Universidad de Valladolid

Facultad de Ciencias

Trabajo Fin de Grado

Grado en Matemáticas

Reconstrucción de trayectorias de aeronaves usando *Simulated Annealing* para resolver una versión del problema del viajante (TSP)

Autor: María Zorita Mínguez

Tutor: Pedro César Álvarez Esteban

"Todos tus sueños pueden hacerse realidad
si tienes el coraje de perseguirlos"

Walt Disney

"La confianza en sí mismo es
el primer secreto del éxito"

Ralph Waldo Emerson

Agradecimientos

Quiero dar las gracias en primer lugar a mi tutor, Pedro César Álvarez, por darme la posibilidad de participar en el proyecto "AIRPORTS" en el que colabora, así como por toda su dedicación y apoyo en el desarrollo del mismo.

También dar las gracias a Boeing Research and Technology Europe (BR&T-E), ya que la realización del presente trabajo se enmarca en la colaboración que llevan a cabo diversos miembros de la Universidad de Valladolid con ellos. Por ello, quiero agradecerles toda la información y datos aportados sobre las trayectorias de los vuelos, dado que han sido de gran importancia para la realización del trabajo.

Dar las gracias también a mi compañero Juan Manuel Velasco Heras por su apoyo y amistad durante estos 5 años que hemos estado juntos, y en particular, por la convivencia durante los últimos años de la carrera.
Además, dar las gracias a toda mi familia y amigos, y en especial a mi madre, por su apoyo incondicional y por darme ánimos en los momentos que más lo necesitaba.

Índice

Capítulo 0

Introducción

La gestión del tráfico aéreo (Air Traffic Management, ATM) consiste en la instalación de nuevos sistemas y tecnologías de comunicación entre el aeropuerto, la aeronave y el centro de control, para así, lograr aumentar la capacidad de los Servicios de Navegación Aérea. Desde hace muchos años existen mecanismos e instituciones que se encargan de gestionar y coordinar el tráfico aéreo de todo el mundo. Considerando el caso europeo, que es uno de los más transitados a nivel mundial, se tiene como gestores de tráfico aéreo más importantes los siguientes: ENAIRE en España, DSNA en Francia, DFS en Alemania, ENAV en Italia, NATS en Reino Unido, EUROCONTROL MUAC en el Noroeste de Alemania, Bélgica, Luxemburgo y los Países Bajos.

Por su gran importancia, destaca Eurocontrol, pues es la Organización Europea para la Seguridad de la Navegación Aérea que fue creada en 1963 con el objetivo de apoyar la aviación europea. Además, junto con la Comunidad Europea (CE), fundaron la empresa SESAR Joint Undertaking (SESAR JU) que está llevando a cabo la iniciativa conocida como Single European Sky ATM Research (SESAR). Dicho proyecto surge como instrumento tecnológico de la iniciativa europea Single European Sky (SES), y se trata de uno de los más innovadores desarrollados por la Unión Europea. Con su desarrollo, se busca reformar la arquitectura actual de la gestión del tráfico aéreo de Europa con el objetivo de crear y poner en práctica una política común de transporte aéreo. Todo esto, es necesario y muy importante (principalmente por la fragmentación del espacio aéreo europeo), y con ello, se pretende lograr mejorar la eficiencia y la seguridad del tráfico aéreo, reducir gastos de funcionamiento, el impacto medioambiental, así como optimizar el uso del espacio aéreo.

Dado que la gestión del tráfico aéreo es un tema bastante complejo y delicado de realizar, ya que intervienen muchos elementos y factores (como la gran cantidad de aeropuertos y de aeronaves y aerolíneas por cada uno de ellos), estos nuevos sistemas buscan introducir mejoras para modernizar los sistemas actuales. Para conseguir dichas mejoras y propósitos mencionados, se requiere hacer uso de varias técnicas de vigilancia. Entre ellas destacan las técnicas radar y el sistema de control de tráfico aéreo Automatic Dependant Surveillance (ADS), el cual, permite que las aeronaves proporcionen de manera automática a través de un enlace de datos todos los datos necesarios sobre la navegación, posición, identificación, etc. al servicio de control de tráfico aéreo (ATS), para así, lograr mejorar la seguridad del tráfico aéreo.

Dentro de este marco de investigación, surge el proyecto "AIRPORTS (CIEN, 2015)" liderado por Boeing Research and Technology Europe (BR&T-E), el cual entre otras muchas líneas de investigación, busca estudiar la eficiencia de los vuelos de aeronaves comerciales en función de la trayectoria que estos describen al realizar dicho vuelo.

Dichas aeronaves están equipadas con un sistema conocido como Automatic Dependant Surveillance Broadcast (ADS-B) que se encarga de enviar diferentes mensajes ADS-B describiendo el estado del vuelo en cada momento. Estos mensajes contienen información de importancia como es: el identificador del mensaje, el tiempo de recepción (*timestamp*), la posición (en términos de longitud, latitud y altitud) de la aeronave, su velocidad (tanto horizontal como vertical), si está o no en suelo, la fecha de realización, etc. Además, se necesitan diferentes redes de sensores distribuidas por distintas partes del planeta que permiten obtener diversas fuentes de datos ADS-B pudiendo ser comerciales o no. Una vez que se reciben los mensajes ADS-B que describen el estado del vuelo, el equipo del proyecto "AIRPORTS" los utiliza para reconstruir las distintas trayectorias de vuelo. Gracias a su reconstrucción es posible realizar su estudio, y para ello, se usa una plataforma Big Data, la cual, es un prototipo que permite calcular una gran variedad de métricas de eficiencia.

La motivación de este proyecto surge cuando al estudiar dichas trayectorias nos damos cuenta de que hay problemas con respecto al alineamiento temporal de las señales obtenidas, siendo las causas principales: el retardo en el tiempo de recepción de dichos mensajes y la falta de sincronización horaria de los dispositivos receptores. Por ello, el presente trabajo tiene como objetivo establecer un modelo matemático que permita resolver este mal alineamiento temporal, estableciendo las trayectorias con una alineación temporal correcta y corrigiendo las que presentan anomalías.

El objeto de estudio presenta cierta relación con problemas ya existentes y estudiados desde hace mucho tiempo como es el caso de la teoría de grafos, la optimización, etc. pudiendo además ser formulado como un problema matemático que es una variante del famoso problema conocido como "El problema del viajante o Travelling Salesman Problem (TSP)".

A lo largo del presente trabajo, se comenzará en el Capítulo 1 describiendo el problema del viajante para introducir y entender el problema que se plantea resolver. Para resolver este problema existen diversos algoritmos que encuentran una ruta factible a través de la búsqueda de ciclos hamiltonianos, pero en nuestro caso, nos centraremos en los algoritmos de mejora. En particular, en el Capítulo 2 estudiaremos el *Simulated Annealing*, que es un algoritmo estocástico y funciona bien en bastantes ocasiones, especialmente, en el caso en que la solución de partida sea próxima a la óptima que es la situación que se tendrá en la práctica.

Luego, en el Capítulo 3 se ilustran y muestran ejemplos generales que se resuelven aplicando el algoritmo *Simulated Annealing*. Tras ello, en el Capítulo 4, dichas aplicaciones se realizan sobre datos de trayectorias reales aportadas por el tutor y para su resolución se aplican diversos algoritmos. Después, a partir de los resultados obte-

nidos, se realiza una pequeña comparación entre ellos para sacar conclusiones sobre su comportamiento frente al problema planteado. Finalmente, en el Capítulo 5, se establecen unas conclusiones generales sobre la realización del trabajo, así como si ha resultado exitoso o no su desarrollo, y por lo tanto, si ha contribuido de manera positiva para la mejora y avance en la gestión del tráfico aéreo.

Conviene decir que la realización del presente Trabajo de Fin de Grado se ha realizado de manera paralela con el Trabajo de Fin de Grado de Informática. Por ello, la temática de ambos es la misma, pero mientras que el de Matemáticas se centra principalmente en el estudio y análisis teórico de la convergencia del algoritmo *Simulated Annealing*, el de Informática busca implementar una solución escalable que permita resolver el problema planteado en la práctica logrando encontrar una ruta lo más próxima posible a la solución óptima.

Ha sido de gran utilidad poder realizar ambos trabajos de manera conjunta y coordinada, puesto que así, se ha podido profundizar más en el estudio del algoritmo y con ello dar una buena solución al problema planteado. Por un lado, el trabajo realizado en Informática me ha permitido corroborar los resultados teóricos descritos en el presente documento sobre la convergencia del algoritmo, ya que se ha podido realizar su visualización práctica. Por otro lado, el de Matemáticas también ha contribuido y ha sido útil para realizar el trabajo de Informática, ya que he podido entender en detalle y profundidad el funcionamiento de dicho algoritmo antes de realizar la implementación del mismo usando el modelo de programación *MapReduce*, lo cual, me ha ayudado a ir más rápido en su realización y entender cada uno de los pasos realizados.

También, conviene señalar que las ejecuciones prácticas realizadas sobre diversos conjuntos de datos en los Capítulos 3 y 4 se han llevado a cabo de forma conjunta y coordinada con mi compañero Juan Manuel Velasco Heras, y por lo tanto los resultados obtenidos en ambos trabajos son los mismos. Para que resulte más claro cuáles son las partes comunes, se volverá a indicar en las mismas. Además, para su realización se ha usado el dashboard creado en el Trabajo de Fin de Grado de Informática usando el entorno *R-Studio*, de manera que las imágenes y resultados mostrados durante el documento sobre trayectorias de vuelo se han obtenido del mismo. Sin embargo, aunque la temática del trabajo de mi compañero y la mía es similar, yo me he centrado en estudiar la heurística de mejora *Simulated Annealing* que se basa en la aleatoriedad o el azar, y él ha estudiado las heurísticas de mejora local, como es el caso del *2-opt* y *Lin-Kernighan*, que se basan en la realización de intercambios. Todo esto ha enriquecido el trabajo realizado y ha permitido realizar una comparativa de los resultados obtenidos por dichos algoritmos para sacar conclusiones sobre su funcionamiento.

Capítulo 1

El problema del viajante

1.1. Definición

El problema del viajante también conocido como "Travelling Salesman Problem (TSP)", consiste en determinar la ruta más corta posible que recorre un conjunto de ciudades (de manera general nodos), de manera que el nodo final coincida con el nodo de partida y que todas las ciudades sean visitadas una única vez.

Dicho problema ha sido estudiado durante muchos años y aún es objeto de estudio dentro de la optimización combinatoria pues, aunque aparentemente parezca un problema fácil de resolver debido a que el número de posibles caminos que existe entre un conjunto de nodos sea finito, en realidad no lo es. Se trata de un problema complejo de resolver, de hecho, es un problema de tipo NP-Duro. Por ello, incluso se considera la posibilidad de no llegar nunca a encontrar un algoritmo que, en todas las situaciones posibles, encuentre la solución óptima.

Para desarrollar el contenido de los apartados de esta sección se ha usado el libro de William J.Cook [1] y el de E.L. Lawler y otros [2].

1.2. Historia

El estudio del problema del viajante surgió hace muchos años y es debido principalmente a la gran utilidad que presenta en las situaciones de la vida real estando muy relacionado con la logística, la distribución de productos, el transporte, etc.

En el año 1832 se dio a conocer en Alemania el primer libro sobre este tema denominado *"El viajante de comercio: cómo debe ser y qué debe hacer para conseguir comisiones y triunfar en su negocio. Por un viajante de comercio veterano*".

Durante los siguientes años, muchos matemáticos se dedicaron a investigar sobre dicho problema, sin embargo, no fue hasta el año 1930 cuando fue definido formalmente en términos matemáticos. Dicha formulación fue realizada por el matemático y economista Karl Menger quien consideró en un primer momento como método de resolución la fuerza bruta, aunque pronto observó que no era un método eficiente y

menos aún óptimo.

Poco después, Hassler Whitney dio a conocer dicho problema con el término anglosajón "Travelling Salesman Problem" y, poco a poco, en las décadas de los 50 y 60 dicho problema ganó mucha popularidad como consecuencia principalmente de la publicidad llevada a cabo por Procter and Gamble en 1962. En ella, se propuso un concurso donde se podía obtener un premio de 10.000$ si se resolvía el problema del viajante para un conjunto de 33 ciudades de EE.UU.

Resultó ser un problema complejo de resolver y, aunque nadie logró llevarse el premio, se pudo demostrar que ya en el año 1954, esto es, 8 años antes, tres matemáticos de Rand Corp. (George Dantzing, Ray Fulkerson y Selmer Johnson) habían encontrado la solución óptima para un conjunto de exactamente 49 ciudades. Para ello desarrollaron el método de los Planos de Corte y lo aplicaron a su estudio obteniendo resultados muy positivos.

Este resultado fue un gran avance y, de hecho, supuso un reto a superar que no se logró hasta 1971, esto es, 17 años después, cuando los investigadores de IBM Michael Held y Richard Karp resolvieron el problema para el caso de 64 ciudades distribuidas al azar en una región cuadrada, donde los costos eran considerados como la distancia en línea recta entre cada par de ellas.

Cuatro años más tarde, concretamente en 1975, Panagiotis Miliotis logró la solución óptima para el caso de 80 puntos distribuidos de manera aleatoria.

Ya en 1977 Grötschel publicó su tesis doctoral en la que determinaba la solución óptima para el caso de 120 ciudades. Fue entonces cuando se asociaron Padberg y el investigador de IBM Harlan Crowder obteniendo la solución óptima para el problema de 318 ciudades distribuidas en un tablero de circuitos. La ocurrencia de todos estos sucesos fueron muy relevantes en la historia del TSP y dieron lugar a un gran avance en el desarrollo de dicho problema, ya que, Grötschel y Padberg de manera independiente lograron hallar la solución óptima para el caso de 532 ciudades en Estados Unidos, 666 localizaciones en el mundo, 1.002 ciudades con problemas de perforación, y posteriormente, de 2.392 ciudades.

