E, w)$ se clasifica como **euleriano** si todos sus vértices tienen grado par. En este caso, existe un circuito cerrado que visita cada arista exactamente una vez y regresa al depósito v_0. En consecuencia, en el Juego del Cartero Chino, los costes no separables de la gran coalición para un grafo euleriano son nulos:

$$c(N) = 0.$$

Como se estableció en la Sección 2.2, un grafo débilmente euleriano es aquel cuyas componentes conexas, tras eliminar todos sus puentes, son grafos eulerianos o vértices aislados. Esta propiedad topológica es relevante en el contexto cooperativo porque determina la aplicabilidad de la regla γ y garantiza la existencia de un núcleo no vacío.

Ejemplo 5.2.3. Consideremos el grafo propuesto por Hamers et al. (1999) ampliado con la arista (v_4, v_6). Al eliminar dicha arista, el grafo se descompone en dos componentes: la componente principal $\{v_1, v_2, v_3, v_4, v_5\}$, que constituye un grafo euleriano, y el vértice aislado $\{v_6\}$. Esta estructura satisface, por tanto, la condición de **débilmente euleriano**.

Aplicando el criterio de conectividad sobre todas las aristas de la red, se obtienen los siguientes resultados representativos:

- **Eliminación de** (v_1, v_2)**:** El grafo permanece conexo, pues los nodos v_1 y v_2 siguen comunicados a través de v_3. Por tanto, no es un puente.
- **Eliminación de** (v_3, v_4)**:** El grafo permanece conexo, pues existe un camino alternativo entre v_3 y v_4 a través de los nodos del ala inferior. No es un puente.
- **Eliminación de** (v_4, v_6)**:** El grafo se divide en dos componentes. El nodo v_6 queda aislado y el subgrafo restante $\{v_1, v_2, v_3, v_4, v_5\}$ conserva su conectividad. Esta arista **sí es un puente**.

El análisis confirma que (v_4, v_6) es el único puente del sistema, es decir, $B(G) = \{(v_4, v_6)\}$, en consecuencia, este grafo cumple las condiciones necesarias para aplicar la regla γ con garantía de estabilidad en el núcleo..

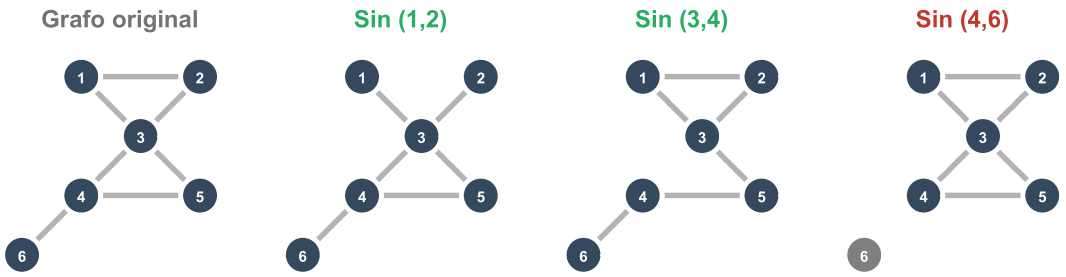


Figura 5.3: Identificación del puente en el grafo débilmente euleriano. De izquierda a derecha: grafo original con 7 aristas y 6 nodos; eliminación de $(1, 2)$, que no desconecta el grafo; eliminación de $(3, 4)$, que tampoco lo desconecta; eliminación de $(4, 6)$, que deja el nodo 6 aislado, confirmando que es el único puente de la red.

Definición 5.2.3 (Grafo débilmente cíclico)**.** Un grafo conexo es **débilmente cíclico** si, al eliminar todos sus puentes, cada componente conexa resultante es un ciclo simple (todos sus vértices tienen grado 2) o un vértice aislado.

De manera intuitiva, una vez eliminados los puentes, el grafo debe descomponerse únicamente en ciclos simples o vértices aislados. Si alguna componente presenta ramificaciones, cruces o ciclos entrelazados, el grafo no es débilmente cíclico. Cabe destacar además que todo grafo débilmente cíclico es débilmente euleriano, pero no al revés.

Ejemplo 5.2.4. El análisis de la estructura débilmente cíclica se desarrolla en cuatro pasos:

1. Grafo original. Se parte de la red inicial compuesta por cuatro vértices y cuatro aristas.

2. Identificación de puentes. Se evalúa cada arista de forma individual. Se comprueba que eliminar cualquiera de las aristas del triángulo (v_1, v_2), (v_2, v_3) o (v_1, v_3) no desconecta la red. Sin embargo, al eliminar la arista (v_0, v_1), el sistema se divide, confirmando que es el único puente: $B(G) = \{(v_0, v_1)\}$.

3. Eliminación del puente. Se suprime la arista (v_0, v_1) del grafo.

4. Análisis de componentes. Tras la eliminación, la red se separa en dos componentes: el vértice v_0, que queda como vértice aislado, y los nodos $\{v_1, v_2, v_3\}$, que forman un ciclo simple.

Dado que los componentes resultantes son un ciclo simple y un vértice aislado, el grafo presenta una estructura débilmente cíclica, que constituye un caso especial de grafo débilmente euleriano.

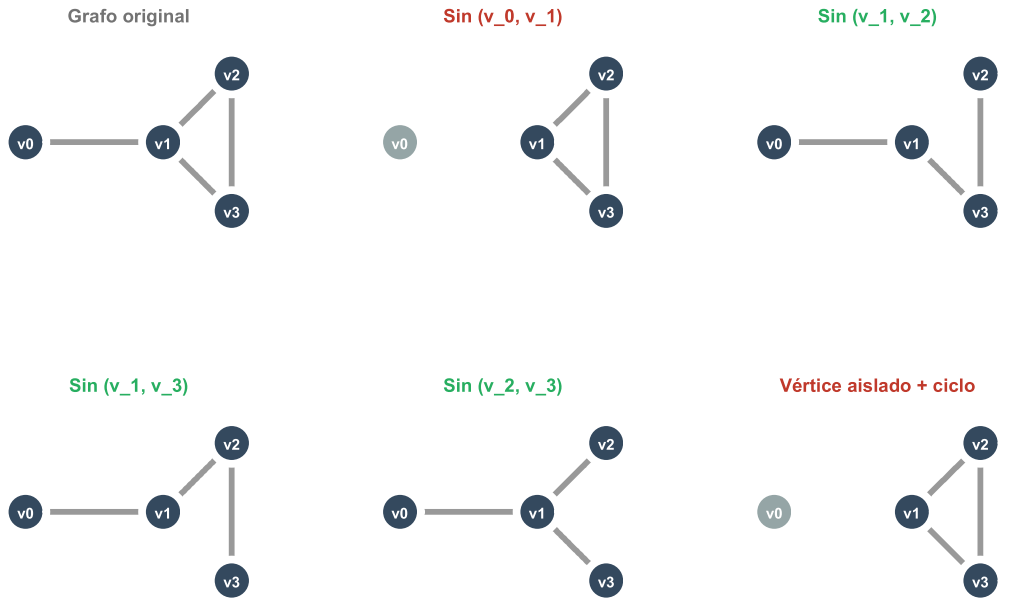


Figura 5.4: Test topológico sobre un grafo débilmente cíclico.

5.3 Reparto del coste y estabilidad cooperativa

El Juego del Cartero Chino se enmarca dentro de la familia de problemas de ruteo de arcos. Aunque el modelo clásico asume que todas las calles deben ser servidas, en aplicaciones reales es más frecuente enfrentarse al **Problema del Cartero Rural** (RPP, *Rural Postman Problem*).

En el RPP, la demanda de servicio no se encuentra en la totalidad de la red, sino en un subconjunto específico de aristas denominado conjunto de aristas requeridas, $E_R \subseteq E$. El objetivo consiste en encontrar el recorrido cerrado de coste mínimo que transite al menos una vez por cada arista perteneciente a E_R (Lenstra y Rinnooy Kan, 1976). El cartero puede además utilizar aristas no pertenecientes a E_R si estas permiten optimizar el recorrido.

Desde la perspectiva de la Teoría de Juegos, el desafío surge al intentar repartir el coste total del servicio entre los agentes asociados a las aristas requeridas. Para resolver este reparto en grafos con estructuras estables, es fundamental identificar la jerarquía de los tramos respecto al depósito central v_0.

En el enfoque tradicional, calcular la función característica $c(S)$ sobre un grafo euleriano es un procedimiento directo. Sin embargo, cuando el grafo no cumple esta propiedad, el problema se vuelve considerablemente más complejo: para determinar el coste de cada coalición S sería necesario resolver un Problema del Cartero Rural específico para ese subconjunto. Este procedimiento es computacionalmente costoso, ya que el RPP es un problema **NP-difícil**. Para evitar esta dificultad, proponen centrar el análisis en la clase de grafos débilmente eulerianos, donde la estructura de los puentes permite simplificar de manera significativa el cálculo de los costes.

En esta clase de grafos, el coste no separable de cualquier coalición S está determinado exclusivamente por los puentes que deben cruzarse para prestar servicio a sus miembros. La principal contribución de Hamers et al. (1999) consiste en demostrar que el mecanismo de reparto resultante no solo se calcula en tiempo polinomial, sino que además la asignación obtenida siempre pertenece al **núcleo** del juego.

5.4 Resolución del reparto: regla γ

Cuando una coalición de jugadores requiere el servicio, el cartero debe trazar una ruta desde el depósito hasta sus ubicaciones. Con frecuencia, esto obliga a transitar por aristas que no pertenecen a la coalición, utilizándolas como vías de enlace. Estos recorridos generan un sobrecoste derivado del uso obligado de puentes.

La asignación o pago asignado a cada jugador i mediante la **regla** γ se obtiene sumando las contribuciones proporcionales de todos los puentes de los cuales su arista e_i es seguidora:

$$\gamma_i = \sum_{\substack{b \in B(G) \\ e_i \in F_b}} \frac{w(b)}{|F_b|},$$

donde $B(G)$ es el conjunto de puentes del grafo, $w(b)$ es el coste del puente b, y $|F_b|$ es el número total de aristas seguidoras que dependen estructuralmente de dicho puente.

Esta regla se fundamenta en dos conceptos clave:

- **Seguidor (*follower*):** una arista e_i es seguidora de un puente b si todo camino desde el depósito v_0 hasta e_i pasa necesariamente por b.
- **Reparto igualitario:** el coste de cada puente se divide equitativamente entre todas sus aristas seguidoras.

A diferencia del valor de Shapley, la regla γ no requiere evaluar el crecimiento exponencial de las 2^n coaliciones posibles, lo que la hace computacionalmente eficiente.

5.5 Propiedades y estabilidad de la regla γ

En los juegos del Cartero Chino definidos sobre grafos débilmente eulerianos, la regla γ es el único mecanismo de reparto que satisface simultáneamente las propiedades de eficiencia, simetría de clusters de puentes e independencia de condensación.