Más tarde en 1988, como consecuencia de los éxitos ocurridos, Vasek Chvátal y William J.Cook se unieron en el estudio del problema logrando en el año 1992 resolverlo para el caso de 3.038 ciudades, para lo que hicieron uso de una amplia red de computadoras que trabajaban en paralelo. Siguiendo con ello, lograron en 1998 encontrar la ruta óptima de 13.509 ciudades en Estados Unidos, otra de 24.978 en Sucia en el año 2004, y finalmente, otra en 2006 de 85.900 ciudades.

Todos estos avances en el estudio y desarrollo del problema del viajante fueron posibles gracias al uso de una herramienta informática denominada *Concorde* que se comenzó a usar ya en el año 1992. Se trata de un programa en *C* muy usado actualmente (intentando su mejora y avance) para este tipo de problemas de optimización

de redes y, que en años anteriores, supuso una gran revolución en el avance del TSP.

En la Figura 1.1 se puede ver el avance producido gracias al uso de la herramienta *Concorde* puesto que se pasó muy rápidamente de encontrar la ruta óptima para el caso de 33 ciudades (se corresponde con la ruta negra) y de 120 ciudades (se corresponde con la azul) a lograr la solución óptima para el caso de 15112 ciudades (se corresponde con la ruta roja).

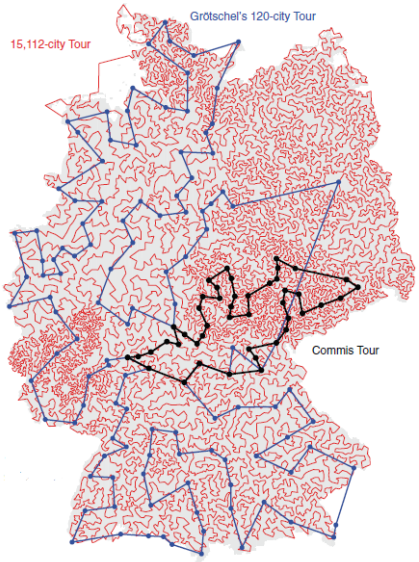


Figura 1.1: Tres recorridos distintos en Alemania
William J.Cook ([1, pág. 14])

En la Figura 1.2 se muestra de manera gráfica la evolución en el avance de los resultados obtenidos a medida que aumenta el número de ciudades consideradas. Se aprecia como a partir del 2006 dicho aumento es exponencial.

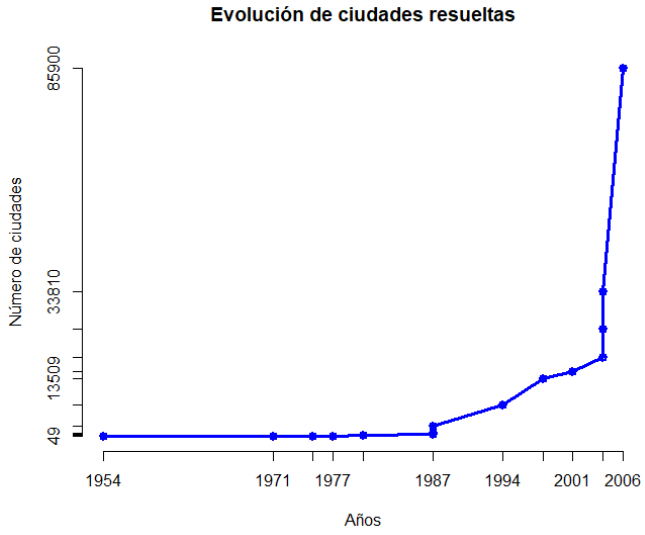


Figura 1.2: Evolución de ciudades resueltas a lo largo de los años

1.3. Aplicaciones

El problema del viajante o TSP ha sido y sigue siendo uno de los principales problemas de estudio, debido principalmente a su contribución y aplicación en diferentes áreas para mejorar y resolver diversas situaciones y problemas de la vida diaria. Las principales áreas de aplicación son la logística y distribución, así como la programación de curvas de producción.

Al inicio, las mejoras del TSP tenían como objetivo conseguir aplicarlo de manera directa, como por ejemplo en rutas de autobuses escolares o de una empresa de lavandería. Poco a poco el ámbito de aplicación fue creciendo y hoy en día incluso es útil para problemas en el ámbito genético.

Se detallan a continuación las aplicaciones más importantes dentro del área de logística así como en la industria:

Logística: El TSP tiene aplicaciones muy abundantes en logística como son las siguientes.

- **Vendedores, Turistas**: Suelen usar algún tipo de sistema para planificar las rutas turísticas de manera que están sean óptimas tanto en tiempo como en coste volviendo al punto de partida. Estos planificadores suelen basarse en algoritmos de resolución de tipo TSP.
- **Rutas escolares y laborales**: Al igual que en el caso anterior se determinan rutas mediante algoritmos de tipo TSP para ahorrar costes y tiempo.
- **Transporte de paquetes y mercancías**: Este tipo de problemas se suele adaptar mejor a problemas de arcos en lugar de problemas de nodos como es el caso del TSP pero, sin embargo, resulta útil cuando las distancias entre los lugares de reparto son lejanas o sólo se desea visitar un lugar concreto.

Sector Industrial: Aunque las aplicaciones del TSP en la industria son menos abundantes también son importantes y se podrían considerar las siguientes.

- **Secuenciación de las tareas**: Consiste en realizar diversas tareas de la manera más rápida posible minimizando el costo de su producción, y para ello, el orden de su realización debe ser independiente entre las mismas. En este caso cada una de las tareas se asemeja al papel de una ciudad y el tiempo que se tarda en realizar una tarea i habiendo hecho antes la tarea j es lo que equivale a la distancia entre las ciudades, luego dicho tiempo es el que se tiene que minimizar.
- **Producción de placas de circuitos electrónicos**: Consiste en realizar agujeros en una placa mediante la perforación automática de la misma usando la técnica del TSP. Para ello, se considera cada uno de los puntos a perforar como una ciudad diferente, de manera que el tiempo en crear dichas placas se reduce al mínimo posible.

Existen más aplicaciones del TSP, pero algunas de ellas, tienen una relación menos intuitiva con dicho problema ya que no requieren de movimientos físicos como es el caso de la búsqueda de planetas o la organización de datos en diferentes grupos que es usado para la minería de datos o para extraer patrones en los datos.

1.4. Complejidad

La clasificación de los problemas a resolver puede realizarse según su complejidad y, según dicho criterio se distingue entre problemas NP, problemas P y problemas NP-Completos. Hasta el momento, los problemas de tipo P y NP-Completos tienen intersección vacía siendo ambos problemas de tipo NP.

Un problema de tipo P es aquel que se resuelve en tiempo polinomial por un algoritmo determinista (ej: una máquina de Turing determinista), mientras que un problema NP es aquel que se resuelve en tiempo polinomial por un algoritmo no determinista (ej: una máquina de Turing no determinista).
Los problemas de tipo NP-Completo son problemas de tipo NP-Duro que están contenidos en la clase de problemas NP, donde x es un problema NP-Duro si cualquier problema que pertenezca a la clase de problemas NP puede reducirse en tiempo polinomial a x.

Para que resulte más claro lo descrito con anterioridad, en la Figura 1.3 se muestra un gráfico acerca de la relación que existe entre los tipos de problemas comentados.

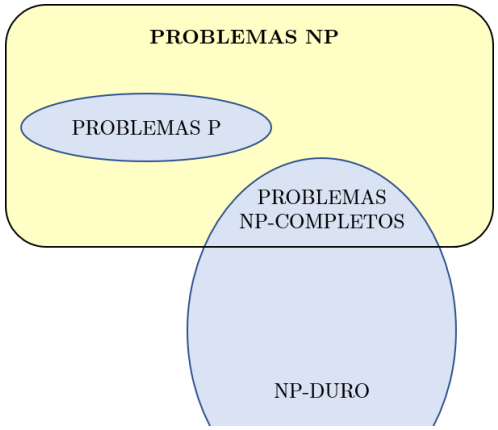


Figura 1.3: Diagrama de Venn siendo P $\neq$ NP

Como ya se comentó, el problema del viajante o TSP es considerado a día de hoy un problema NP-Duro, luego, todo problema que pertenezca a la clase de problemas de tipo NP puede transformarse en tiempo polinomial en él.

Frente a estos conceptos surge la cuestión de encontrar algún problema NP-Completo que sea de tipo P ya que, en ese caso, se obtendrá que P = NP. Dicho resultado se dio a conocer por Stephen Cook en 1971 (William J.Cook [1, pág.9]) y todavía no ha sido posible su demostración.

Con respecto al problema del viajante, al ser un problema de tipo NP-Completo, desde hace años se está estudiando si se puede resolver en tiempo polinomial para poder con ello demostrar que P = NP. Esto, que aún no se ha podido demostrar, tiene una gran importancia, ya que en ese caso, se tendría que todos los problemas NP-Completos, y por lo tanto, todos los problemas NP podrían ser resueltos en tiem-

po polinomial permitiendo resolver muchos problemas que hoy en día son intratables.

Por lo tanto, aunque a simple vista el problema del viajante parezca un problema sencillo, hoy en día aún no existe un algoritmo eficiente para resolverlo, pero como se verá, si existen buenos algoritmos que obtienen soluciones muy próximas a la óptima en un tiempo razonable.

1.5. Teoría de Grafos

Es una de las ramas matemáticas y de las ciencias de computación que se centra en estudiar las propiedades de los grafos.

Para poder entender algunos de los algoritmos propuestos para resolver el TSP es necesario conocer antes los conceptos fundamentales de *Teoría de Grafos* y, en este apartado, se hace una pequeña introducción a alguno de ellos.
Para ello, se ha usado principalmente el libro de A. Caicedo, G. Wagner y R.M Méndez [9] junto con el de L.W. Beineke y R.J. Wilson [10].

Grafo: Un grafo, denotado por $G = (V, A)$, es una pareja ordenada en la que V es un conjunto de vértices no vacío $\{v_1, ..., v_n\}$ y A es el conjunto de aristas o arcos entre dichos vértices, esto es, A está formado por pares no ordenados de vértices de la forma $a_{ij} = (v_i, v_j)$ siendo $i \neq j$.

Las aristas son las líneas que unen un par de vértices y pueden ser de varios tipos:

Adyacentes: Aquellas que convergen en el mismo vértice.

Paralelas: Aquellas aristas conjuntas para las que el vértice inicial y final es el mismo.

Cíclicas: Aquellas que parten de un vértice y entran en sí mismo.

Cruzadas: Aquellas que se cruzan en el mismo punto.

También existen diferentes tipos de grafos, siendo algunos los siguientes:

Grafo dirigido u orientado: Se suele denominar dígrafo y es aquel en el que las aristas son pares ordenados, esto es, $a_{ij} \neq a_{ji}$, $\forall\ i \neq j$. Dichas aristas se suelen representar con una flecha que va del vértice de partida al vértice final.

Grafo no dirigido o no orientado: Es aquel en el que los pares a_{ij} y a_{ji} se corresponden con la misma arista, esto es, las aristas no tienen dirección. Durante el desarrollo del trabajo se considera este tipo de grafos.

Grafo completo: Es aquel en el que aparecen trazadas todas las posibles aristas.

Grafo etiquetado: Es aquel en el que las aristas del mismo tienen un determinado peso asociado. Por ejemplo en el caso de las ciudades sería la distancia que hay entre cada par de ellas.

En la Figura 1.4 se muestra un ejemplo de cada uno de ellos.

A continuación se detallan unos conceptos que serán utilizados para describir el problema del viajante o TSP en términos matemáticos.

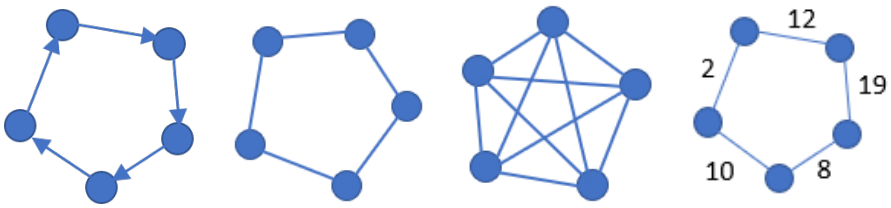
Camino: Es un conjunto de vértices conectados a través de aristas que van de un vértice inicial a un vértice final. Se trata de sucesiones en las que aparecen elementos de V y A de forma alternativa, esto es, $v_0, a_{01}, v_1, ..., v_{n-1}, a_{n-1n}, v_n$ donde $v_i \in V$, $\forall\, i = 0, ..., n$ y $a_{ij} \in A$, $\forall\, i = 0, ..., n-1$ y $j = i+1$.
En el caso en el que el nodo o vértice inicial coincida con el vértice final, esto es, $v_0 = v_n$ se denomina circuito o camino cerrado. Se pueden considerar distintos tipos:

Camino hamiltoniano: Es aquel camino que pasa por todos los nodos del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_i \neq v_j$, $\forall\, i, j = 0, ..., n$ con $i \neq j$, $v_i \in V$ y $a_{ij} \in A$, $\forall\, i = 0, ..., n-1$ y $j = i+1$.

Camino euleriano: Es aquel camino que recorre todas las aristas del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $a_{ij} \neq a_{ji}$, $\forall\, i = 0, ..., n-1$ y $j = i+1$, $v_i \in V$ y $a_{ij} \in A$, $\forall\, i = 0, ..., n-1$ y $j = i+1$.

Circuito hamiltoniano: Es aquel circuito que pasa por todos los nodos del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_0 = v_n$, $v_i \neq v_j$, $\forall\, i, j = 1, ..., n-1$ con $i \neq j$, $v_i \in V$ y $a_{ij} \in A$,
$\forall\, i = 0, ..., n-1$ y $j = i+1$.

Circuito euleriano: Es aquel circuito que recorre todas las aristas del grafo en una única ocasión. Más formalmente es una secuencia de la forma $v_0 a_{01} ... a_{n-1n} v_n$ donde $v_0 = v_n$, $a_{ij} \neq a_{ji}$, $\forall\, i = 0, ..., n-1$ y $j = i+1$, $v_i \in V$ y $a_{ij} \in A$, $\forall\, i = 0, ..., n-1$ y $j = i+1$.



(a) Dirigido (b) No dirigido (c) Completo (d) Etiquetado

Figura 1.4: Tipos de grafos

A partir de lo anteriormente descrito, se puede observar la relación existente entre la *Teoría de Grafos* y el TSP, de manera que puede describirse como un problema de *Teoría de Grafos* donde el objetivo es encontrar un circuito hamiltoniano de coste mínimo.

Para ello, se debe considerar un grafo completo $G = (V, A)$ donde V denota el conjunto de ciudades a visitar y A son las aristas entre ellas. Como ya se comentó, dicho grafo debe estar etiquetado, esto es, a cada arista de unión entre dos ciudades se la asocia un peso que, en este caso, es una distancia de la forma $d_{v_i v_j}$ (distancia de la ciudad en el vértice v_i a la ciudad en el vértice v_j).
Por lo tanto, lo que se busca es encontrar en dicho grafo un circuito que sea hamiltoniano y de coste el menor posible, pues así, todas las ciudades serán visitadas exactamente una sola vez y con la ruta de menor distancia posible.

Todo ello nos lleva a considerar dos tipos de problemas TSP:

TSP simétrico: Es el que se considera en el trabajo y se basa en que G es un grafo no dirigido, por lo que $d_{v_iv_j} = d_{v_jv_i}$ para todo $(v_i, v_j) \in A$ con $i \neq j$, esto es, la distancia entre dos ciudades es la misma independientemente de la ciudad de partida y de destino.

En términos matemáticos el propósito es minimizar la función objetivo siguiente:

$$\sum_i \sum_j d_{v_iv_j} x_{ij}$$

donde x_{ij} toma el valor 0 si la arista (v_i, v_j) no forma parte de la solución y 1 en caso contrario.

Además se tienen una serie de restricciones como son:

$$\sum_i x_{ij} = 1 \quad \forall\, j, \;\; \sum_j x_{ij} = 1 \quad \forall\, i$$

De esta manera, se establece que sólo una arista puede entrar en un vértice del grafo (desde cada ciudad v_j sólo se puede llegar a una ciudad v_i) y, que sólo una arista puede salir de cada vértice (desde cada ciudad v_i sólo se puede llegar a una ciudad v_j). Así, se logra que cada ciudad sea visitada en una única ocasión.

Sin embargo, con esto no es suficiente, ya que se necesita considerar una serie de restricciones a mayores para evitar que haya subciclos.
Sea W un subconjunto de vértices del conjunto V considerado. Considerando los siguientes conjuntos,

$$\begin{aligned} A(W) &= \{a_{ij} = (v_i, v_j) \in A : v_i, v_j \in W\} \\ \delta^-(W) &= \{a_{ij} = (v_i, v_j) \in A : v_i \notin W, v_j \in W\} \\ \delta^+(W) &= \{a_{ij} = (v_i, v_j) \in A : v_i \in W, v_j \notin W\} \end{aligned}$$

se tiene que satisfacer la siguiente condición:

$$\sum_{(v_i,v_j)\in A(W)} x_{ij} \leq |W| - 1\ , \;\; \forall \;\; W \subset V \quad \equiv \quad \sum_{v_i \in W, v_j \notin W} x_{ij} \geq 1\ , \;\; \forall \;\; W \subset V$$

Las condiciones anteriores son equivalentes y denotan que en todo subconjunto de nodos debe haber al menos un arco que salga del mismo para así evitar subciclos.

De esta manera quedaría descrito el problema del viajante de comercio en términos matemáticos.

TSP asimétrico: En este caso G es un grafo dirigido. No se considera en el trabajo ya que no es de interés para resolver el problema de las trayectorias que lo motiva.

Para resolver el problema del TSP se plantean varias alternativas.
Una de ellas, consiste en buscar un método o algoritmo que permita determinar la solución óptima del problema planteado, pero, hasta el momento, esta posibilidad no ha resultado factible para el caso en que se tienen más de 20 vértices. Por lo tanto, se ha optado por una segunda alternativa que consiste en buscar un método con el que se obtenga una primera ruta factible, y tras ello, mejorar el coste de la misma intentando así estar lo más cerca posible de la solución óptima, esto es, se buscan soluciones aproximadas. Estos métodos se denominan heurísticos y se caracterizan por ser eficientes, buenos (solución próxima a la óptima) y robustos (probabilidad de que la solución no sea próxima a la óptima es baja). Así, mediante su uso se logra encontrar una solución factible o satisfactoria del problema de manera bastante rápida y eficiente.

1.6. Heurísticas para encontrar una ruta factible

Existen diversos métodos heurísticos usados para resolver el TSP que conviene mencionar. Se puede distinguir entre heurísticas constructivas, heurísticas de mejora, así como otros. Todos ellos permiten acercarse lo más posible a la solución óptima.

1.6.1. Heurísticas constructivas

Existen diversas heurísticas constructivas que se basan en considerar la mejor solución en cada iteración, esto es, la que cueste menos de acuerdo a un determinado criterio establecido.

Heurística vecinos más próximo: Este método busca encontrar un circuito hamiltoniano de coste mínimo apoyándose en la idea de elegir siempre el vértice del grafo que esté más cercano del actual (aristas de menor coste) y no haya sido visitado. Una vez visitados todos los vértices se termina uniendo el último vértice visitado con el de partida. Por ir siempre en busca de lo mejor, se suele considerar un método ambicioso o voraz.

De manera genérica, sea un grafo $G = (V, A)$ de n vértices tal y como fue descrito en la Sección 1.5. Los pasos seguidos por este algoritmo son los siguientes:

1. Se selecciona un vértice v_i del grafo al azar donde $i \in \{0, ..., n\}$.
2. Se considera $p = i$ y el subgrafo $W = V \setminus \{i\}$.
3. Mientras $W \neq \emptyset$
    a. Se selecciona un vértice $v_i \in W$ tal que $d_{pi} = min\{d_{pj} : \forall\, j \in W\}$.
    b. Se une el vértice v_p con v_i.
    c. Se considera $W = W \setminus \{i\}$ y $p = i$

En la Figura 1.5 se muestra un ejemplo muy sencillo para ilustrar el funcionamiento del método anterior. Para ello, se indica en color amarillo el nodo de partida y de llegada. En este caso el camino encontrado es el óptimo pero no siempre es así.

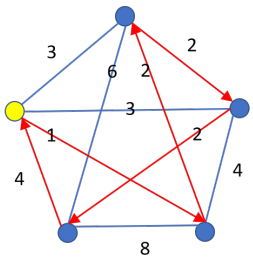
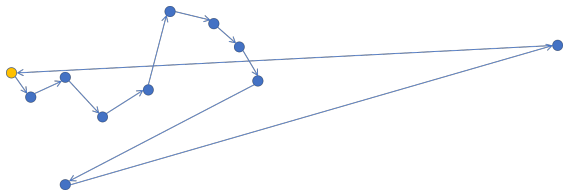


Figura 1.5: Ejemplo vecino más próximo

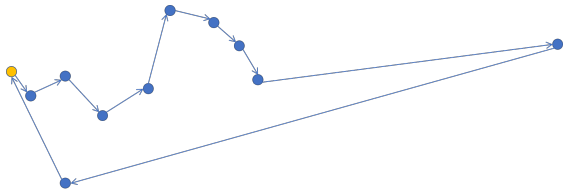
Para situaciones en las que se tienen pocos nodos o ciudades dicho algoritmo resulta útil debido a su sencillez y rapidez pero, en general, no es recomendable pues lo habitual es tener una gran cantidad de nodos, y en este caso, suele obtener recorridos bastante malos con respecto a la solución óptima.

Una de las principales consecuencias por la que se obtienen recorridos lejanos al óptimo se debe a que cuando se realiza la elección de los nodos sólo se tiene en cuenta el nodo que se encuentra más próximo. Esto provoca que la distancia entre el último nodo considerado y el de partida sea demasiado grande haciendo ineficiente dicho algoritmo. Si el objetivo fuera considerar un camino hamiltoniano en lugar de un circuito hamiltoniano dicho método resultaría muy adecuado, ya que, en la mayoría de los casos se encontraría la solución óptima o una muy próxima a la misma.

Dicho problema descrito se muestra en la Figura 1.6, donde se considera el coste de las aristas como la distancia existente entre pares de vértices, siendo el nodo amarillo el de partida. Es un ejemplo muy sencillo donde se puede ver cómo esta situación afecta de manera negativa para encontrar la ruta óptima.



(a) Método vecinos más próximo



(b) Posible mejor solución

Figura 1.6: Heurística vecinos más próximo

Heurística de inserción: Son un conjunto de métodos que se basan en construir circuitos usando un determinado conjunto de vértices, y posteriormente, se van insertando uno a uno los restantes vértices en dicho circuito hasta formar un circuito hamiltoniano.

De manera genérica, sea un grafo $G = (V, A)$ de n vértices tal y como fue descrito en la Sección 1.5. Los pasos seguidos por este algoritmo son los siguientes:

1. Se selecciona un conjunto inicial con j vértices.
2. Se considera el subgrafo $W = V \setminus \{\text{vértices seleccionados del conjunto}\}$.
3. Mientras $W \neq \emptyset$
    a. Se considera un vértice $v_i \in W$ según un determinado método (se explican a continuación los más usados).
    b. Se inserta dicho vértice v_i de manera que el coste del circuito se incremente lo menos posible.
    c. Se considera $W = W \setminus \{i\}$.

Existen diferentes métodos de inserción según el criterio usado para añadir los nodos. Estos fueron descritos por Robacker y son los siguientes:

Inserción más cercana: Consiste en elegir la ciudad o vértice v_i más cercana a las ciudades del circuito actual, esto es, si W es el circuito actual,
$d_{min}(v_i) = min\{d_{min}(v_j) : v_j \in W\}$.

Inserción más lejana: Consiste en elegir la ciudad o vértice v_i más alejada de las ciudades del circuito actual, esto es, si W es el circuito actual,
$d_{min}(v_i) = max\{d_{min}(v_j) : v_j \in W\}$.

Inserción aleatoria: Consiste en elegir la ciudad o vértice v_i al azar, esto es, sin seguir ningún criterio.

Inserción más barata: Consiste en elegir la ciudad o vértice v_i que produce el menor incremento de coste posible, esto es, que mantiene el circuito existente lo más corto posible.

A continuación, en la Figura 1.7 se muestran las diferentes elecciones de nodos según el método de inserción más lejano, más cercano y más barato para insertar al ciclo de 4 vértices actual. Para el caso de la inserción más lejana habría que añadir el nodo j al circuito actual, en el caso más cercano el nodo k y en el caso más barato el nodo i, pues con él se obtiene el circuito menor posible a partir del actual.

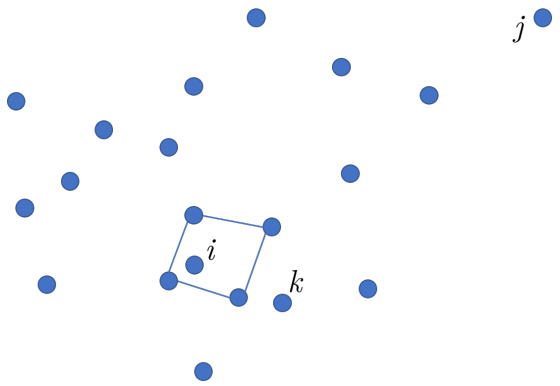


Figura 1.7: Elecciones según el método elegido

Heurística de Christofides: Fue creada por Christofides en 1976 (William J.Cook [1, pág. 72]) y está muy relacionada con los árboles de coste mínimo, por ello, antes de explicar el método es necesario entender este tipo de árboles.

Un árbol de coste mínimo de un grafo es un subgrafo que es un árbol y, además, contiene todos los vértices del grafo inicial con el mínimo coste posible. La búsqueda de este tipo de árboles es un problema que se puede resolver en tiempo polinomial mediante algoritmos eficientes a diferencia de lo que ocurre con el TSP y, es útil, ya que la solución de este tipo de problemas proporciona una cota del coste de la solución óptima para el TSP que es mínima.

Esto se debe a que si de un circuito que es solución se elimina una arista se obtiene un árbol, el cual, posee un único camino de unión entre las distintas ciudades. Por ello, como la solución óptima debe tener una arista más que el árbol anterior, ya que debe ser un circuito cerrado, el coste de dicha solución óptima va a ser necesariamente mayor que el del árbol de mínimo coste.

El método de Christofides es un algoritmo que busca soluciones aproximadas a la óptima, de manera que si el coste de la solución aproximada es x y el de la solución óptima es y, entonces $x \leq \frac{3}{2}y$. Comienza buscando el árbol de mínimo coste L de un grafo G completo y etiquetado. Tras ello se elige el conjunto de vértices de grado impar del árbol L (el grado de un vértice es el número de aristas que inciden en él) y se halla un apareamiento perfecto M (conjunto de aristas sin vértices en común) de mínimo peso en G sobre dichos vértices considerados. Tras ello se forma un multigrafo (en él dos nodos pueden estar conectados por más de una arista) mediante la combinación de las aristas de M y L. Finalmente se obtiene un circuito euleriano en dicho multigrafo y, quitando los nodos ya visitados, se obtiene el circuito hamiltoniano buscado.

A continuación, en la Figura 1.8 se muestra un ejemplo de lo anteriormente comentado. El primer dibujo se corresponde con el grafo completo G del cual se quiere hallar la ruta óptima. Para ello, primero se busca el árbol de mínimo coste que se muestra en el segundo dibujo, y después, tras encontrar el par de vértices de grado impar se forma el emparejamiento M a partir del grafo G sobre esos vértices. Finalmente el último dibujo se corresponde con el multigrafo unión de L y M que como resulta ser un circuito hamiltoniano de mínimo coste termina la búsqueda.