Teorema 5.5.1 (Caracterización axiomática)**.** Sea (N, c) un juego del Cartero Chino sobre un grafo débilmente euleriano. La regla γ es el **único** valor de reparto que satisface simultáneamente las siguientes tres propiedades:

- **Eficiencia:** El reparto agota exactamente el coste no separable de la gran coalición, es decir, $\sum_{e \in N} \gamma_e = c(N)$.
- **Simetría de clusters de puentes (*bridge-cluster symmetry*):** Los jugadores con la misma dependencia estructural respecto de los puentes asumen la misma fracción del coste no separable.
- **Independencia de condensación (*condensation independence*):** El reparto no varía al condensar las componentes del grafo que no contienen puentes, es decir, al sustituir cada componente cíclica por un único supervértice.

Teorema 5.5.2 (Estabilidad en el núcleo)**.** Sea Γ un juego del Cartero Chino definido sobre un grafo conexo débilmente euleriano. Entonces la asignación producida por la regla γ siempre pertenece al núcleo (Hamers et al., 1999):

$$\gamma(\Gamma) \in \mathrm{Core}(c).$$

Estos resultados garantizan que γ no es una regla arbitraria, sino que es **justa** (por sus axiomas) y **estable** (al pertenecer al núcleo, ninguna coalición tiene incentivos para abandonar la gran coalición).

5.6 Análisis Práctico de Estabilidad y Reparto

En esta sección se valida empíricamente la metodología de Hamers et al. (1999) mediante una serie de casos prácticos. Para cada escenario, se calculan los costes no separables, se aplica la regla γ y se verifica la estabilidad de la solución.

Ejemplo 5.6.1 (Grafo euleriano sin puentes)**.** A diferencia del ejemplo anterior, se analiza ahora un grafo sin puentes. Esta topología corresponde a un circuito cerrado donde todas las aristas son requeridas ($E_R = E$) y ninguna de ellas desconecta el grafo al ser eliminada. Consideramos un depósito v_0 y tres aristas de coste unitario $w(e_i) = 1$ formando un ciclo:

$$e_1 = (v_0, v_1), \qquad e_2 = (v_1, v_2), \qquad e_3 = (v_0, v_2).$$

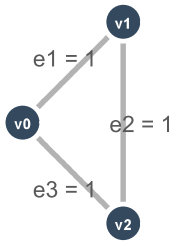


Figura 5.5: Grafo triangular sin puentes.

Verificación topológica.

Paso 1. Grados de los vértices. Se analiza la conectividad de la red: $d(v_0) = 2$ (conectado a e_1 y e_3), $d(v_1) = 2$ (conectado a e_1 y e_2) y $d(v_2) = 2$ (conectado a e_2 y e_3).

Dado que todos los vértices presentan grado par, el grafo es **euleriano**. Al existir un circuito euleriano que visita cada arista exactamente una vez y regresa al depósito, los costes no separables de la gran coalición son nulos: $c(N) = 0$.

Paso 2. Identificación de puentes. Al eliminar cualquiera de las tres aristas el grafo permanece conexo, por lo que el conjunto de puentes es vacío: $B(G) = \emptyset$. Al ser un ciclo simple, constituye un caso de grafo **débilmente cíclico**. En esta clase de grafos el juego de reparto asociado es **cóncavo**, lo que garantiza la existencia de un núcleo no vacío.

Cálculo exhaustivo de la función característica $c(S)$.

Con tres jugadores, el número de coaliciones posibles es $2^3 - 1 = 7$. Calculamos el sobrecoste no separable para cada una:

- **Coaliciones individuales ($|S| = 1$):**

  $$c(\{e_1\}) = (1 + 1) - 1 = 1, \quad c(\{e_2\}) = (1 + 1 + 1) - 1 = 2, \quad c(\{e_3\}) = (1 + 1) - 1 = 1.$$

  (*Nota: para servir a e_2, el cartero debe ir y volver del depósito usando e_1 o e_3*).

- **Coaliciones de dos jugadores ($|S| = 2$):**

  $$c(\{e_1, e_2\}) = (1+1+1)-2 = 1, \quad c(\{e_1, e_3\}) = (1+1+1)-2 = 1, \quad c(\{e_2, e_3\}) = (1+1+1)-2 = 1.$$

- **Gran coalición ($|S| = 3$):**

  $$c(N) = c(\{e_1, e_2, e_3\}) = (1 + 1 + 1) - 3 = 0.$$

Coalición S	$\{e_1\}$	$\{e_2\}$	$\{e_3\}$	$\{e_1, e_2\}$	$\{e_1, e_3\}$	$\{e_2, e_3\}$	N
$c(S)$	1	2	1	1	1	1	0

Cuadro 5.4: Función característica del juego sobre el grafo triangular.

Concavidad y estabilidad.

Un juego de costes es **cóncavo** si las contribuciones marginales de los jugadores decrecen a medida que la coalición aumenta de tamaño (Peleg y Sudhölter, 2007). Verificamos esta propiedad para el jugador e_3:

Coalición S	$c(S)$	$c(S \cup \{e_3\})$	**Contribución marginal**
$\emptyset$	0	1	1
$\{e_1\}$	1	1	0
$\{e_2\}$	2	1	-1
$\{e_1, e_2\}$	1	0	-1

Cuadro 5.5: Análisis de contribuciones marginales del jugador e_3.

Como se observa, las contribuciones marginales son no crecientes ($1 \geq 0 \geq -1 \geq -1$), lo que confirma la concavidad del juego. Esta propiedad asegura que siempre existen incentivos económicos para ampliar la cooperación.

Dado que $c(N) = 0$, el reparto trivial mediante la regla $\gamma = (0, 0, 0)$ es eficiente y pertenece al núcleo:

$$\sum_{i \in S} \gamma_i = 0 \leq c(S) \quad \forall\, S \subseteq N. \quad \checkmark$$

Este resultado es consistente con Hamers et al. (1999): en grafos eulerianos, los costes no separables se anulan y el reparto estable es inmediato.

Ejemplo 5.6.2 (Grafo débilmente euleriano con pesos heterogéneos)**.** Este caso reproduce la estructura topológica del ejemplo fundamental de Hamers et al. (1999): un grafo formado por un puente de acceso e_1 y un ciclo euleriano $\{e_2, e_3, e_4\}$. A diferencia del modelo original de costes unitarios, aquí se asignan pesos heterogéneos para validar que la estabilidad del núcleo se mantiene bajo condiciones generales de coste.

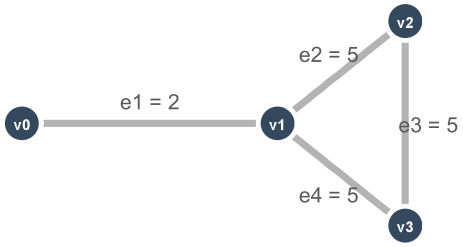


Figura 5.6: Grafo débilmente euleriano con pesos heterogéneos.

Verificación topológica.

Paso 1. Grados de los vértices.

$d(v_0) = 1$ (conectado a e_1), $d(v_1) = 3$ (conectado a e_1, e_2 y e_4), $d(v_2) = 2$ (conectado a e_2 y e_3), $d(v_3) = 2$ (conectado a e_3 y e_4).

Los vértices v_0 y v_1 tienen grado impar, por lo que el grafo **no es euleriano** y se producen costes adicionales en el recorrido.

Paso 2. Identificación de puentes.

Al eliminar e_1 el grafo se desconecta, mientras que las restantes aristas no lo hacen: por tanto, $B(G) = \{e_1\}$. Las componentes resultantes son el vértice aislado v_0 y el ciclo simple $\{v_1, v_2, v_3\}$, lo que confirma que el grafo es débilmente euleriano y débilmente cíclico. En consecuencia, el juego asociado es cóncavo y todas las aristas del ciclo, al depender exclusivamente de e_1, forman un único cluster que asume el coste del puente en partes iguales por la propiedad de simetría.

Cálculo exhaustivo de la función característica $c(S)$.

Con cuatro jugadores, el número de coaliciones posibles es $2^4 - 1 = 15$. Aunque el cálculo exhaustivo aún es abordable, comienza a evidenciarse el crecimiento exponencial del problema. Calculamos el sobrecoste para cada subconjunto:

- **Coaliciones individuales ($|S| = 1$):**

$$c(\{e_1\}) = (10 + 10) - 10 = 10, \quad c(\{e_2\}) = (10 + 5 + 5 + 10) - 5 = 25,$$

$$c(\{e_3\}) = (10 + 5 + 5 + 5 + 10) - 5 = 30, \quad c(\{e_4\}) = (10 + 5 + 5 + 10) - 5 = 25.$$

- **Coaliciones de dos jugadores ($|S| = 2$):**

$$c(\{e_1, e_2\}) = c(\{e_1, e_4\}) = 15, \quad c(\{e_1, e_3\}) = 20,$$

$$c(\{e_2, e_3\}) = c(\{e_2, e_4\}) = c(\{e_3, e_4\}) = 25.$$

- **Coaliciones de tres jugadores ($|S| = 3$):**

$$c(\{e_1, e_2, e_3\}) = c(\{e_1, e_2, e_4\}) = c(\{e_1, e_3, e_4\}) = 15, \quad c(\{e_2, e_3, e_4\}) = 20.$$

- **Gran coalición ($|S| = 4$):**

$$c(N) = (10 + 5 + 5 + 5 + 10) - 25 = 10.$$

Este resultado es consistente con la propiedad teórica demostrada por Hamers et al. (1999): en grafos débilmente eulerianos, el coste de la gran coalición es igual a la suma de los pesos de sus puentes:

$$c(N) = \sum_{b \in B(G)} w(b) = w(e_1) = 10.$$

Coalición S	**Recorrido total**	**Coste separable**	$c(S)$
$\{e_1\}$	20	10	10
$\{e_2\}$	30	5	25
$\{e_3\}$	35	5	30
$\{e_4\}$	30	5	25
$\{e_1, e_2\}$	30	15	15
$\{e_1, e_3\}$	35	15	20
$\{e_1, e_4\}$	30	15	15
$\{e_2, e_3\}$	35	10	25
$\{e_2, e_4\}$	35	10	25
$\{e_3, e_4\}$	35	10	25
$\{e_1, e_2, e_3\}$	35	20	15
$\{e_1, e_2, e_4\}$	35	20	15
$\{e_1, e_3, e_4\}$	35	20	15
$\{e_2, e_3, e_4\}$	35	15	20
N	35	25	10

Cuadro 5.6: Función característica del juego sobre el grafo débilmente euleriano con pesos heterogéneos.

Los valores de la función característica calculados anteriormente son necesarios para verificar la estabilidad del reparto, pero **no para calcularlo**. Esta es la principal ventaja competitiva de la metodología de Hamers et al. (1999): el reparto se obtiene directamente desde la topología del grafo en tiempo polinomial, evitando la complejidad exponencial de evaluar todas las coaliciones.

Aplicación de la regla γ.

Identificamos el conjunto de seguidores para el único puente del grafo respecto a v_0: $F_{e_1} = \{e_1, e_2, e_3, e_4\}$. Al depender los cuatro jugadores de dicho puente ($|F_{e_1}| = 4$), el coste se distribuye equitativamente:

$$\gamma_1 = \gamma_2 = \gamma_3 = \gamma_4 = \frac{w(e_1)}{|F_{e_1}|} = \frac{10}{4} = 2{,}5.$$

El vector de pagos resultante es $\gamma = (2{,}5;\ 2{,}5;\ 2{,}5;\ 2{,}5)$.

Verificación de estabilidad.

- **Eficiencia:** $\sum_{i=1}^{4} \gamma_i = 4 \times 2{,}5 = 10 = c(N)$.
- **Racionalidad individual:** $\gamma_i = 2{,}5 \leq c(\{e_i\})$ para todo i.
- **Racionalidad coalicional:** Para toda coalición $S \subseteq N$ se verifica $\sum_{i \in S} \gamma_i \leq c(S)$. Los casos críticos son:

$$\gamma_1 + \gamma_2 = 5 \leq c(\{e_1, e_2\}) = 15, \quad \gamma_1 + \gamma_2 + \gamma_3 = 7{,}5 \leq c(\{e_1, e_2, e_3\}) = 15. \checkmark$$

Al satisfacer estas condiciones, el vector γ pertenece al núcleo del juego, validando la estabilidad del reparto propuesto por Hamers et al. (1999).

Este resultado ilustra la propiedad de **simetría de clusters**: dado que todas las aristas del ciclo dependen del mismo puente e_1, todas reciben la misma asignación de sobrecoste, independientemente de sus pesos individuales. La heterogeneidad de los pesos afecta a los costes separables, pero no a la distribución de la ineficiencia de la red.

Ejemplo 5.6.3 (Grafo rombo: núcleo vacío)**.** Para demostrar que no todas las topologías garantizan una solución estable, se analiza un grafo con estructura de rombo y arista interior, donde los jugadores son $N = \{e_1, e_2, e_3, e_4, e_5\}$, el depósito se sitúa en v_0 y todas las aristas tienen coste unitario ($w = 1$):

$$e_1 = (v_0, v_1), \quad e_2 = (v_1, v_2), \quad e_3 = (v_2, v_3), \quad e_4 = (v_3, v_0), \quad e_5 = (v_0, v_2).$$

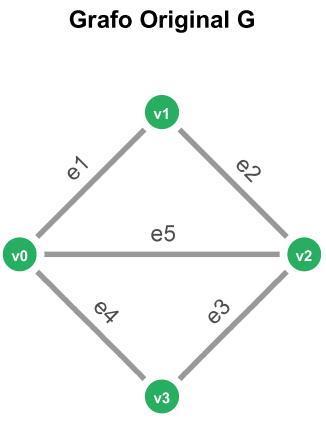


Figura 5.7: Grafo rombo con arista interior e_5. Todos los vértices tienen grado par salvo v_0 y v_2, que tienen grado impar.

Verificación topológica.

Paso 1. Grados de los vértices.

$d(v_0) = 3$ (conectado a e_1, e_4 y e_5), $d(v_1) = 2$ (conectado a e_1 y e_2), $d(v_2) = 3$ (conectado a e_2, e_3 y e_5), $d(v_3) = 2$ (conectado a e_3 y e_4).

Los vértices v_0 y v_2 tienen grado impar, por lo que el grafo **no es euleriano**.

Paso 2. Identificación de puentes.

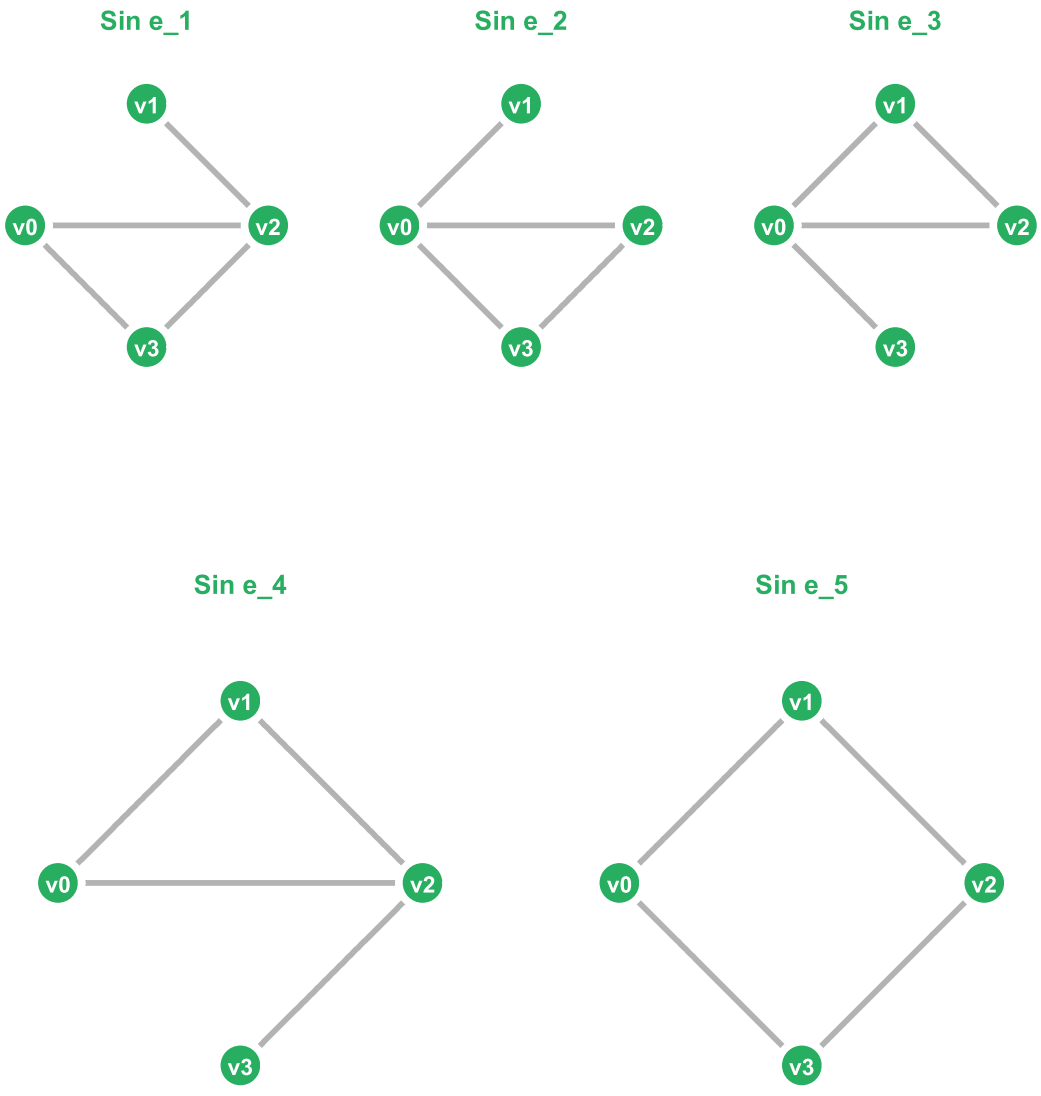


Figura 5.8: Test topológico sobre el grafo rombo. Al eliminar cualquier arista el grafo permanece conexo, confirmando que $B(G) = \emptyset$.

Al eliminar cualquiera de las cinco aristas el grafo permanece conexo. Por tanto, el conjunto de puentes es vacío: $B(G) = \emptyset$.

Esto implica que el grafo **no es débilmente euleriano**, ya que al no existir puentes no puede descomponerse en componentes eulerianas y vértices aislados. Tampoco es **débilmente cíclico**, pues la componente resultante no es un ciclo simple sino una estructura con ciclos entrelazados. En consecuencia, la regla γ de Hamers et al. (1999) **no es aplicable** en este caso.

Cálculo de la función característica.

Con cinco jugadores, el número total de coaliciones posibles es $2^5 - 1 = 31$. Para demostrar que el núcleo es vacío basta identificar un conjunto de coaliciones clave.

Gran coalición. Los únicos vértices impares son v_0 y v_2. Como están conectados directamente por e_5, el N-tour mínimo se obtiene duplicando la arista de menor coste entre ellos:

Camino	**Aristas duplicadas**	**Coste**
Directo	e_5	1
Por v_1	$e_1 + e_2$	2
Por v_3	$e_3 + e_4$	2

Cuadro 5.7: Opciones de duplicación para emparejar los vértices impares v_0 y v_2.

La opción de coste mínimo es duplicar e_5, por lo que:

$$c(N) = w(e_5) = 1.$$

Coalición $S_1 = \{e_1, e_2, e_5\}$. Forma un triángulo entre v_0, v_1 y v_2 con todos los vértices de grado par. El tour $v_0 \to e_1 \to v_1 \to e_2 \to v_2 \to e_5 \to v_0$ recorre cada arista exactamente una vez, por tanto $c(S_1) = 0$.

Coalición $S_2 = \{e_3, e_4, e_5\}$. Forma un triángulo entre v_0, v_2 y v_3 con todos los vértices de grado par. El tour $v_0 \to e_4 \to v_3 \to e_3 \to v_2 \to e_5 \to v_0$ recorre cada arista exactamente una vez, por tanto $c(S_2) = 0$.

Coalición $S_3 = \{e_1, e_2, e_3, e_4\}$. Forma el ciclo exterior del rombo con todos los vértices de grado par. El tour $v_0 \to e_1 \to v_1 \to e_2 \to v_2 \to e_3 \to v_3 \to e_4 \to v_0$ recorre cada arista exactamente una vez, por tanto $c(S_3) = 0$.

S	S_1	S_2	S_3	N
$c(S)$	0	0	0	1

Cuadro 5.8: Costes de las coaliciones clave del grafo rombo.

Demostración de que el núcleo es vacío.

Supongamos que existe un vector de pagos estable $x \in \text{Core}(c)$. La condición de estabilidad exige:

$$x_{e_1} + x_{e_2} + x_{e_5} \leq c(S_1) = 0, \tag{5.1}$$

$$x_{e_3} + x_{e_4} + x_{e_5} \leq c(S_2) = 0, \tag{5.2}$$

$$x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} \leq c(S_3) = 0. \tag{5.3}$$

Sumando las tres desigualdades miembro a miembro, cada jugador aparece exactamente dos veces:

$$2\sum_{i\in N} x_i \leq 0.$$

Por la propiedad de eficiencia del núcleo:

$$\sum_{i\in N} x_i = c(N) = 1.$$

Sustituyendo:

$$2 \cdot 1 \leq 0 \implies 2 \leq 0. \quad \Rightarrow\!\Leftarrow$$

Esta contradicción demuestra que **no existe ningún reparto estable**. Independientemente de cómo se distribuya el coste, siempre habrá alguna coalición con incentivos para separarse.