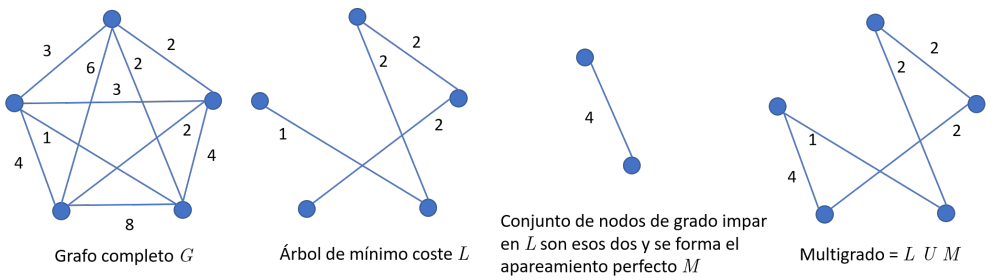


Figura 1.8: Ejemplo heurística de Christofides

1.6.2. Heurísticas de mejora

Se trata de una serie de métodos que buscan una solución factible y, para ello, siguen diversas técnicas. Se distingue entre aquellos que se basan en realizar diversos intercambios y otros que se basan en el azar o aleatoriedad.

1.6.2.1. Heurísticas de mejora *k-opt* o *Lin-Kernighan*

Son procedimientos que se conocen gracias a las primeras definiciones desarrolladas por Flood, los cuales, consisten en intercambiar diversas aristas de una solución inicial de partida buscando mejorarla y conseguir una nueva solución más próxima a la óptima. Por lo tanto, dichos métodos se basan en realizar k-intercambios de aristas e ir generando rutas k-óptimas hasta que no sea posible mejorarlas más. Para entenderlos es necesario describir unos conceptos previos, que se detallan a continuación.

El proceso de realizar un k-intercambio de aristas en una ruta inicial dada consiste en eliminar exactamente k aristas de dicha ruta y reemplazarlas por otras k aristas diferentes de manera que la nueva ruta obtenida sea mejor que la anterior, esto es, de menor coste. En ese caso, dicha ruta se conoce como k-óptima.

La complejidad de este tipo de métodos es $\mathcal{O}(n^k)$ (siendo n el número de nodos), ya que en cada paso, el número de posibles elecciones es $\binom{n}{k}$. Sin embargo, aunque a mayor valor de k mejores soluciones se esperan obtener, el número de operaciones a realizar crece mucho. Por ello, lo más usual es usar un valor de k no mayor que 3, pues en otro caso el coste temporal sería muy grande, no siendo recomendable.
Para que resulte más claro dicho procedimiento se describe el caso en que $k = 2$. El proceso comienza con un ciclo hamiltoniano inicial y con el valor de la variable mejora $= 1$. Tras ello, mientras mejora valga 1, esto es, se encuentren soluciones mejores, se establece mejora a 0 y se van seleccionando los vértices que no han sido explorados. Para cada uno de ellos se realizan todos los posibles movimientos de dos intercambios que incluyan a dicho vértice y uno sucesor. Si alguno de dichos intercambios reduce la distancia actual, se elige el mejor de ellos y mejora pasa a valer 1. Tras ello, dicho vértice se considera explorado y se sigue con el resto hasta que mejora vale 0 pues, en ese caso, ningún intercambio mejora la distancia actual. El coste computacional en cada paso no es grande siendo del orden de $\mathcal{O}(n^2)$.

A continuación, se muestra un ejemplo sencillo de este método.

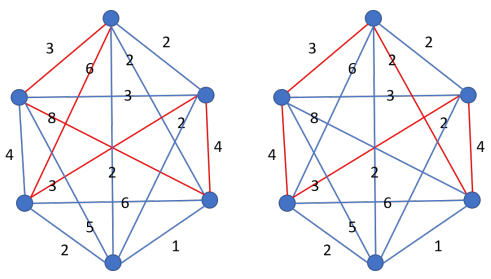


Figura 1.9: Ejemplo heurística *2-opt*

Se puede observar que se basa en la idea de eliminar "cruces" entre aristas, aunque en la práctica, esto es difícil de visualizar. En el ejemplo de la Figura 1.9 se puede ver en el primer dibujo que las aristas en color rojo forman la ruta de partida con un coste de 24 unidades, mientras que en el segundo dibujo, al intercambiar las aristas de mayor coste (la de 8 y 6) por otras de menor coste (la de 4 y 2) se obtiene una mejor solución, siendo el coste de esta de 16 unidades.

Una variante de esta heurística se denomina *V-opt* y difiere del *k-opt* en que las aristas que son eliminadas no están fijas, sino que, dicho número aumenta con el número de iteraciones que se hacen.

Dentro de esta heurística destaca el método *Lin-Kernighan*. Se trata de una de las mejores heurísticas que se conocen para resolver el problema del viajante. Consiste en ir intercambiando un número diferente de aristas según resulte más conveniente en cada caso.

1.6.2.2. Heurísticas de mejora aleatorias

Estos métodos usan diversas técnicas para ir generando soluciones o rutas que estén cada vez más próximas a la ruta óptima logrando conseguir buenos resultados en un tiempo reducido.

Algoritmos Genéticos: Son métodos que se basan en simular los fenómenos naturales de evolución. Se parte de una población inicial generada de manera aleatoria (conjunto de nodos al azar) que sigue un proceso con las siguientes etapas:

Selección: Consiste en elegir de la población actual aquellos descendientes que poseen las mejores características. Para ello, existe una función *fitness* que mide la calidad de cada una de las distintas alternativas.

Mezcla: Consiste en el traspaso de información genética entre cromosomas de los padres a los descendientes, lo cual, en el problema del TSP equivale a realizar saltos entre los distintos estados del espacio de búsqueda.
Conviene mencionar que serán usados de manera equivalente los términos estado y solución a lo largo del presente trabajo.

Mutación: Tras el cruce, cada uno de los nuevos individuos generados puede sufrir mutaciones con una determinada probabilidad p, que si es menor que la tasa de mutación (se elige en el rango $[0.001, 0.05]$), entonces se lleva a cabo dicha mutación.

Una vez realizadas estas etapas se eligen las mejores soluciones entre las existentes (las anteriores más las nuevas obtenidas) volviendo a realizar el proceso descrito. Así, se logra obtener diversas soluciones y a medida que aumentan las iteraciones están más próximas a la solución óptima.

El procedimiento seguido en los algoritmos genéticos para buscar soluciones óptimas se muestra en la Figura 1.10 de manera gráfica.

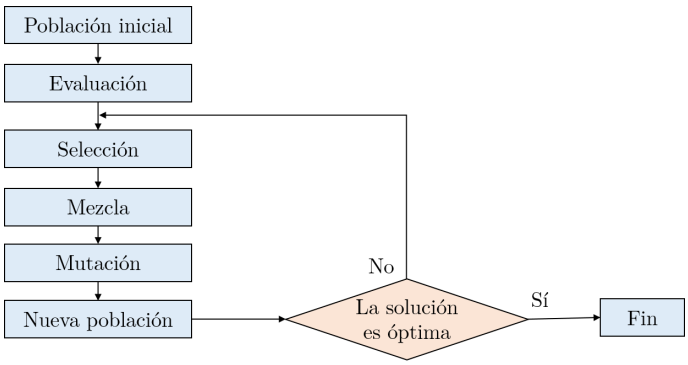


Figura 1.10: Procedimiento algoritmos genéticos

Búsqueda tabú: Es un algoritmo de búsqueda local desarrollado por Fred Glover que trata de evitar que la búsqueda se quede bloqueada en óptimos locales no llegando a encontrar soluciones próximas a la óptima. Para ello, hace uso de estructuras de memoria que pueden ser a corto (denominado *lista tabú*) o largo plazo permitiendo moverse a soluciones que sean peores que la actual para poder escapar de esos óptimos locales.

Colonia de hormigas: Al igual que los algoritmos genéticos se basa en imitar los procesos naturales.

Si se tiene un conjunto de n hormigas que salen del hormiguero, cada una de ellas hace su propio recorrido marcando cuál es el camino seguido mediante el rastro de feronomas que dejan. Como a medida que pasa el tiempo el rastro de feromonas se evapora, los caminos que son más largos tienen menos probabilidades de ser seguidos, pues en ellos, se reduce la fuerza de atracción que mueve a las hormigas en su elección. Este proceso de evaporación es útil ya que permite que se detenga convergiendo en óptimos locales. Por ello, cuando se encuentra un camino que es bueno, esto es, de menor distancia hay más posibilidades de que sea elegido y seguido.

Este hecho fue considerado para aplicarlo a la resolución del TSP buscando encontrar en un grafo completo el camino hamiltoniano de menor coste. En este caso, el agente que se mueve entre las ciudades juega el papel de la hormiga.

Simulated Annealing: Dicho método fue descrito de manera independiente por Scott Kirkpatrick, C. Daniel Gelatt y Mario P. Vecchi, así como por Vlado Černý en los años 80 y está relacionado con el campo de la termodinámica.
Es un método simple y muy usado debido a su gran potencial para buscar soluciones próximas a la óptima en problemas generales de optimización combinatoria.

No consume mucha memoria, produce muy buenos resultados para problemas de tamaño grande y tiene la ventaja de que no se bloquea en óptimos locales, ya que permite considerar soluciones peores a la actual sobretodo en las primeras etapas.

Este trabajo se centra en el estudio de dicho algoritmo por lo que las explicaciones detalladas se muestran en el Capítulo 2.

Capítulo 2

Algoritmo *Simulated Annealing*

Este apartado del trabajo tiene como objetivo dar una descripción del algoritmo *Simulated Annealing*, así como establecer los resultados necesarios para probar su convergencia hacia el conjunto de soluciones óptimas de una función objetivo dada.

Para ello, en la primera sección se establece de manera formal en qué consiste dicho algoritmo. Una vez descrito el algoritmo, en la siguiente sección se comienza introduciendo las definiciones y resultados previos para probar la convergencia del mismo, y posteriormente, se prueba dicha convergencia. Para finalizar el presente capítulo, se consideran varias alternativas de elección de los parámetros que forman parte del algoritmo, de manera que la convergencia se realice en tiempo finito.

2.1. Descripción formal del algoritmo

El algoritmo *Simulated Annealing* es una heurística que tiene como propósito encontrar un valor lo más próximo posible al valor óptimo de una función objetivo determinada. Por lo tanto, se trata de un método de optimización global que tiene la ventaja de resolver problemas de optimización genéricos de manera rápida y eficiente en espacios de estados grandes.

A continuación, se introduce la notación que se va a usar para describir el algoritmo:

$S = \{$soluciones posibles del problema a optimizar$\}$, siendo finito y denominado espacio de soluciones.

f: Es la función objetivo a optimizar, que va del espacio de soluciones posibles a la recta real, esto es,

$$f : S \to \mathbb{R}$$

(S, f): Es un par que simboliza una determinada instancia del problema de optimización combinatoria. Para cada solución o estado $i \in S$ se considera el conjunto $S_i \subseteq S$ como el conjunto de las soluciones próximas a i, esto es, el entorno o estructura de vecinos de i. Se tiene que $j \in S_i \Leftrightarrow i \in S_j$.

$S_{opt} = \{$soluciones óptimas del problema a optimizar$\}$.

f_{opt}: Es el valor de la función objetivo en una solución óptima del problema de optimización. Por ejemplo, para el caso del problema del viajante se debe buscar $i_{opt} \in S_{opt}$ tal que $f(i_{opt}) \leq f(i)$, $\forall\ i \in S$ (pues se trata de un problema de minimización), donde i_{opt} es una solución global óptima de dicho problema.

c_k: Es el valor del parámetro de control en la iteración k o paso k-ésimo.

L_k: Es el número de transiciones o movimientos a realizar en la iteración k o paso k-ésimo.

Definición 2.1. *Una transición es un proceso formado por dos etapas donde, en la primera se aplica un mecanismo para generar una solución o estado sucesor del actual, y tras ello, se aplica el criterio de aceptación (Definición 2.2) para ver si se sigue con la solución o estado actual, o se elige el sucesor.*

Una vez que se tienen los estados, el espacio de soluciones, la función objetivo, etc. es necesario determinar cómo se va a llevar a cabo el proceso de aceptación de los diferentes estados. Dicho proceso sigue el siguiente criterio.

Definición 2.2 (**Criterio de aceptación**). *Sea (S, f) una instancia de un problema de optimización combinatoria y sean $i, j \in S$ dos soluciones con valores o costes asociados $f(i)$, $f(j)$ respectivamente.*
Se define el criterio de aceptación para pasar del estado i al estado j mediante la siguiente probabilidad:

$$\mathbb{P}_c\ (\text{aceptar j}) = \begin{cases} 1 & \text{si } f(j) \leq f(i) \\ exp\left(\frac{f(i)-f(j)}{c}\right) & \text{si } f(j) > f(i) \end{cases}, \tag{2.1}$$

donde $c \in \mathbb{R}^+$ es denominado el parámetro de control.

Es importante señalar que, a la hora de determinar la convergencia o no del algoritmo en las secciones siguientes, el comportamiento de dicho parámetro c va a tener gran importancia.

Una vez que ya se tienen todos los conceptos y criterios anteriores, se describe el proceso seguido para llevar a cabo el algoritmo *Simulated Annealing*.
Dada (S, f) una instancia de un problema de optimización combinatoria, para encontrar un valor próximo al valor óptimo de la función objetivo f, se siguen los siguientes pasos:

1. Se inicializa el estado inicial $i_{inicial}$, el valor inicial del parámetro de control c_0, así como el número de transiciones o movimientos a realizar en la iteración inicial, denotado por L_0.

2. Se fija $k = 0$, $i = i_{inicial}$.

3. Se repite lo siguiente hasta que se cumpla el criterio de parada fijado:

    a. Para $l = 1, ..., L_k$.

a.1. Se genera un nuevo estado $j \in S_i$.

a.2. Si $f(j) \leq f(i) \implies i = j$

Si $f(j) > f(i)$ y $exp\left(\frac{f(i)-f(j)}{c_k}\right) > random[0,1) \implies i = j$

b. $k = k + 1$.

c. Se calcula el valor L_k.

d. Se calcula el valor c_k.