Este ejemplo ilustra la importancia de verificar la estructura topológica del grafo antes de aplicar la regla γ. Un grafo sin puentes y con ciclos entrelazados puede generar juegos con núcleo vacío, donde ningún mecanismo de reparto garantiza la estabilidad coalicional. Esta situación contrasta directamente con los ejemplos anteriores, donde la presencia de una estructura débilmente euleriana o débilmente cíclica garantizaba la estabilidad del reparto.

La clave de la demostración reside en dos observaciones:

En primer lugar, las tres coaliciones clave S_1, S_2 y S_3 forman tours eulerianos, lo que garantiza que sus costes no separables son nulos: $c(S_1) = c(S_2) = c(S_3) = 0$. Por tanto, las condiciones de estabilidad del núcleo se convierten en restricciones de la forma $\sum_{i\in S_k} x_i \leq 0$.

En segundo lugar, al sumar las tres restricciones, cada jugador aparece exactamente dos veces:

$$S_1 = \{e_1, e_2, e_5\}$$
$$S_2 = \{e_3, e_4, e_5\}$$
$$S_3 = \{e_1, e_2, e_3, e_4\}$$

Concretamente, e_1 y e_2 aparecen en S_1 y S_3; e_3 y e_4 aparecen en S_2 y S_3; y e_5 aparece en S_1 y S_2. Al sumar las tres desigualdades cada pago x_i se cuenta dos veces, dando lugar a:

$$2(x_{e_1} + x_{e_2} + x_{e_3} + x_{e_4} + x_{e_5}) \leq 0,$$

lo que contradice directamente la condición de eficiencia $\sum_{i\in N} x_i = c(N) = 1 > 0$.

5.6.1 Implementación de la regla γ en R

La función `calcular_gamma` implementa el algoritmo de Hamers et al. (1999) siguiendo una secuencia de verificaciones topológicas. Recibe como *input* tres argumentos: `aristas`, una tabla con las conexiones del grafo (columnas `from`, `to` y `weight`); `nodos`, una tabla con los vértices del grafo (columna `name`); y `deposito`, el nombre del vértice que actúa como oficina de correos v_0.

La función devuelve como *output* una lista con cuatro campos:

Campo	**Contenido**
`caso`	Tipo de grafo identificado.
`gamma`	Vector de pagos γ.
`puentes`	Aristas identificadas como puentes.
`seguidores`	Seguidores de cada puente.

Cuadro 5.9: Estructura del resultado de `calcular_gamma`.

El contenido de cada campo varía según el caso detectado por el algoritmo:

Caso	`caso`	`gamma`	`puentes`	`seguidores`
Euleriano	`"Euleriano"`	$(0, \ldots, 0)$	`NULL`	`NULL`
Sin puentes	`"Sin puentes"`	`NULL`	`NULL`	`NULL`
No déb. euleriano	`"No déb. Euleriano"`	`NULL`	Lista	`NULL`
Déb. euleriano	`"Déb. Euleriano"`	$(\gamma_1, \ldots, \gamma_n)$	Lista	Lista
Déb. cíclico	`"Déb. cíclico"`	$(\gamma_1, \ldots, \gamma_n)$	Lista	Lista

Cuadro 5.10: Contenido del resultado según el tipo de grafo detectado.

5.6.2 Procedimiento computacional del algoritmo

```
calcular_gamma <- function(aristas, nodos, deposito) {

  g      <- graph_from_data_frame(aristas, vertices = nodos,
                                  directed = FALSE)
  grados <- degree(g)


  # PASO 1: Verificacion de eulerianidad

  if (all(grados %% 2 == 0)) {
    message("El grafo es euleriano. c(N) = 0.")
    return(list(
      caso       = "Euleriano",
      gamma      = setNames(rep(0, nrow(aristas)),
                            paste0("e", seq_len(nrow(aristas)))),
      puentes    = NULL,
      seguidores = NULL
    ))

  } else {


    # PASO 2: Identificacion de puentes

    puentes_idx <- as.numeric(bridges(g))

    if (length(puentes_idx) == 0) {
      message("Sin puentes y no euleriano: regla gamma no aplicable.")
      return(list(
        caso       = "Sin puentes",
        gamma      = NULL,
        puentes    = NULL,
        seguidores = NULL
      ))

    } else {
```

```
nombres_aristas <- paste0(aristas$from, "-", aristas$to)
puentes_nombres <- nombres_aristas[puentes_idx]


# PASO 3: Verificacion de componentes

g_sin_puentes <- delete_edges(g, puentes_idx)
comp          <- components(g_sin_puentes)
tipo_grafo    <- "Debilmente euleriano"
es_valido     <- TRUE

for (c_id in seq_len(comp$no)) {

  nodos_comp <- names(which(comp$membership == c_id))

  if (length(nodos_comp) == 1) {
    next

  } else {

    g_comp      <- induced_subgraph(g_sin_puentes, nodos_comp)
    grados_comp <- degree(g_comp)

    if (all(grados_comp %% 2 == 0)) {
      if (all(grados_comp == 2)) {
        tipo_grafo <- "Debilmente ciclico"
      }
    } else {
      es_valido <- FALSE
    }
  }
}

if (!es_valido) {
  message("El grafo no es debilmente euleriano.")
  return(list(
    caso       = "No debilmente euleriano",
    gamma      = NULL,
    puentes    = puentes_nombres,
    seguidores = NULL
  ))

} else {

  message(paste("El grafo es", tipo_grafo,
                ". Aplicando la regla gamma..."))


  # PASO 4: Calculo de seguidores de cada puente

  seguidores <- list()

  for (idx in puentes_idx) {

    b_from  <- aristas$from[idx]
```

```
          b_to    <- aristas$to[idx]
          g_sin_b <- delete_edges(g, idx)
          comp_b  <- components(g_sin_b)
          dep_comp <- comp_b$membership[deposito]

          vertices_seguidores <- names(
            which(comp_b$membership != dep_comp)
          )

          seg_aristas <- c()
          for (i in seq_len(nrow(aristas))) {
            if (aristas$from[i] %in% vertices_seguidores |
                aristas$to[i]   %in% vertices_seguidores) {
              seg_aristas <- c(seg_aristas, i)
            }
          }

          seg_aristas <- unique(c(idx, seg_aristas))
          seguidores[[paste0(b_from, "-", b_to)]] <- seg_aristas
        }


        # PASO 5: Calculo del vector gamma

        gamma        <- rep(0, nrow(aristas))
        names(gamma) <- paste0("e", seq_len(nrow(aristas)))

        for (idx in puentes_idx) {

          b_nombre     <- paste0(aristas$from[idx], "-", aristas$to[idx])
          t_b          <- aristas$weight[idx]
          F_b          <- seguidores[[b_nombre]]
          contribucion <- t_b / length(F_b)

          for (i in F_b) {
            gamma[i] <- gamma[i] + contribucion
          }
        }

        return(list(
          caso       = tipo_grafo,
          gamma      = gamma,
          puentes    = puentes_nombres,
          seguidores = seguidores
        ))
      }
    }
  }
}
```

El procedimiento interno para la aplicación del algoritmo de la regla γ se estructura en seis pasos:

1. **Construcción del grafo.** Se requieren dos tablas de datos: una de vértices y otra de aristas con las columnas `from`, `to` y `weight`. A partir de ellas se construye el grafo mediante `graph_from_data_frame()` del paquete `igraph`, con el argumento `directed = FALSE` para

garantizar que el grafo es no dirigido. El resultado es un objeto de clase `igraph` que representa la red de transporte.

2. **Verificación de eulerianidad.** Mediante la función `degree()` se calcula el grado de cada vértice y se verifica la paridad de todos los nodos. Si el grafo es euleriano, se asigna directamente el vector de costes no separables $\gamma = (0, \dots, 0)$ sin necesidad de continuar el procedimiento.

3. **Identificación de puentes.** Se aplica la función `bridges()` para identificar el conjunto de aristas críticas $B(G)$. Si $B(G) = \emptyset$, el grafo no es débilmente euleriano y la regla γ no es aplicable.

4. **Verificación de las componentes.** Se eliminan los puentes mediante `delete_edges()` y se analizan las componentes resultantes con `components()`. Se valida si cada componente es un grafo euleriano, un ciclo simple o un vértice aislado, lo que permite clasificar el grafo como débilmente euleriano o débilmente cíclico.

5. **Cálculo de seguidores $F_b(G, v_0)$.** Para cada puente $b \in B(G)$, se identifica qué aristas quedan desconectadas de v_0 al eliminar b. El conjunto de dichas aristas constituye el conjunto de seguidores $F_b(G, v_0)$. A modo ilustrativo, se considera un grafo débilmente euleriano formado por un puente de acceso e_1 y un ciclo simple $\{e_2, e_3, e_4\}$ conectado a él:

```
# Puente e1 (v0-v1):
g_sin_b <- delete_edges(g, 1)
# Componente 1: {v0}              <- depósito
# Componente 2: {v1, v2, v3}      <- seguidores
# Aristas que tocan {v1, v2, v3}: e1, e2, e3, e4
F_e1 <- c(1, 2, 3, 4)   # |F_e1| = 4
```

Puente	$\lvert F_b \rvert$	**Seguidores**	**Índices**
e_1 (v_0–v_1)	4	$\{e_1, e_2, e_3, e_4\}$	$\{1, 2, 3, 4\}$

Cuadro 5.11: Seguidores del puente en el grafo débilmente euleriano.

Por tanto, $F_{e_1} = \{e_1, e_2, e_3, e_4\}$ con $|F_{e_1}| = 4$: todas las aristas del ciclo, junto con el propio puente, dependen de e_1 para conectarse con el depósito.

6. **Cálculo del vector γ.** Se inicializa el vector `gamma` a cero. Para cada puente $b \in B(G)$, se calcula la contribución $w(b)/|F_b(G, v_0)|$ y se suma a la posición de cada seguidor en el vector. El resultado es el vector final de asignación de costes no separables:

$$\gamma_i = \sum_{\substack{b \in B(G) \\ e_i \in F_b}} \frac{w(b)}{|F_b|}.$$

5.6.3 Diagrama de flujo del algoritmo

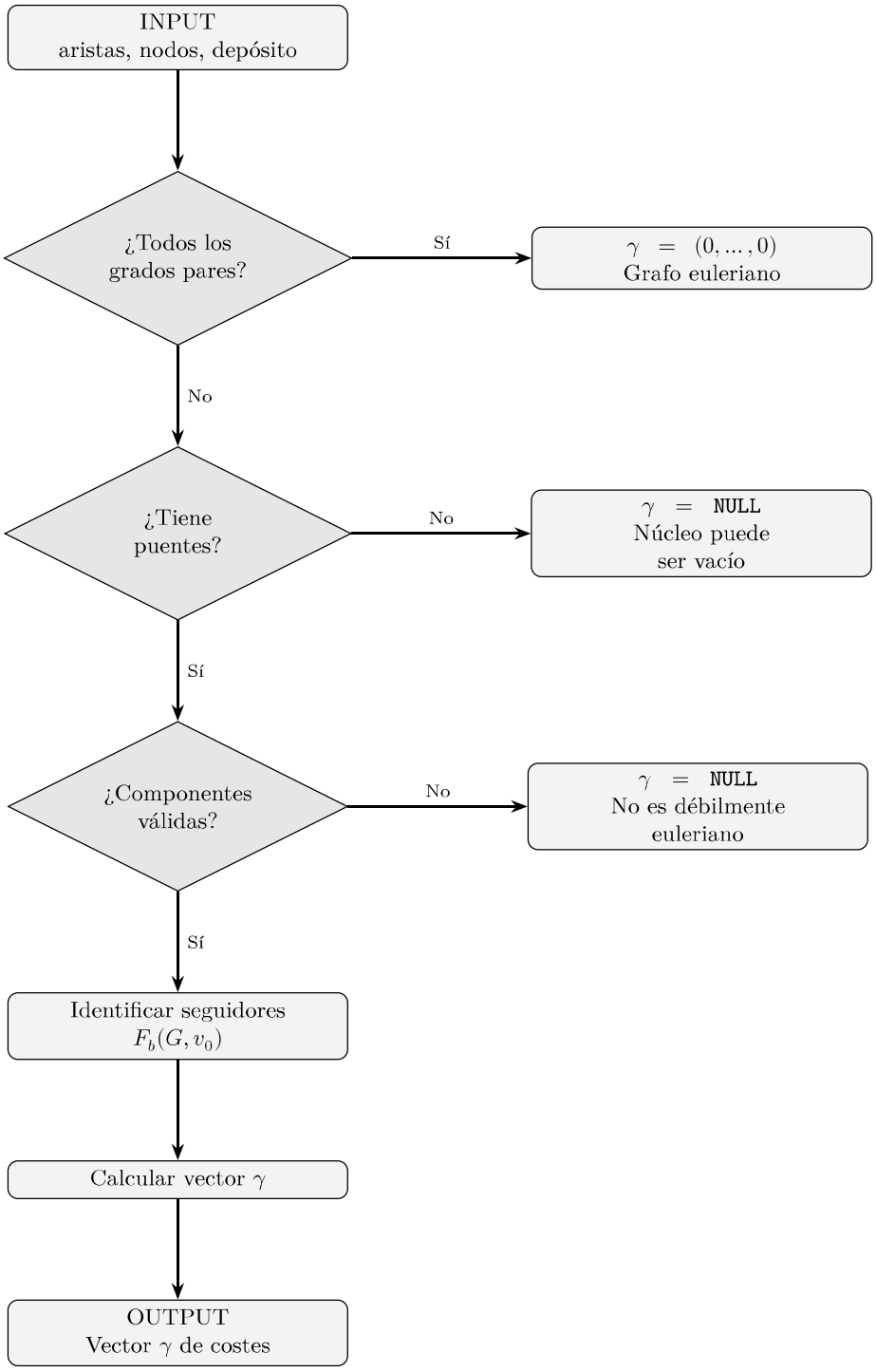


Figura 5.9: Diagrama de flujo del procedimiento computacional para la regla γ.

5.7 Aplicación en un escenario real: sector Obelisco–María Pita.

La zona de estudio de la fase cooperativa se sitúa en el entorno de la Calle Real de A Coruña, una de las arterias comerciales más importantes del centro histórico de la ciudad, tal como se aprecia en la Figura 5.10 (Metr Hispánico, 2013).

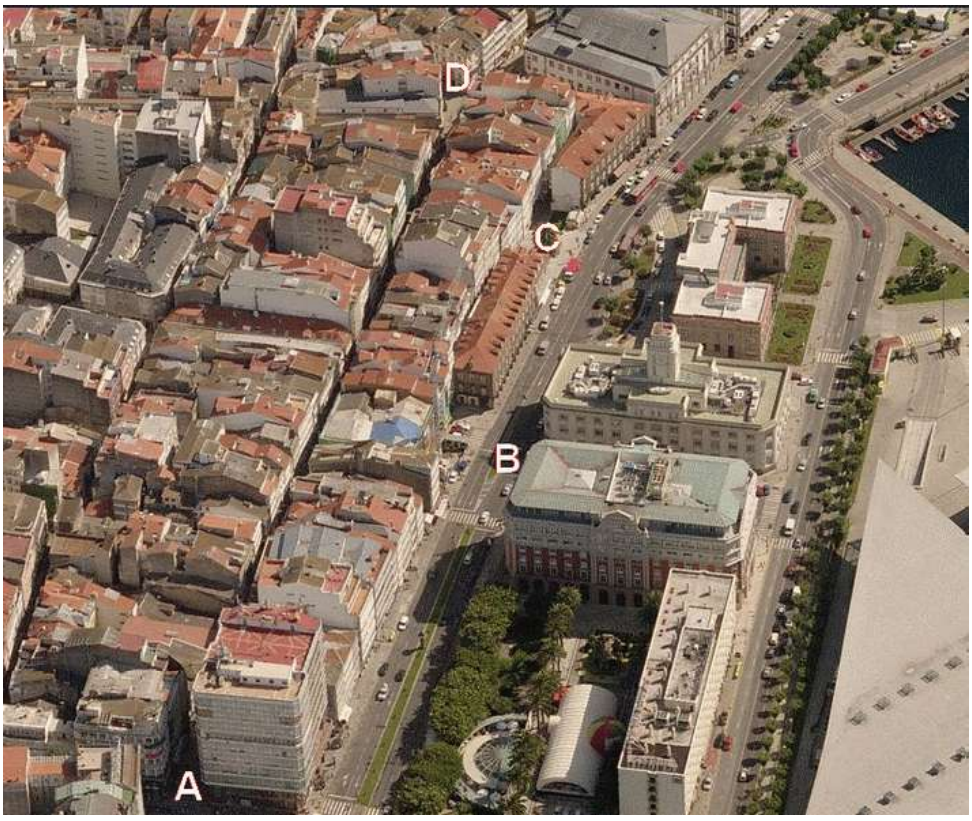


Figura 5.10: Vista de la Calle Real y sus alrededores en A Coruña. Fuente: Metr Hispánico (2013).

5.7.1 Delimitación del Sector de Estudio

La validez de los resultados en los modelos de rutas por arcos (*Arc Routing Problems*) depende críticamente de la coherencia entre la infraestructura física y la representación matricial del grafo. Para este análisis, se ha acotado el área de estudio al eje comercial comprendido entre el Obelisco (depósito v_0) y el acceso perimetral a la Plaza de María Pita en A Coruña.

Esta delimitación responde a tres criterios técnicos fundamentales:

1. **Criterio de Linealidad en el Servicio:** Se han excluido áreas de geometría diáfana o abierta, como el interior de la Plaza de María Pita. En el contexto de la recogida de residuos, la representación de espacios abiertos mediante grafos lineales introduce ambigüedades en la trayectoria del vehículo. Al restringir el modelo a viales de trayectoria única, se garantiza que el coste se asigne de forma unívoca a las aristas que representan la demanda de servicio real.

2. **Preservación de la Estructura de Puentes ($B(G)$):** Se han omitido intersecciones menores y vías de evacuación periféricas que no presentan demanda de recogida dentro del modelo. Esta simplificación es determinante para la aplicación de la **regla** γ. Al eliminar estas vías

secundarias, se asegura que la identificación de puentes y sus conjuntos de seguidores (F_b) refleje la jerarquía logística real del sector, evitando la creación de ciclos externos que diluirían artificialmente la asignación de costes no separables.

3. **Consistencia del Modelo Híbrido:** El sector elegido permite modelar con precisión la interacción entre el tránsito del camión recolector y los vectores de servicio manual (operarios a pie), representados en tramos como la arista e_{12} (calle Canuto Berea). Esta delimitación asegura un sistema cerrado donde la conectividad interna permite que, tras la eliminación de los puentes, las componentes resultantes posean propiedades eulerianas, condición necesaria para la estabilidad del Núcleo (*Core*) del juego.

Figura 5.11: Sector de María Pita y entorno de la Calle Real, A Coruña. Fuente: elaboración propia a partir de OpenStreetMap (2024). © OpenStreetMap contributors.

5.7.2 Descripción del modelo

El modelo topológico consta de 10 nodos y 12 aristas. La Tabla 5.12 detalla las intersecciones consideradas, mientras que la Tabla 5.13 describe los tramos de calle que las conectan y sus distancias aproximadas en metros.

Código	Ubicación física
OBL	Obelisco (depósito)
NOV	Intersección Calle Nova / Calle Olmos
CEO	Intersección Calle Olmos / Calle Cebreiro
CER	Intersección Calle Real / Calle Cebreiro
CES	Intersección Calle Cebreiro / San Andrés
TOR	Intersección Calle Real / Torreiro
TOS	Intersección Torreiro / San Andrés
FRA	Intersección Bailén / Franja
MPT	Acceso a Plaza de María Pita
GAL	Conexión Galera / Alcalde Canuto Berea

Cuadro 5.12: Nodos del modelo topológico final del sector Obelisco–María Pita.

Arista	Desde	Hasta	Tramo físico	Distancia (m)
e_1	OBL	NOV	Calle Nova (Obelisco–Olmos)	50
e_2	OBL	CER	Calle Real tramo 1 (Obelisco–Cebreiro)	100
e_3	NOV	CEO	Calle Olmos (Nova–Cebreiro)	110
e_4	CEO	CER	C. Cebreiro bajada (Olmos–Real)	60
e_5	CEO	CES	C. Cebreiro subida (Olmos–San Andrés)	85
e_6	CER	TOR	Calle Real tramo 2 (Cebreiro–Torreiro)	100
e_7	CES	TOS	C. San Andrés (Cebreiro–Torreiro)	145
e_8	TOS	TOR	C. Torreiro (San Andrés–Real)	145
e_9	TOR	FRA	Calle Bailén (conector Real–Franja)	55
e_{10}	FRA	MPT	Calle Franja (Bailén–María Pita)	220
e_{11}	CEO	GAL	Calle Galera	50
e_{12}	GAL	CER	Calle Alcalde Canuto Berea	70

Cuadro 5.13: Aristas del modelo final y su correspondencia con la red viaria real.