Se tiene que $random[0,1)$ es una variable aleatoria uniforme en $[0,1)$, el paso a.1 es el mecanismo de generación de nuevos estados (se eligen estados dentro del entorno del estado actual) y el paso a.2 es el mecanismo llevado a cabo para aceptar o no dichos estados generados.

El criterio de parada usado para finalizar dicho proceso, puede venir dado fijando un número máximo de iteraciones posibles a realizar, o bien cuando tras un número de iteraciones, no se consigue mejorar el estado o solución actual.

Por lo tanto, dicho algoritmo se trata de una heurística de búsqueda local que permite llevar a cabo movimientos que lleven del estado actual a otros peores durante el inicio del proceso, y a medida que desciende de forma gradual el valor del parámetro de control c, dicha probabilidad se reduce para evitar alejarse del valor óptimo de la función objetivo.

El interés de permitir aceptar estados peores que el actual en las primeras etapas, es lo que permite al algoritmo escapar de óptimos locales, permitiendo así explorar todo el espacio de estados S. Además, si la disminución de c se realiza lentamente, se puede garantizar que el algoritmo encuentra el óptimo global con una probabilidad cercana a 1.

Como consecuencia de ello, supone una mejora con respecto a otros algoritmos de búsqueda local, como es el caso del Hill Climbing. Este algoritmo presenta el inconveniente de quedarse bloqueado en óptimos locales por no permitir ir a soluciones peores que la actual. En la Figura 2.1 se muestra dicho problema, ya que dicho algoritmo, se quedaría bloqueado en el mínimo local sin llegar al global.

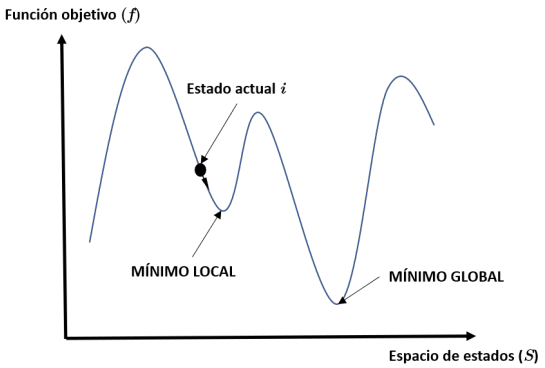


Figura 2.1: Inconveniente de la búsqueda local determinista

2.2. Convergencia asintótica del algoritmo

Como ya se explicó en la sección anterior, el algoritmo *Simulated Annealing*, dado un estado $i \in S$ busca otro estado $j \in S_i$, esto es, dentro del espacio de estados o soluciones próximas a la actual.

Por lo tanto, se puede observar que la generación de un nuevo estado sólo depende del estado anterior. Como las cadenas de Markov sirven para modelizar las transiciones que se llevan a cabo en un determinado sistema de estados, debido a su analogía con el proceso seguido por el *Simulated Annealing*, dicho algoritmo puede describirse matemáticamente usando cadenas de Markov.

Así, en esta sección, se formaliza la convergencia asintótica del algoritmo al espacio de soluciones globales, haciendo uso de cadenas de Markov. Para ello, durante la primera parte se introduce la teoría sobre cadenas de Markov desarrollada por Feller y Seneta, y posteriormente, se estudia bajo qué condiciones se da la convergencia buscada. La convergencia de dicho algoritmo se comenzará probando usando cadenas de Markov homogéneas, y tras ver que en la práctica no es lo más conveniente, se probará usando cadenas de Markov no homogéneas.

Para su desarrollo, se ha usado principalmente los libros de P.J.M. Van Laarhoven y Emile Aarts [3] y Emile Aarts y Jan Korst [4].

2.2.1. Teoría de cadenas de Markov

Definición 2.3. *Un proceso estocástico es una sucesión de observaciones X_1, X_2,... (variables estocásticas) cuyos valores no se pueden predecir exactamente, esto es, son aleatorios, pero sin embargo, sí es posible especificar las probabilidades para los distintos posibles valores en cada instante determinado.*

Definición 2.4. *Una cadena de Markov es un proceso estocástico en el que si se conoce el estado actual X_n y los estados previos X_1, ..., X_{n-1} entonces la probabilidad del estado futuro sólo depende del actual, esto es,*

$$\mathbb{P}\left(X_{n+1} = s_{n+1} \mid X_1 = s_1, ..., X_n = s_n\right) = \mathbb{P}(X_{n+1} = s_{n+1} \mid X_n = s_n), \tag{2.2}$$

siendo s_i el valor en el estado i-ésimo.

Definición 2.5. *Se consideran las siguientes definiciones:*

1. *Un vector $\boldsymbol{v} = (v_1, ..., v_n, ...)$ es estocástico si cumple lo siguiente:*

$$0 \leq v_i \leq 1, \ \ \forall\, i \quad \ y \quad \ \sum_i v_i = 1. \tag{2.3}$$

2. *Una matriz cuadrada, P, será estocástica por filas si cumple lo siguiente:*

$$0 \leq P_{ij} \leq 1, \ \ \forall\, i, j \quad \ y \quad \ \sum_j P_{ij} = 1, \ \ \forall\, i, \tag{2.4}$$

   esto es, los elementos son no negativos y por cada fila suman 1, siendo P_{ij} la probabilidad de transición del estado i al estado j.

A cada cadena de Markov se la puede asignar una única matriz de transición cuya dimensión dependerá del número de estados considerados.
Dicha matriz de transición es una matriz cuadrada y estocástica por filas, luego, suponiendo una cadena de Markov de n estados, la matriz de transición asociada será la siguiente:

$$P = \begin{bmatrix} P_{11} & P_{12} & . & . & P_{1n} \\ P_{21} & P_{22} & . & . & P_{2n} \\ . & . & . & . & . \\ P_{n1} & P_{n2} & . & . & P_{nn} \end{bmatrix} \in \mathcal{M}_{n\times n}, \text{cumpliendo las expresiones dadas en (2.4).}$$

Para el caso del algoritmo *Simulated Annealing*, interesa conocer cual es la probabilidad de transición del estado i al j en el paso k-ésimo que se representa como:

$$P_{ij}(k) = \mathbb{P}(X_k = j \mid X_{k-1} = i). \tag{2.5}$$

Dichos valores determinan la matriz de transición de estados $P(k)$ que es una matriz estocástica.

Notación: Se denota la probabilidad de que ocurra el estado i en el k-ésimo paso por:

$$a_i(k) = \mathbb{P}(X_k = i) = \sum_l \mathbb{P}(X_{k-1} = l) \cdot \mathbb{P}(X_k = i \mid X_{k-1} = l) = \sum_l a_l(k-1)P_{li}(k).$$

Definición 2.6. *Una cadena de Markov puede ser de distintos tipos:*

Finita: *Sólo existe un número finito k de estados posibles (en el caso del algoritmo Simulated Annealing el conjunto finito de posibles soluciones), y en cualquier instante de tiempo, la cadena se encuentra en uno de ellos.*

Homogénea: *Es aquella en la que la probabilidad de ir del estado i al estado j en un determinado paso, no depende del momento del tiempo en el cual se encuentre la cadena, esto es,*

$$\mathbb{P}(X_n = j \mid X_{n-1} = i) = \mathbb{P}(X_1 = j \mid X_0 = i), \quad \forall\, n, i, j. \tag{2.6}$$

No homogénea: *Es aquella en la que ocurre lo contrario que en el caso anterior, esto es, la probabilidad de ir del estado i al estado j en un determinado paso, depende del mismo.*

Definición 2.7. *Se considera $f_{ij}^{(k)}$ la probabilidad de que empezando en el estado i se vaya por primera vez al estado j en el k-ésimo paso, esto es,*

$$f_{ij}^{(k)} = \mathbb{P}(X_k = j, X_{k-1} \neq j, ..., X_1 \neq j \mid X_0 = i).$$

Por convención se considera que, $f_{ij}^{(0)} = 0$.
Además, se tiene que $(P^k)_{ij} = \sum_{m=1}^{k} f_{ij}^{(m)}(P^{k-m})_{jj}$, lo cual se corresponde con la probabilidad de volver al estado j en el paso k-ésimo, esto es, P^k denota la matriz

de transición tras k pasos. Sus componentes son $(P^k)_{ij} = \mathbb{P}(X_k = j \mid X_0 = i)$. *También se considera por convención que:*

$$(P^0)_{ij} = \begin{cases} 0 & \text{si } i \neq j \\ 1 & \text{si } i = j \end{cases}$$

Definición 2.8. *Se define* $f_{ij} = \sum_{k=1}^{\infty} f_{ij}^{(k)}$ *y* $\mu_j = \sum_{k=1}^{\infty} k f_{jj}^{(k)}$, *donde* $f_{ij} \leq 1$ *es la probabilidad de que el sistema vaya del estado i al estado j por lo menos una vez y* μ_j *es el tiempo medio de recurrencia del estado j.*

Nota: Se observa que la definición de μ_j sólo tiene sentido si $f_{jj} = 1$.

Definición 2.9. *Se considera la siguiente clasificación de estados:*

Un estado j es persistente si $f_{jj} = 1$, *esto es, si después de haber llegado al estado j, el proceso definitivamente regresará a dicho estado. Por lo tanto, la probabilidad de volver al estado j habiendo empezado en dicho estado es 1.*

Un estado j es transitorio si $f_{jj} < 1$, *esto es, si después de haber alcanzado dicho estado el proceso nunca regresa a él, o lo que es lo mismo, sólo es visitado un número finito de veces. Por lo tanto, la probabilidad de volver al estado j habiendo comenzado en él es menor que 1.*

Un estado persistente j es nulo si $\mu_j = \infty$ *(el número esperado de transiciones para regresar a él es* ∞*) y persistente no nulo si* $\mu_j < \infty$ *(se regresa a dicho estado en un número finito de transiciones).*

Lema 2.10 (**Feller, 1950 [6]**). *A partir de las definiciones anteriores se tienen los siguientes resultados:*

a) Un estado j es transitorio $\iff \sum_{k=0}^{\infty}(P^k)_{jj} < \infty$. *Además, en dicho caso, se tiene que* $\sum_{k=1}^{\infty}(P^k)_{ij} < \infty$, $\forall\, i \in S$.

b) Un estado j es persistente nulo $\iff \sum_{k=0}^{\infty}(P^k)_{jj} = \infty$, *pero* $(P^k)_{jj} \overset{k\to\infty}{\longrightarrow} 0$. *Además, en dicho caso, se tiene que* $(P^k)_{ij} \overset{k\to\infty}{\longrightarrow} 0$, $\forall\, i \in S$.

Como ya se ha comentado, en el algoritmo *Simulated Annealing* se va decrementando el valor del parámetro de control c. En función del tipo de cadena de Markov usada (homogénea o no homogénea) y de cómo se lleve a cabo dicho decremento de c, se puede probar la convergencia asintótica del algoritmo de dos formas diferentes:

1ª forma: Consiste en modelizar el algoritmo usando una secuencia infinita de cadenas de Markov homogéneas, donde cada una de ellas es generada considerando un valor de c fijo que se decrementa al cambiar a otra cadena de la secuencia ($c_1 > ... > c_{k-1} > c_k > ...$, donde c_i es el parámetro de control correspondiente a la cadena i-ésima). Además, conviene mencionar que se va a considerar una cadena de Markov diferente para cada paso del algoritmo, esto es, se asociará a cada paso del mismo una determinada cadena de Markov.

Aunque dicha modelización es sencilla, al no poderse realizar en la práctica, se considera una 2ª forma.

2ª forma: Consiste en modelizar el algoritmo usando una única cadena de Markov no homogénea donde el valor del parámetro de control c se decrementa en cada paso o transición de la misma.

Definición 2.11. *Dada (S, f) una instancia de un problema de optimización combinatoria, las componentes de la matriz de transición de probabilidades $P(c_k)$ para el algoritmo Simulated Annealing vienen dadas por:*

$$P_{ij}(k) = P_{ij}(c_k) = \begin{cases} G_{ij}A_{ij}(c_k) & \text{si } i \neq j \\ 1 - \sum_{l \in S, l \neq i} P_{il}(c_k) & \text{si } i = j \end{cases} \quad \forall\, i, j \in S, \tag{2.7}$$

donde G es la matriz de generación de probabilidades y $A(c_k)$ la matriz de aceptación.

Las componentes de la matriz de G vienen definidas como:

$$G_{ij} = \frac{1}{\Theta}\chi_{S_i}(j),\ \forall\, i, j \in S,\ \ \textit{siendo}\ \ \chi_{S_i}(j) = \begin{cases} 1 & \text{si } j \in S_i \\ 0 & \text{si } j \notin S_i \end{cases} \tag{2.8}$$

y $\Theta = |S_i|,\ \forall\, i \in S$, esto es, los elementos de la matriz de generación de estados son independientes del parámetro de control c_k y uniformes sobre los elementos de su entorno. Además se supone que para todos ellos, el entorno considerado es del mismo tamaño, por ello, $\Theta = |S_i|,\ \forall\, i \in S$.

Por otro lado las componentes de $A(c_k)$ vienen definidas por:

$$A_{ij}(c_k) = exp\left(\frac{-(f(j) - f(i))^+}{c_k}\right),\ \forall\, i, j \in S, \tag{2.9}$$

donde $\forall\, a \in \mathbb{R},\ a^+ = a$ si $a > 0$ y 0 en otro caso.

Además, de (2.7), (2.8) y (2.9) se obtiene que, tanto la matriz de probabilidades $P(c_k)$ como la matriz de generación de probabilidades G son matrices estocásticas, mientras que la matriz de aceptación de probabilidades $A(c_k)$ no lo es.

El objetivo perseguido, es probar la convergencia asintótica del algoritmo hacia el conjunto de soluciones óptimas S_{opt} de una función objetivo dada f (bajo un cierto comportamiento asintótico del parámetro de control c_k), esto es:

$$\lim_{k \to \infty} \mathbb{P}(X_k \in S_{opt}) = 1. \tag{2.10}$$

2.2.2. Convergencia usando cadenas homogéneas

Como ya se comentó en el apartado 2.2.1, el algoritmo *Simulated Annealing* puede modelarse de manera matemática usando la teoría de cadenas de Markov.
Este apartado se centra en probar su convergencia modelizándolo mediante una secuencia infinita de cadenas de Markov homogéneas.

Considerando dicha modelización, se obtendrá la expresión dada en (2.10), esto es, la convergencia asintótica del algoritmo. Para probar dicha convergencia, se verá que es muy importante, la existencia de una distribución estacionaria única para cadena de Markov considerada en la representación del comportamiento de dicho algoritmo. Además, se tendrá que para que exista, deben cumplirse una serie de condiciones: cada cadena de Markov de la secuencia considerada ha de tener longitud infinita, las matrices de generación de probabilidades y de aceptación definidas en (2.8) y (2.9) deben satisfacer ciertas condiciones y $\lim_{k\to\infty} c_k = 0$, siendo c_k el valor del parámetro de control correspondiente a la cadena k-ésima de Markov de la secuencia considerada, esto es, la correspondiente al paso k-ésimo del algoritmo.

Como las cadenas de Markov consideradas son homogéneas, el valor del parámetro de control para cada una de ellas, tiene que ser constante.
Por lo tanto, al ser constante para cada cadena de Markov de la secuencia considerada, se tiene que, por ejemplo para la cadena k-ésima, $c_k = c$ y que además, la matriz de transición asociada a la misma será $P = P(c)$ cuyas componentes vendrán dadas por:

$$P_{ij}(c) = \begin{cases} G_{ij}A_{ij}(c) & \text{si } i \neq j \\ 1 - \sum_{l\in S, l\neq i} P_{il}(c) & \text{si } i = j \end{cases} \quad \forall\, i, j \in S.$$

A continuación, se realiza dicho estudio. Primero se establecen una serie de definiciones y resultados previos, y posteriormente, se prueba dicha convergencia.

Definición 2.12 (**Feller, 1950 [6]**). *La distribución estacionaria correspondiente a una cadena de Markov finita y homogénea con matriz de transición P viene dada por el vector $\boldsymbol{q}$ cuya componente i-ésima es de la forma:*

$$q_i = \lim_{k\to\infty} \mathbb{P}(X_k = i \mid X_0 = j), \;\; \forall\, j \in S. \tag{2.11}$$

Por lo tanto, si existe dicha distribución estacionaria $\boldsymbol{q}$, se corresponderá con la distribución de probabilidades sobre el espacio de soluciones S tras un número infinito de pasos (transiciones), ya que:

$$\begin{aligned} \lim_{k\to\infty} a_i(k) &= \lim_{k\to\infty} \mathbb{P}(X_k = i) = \lim_{k\to\infty} \sum_{j\in S} \mathbb{P}(X_k = i \mid X_0 = j) \cdot \mathbb{P}(X_0 = j) \\ &= q_i \sum_{j\in S} \mathbb{P}(X_0 = j) = q_i. \end{aligned}$$

Además, considerando el vector inicial de la distribución de probabilidades
$\mathbf{a}(0) = (\mathbf{a}_i(0))$ donde $i \in S$, se tiene que $\boldsymbol{q}$ es un autovector por la izquierda de la

matriz de transición P con autovalor asociado 1:

$$\begin{aligned} \boldsymbol{q}^T &= \lim_{k\to\infty} \mathbf{a}(0)^T \prod_{n=1}^{k} P(n) = \lim_{k\to\infty} \mathbf{a}(0)^T P^k = \lim_{k\to\infty} \mathbf{a}(0)^T P^{k-1} P \\ &= \lim_{n\to\infty} \mathbf{a}(0)^T P^n P = \boldsymbol{q}^T P. \end{aligned} \tag{2.12}$$

En la última igualdad de (2.12) se ha usado la propiedad de que la distribución de probabilidades inicial $\mathbf{a}(0)$ y las potencias de la matriz de transición P determinan la distribución de las variables de la cadena de Markov. Además, este hecho prueba que $\boldsymbol{q}$ es el vector asociado a la distribución estacionaria para la cadena de Markov.

Para el caso del algoritmo *Simulated Annealing* dado que la matriz P depende de c, $\boldsymbol{q}$ también lo hará, esto es, se tendrá que $\boldsymbol{q} = \boldsymbol{q}(c)$. Ahora, se muestran una serie de resultados que son necesarios para poder probar la existencia de una distribución estacionaria para cadena de Markov homogénea considerada, y con ello, poder concluir la convergencia asintótica del *Simulated Annealing*.

Definición 2.13. *Una cadena de Markov con matriz de transición P es irreducible si para cada par de estados $i, j \in S$ hay una probabilidad positiva de ir de i a j en un número finito de pasos, esto es,*

$$\forall\, i, j \in S, \exists\, n:\ 1 \leq n < \infty \text{ tal que } (P^n)_{ij} > 0. \tag{2.13}$$

Definición 2.14. *Una cadena de Markov con matriz de transición P es aperiódica si todos sus estados son periódicos de periodo 1, esto es,*

$$\forall\, i \in S,\ mcd(D_i) = 1 \text{ donde } D_i = \{n \in \mathbb{N}^+ : (P^n)_{ii} > 0\}, \tag{2.14}$$

siendo $mcd(D_i)$ el máximo común divisor de D_i que se conoce como periodo de la solución o estado i.

Lema 2.15. *Una cadena de Markov irreducible con matriz de transición P es aperiódica si $\exists\, j \in S : P_{jj} > 0$.*

Demostración: Se sabe por la Definición 2.13 que:

$$\forall\, i, j \in S, \exists\, k, l:\ 1 \leq k, l < \infty \text{ tal que } (P^k)_{ij} > 0,\ (P^l)_{ji} > 0.$$

Por lo tanto, si $n = k + l$, $(P^n)_{ii} \geq (P^k)_{ij}(P^l)_{ji} > 0$ y $(P^{n+1})_{ii} \geq (P^k)_{ij} P_{jj} (P^l)_{ji} > 0$ ya que por hipótesis $\exists\, j \in S : P_{jj} > 0$. Entonces se tiene que $n, n+1 \in D_i$ y, como $1 \leq mcd(D_i) \leq mcd(n, n+1) = 1 \implies mcd(D_i) = 1$. □

Lema 2.16. *Dado un estado j con periodo d entonces d es el máximo común divisor del conjunto de enteros n para los que $f_{jj}^{(n)} > 0$.*

Demostración: Dado que $(P^s)_{jj} = \sum_{m=1}^{s} f_{jj}^{(m)} (P^{s-m})_{jj}$ es claro que si s es el entero más pequeño para el que $f_{jj}^{(s)} > 0$, entonces también es el entero más pequeño para el que $(P^s)_{jj} > 0$. Sea d_N' el máximo común divisor de los enteros positivos $s \leq n \leq N$

para los que $f_{jj}^{(n)} > 0$ y $d_N^{''}$ el máximo común divisor para los que $(P^n)_{jj} > 0$.

Se prueba por inducción sobre N:
Es claro que $f_{jj}^{(s)} = (P^s)_{jj}$ (debido a como fue definido s), por lo que $d_s^{'} = d_s^{''}$.
Ahora, se supone que $d_N^{'} = d_N^{''}$ y se debe ver que $d_{N+1}^{'} = d_{N+1}^{''}$:

$$(P^{N+1})_{jj} = \sum_{m=1}^{N+1} f_{jj}^{(m)}(P^{N+1-m})_{jj} = \sum_{m=1}^{N} f_{jj}^{(m)}(P^{N+1-m})_{jj} + f_{jj}^{(N+1)}$$

- Si $f_{jj}^{(N+1)}$, $(P^{N+1})_{jj} > 0 \implies d_{N+1}^{'} = d_{N+1}^{''}$.
- Si $f_{jj}^{(N+1)}$, $(P^{N+1})_{jj} = 0 \implies d_{N+1}^{'} = d_{N+1}^{''}$
- Si $f_{jj}^{(N+1)} = 0$ y $(P^{N+1})_{jj} > 0 \implies \exists\ m,\ 1 \leq m \leq N : f_{jj}^{(m)}(P^{N+1-m})_{jj} > 0$, luego $d_N^{'}$ divide a m y $d_N^{''}$ divide a $N+1-m$. Así que, tanto $d_N^{'}$ como $d_N^{''}$ dividen a $N+1$, ya que $d_N^{'} = d_N^{''}$. Como consecuencia, $d_{N+1}^{'} = d_N^{'} = d_N^{''} = d_{N+1}^{''}$.

□

Lema 2.17. *Dado un número finito de enteros positivos n_s con $1 \leq s \leq t$ y d el máximo común divisor de ellos, entonces existe un N tal que $\forall\ n \geq N$ existen enteros positivos c_s con $1 \leq s \leq t$ satisfaciendo que:*

$$nd = \sum_{s=1}^{t} c_s n_s.$$

Demostración: Fijados los elementos n_s con $1 \leq s \leq t$, entonces el conjunto de elementos $c_1n_1 + ... + c_tn_t$ con $c_s \geq 0,\ \forall\ s$ es un semigrupo generado por dichos elementos, el cual se denota por $< n_1, ..., n_t >$.
Además, será un semigrupo numérico o de complemento finito si su complementario en $\mathbb{N}$ es finito, esto es, si $\exists\ N : \forall\ n \geq N,\ n \in < n_1, ..., n_t >$.
Por el Lema 2.1 del libro de Rosales y García-Sánchez [12], se sabe que $< n_1, ..., n_t >$ es un semigrupo numérico si y solo si $mcd(\{n_1, ..., n_t\}) = 1$.
Por lo tanto, considerando $m_s = \frac{n_s}{d}$ y $< m_1, ..., m_t >$ semigrupo numérico, se sabe que $\exists\ n \geq N : n \in < m_1, ..., m_t >$. Luego, existen $c_1, ..., c_t$ con $c_s \geq 0,\ \forall\ s$ tal que $n = c_1m_1 + ... + c_sm_s = \frac{1}{d}(c_1n_1 + ... + c_tn_t) \implies nd = \sum_{s=1}^{t} c_sn_s$. □

Lema 2.18 (**Kai Lai Chung [11]**). *Si el estado j es persistente no nulo y aperiódico, $(P^k)_{jj} \overset{k\to\infty}{\longrightarrow} \frac{1}{\mu_j}$.*

Demostración: Sea $\forall\ k \geq 0,\ r_k = \sum_{l=k+1}^{\infty} f_{jj}^{(l)}$. Luego, por ser j un estado persistente es claro que $r_0 = \sum_{l=1}^{\infty} f_{jj}^{(l)} = 1$ y $\sum_{k=0}^{\infty} r_k = \sum_{l=1}^{\infty} l f_{jj}^{(l)} = \mu_j$.
Como $r_l - r_{l-1} = -f_{jj}^{(l)}$ se tiene que:

$$(P^k)_{jj} = \sum_{l=1}^{k} f_{jj}^{(l)}(P^{k-l})_{jj} = -\sum_{l=1}^{k}(r_l - r_{l-1})(P^{k-l})_{jj}$$

$$\implies \sum_{l=0}^{k} r_l(P^{k-l})_{jj} = \sum_{l=0}^{k-1} r_l(P^{k-1-l})_{jj}$$

En consecuencia, $\forall\, k \geq 0$, la suma anterior vale siempre lo mismo con independencia del número de sumandos. Además, como $r_0(P^0)_{jj} = 1$,

$$\sum_{l=0}^{k=0} r_l(P^{k-l})_{jj} = r_0(P^0)_{jj} = 1.$$

Por lo tanto, al ser dicha suma independiente de k y valer 1 para $k = 0$, se deduce que:

$$\forall\, k \geq 0\ , \quad \sum_{l=0}^{k} r_l(P^{k-l})_{jj} = 1. \tag{2.15}$$

Dado $\lambda = \lim\sup_{k\to\infty}(P^k)_{jj}$, existe una subsucesión $\{k_n\}$ de manera que $\lim_{n\to\infty}(P^{k_n})_{jj} = \lambda$. Sea s tal que $f_{jj}^{(s)} > 0$:

$$\begin{aligned}
\lambda = \liminf_{n\to\infty}(P^{k_n})_{jj} &= \liminf_{n\to\infty}\left(f_{jj}^{(s)}(P^{k_n-s})_{jj} + \sum_{l=1,l\neq s}^{k_n} f_{jj}^{(l)}(P^{k_n-l})_{jj}\right) \\
&\leq f_{jj}^{(s)} \liminf_{n\to\infty}(P^{k_n-s})_{jj} + \sum_{l=1,l\neq s}^{\infty} f_{jj}^{(l)} \limsup_{n\to\infty}(P^{k_n-l})_{jj} \\
&\leq f_{jj}^{(s)} \liminf_{n\to\infty}(P^{k_n-s})_{jj} + (1 - f_{jj}^{(s)})\lambda
\end{aligned}$$

Luego $\lim\inf_{n\to\infty}(P^{k_n-s})_{jj} \geq \lambda$, y por la propia definición de λ se tiene que $\lim_{n\to\infty}(P^{k_n-s})_{jj} = \lambda$.
Este resultado es cierto $\forall\ s$ tal que $f_{jj}^{(s)} > 0$ y cada subsucesión $\{k_n\}$ para la que $\lim_{n\to\infty}(P^{k_n})_{jj} = \lambda$. Por ello, aplicando el resultado anterior un número finito de veces, se concluye que:

$$\lim_{n\to\infty}(P^{k_n-t})_{jj} = \lambda,\ \forall\ t \text{ de la forma} \sum_{m=1}^{l} c_m s_m,\ \text{donde } c_m, s_m \in \mathbb{N}^+$$
$$\text{y tal que } f_{jj}^{(s_m)} > 0,\ 1 \leq m \leq l.$$

Por el Lema 2.16, existen s_m con $1 \leq m \leq l$ y $f_{jj}^{(s_m)} > 0$ tal que su máximo común divisor es $d = 1$ ya que el estado j es aperiódico.