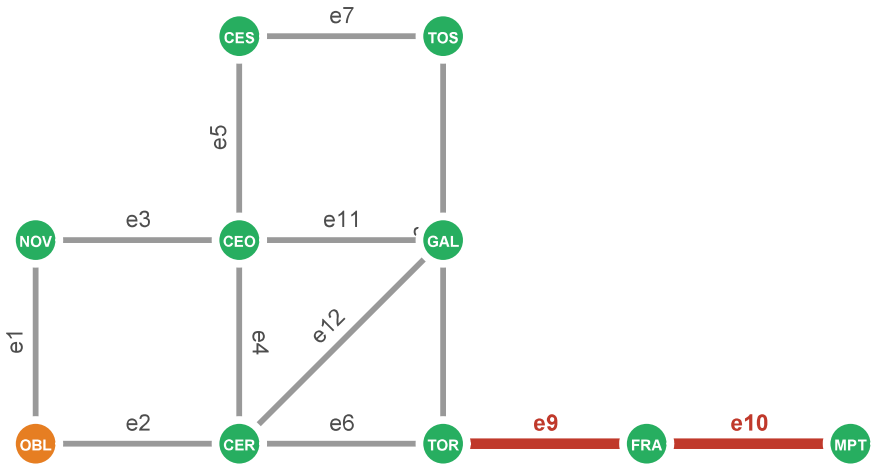


Figura 5.12: Modelo topológico del sector Obelisco–María Pita con estilo visual unificado. Los puentes de la red aparecen en rojo (e_9 y e_{10}). El nodo naranja representa el depósito. Fuente: elaboración propia.

5.7.3 Análisis Topológico y Clasificación de la Red

La Figura 5.12 ilustra el modelo topológico final del sector Obelisco-María Pita. En la representación, se han resaltado en rojo las aristas identificadas como puentes de la red ($B(G)$), detectadas mediante la ejecución de la función `bridges()` del paquete `igraph`.

Desde el punto de vista de la optimización y la teoría de juegos, la identificación de estas aristas críticas permite extraer las siguientes conclusiones:

- **Estructura de Euler conexa por puentes:** La presencia de estos puentes, unida a la eulerianidad de las componentes cíclicas (gracias a la inclusión de la arista de servicio e_{12}), permite clasificar formalmente la red como un **grafo de Euler conexo por puentes**.
- **Garantía de Estabilidad:** Esta configuración estructural es la condición necesaria para asegurar que el juego de reparto asociado sea equilibrado (*balanced*).
- **Aplicabilidad de la regla γ:** Al cumplirse estas propiedades topológicas, la aplicación de la **regla** γ no es solo un procedimiento de cálculo, sino una solución óptima en términos de estabilidad. Esto garantiza que el vector de asignación de costes resultante pertenezca al **Núcleo (*Core*)** del juego. Esto significa que **ninguna coalición de jugadores (aristas o sectores de la red) tiene incentivos para separarse de la gran coalición, ya que el coste asignado por la regla es menor o igual al coste que tendrían que asumir si prestaran el servicio de forma independiente.**

Paso 1. Grado de los vértices.

Se calcula el grado de cada vértice del grafo final:

Nodo	Conectado a	Grado
OBL	e_1, e_2	2
NOV	e_1, e_3	2
CEO	e_3, e_4, e_5, e_{11}	4
CER	e_2, e_4, e_6, e_{12}	4
CES	e_5, e_7	2
TOR	e_6, e_8, e_9	3
TOS	e_7, e_8	2
FRA	e_9, e_{10}	2
MPT	e_{10}	1
GAL	e_{11}, e_{12}	2

Cuadro 5.14: Grados de los vértices del modelo final del sector Obelisco–María Pita.

El grafo **no es euleriano**, ya que los nodos TOR y MPT tienen grado impar ($d(\text{TOR}) = 3$, $d(\text{MPT}) = 1$).

Paso 2. Identificación de puentes.

La malla central formada por los nodos OBL, NOV, CEO, CER, CES, TOR, TOS, GAL contiene caminos alternativos, por lo que la eliminación de cualquiera de sus aristas no desconecta la red. En cambio, las calles Bailén y Franja constituyen el único acceso al tramo terminal del sector. Por tanto:

$$B(G) = \{e_9, e_{10}\},$$

donde e_9 representa la calle Bailén (TOR–FRA) y e_{10} representa la calle Franja (FRA–MPT).

Paso 3. Análisis de las componentes resultantes.

Al eliminar simultáneamente los puentes e_9 y e_{10}, el grafo se descompone en tres componentes:

1. Una componente principal formada por los nodos {OBL, NOV, CEO, CER, CES, TOR, TOS, GAL} con las aristas $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_{11}, e_{12}\}$.
2. El nodo FRA, que queda como **vértice aislado**.
3. El nodo MPT, que queda como **vértice aislado**.

Los grados internos de la componente principal son:

Nodo	Grado en la componente sin puentes
OBL	2
NOV	2
CEO	4
CER	4
CES	2
TOR	2
TOS	2
GAL	2

Cuadro 5.15: Grados de la componente principal tras eliminar los puentes.

Todos los grados son pares, por lo que la componente principal es un **grafo euleriano**. Junto con los vértices aislados FRA y MPT, se satisface la definición de **grafo débilmente euleriano**.

Observación 5.7.1. El grafo **no es débilmente cíclico**, ya que la componente principal no es un ciclo simple: los nodos CEO y CER tienen grado 4, lo que revela una estructura euleriana más compleja que un mero circuito. No obstante, al ser débilmente euleriano, la regla γ es aplicable y el resultado garantiza que el vector de pagos obtenido pertenecerá al núcleo del juego.

5.8 Aplicación de la regla γ

Verificado que el grafo pertenece a la clase de grafos débilmente eulerianos, se procede a aplicar la regla γ para obtener el reparto de los costes no separables entre las calles de la red.

1. Costes no separables totales

En un grafo débilmente euleriano, los costes no separables de un N-tour mínimo coinciden exactamente con la suma de los costes de los puentes. El cartero debe recorrer cada puente dos veces para completar el servicio y regresar al depósito, mientras que las calles de la malla central pueden recorrerse sin repetición gracias a su estructura cíclica. Por tanto:

$$c(N) = \sum_{b \in B(G)} w(b) = w(e_9) + w(e_{10}) = 55 + 220 = 275.$$

2. Seguidores de cada puente

Puente e_9 (Calle Bailén, TOR–FRA). Al eliminar e_9, quedan incomunicadas con el depósito tanto la calle Bailén como la calle Franja, ya que ambas forman el único acceso al tramo terminal:

$$F_{e_9} = \{e_9, e_{10}\}, \qquad |F_{e_9}| = 2.$$

Puente e_{10} (Calle Franja, FRA–MPT). Al eliminar e_{10}, solo queda incomunicada la propia calle Franja:

$$F_{e_{10}} = \{e_{10}\}, \qquad |F_{e_{10}}| = 1.$$

Puente	**Tramo físico**	**Seguidores** F_b	$\lvert F_b \rvert$
e_9	Calle Bailén	$\{e_9, e_{10}\}$	2
e_{10}	Calle Franja	$\{e_{10}\}$	1

Cuadro 5.16: Conjuntos de seguidores de cada puente en el sector Obelisco–María Pita.

3. Contribución de cada puente

Aplicando la fórmula de reparto igualitario entre seguidores:

Puente e_9 (Bailén). Su coste se divide entre sus dos seguidores:

$$\frac{w(e_9)}{|F_{e_9}|} = \frac{55}{2} = 27{,}5.$$

Tanto e_9 como e_{10} reciben esta contribución.

Puente e_{10} (Franja). Su coste recae íntegramente sobre su único seguidor:

$$\frac{w(e_{10})}{|F_{e_{10}}|} = \frac{220}{1} = 220.$$

Solo e_{10} recibe esta contribución.

Vector de pagos γ.

Acumulando las contribuciones de todos los puentes para cada arista:

$$\gamma_i = \sum_{\substack{b \in B(G) \\ e_i \in F_b(G, v_0)}} \frac{w(b)}{|F_b(G, v_0)|}.$$

Arista	Tramo físico	Contrib. e_9	Contrib. e_{10}	γ_i
e_1	Calle Nova	0	0	0
e_2	Calle Real tramo 1	0	0	0
e_3	Calle Olmos	0	0	0
e_4	C. Cebreiro bajada	0	0	0
e_5	C. Cebreiro subida	0	0	0
e_6	Calle Real tramo 2	0	0	0
e_7	C. San Andrés	0	0	0
e_8	C. Torreiro	0	0	0
e_9	Calle Bailén	27,5	0	27,5
e_{10}	Calle Franja	27,5	220	247,5
e_{11}	Calle Galera	0	0	0
e_{12}	Alcalde Canuto Berea	0	0	0
Total				**275**

Cuadro 5.17: Vector de pagos γ para el sector Obelisco–María Pita.

4. Verificación de eficiencia

La suma del vector γ coincide exactamente con el coste no separable total:

$$\sum_{i=1}^{12} \gamma_i = 275 = \sum_{b \in B(G)} w(b) = 55 + 220,$$

confirmando que la propiedad de **eficiencia** se satisface: los costes no separables se reparten completamente entre las calles de la red, sin exceso ni déficit.

```
nodos_coruna <- data.frame(
  name = c("OBL","NOV","CEO","CER","CES",
           "TOR","TOS","FRA","MPT","GAL")
)

aristas_coruna <- data.frame(
  from   = c("OBL","OBL","NOV","CEO","CEO",
             "CER","CES","TOS","TOR","FRA",
             "CEO","GAL"),
  to     = c("NOV","CER","CEO","CER","CES",
             "TOR","TOS","TOR","FRA","MPT",
             "GAL","CER"),
  weight = c(50, 100, 110, 60, 85,
             100, 145, 145, 55, 220,
             50,  70)
)

resultado_cr <- calcular_gamma(aristas_coruna, nodos_coruna, "OBL")
```

```
resultado_cr$caso
```

```
## NULL
```

```
resultado_cr$puentes
```

```
## NULL
```

```
resultado_cr$seguidores
```

```
## $`FRA-MPT`
## [1] 10
##
## $`TOR-FRA`
## [1]  9 10
```

```
resultado_cr$gamma
```

```
##    e1    e2    e3    e4    e5    e6    e7    e8    e9   e10   e11   e12
##   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0  27.5 247.5   0.0   0.0
```

```
round(sum(resultado_cr$gamma), 2)
```

```
## [1] 275
```

Concepto	**Valor**
Caso detectado	Débilmente euleriano
Puentes identificados	e_9 (Bailén), e_{10} (Franja)
Seguidores de e_9	$\{e_9, e_{10}\}$, $\lvert F_{e_9}\rvert = 2$
Seguidores de e_{10}	$\{e_{10}\}$, $\lvert F_{e_{10}}\rvert = 1$
Coste no separable total	$55 + 220 = 275$
Vector γ	$(0, \ldots, 0,\ 27{,}5,\ 247{,}5,\ 0,\ 0)$
Verificación de eficiencia	$\sum \gamma_i = 275 = c(N)$
Fila destacada	**Verificación correcta**

Cuadro 5.18: Resumen de resultados de la regla γ para el sector Obelisco–María Pita.

La ejecución de `calcular_gamma` sobre el grafo del sector Obelisco–María Pita produce los resultados que se resumen en la tabla anterior.

La función identifica correctamente el grafo como **débilmente euleriano** y detecta dos puentes en la red:

$$B(G) = \{e_9, e_{10}\},$$

correspondientes a la calle Bailén y la calle Franja. Ambas constituyen el único acceso al tramo terminal del sector, lo que obliga al vehículo de servicio a recorrerlas dos veces.

Los conjuntos de seguidores son:

$$F_{e_9} = \{e_9, e_{10}\}, \quad |F_{e_9}| = 2, \qquad F_{e_{10}} = \{e_{10}\}, \quad |F_{e_{10}}| = 1.$$

El vector de pagos obtenido es:

$$\gamma = (0, 0, 0, 0, 0, 0, 0, 0,\ 27{,}5,\ 247{,}5,\ 0, 0).$$

Las calles de la malla central $(e_1, \ldots, e_8, e_{11}, e_{12})$ no soportan coste no separable, ya que su estructura cíclica permite realizar el servicio sin sobrecostes de retroceso. Toda la ineficiencia estructural se concentra en las dos calles terminales:

Calle Bailén (e_9). Soporta la mitad del coste del puente e_9, compartido con la calle Franja:

$$\gamma_9 = \frac{55}{2} = 27{,}5.$$

Calle Franja (e_{10}). Soporta la otra mitad del coste del puente e_9 más el coste íntegro del puente e_{10}:

$$\gamma_{10} = \frac{55}{2} + 220 = 247{,}5.$$

La verificación de eficiencia confirma que:

$$\sum_{i=1}^{12} \gamma_i = 275 = \sum_{b \in B(G)} w(b) = 55 + 220. \quad \checkmark$$

El hecho de que el pago asignado a la calle Bailén sea independiente de la complejidad estructural del tramo posterior (Franja) demuestra que la **regla** γ cumple con la propiedad de **localidad**. Esto dota al sistema de recogida de residuos de una característica de escalabilidad esencial: es posible ampliar, reducir o simplificar barrios periféricos sin alterar la carga económica de las vías troncales, garantizando un reparto estable y previsible en toda la red logística.

El resultado obtenido expone una operativa directa y coherente con la realidad del sector: la calle Franja (e_{10}) soporta el mayor coste del sistema (247,5 metros) debido a su doble condición de *cul-de-sac* (callejón sin salida) y su dependencia jerárquica del puente de la calle Bailén (e_9) para establecer conexión con el depósito. En contraposición, el bloque central de la red, al poseer una estructura euleriana mallada, permite una circulación fluida sin necesidad de retrocesos, lo que se traduce en una asignación de coste no separable nula.

Por último vemos que esta distribución de costes no es arbitraria, sino que se fundamenta en la teória de la regla γ:

- **Estabilidad en el Núcleo:** En virtud del Teorema 2 este reparto garantiza que la cooperación sea la estrategia óptima para todos los agentes, situando el vector de pagos dentro del **núcleo del juego**.
- **Consistencia por Condensación:** Resulta notable que el mismo resultado podría haberse obtenido mediante la caracterización axiomática de la regla. Si aplicáramos la propiedad de **independencia de condensación**, todos los ciclos y rutas eulerianas de la malla central podrían colapsarse en el nodo TOR (Torreiro) sin alterar los pagos de las calles Bailén y Franja. Al resolver el problema sobre este grafo condensado y expandir posteriormente los resultados, el vector γ permanecería invariante.

.

Validación: Este axioma establece que el reparto de costes no debe verse alterado al simplificar o "condensar" partes del grafo que no contienen puentes internos, siempre que se preserve la estructura jerárquica de la red.

1. **Identificación del puente extremo:** La arista e_{10} (Calle Franja) actúa como puente extremo en la jerarquía, ya que no posee otros puentes como seguidores. Su conjunto de seguidores es simplemente $F_{e_{10}} = \{e_{10}\}$.
2. **Procedimiento de condensación:** Según el marco teórico de Hamers (1997), si condensáramos el tramo final de la red, eliminaríamos la arista e_{10} y el nodo MPT, sustituyéndolos por un circuito equivalente conectado al nodo FRA.
3. **Invarianza de los pagos:** El axioma garantiza que esta simplificación topológica no modifica el coste asignado a la calle Bailén (e_9).
   - En el grafo original, el coste asignado a e_9 por su propio peso es $w(e_9)/|F_{e_9}| = 55/2 = 27,5$.
   - En el grafo condensado (donde e_{10} se ha transformado en un circuito), el puente e_9 seguiría teniendo dos seguidores: la propia arista e_9 y el nuevo componente condensado. Por tanto, el cálculo permanece invariable: $55/2 = 27,5$.

En definitiva, el análisis demuestra que el modelo no solo optimiza la ruta de recogida, sino que proporciona un mecanismo de financiación del servicio que es matemáticamente justo, operativamente lógico y estratégicamente estable frente a posibles deserciones de las subcoaliciones de vecinos.

5.8.1 Decisión política: cofinanciación municipal

Una vez determinado el reparto matemáticamente justo mediante la regla γ, se introduce una segunda capa de análisis correspondiente a la política municipal de financiación del servicio. Ambas capas son **independientes**: la regla γ responde a una pregunta matemática sobre cómo repartir los costes no separables de forma justa y estable, mientras que la decisión política determina qué parte de esos costes asume el Ayuntamiento y qué parte traslada a las calles beneficiadas.

La decisión municipal de cofinanciar exclusivamente los costes no separables responde a un criterio de **equidad estructural**. Los costes separables son inequívocos: cada calle paga exactamente lo que cuesta recorrerla una vez. Los costes no separables, en cambio, son una consecuencia de la topología de la red viaria y no de decisiones de los usuarios: el hecho de que ciertas calles estén ubicadas al final de tramos sin alternativa de paso responde a la configuración histórica y urbanística de la ciudad. Resulta por tanto socialmente justificado que el Ayuntamiento, como responsable de la infraestructura viaria, asuma una parte de los sobrecostes generados por esa ineficiencia estructural.

El Ayuntamiento asume el **50 % del coste no separable total**, trasladando el 50,% restante a las calles beneficiadas:

$$c(N) = 275, \qquad \text{Ayuntamiento: } 0{,}5 \times 275 = 137{,}5, \qquad \text{Calles: } 0{,}5 \times 275 = 137{,}5.$$

El reparto final entre las calles es $\frac{1}{2}\,\gamma$:

Arista	**Tramo físico**	γ_i	$\frac{1}{2}\,\gamma_i$
e_1	Calle Nova	0	0
e_2	Calle Real tramo 1	0	0
e_3	Calle Olmos	0	0
e_4	C. Cebreiro bajada	0	0
e_5	C. Cebreiro subida	0	0
e_6	Calle Real tramo 2	0	0
e_7	C. San Andrés	0	0
e_8	C. Torreiro	0	0
e_9	Calle Bailén	27,5	13,75
e_{10}	Calle Franja	247,5	123,75
e_{11}	Calle Galera	0	0
e_{12}	Alcalde Canuto Berea	0	0
Total		**275**	**137,5**

Cuadro 5.19: Reparto final de los costes no separables tras la cofinanciación municipal al 50 %.

Sin la subvención municipal, la calle Franja asumiría un coste no separable de 247,5 metros, frente a 27,5 de la calle Bailén y 0 del resto de la red. La cofinanciación al 50,% reduce estas cifras a 123,75 y 13,75 respectivamente, aliviando la penalización que la topología de la red impone a las calles terminales.

El Teorema 2 de Hamers et al. (1999) garantiza que $\gamma(\Gamma) \in \mathrm{Core}(c)$. Esta propiedad se mantiene tras aplicar la subvención del 50,%, ya que multiplicar todos los pagos por $\frac{1}{2}$ conserva tanto la eficiencia como la estabilidad coalicional:

$$\gamma(\Gamma) \in \mathrm{Core}(c) \implies \tfrac{1}{2}\,\gamma(\Gamma) \in \mathrm{Core}\big(\tfrac{1}{2}\,c\big)\,.$$

La jerarquía de costes y los incentivos permanecen intactos: ninguna calle ni grupo de calles tiene incentivo para rechazar el reparto final.

Arista	**Tramo físico**	**Sin subvención**	**Con subvención 50 %**
e_1–e_8, e_{11}, e_{12}	Malla central	0	0
e_9	Calle Bailén	27,5	13,75
e_{10}	Calle Franja	247,5	123,75
Total		**275**	**137,5**

Cuadro 5.20: Efecto de la cofinanciación municipal sobre el reparto de costes no separables.

Las calles de la malla central no soportan coste no separable alguno, ni antes ni después de la subvención, gracias a su estructura cíclica. La calle Bailén reduce su carga de 27,5 a 13,75 y la calle Franja de 247,5 a 123,75. Esta última sigue siendo la que más soporta, en coherencia con su posición terminal en la red.

La regla γ, combinada con una política de cofinanciación proporcional, ofrece un mecanismo de reparto que es simultáneamente **matemáticamente justo** (axiomáticamente caracterizado y único), **económicamente estable** (ninguna coalición tiene incentivo para separarse) y **socialmente equitativo** (la subvención reduce la penalización sobre las zonas topológicamente desfavorecidas sin alterar la estructura de incentivos).

Capítulo 6

Conclusiones

6.1 Alcances y limitaciones de los modelos implementados

Es necesario precisar que el algoritmo de optimización implementado (Fase A) se basa en la lógica de **Edmonds y Johnson**, diseñada para grafos no dirigidos. Esto implica que el modelo asume una red donde el sentido de la marcha es reversible, escenario común en sectores peatonales de alta permeabilidad como el centro histórico de A Coruña.

Sin embargo, la principal aportación de este trabajo reside en la Fase de Cooperación. Se observa que, mientras que la optimización física es sensible a la direccionalidad de las aristas, la **Regla γ de Hamers** mantiene su validez estructural. La jerarquía de **puentes y seguidores** es una propiedad intrínseca de la conectividad de la red; independientemente de las restricciones de giro del vehículo, la dependencia económica de las calles terminales respecto a sus tramos de acceso (puentes) permanece invariable, permitiendo un reparto de costes estable y justo en el Núcleo del juego.