Por el Lema 2.17 de *Teoría de Números*, $\exists\ s_0 : \forall\ s \geq s_0,\ \exists\ c_m \geq 0$ con $1 \leq m \leq l$ de manera que $s = \sum_{m=1}^{l} c_m s_m$. Luego $\forall\ s \geq s_0$, $\lim_{n\to\infty}(P^{k_n-s})_{jj} = \lambda$.
Si se toma $k = k_n - s_0$ en (2.15), se obtiene que:

$$\sum_{l=0}^{k_n-s_0} r_l(P^{k_n-s_0-l})_{jj} = 1 \overset{n\to\infty}{\longrightarrow} \lambda\sum_{l=0}^{\infty} r_l = 1 \implies \lambda = \frac{1}{\sum_{l=0}^{\infty} r_l} = \frac{1}{\mu_j}$$

Luego se puede concluir que $\lim\sup_{k\to\infty}(P^k)_{jj} = \frac{1}{\mu_j}$ y por analogía se tiene que $\lim\inf_{k\to\infty}(P^k)_{jj} = \frac{1}{\mu_j}$. Como consecuencia, $\lim_{k\to\infty}(P^k)_{jj} = \frac{1}{\mu_j}$.

□

Lema 2.19. *Todos los estados de una cadena de Markov irreducible son del mismo tipo.*

Demostración: Sean j, k dos estados arbitrarios de una cadena de Markov irreducible. Por ser dicha cadena irreducible se sabe que existen $r, s \in \mathbb{N}^+$ tal que $(P^r)_{jk} = \alpha > 0$ y $(P^s)_{kj} = \beta > 0$. Luego se tiene que:

$$(P^{n+r+s})_{jj} \geq (P^r)_{jk}(P^n)_{kk}(P^s)_{kj} = \alpha\beta(P^n)_{kk},\ j, k, r, s \text{ fijos y } n \text{ arbitrario.}$$

Si j es transitorio, por el Lema 2.10 se sabe que $\sum_{n=0}^{\infty}(P^{n+r+s})_{jj} < \infty$, luego es una serie convergente, y por lo tanto, $\sum_{n=0}^{\infty}(P^n)_{kk} < \infty$, esto es, k es transitorio.

Si j es persistente nulo, por el Lema 2.10 se sabe que $(P^n)_{jj} \overset{\text{n}\to\infty}{\longrightarrow} 0$, y también se tendrá que $(P^n)_{kk} \overset{\text{n}\to\infty}{\longrightarrow} 0$, luego k será persistente nulo. Igual para el caso no nulo.

Si j es aperiódico, entonces su periodo es 1, esto es, $mcd(D_j) = 1$. En consecuencia, $P_{jj} > 0$, y por el Lema 2.15, dicha cadena de Markov es aperiódica, luego todos sus estados son aperiódicos, y en particular, k lo es.

Todo esto es igualmente válido si se intercambian los papeles de j y k. □

Lema 2.20. *Dado un estado j persistente, existe un único conjunto cerrado irreducible C que contiene a j y tal que para cada par de estados $i, k \in C, f_{ik} = 1$ y $f_{ki} = 1$.*

Demostración: Sea k un estado alcanzable desde j y sea α la probabilidad de llegar a k desde j, sin poder ir a j desde k. Luego, una vez alcanzado el estado k, la probabilidad de no volver nunca a j es $1 - f_{kj}$. Por lo tanto, la probabilidad de que empezando en el estado j nunca se regrese a dicho estado es al menos $\alpha(1 - f_{kj})$. Dado que j es un estado persistente, la probabilidad de no retorno es 0, por lo que $f_{kj} = 1$, $\forall$ estado k alcanzable desde j.

Sea C el conjunto de todos los estados a los cuales se puede llegar desde el estado j e $i, k \in C$. Al igual que vimos que se puede llegar a j desde k, también es posible llegar a i desde k. Por lo tanto, cada estado en C puede ser alcanzado desde otro estado en C, esto es, C es irreducible. Como j es persistente todos los estados en C lo son (Lema 2.19), luego cambiando el papel de j por i en el argumento del principio se tiene que $f_{ki} = 1$, $\forall$ estado k alcanzable desde i. □

Como consecuencia del lema anterior, deducimos que ningún estado transitorio puede alcanzarse desde un estado persistente.

Lema 2.21. *Si una cadena de Markov es finita entonces no existen estados nulos y es imposible que todos sus estados sean transitorios.*

Demostración: Dada la matriz de transición P^k, se sabe que es estocástica por filas, luego es imposible que $(P^k)_{ij} \overset{\text{k}\to\infty}{\longrightarrow} 0$, $\forall\ i, j \in S$ (condición necesaria de convergencia) ya que, por ser finita, tiene un número fijo de elementos. Por lo tanto, todos los estados no pueden ser transitorios.

Por otro lado un estado persistente va a pertenecer a un conjunto C irreducible (Lema 2.20), y por lo tanto, todos sus estados han de ser del mismo tipo (Lema 2.19). Como C tiene al menos un estado persistente no nulo, no puede tener ningún estado nulo. □

Por lo tanto, del lema anterior se deduce que todos los estados de una cadena de Markov finita son persistentes no nulos.

Teorema 2.22 (**Feller, 1950 [6]**). *Sea P la matriz de transición de una cadena de Markov finita y homogénea. Entonces existe la distribución estacionaria $\boldsymbol{q}$ asociada a dicha cadena si esta es irreducible y aperiódica. Además sus componentes están definidas de manera única por:*

$$q_i = \sum_j q_j P_{ji}, \quad \forall\, i \in S \ \ (\textit{Ecuación de balance general}), \tag{2.16}$$

donde $\sum_{i\in S} q_i = 1$ con $q_i \geq 0$.

Nota: Se puede observar que dicho vector $\boldsymbol{q}$ cumple la expresión dada en (2.12).

Demostración: Para poder realizar la demostración lo primero que hay que ver es que, por ser la cadena de Markov irreducible y aperiódica, dado $i \in S$, $\forall\ j \in S$ existe $q_i = \lim_{k\to\infty} \mathbb{P}(X_k = i \mid X_0 = j)$.

Por ser la cadena de Markov finita, del Lema 2.21, se puede considerar un estado j persistente no nulo, tal que $f_{ij} = \sum_{m=1}^{\infty} f_{ij}^{(m)} = 1$ (Lema 2.20).
Además como $(P^k)_{ij} = \sum_{m=1}^{k} f_{ij}^{(m)} (P^{k-m})_{jj}$ se tiene que,

$$(P^k)_{ij} \leq \sum_{m=1}^{N} f_{ij}^{(m)} (P^{k-m})_{jj} + \sum_{m=N+1}^{k} f_{ij}^{(m)} \leq \sum_{m=1}^{N} f_{ij}^{(m)} (P^{k-m})_{jj} + \sum_{m=N+1}^{\infty} f_{ij}^{(m)}$$

Por lo tanto, sabiendo por el Lema 2.18 que $\lim_{k\to\infty} (P^k)_{jj} = \frac{1}{\mu_j}$,

$$\limsup_{k\to\infty} (P^k)_{ij} \leq \frac{1}{\mu_j} \sum_{m=1}^{N} f_{ij}^{(m)} + \sum_{m=N+1}^{\infty} f_{ij}^{(m)}, \text{ cuando } N \to \infty, \ \limsup_{k\to\infty} (P^k)_{ij} \leq \frac{1}{\mu_j}$$

Por otro lado, se toma $N < k$ tal que $\sum_{k=1}^{N} f_{ij}^{(k)} > 1 - \epsilon$. Entonces,

$$(P^k)_{ij} \geq \sum_{m=1}^{N} f_{ij}^{(m)} (P^{k-m})_{jj} \geq \min_{k-N\leq m\leq k-1} (P^m)_{jj} (1-\epsilon) \implies \liminf_{k\to\infty} (P^k)_{ij} \geq \frac{1}{\mu_j}$$

Finalmente se tiene que:

$$\limsup_{k\to\infty} (P^k)_{ij} \leq \frac{1}{\mu_j} \leq \liminf_{k\to\infty} (P^k)_{ij} \implies q_i = \lim_{k\to\infty} (P^k)_{ij} = \frac{1}{\mu_j} < \infty$$

por ser j persistente no nulo.

Tras ello queda probar que la distribución $\boldsymbol{q}$ con componentes q_i es estacionaria, así como la unicidad de la misma. Se realiza en varias partes:

Primero se debe ver que $\sum_{i\in S} q_i = 1$: Sea $j \in S$ fijo, se sabe que para todo $k \in \mathbb{N}$

$$\sum_{i\in S} \mathbb{P}(X_k = i \mid X_0 = j) = 1 \implies \lim_{k\to\infty} \left(\sum_{i\in S} \mathbb{P}(X_k = i \mid X_0 = j) \right) = 1$$

Por ser S un conjunto finito, se tiene que:

$$1 = \lim_{k\to\infty} \left(\sum_{i\in S} \mathbb{P}(X_k = i \mid X_0 = j) \right) = \sum_{i\in S} \Big(\lim_{k\to\infty} \mathbb{P}(X_k = i \mid X_0 = j) \Big) = \sum_{i\in S} q_i$$

Ahora se debe ver que $q_i = \sum_j q_j P_{ji},\ \forall\, i \in S$: Sea $i \in S$ y $l \in S$ un estado arbitrario, se tiene que:

$$\begin{aligned}
\sum_{j\in S} q_j P_{ji} &= \sum_{j\in S} \Big(\lim_{k\to\infty} \mathbb{P}(X_k = j \mid X_0 = l) \cdot \mathbb{P}(X_1 = i \mid X_0 = j) \Big) \\
&= \lim_{k\to\infty} \left(\sum_{j\in S} \mathbb{P}(X_k = j \mid X_0 = l) \cdot \mathbb{P}(X_1 = i \mid X_0 = j) \right) \\
&= \lim_{k\to\infty} \left(\sum_{j\in S} \mathbb{P}(X_k = j \mid X_0 = l) \cdot \mathbb{P}(X_{k+1} = i \mid X_k = j) \right) \\
&= \lim_{k\to\infty} \left(\sum_{j\in S} \mathbb{P}(X_{k+1} = i \mid X_k = j) \cdot \mathbb{P}(X_k = j \mid X_0 = l) \right) \\
&= \lim_{k\to\infty} \mathbb{P}(X_{k+1} = i \mid X_0 = l) = \lim_{m\to\infty} \mathbb{P}(X_m = i \mid X_0 = l) = q_i
\end{aligned}$$

Finalmente falta probar la unicidad de la distribución: Para ello se supone que existe otra distribución estacionaria $\widetilde{\boldsymbol{q}}$ y se debe ver que $\widetilde{\boldsymbol{q}} = \boldsymbol{q}$. Para ello bastará ver que $\forall\, i \in S,\ q_i = \widetilde{q}_i$. Sea $i \in S$, luego:

$$\widetilde{q}_i = \sum_{j\in S} \widetilde{q}_j P_{ji} = \sum_{j\in S} \left(\sum_{l\in S} \widetilde{q}_l P_{lj} \right) P_{ji} = \sum_{l\in S} \widetilde{q}_l \left(\sum_{j\in S} P_{lj} P_{ji} \right) = \sum_{l\in S} \widetilde{q}_l P_{li}^2$$

Por lo tanto, $\forall\, k \geq 1,\ \widetilde{q}_i = \sum_{j\in S} \widetilde{q}_j P_{ji}^k = \sum_{j\in S} \widetilde{q}_j\, \mathbb{P}(X_k = i \mid X_0 = j)$.

Ahora tomando el límite cuando $k \to \infty$ se tiene que:

$$\begin{aligned}
\widetilde{q}_i &= \lim_{k\to\infty} \left(\sum_{j\in S} \widetilde{q}_j\, \mathbb{P}(X_k = i \mid X_0 = j) \right) = \sum_{j\in S} \widetilde{q}_j \Big(\lim_{k\to\infty} \mathbb{P}(X_k = i \mid X_0 = j) \Big) \\
&= \sum_{j\in S} \widetilde{q}_j q_i = q_i \sum_{j\in S} \widetilde{q}_j = q_i,\ \forall\, i \in S.
\end{aligned}$$

□

Lema 2.23. *Sea P la matriz de transición de una cadena de Markov homogénea y finita, siendo irreducible y aperiódica. Consideremos la distribución de probabilidades $\boldsymbol{q}$ sobre el espacio de estados S, cuyas componentes verifican lo siguiente:*

$$q_i P_{ij} = q_j P_{ji}, \quad \forall\, i, j \in S \ \ (\textit{Ecuación de balance detallado}). \tag{2.17}$$

Entonces se tiene que $\boldsymbol{q}$ *es la distribución estacionaria asociada a dicha cadena de Markov.*

Demostración: Por ser la cadena de Markov irreducible y aperiódica, se sabe que existe la distribución estacionaria asociada a dicha cadena.
Se debe probar si es la dada en el lema: Para ello, se debe garantizar que la distribución $\boldsymbol{q}$ dada, verifica la expresión (2.16).

Se sabe que: $\forall\, i, j \in S,\ q_i P_{ij} = q_j P_{ji}$, luego sumando en $j \in S$ en ambos lados de la igualdad, se obtiene que:

$$\sum_{j\in S} q_i P_{ij} = \sum_{j\in S} q_j P_{ji} \overset{\text{P estocástica}}{\Longrightarrow} q_i = \sum_{j\in S} q_j P_{ji},$$

y por lo tanto, se tiene que $\boldsymbol{q}$ es la distribución estacionaria asociada a la cadena de Markov dada con matriz de transición P, y además, es única. □

Como consecuencia del Lema 2.23, se deduce que para una cadena de Markov homogénea y finita con matriz de transición P, existirá una distribución estacionaria $\boldsymbol{q}$ única, siempre que sus componentes verifiquen la expresión (2.17) y la cadena de Markov sea irreducible y aperiódica.