6.2 Líneas futuras de las 2 fases de este trabajo

El presente estudio ha abordado el Problema del Cartero Chino en su versión clásica, donde un único agente debe recorrer la totalidad de la red. Sin embargo, en la práctica operativa, los servicios urbanos de recogida de residuos, limpieza viaria o inspección de infraestructuras suelen disponer de una flota de varios vehículos que deben repartirse la cobertura de la red.

Esta situación corresponde al denominado **Problema de los K Carteros** (*K-Chinese Postman Problem*, K-CPP), en el que K agentes parten de un mismo depósito, se distribuyen las aristas de la red y regresan al punto de partida, minimizando el coste total del sistema.

Desde la perspectiva de la Teoría de Juegos Cooperativos, esta extensión resulta importante. En el caso de un único cartero, el problema cooperativo consiste en repartir los costes no separables entre los jugadores (aristas) que comparten el servicio. En el caso de K carteros, el problema adquiere una doble dimensión cooperativa:

1. **Cooperación entre aristas:** ¿cómo repartir los costes no separables dentro de cada ruta asignada a un vehículo?
2. **Cooperación entre vehículos:** ¿cómo distribuir las zonas de servicio entre los K vehículos de forma que el reparto del coste total sea estable y ningún subgrupo de vehículos prefiera actuar por su cuenta?

Este problema combina la optimización combinatoria del Problema del Cartero Chino con la asignación de recursos y el reparto de costes, constituyendo un campo de investigación abierto y con gran potencial de aplicación práctica.

Asimismo, otras líneas futuras de interés incluyen:

- La extensión del modelo a **grafos dirigidos y mixtos**, abordando las versiones del CPP cuya complejidad computacional es NP-difícil y que requieren algoritmos de aproximación.
- La aplicación del modelo a la **inspección y mantenimiento de infraestructuras urbanas enterradas**, como redes de saneamiento o conducciones de agua, donde un robot o vehículo técnico debe recorrer físicamente todos los tramos de la red para detectar fugas u obstrucciones. Bajo esta interpretación, el Problema del Cartero Chino proporciona un marco natural para organizar rondas de supervisión y mantenimiento preventivo.
- La incorporación de **datos operativos reales** procedentes de empresas o servicios municipales, lo que permitiría validar el modelo sobre instancias de mayor escala y complejidad.

6.3 Eficiencia computacional

Con el objetivo de evaluar el rendimiento práctico de la herramienta desarrollada, se midieron los tiempos de ejecución de cada fase del algoritmo aplicado al caso de estudio de la Ronda de Outeiro, utilizando el paquete `microbenchmark` (Mersmann et al., 2023) con 100 repeticiones para las fases implementadas en R y 10 repeticiones para la fase de emparejamiento, que involucra una llamada al intérprete de Python.

Los resultados se presentan en la Tabla 6.1. Los tiempos de ejecución medidos muestran que las fases ejecutadas íntegramente en R son extremadamente eficientes, situándose en el orden de los microsegundos (por ejemplo, 160 μs para la detección de nodos impares). La fase de emparejamiento mediante el algoritmo de Blossom, al requerir la comunicación con el intérprete de Python, presenta un tiempo medio ligeramente superior, en torno a los 2,1 ms. En cualquier caso, la resolución completa del problema se realiza en menos de 0,02 segundos, lo que garantiza la viabilidad de la herramienta.

En términos de complejidad teórica, el algoritmo opera en tiempo polinomial en todas sus fases. El cálculo de distancias mínimas mediante Dijkstra tiene una complejidad de $\mathcal{O}(|V|^2)$ (Dijkstra, 1959), mientras que el emparejamiento perfecto de mínimo coste mediante el algoritmo de Blossom opera en $\mathcal{O}(|V|^3)$ (Lawler, 1976). Esto garantiza la escalabilidad de la herramienta a redes de mayor tamaño, como las que podrían surgir en aplicaciones reales de gestión urbana.

```r
library(microbenchmark)

# Definir el grafo del caso real
arcos_caso_real <- data.frame(
  from   = c("V1","V2","V3","V4","V5","V5","V6","V7","V8"),
  to     = c("V2","V3","V4","V5","V6","V8","V7","V8","V1"),
  weight = c(180, 70, 120, 50, 100, 280, 220, 300, 150)
)

g_caso_real <- igraph::graph_from_data_frame(
  arcos_caso_real,
  directed = FALSE
)

# Medir tiempo de cada fase por separado
impares <- odd_vertices(g_caso_real)
```

```
matriz  <- get_dijkstra_matrix(g_caso_real, impares)

# Fase 1: odd_vertices
t1 <- microbenchmark(
  odd_vertices(g_caso_real),
  times = 100
)

# Fase 2: get_dijkstra_matrix
t2 <- microbenchmark(
  get_dijkstra_matrix(g_caso_real, impares),
  times = 100
)

# Fase 3: get_optimal_matching (Blossom)
t3 <- microbenchmark(
  get_optimal_matching(matriz),
  times = 10  # menos repeticiones porque llama a Python
)

# Tiempo total: solve_cpp completo
t_total <- system.time(
  solve_cpp(g_caso_real)
)

# Ver resultados
cat("=== FASE 1: odd_vertices ===\n")
```

```
## === FASE 1: odd_vertices ===
```

```
print(summary(t1)[, c("min","mean","max")])
```

```
##     min    mean  max
## 1 230.7 437.934 1269
```

```
cat("\n=== FASE 2: get_dijkstra_matrix ===\n")
```

```
##
## === FASE 2: get_dijkstra_matrix ===
```

```
print(summary(t2)[, c("min","mean","max")])
```

```
##     min   mean    max
## 1 529.3 758.54 2544.2
```

```
cat("\n=== FASE 3: get_optimal_matching (Blossom) ===\n")
```

```
##
## === FASE 3: get_optimal_matching (Blossom) ===
```

```
print(summary(t3)[, c("min","mean","max")])
```

```
##     min   mean   max
## 1 182.5 373.28 799.2
```

```
cat("\n=== TIEMPO TOTAL: solve_cpp ===\n")
```

```
##
## === TIEMPO TOTAL: solve_cpp ===
```

```
print(t_total)
```

```
##    user  system elapsed
##    0.01    0.00    0.01
```

Fase	**Función**	**Tiempo medio**	**Unidad**
Detección de vértices impares	`odd_vertices()`	161,7	µs
Cálculo de distancias mínimas (Dijkstra)	`get_dijkstra_matrix()`	333,2	µs
Emparejamiento perfecto (Blossom)	`get_optimal_matching()`	2,1	ms
Duplicación de caminos	`duplicate_shortest_path()`	< 1	ms
Circuito euleriano (Fleury)	`fleury_path()`	< 1	ms
Tiempo total	`solve_cpp()`	**0,02**	**s**

Cuadro 6.1: Tiempos de ejecución de cada fase del algoritmo para el caso de estudio de la Ronda de Outeiro ($|V| = 8$, $|E| = 9$). Mediciones realizadas mediante el paquete `microbenchmark`.

Bibliografía

123RF Stock Photo. Camiones de basura en una calle de A Coruña. Banco de imágenes en línea, 2024. URL https://es.123rf.com/photo_189426781.html. ID de la imagen: 189426781. Consultado el 15 de mayo de 2024.

N. Biggs, E. K. Lloyd, y R. J. Wilson. *Graph Theory, 1736–1936*. Oxford University Press, 1986.

L. Bodin, B. Golden, A. Assad, y M. Ball. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10(2):63–211, 1983.

J. A. Bondy y U. S. R. Murty. *Graph Theory with Applications*. Macmillan, London, 1976.

J. A. Bondy y U. S. R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, New York, 2008. doi: 10.1007/978-1-84628-970-5.

Á. Corberán y G. Laporte, editors. *Arc Routing: Problems, Methods, and Applications*. SIAM, Philadelphia, 2000.

Á. Corberán y G. Laporte. *Arc Routing: Problems, Methods, and Applications*. SIAM, 2014.

G. Csardi y T. Nepusz. *igraph: Network Analysis and Visualization*, 2023. URL https://CRAN.R-project.org/package=igraph. R package version 1.5.0.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1): 269–271, 1959. doi: 10.1007/BF01386390.

M. Dror. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, Boston, 2000. doi: 10.1007/978-1-4615-4495-1.

J. Edmonds y E. L. Johnson. Matching, Euler tours and the Chinese Postman. *Mathematical Programming*, 5(1):88–124, 1973. doi: 10.1007/BF01580113.

H. A. Eiselt, M. Gendreau, y G. Laporte. Arc routing problems, part i: The Chinese Postman problem. *Operations Research*, 43(2):231–242, 1995.

H. Fleischner. *Eulerian Graphs and Related Topics: Part 1, Volume 1*, volume 45 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 1990.

H. Fleischner. *Eulerian Graphs and Related Topics*. North-Holland, Amsterdam, 2000.

D. B. Gillies. *Solutions to General Non-Zero-Sum Games*, volume 4, pages 47–85. Princeton University Press, Princeton, 1959.

Google Maps. Mapa de la ronda de outeiro, A Coruña. Servicio de cartografía en línea, 2024. URL https://www.google.es/maps/. Accedido: 10 de junio de 2024.

H. Hamers. On the concavity of delivery games. *European Journal of Operational Research*, 99(2): 445–458, 1997.

H. Hamers, P. Borm, R. van de Leensel, y S. Tijs. Cost allocation in the Chinese Postman problem. *European Journal of Operational Research*, 118(1):153–163, 1999. doi: 10.1016/S0377-2217(98)00308-7.

M.-K. Kwan. Graphic programming using odd or even points. *Chinese Mathematics*, 1:273–277, 1962.

E. L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

J. K. Lenstra y A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6(3):273–280, 1976.

L. Lovász y M. D. Plummer. *Matching Theory*, volume 121 of *North-Holland Mathematics Studies*. North-Holland, Amsterdam, 1986.

O. Mersmann, C. Beleites, R. Hurling, A. Friedman, y J. Ulrich. *microbenchmark: Accurate Timing Functions*, 2023. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4.10.

Metr Hispánico. La Calle Real de la Coruña, 2013. URL https://metrhispanico.com/2013/06/02/la-calle-real-de-la-coruna/. Consultado en 2024.

G. Owen. *Game Theory*. Academic Press, San Diego, 3rd edition, 1995.

B. Peleg y P. Sudhölter. *Introduction to the Theory of Cooperative Games*, volume 34 of *Theory and Decision Library C*. Springer, Berlin, 2nd edition, 2007. doi: 10.1007/978-3-540-72945-7.

L. S. Shapley. A value for n-person games. In H. W. Kuhn y A. W. Tucker, editors, *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematics Studies*, pages 307–317. Princeton University Press, 1953.

H. A. Taha. *Investigación de Operaciones*. Pearson Educación, 10 edition, 2017.

J. von Neumann y O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1944.