Nota 2.24. *Se denota por* $\boldsymbol{q}^*$ *a una distribución de probabilidades sobre el espacio de soluciones* S *cuyas componentes vienen dadas por:*

$$q_i^*(c) = \frac{1}{N_0(c)} exp\left(\frac{-f(i)}{c}\right),\ \ \forall\, i \in S \quad donde \quad N_0(c) = \sum_{j\in S} exp\left(\frac{-f(j)}{c}\right), \quad (2.18)$$

donde $i \in S$ *y* $N_0(c)$ *es una constante de normalización.*

Antes de realizar la demostración del teorema que nos garantizará la convergencia del algoritmo hacia el espacio de soluciones óptimas S_{opt}, se van a definir y demostrar una serie de resultados que nos van a hacer falta. Estos resultados, nos van a permitir probar ciertas propiedades de la distribución dada en la Nota 2.24.

Corolario 2.25. *Dada* (S, f) *una instancia de un problema de optimización combinatoria y una adecuada estructura de estados vecinos. Dada la distribución de probabilidades* $\boldsymbol{q}^*$ *de la Nota 2.24, se tiene que:*

$$q_i^{**} \overset{\text{def}}{=} \lim_{c\to 0} q_i^*(c) = \frac{1}{|S_{opt}|}\chi_{S_{opt}}(i) \ \ donde \ \ \chi_{S_{opt}}(i) = \begin{cases} 1 & \text{si } i \in S_{opt} \\ 0 & \text{si } i \notin S_{opt} \end{cases}. \quad (2.19)$$

Demostración: Se sabe que

$$\lim_{c\to 0} e^{\frac{a}{c}} = \begin{cases} 1 & \text{si } a = 0 \\ 0 & \text{si } a < 0 \end{cases},$$

lo cual, va a ser usado a lo largo de la demostración.

$$\lim_{c\to 0} q_i^*(c) = \lim_{c\to 0} \frac{exp\left(\frac{-f(i)}{c}\right)}{\sum_{j\in S} exp\left(\frac{-f(j)}{c}\right)} \overset{(1)}{=} \lim_{c\to 0} \frac{exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S} exp\left(\frac{f_{opt}-f(j)}{c}\right)}$$

$$\overset{(2)}{=} \lim_{c\to 0} \frac{\chi_{S_{opt}}(i)}{\sum_{j\in S} exp\left(\frac{f_{opt}-f(j)}{c}\right)} + \lim_{c\to 0} \frac{exp\left(\frac{f_{opt}-f(i)}{c}\right)}{\sum_{j\in S} exp\left(\frac{f_{opt}-f(j)}{c}\right)} \chi_{S\setminus S_{opt}}(i) \overset{(3)}{=} \frac{\chi_{S_{opt}}(i)}{|S_{opt}|}.$$

Para resolver el límite, en el paso 1 se multiplica arriba y abajo por $e^{\frac{f_{opt}}{c}}$, y en el paso 2, se separa el límite resultante en dos según si $i \in S_{opt}$ o no.
Tras el paso (2) se obtiene que:

- El primer término será no nulo cuando $i \in S_{opt}$, y en ese caso, se tiene que si $j \in S_{opt}$ entonces la exponencial vale 1, y en caso contrario, como $f_{opt} - f(j) < 0$ dicho límite valdrá 0. Entonces el denominador del primer término valdrá $|S_{opt}|$.
- Por otro lado, el segundo término de la expresión será no nulo si $i \notin S_{opt}$, pero en ese caso, $f_{opt} - f(i) < 0$, así que el límite de la exponencial vale 0, y como el del denominador es finito, dicho límite es nulo.

Por lo tanto, la suma de los límites será la expresión que queda en el paso 3, finalizando así la demostración. □

El resultado que se obtiene del corolario anterior tiene mucha importancia, ya que cuando el valor de c decrece a 0, la distribución $\boldsymbol{q}^*$ definida en la Nota 2.24 se comporta de manera adecuada. Este comportamiento se debe a que dicha distribución en el límite es uniforme sobre el conjunto de soluciones óptimas S_{opt} y, por ser un conjunto finito, se podrá obtener la expresión dada en (2.10). Por lo tanto, el objetivo perseguido será probar que dicha distribución es la estacionaria asociada a una cadena de Markov finita y homogénea, ya que así, se podrá deducir la convergencia del algoritmo hacia el conjunto de soluciones óptimas S_{opt} de una determinada función f.

Para ello, antes es necesario enunciar y probar una serie de resultados adicionales.

Definición 2.26. *Se define el coste esperado $E_c(f)$, el coste cuadrático esperado $E_c(f^2)$, la varianza de coste $Var_c(f)$ y la entropía $\mathbb{S}_c$ considerando la distribución dada en la Nota 2.24 como:*

$$\begin{aligned} E_c(f) &\overset{\text{def}}{=} \sum_{i\in S} f(i) q_i^*(c). \\ E_c(f^2) &\overset{\text{def}}{=} \sum_{i\in S} f^2(i) q_i^*(c). \\ Var_c(f) &\overset{\text{def}}{=} \sigma_c^2 = E_c(f^2) - E_c(f)^2. \\ \mathbb{S}_c &\overset{\text{def}}{=} -\sum_{i\in S} q_i^*(c) \ln(q_i^*(c)). \end{aligned} \tag{2.20}$$

Corolario 2.27. *Dada la distribución $q_i^*(c)$ se tienen los siguientes resultados:*

$$\begin{aligned}
&a)\ \frac{\partial}{\partial c}E_c(f) = \frac{\sigma_c^2}{c^2} \quad y \quad \frac{\partial}{\partial c}\mathbb{S}_c = \frac{\sigma_c^2}{c^3}.\\
&b)\ \lim_{c\to\infty} E_c(f) \stackrel{\text{def}}{=} E_\infty(f) = \frac{1}{|S|}\sum_{i\in S} f(i).\\
&c)\ \lim_{c\to 0} E_c(f) = f_{opt}.\\
&d)\ \lim_{c\to\infty} \mathbb{S}_c \stackrel{\text{def}}{=} \mathbb{S}_\infty = \ln|S|.\\
&e)\ \lim_{c\to 0} \mathbb{S}_c \stackrel{\text{def}}{=} \mathbb{S}_0 = \ln|S_{opt}|.
\end{aligned} \tag{2.21}$$

Demostración:

$a)$
$$\frac{\partial}{\partial c}N_0(c) = \frac{\partial}{\partial c}\left(\sum_{j\in S} exp\left(\frac{-f(j)}{c}\right)\right) = \sum_{j\in S} exp\left(\frac{-f(j)}{c}\right)\frac{f(j)}{c^2}.$$

$$\begin{aligned}
\frac{\partial}{\partial c}q_i^*(c) &= \frac{\partial}{\partial c}\left(\frac{exp\left(\frac{-f(i)}{c}\right)}{N_0(c)}\right) = \frac{exp\left(\frac{-f(i)}{c}\right)\frac{f(i)}{c^2}N_0(c) - exp\left(\frac{-f(i)}{c}\right)\frac{\partial}{\partial c}N_0(c)}{N_0(c)^2}\\
&= q_i^*(c)\left(\frac{f(i)}{c^2} - \frac{\sum_{j\in S} exp\left(\frac{-f(j)}{c}\right)\frac{f(j)}{c^2}}{N_0(c)}\right)\\
&= \frac{q_i^*(c)}{c^2}\left(f(i) - \sum_{j\in S} q_j^*(c)f(j)\right) = \frac{q_i^*(c)}{c^2}(f(i) - E_c(f)).
\end{aligned}$$

Por lo tanto, se tiene que:

$$\begin{aligned}
\frac{\partial}{\partial c}E_c(f) &= \frac{\partial}{\partial c}\left(\sum_{i\in S} f(i)q_i^*(c)\right) = \sum_{i\in S} f(i)\frac{q_i^*(c)}{c^2}(f(i) - E_c(f))\\
&= \frac{1}{c^2}\left(\sum_{i\in S} f^2(i)q_i^*(c) - f^2(i)(q_i^*(c))^2\right) = \frac{E_c(f^2) - E_c(f)^2}{c^2} = \frac{\sigma_c^2}{c^2}.\\
\frac{\partial}{\partial c}\mathbb{S}_c &= -\sum_{i\in S}\left(\frac{\partial}{\partial c}q_i^*(c)\ln(q_i^*(c))\right) = -\sum_{i\in S}\left(\frac{\partial}{\partial c}q_i^*(c)\ln(q_i^*(c)) + \frac{\partial}{\partial c}q_i^*(c)\right)\\
&= -\sum_{i\in S}\left(\frac{q_i^*(c)}{c^2}(f(i) - E_c(f))(\ln(q_i^*(c)) + 1)\right)\\
&= -\sum_{i\in S}\left(\frac{q_i^*(c)}{c^2}(f(i) - E_c(f))\ \frac{-f(i)}{c}\right)\\
&= \frac{1}{c^3}\left(\sum_{i\in S} q_i^*(c)f(i)^2 - q_i^*(c)f(i)E_c(f)\right) = \frac{E_c(f^2) - E_c(f)^2}{c^3} = \frac{\sigma_c^2}{c^3}.
\end{aligned}$$

$$b)\ \lim_{c\to\infty} E_c(f) \overset{\text{def}}{=} \lim_{c\to\infty} \sum_{i\in S} f(i)q_i^*(c) = \sum_{i\in S} f(i) \left(\lim_{c\to\infty} \frac{exp\left(\frac{-f(i)}{c}\right)}{\sum_{j\in S} exp\left(\frac{-f(j)}{c}\right)} \right)$$
$$\overset{(*)}{=} \frac{1}{|S|} \sum_{i\in S} f(i).$$

En la igualdad identificada con $(*)$, se ha realizado el límite de $q_i^*(c)$. Como cuando $c \to \infty$ las exponenciales tienen a 1, se tiene que el numerador vale 1 y el denominador $|S|$.

$$c)\ \lim_{c\to 0} E_c(f) \overset{\text{def}}{=} \lim_{c\to 0} \sum_{i\in S} f(i)q_i^*(c) = \sum_{i\in S} f(i) \left(\lim_{c\to 0} q_i^*(c) \right) \overset{(2.19)}{=} \sum_{i\in S} f(i) \frac{\chi_{S_{opt}}(i)}{|S_{opt}|} \overset{(*)}{=} f_{opt}.$$

En la igualdad identificada con $(*)$, se ha tenido en cuenta que los términos de la suma son nulos excepto para $i \in S_{opt}$, luego dicha suma quedará $\frac{|S_{opt}|}{|S_{opt}|} f_{opt} = f_{opt}$.

$$d)\ \lim_{c\to\infty} \mathbb{S}_c \overset{\text{def}}{=} -\lim_{c\to\infty} \sum_{i\in S} q_i^*(c) \cdot \ln(q_i^*(c)) \overset{(*)}{=} -\sum_{i\in S} \frac{1}{|S|} \ln\left(\frac{1}{|S|}\right) = -\ln\left(\frac{1}{|S|}\right) = \ln|S|.$$

En la igualdad identificada con $(*)$ se ha aplicado que $q_i^*(c) \overset{c\to\infty}{\longrightarrow} \frac{1}{|S|}$.

$$e)\ \lim_{c\to 0} \mathbb{S}_c \overset{\text{def}}{=} -\lim_{c\to 0} \sum_{i\in S} q_i^*(c) \cdot \ln(q_i^*(c)) \overset{(*)}{=} -\sum_{i\in S} \frac{\chi_{S_{opt}(i)}}{|S_{opt}|} \ln\left(\frac{\chi_{S_{opt}(i)}}{|S_{opt}|}\right)$$
$$\overset{(**)}{=} \frac{-|S_{opt}|}{|S_{opt}|} \ln\left(\frac{1}{|S_{opt}|}\right) = \ln|S_{opt}|.$$

En la igualdad identificada con $(*)$, se ha aplicado que $q_i^*(c) \overset{c\to 0}{\longrightarrow} \frac{\chi_{S_{opt}(i)}}{|S_{opt}|}$, y en $(**)$, que el número de elementos que no se anulan son exactamente $|S_{opt}|$ ya que sólo para $i \in S_{opt}$, $\chi_{S_{opt}(i)} = 1$. □

Corolario 2.28. *Sea (S, f) una instancia de un problema de optimización combinatoria donde $S_{opt} \neq S$ y $q_i^*(c)$ la distribución dada por la Nota 2.24. Entonces se tiene que:*

$$\begin{aligned}
&a)\ \forall\ i \in S_{opt}, \quad \frac{\partial}{\partial c} q_i^*(c) < 0, \\
&b)\ \forall\ i \notin S_{opt}\ con\ f(i) \geq\ E_\infty(f), \quad \frac{\partial}{\partial c} q_i^*(c) > 0, \\
&c)\ \forall\ i \notin S_{opt}\ con\ f(i) <\ E_\infty(f),\ \exists\ \hat{c}_i > 0, \quad \frac{\partial}{\partial c} q_i^*(c) \begin{cases} > 0 & \text{si } c < \hat{c}_i \\ = 0 & \text{si } c = \hat{c}_i \\ < 0 & \text{si } c > \hat{c}_i \end{cases}.
\end{aligned} \tag{2.22}$$

Demostración: Por el Corolario 2.27 se sabe que $\frac{\partial}{\partial c} q_i^*(c) = \frac{q_i^*(c)}{c^2}(f(i) - E_c(f))$, luego el signo de dicha derivada parcial vendrá determinado por el de $f(i) - E_c(f)$, ya que $\forall\ i \in S,\ \frac{q_i^*(c)}{c^2} > 0$.

Además, por el Corolario 2.27 se tiene también que a medida que crece c, $E_c(f)$ aumenta de valor de f_{opt} a $E_\infty(f)$, siempre que $S_{opt} \neq S$.

Si $i \in S_{opt}$, entonces $f(i) = f_{opt}$ por lo que, $f(i) < f(j)\ \ \forall\, j \in S$ y $f(i) <\ E_c(f)$ por lo que $\frac{\partial}{\partial c} q_i^*(c) < 0$.

Si $i \notin S_{opt}$ se tiene que $f(i) \neq f_{opt}$ y se pueden dar varias situaciones:

- Si $f(i) \geq\ E_\infty(f)$, entonces $f(i) - E_c(f)\ > 0$ por lo que $\frac{\partial}{\partial c} q_i^*(c) > 0$.
- Si $f(i) <\ E_\infty(f)$, entonces $\exists\ \hat{c}_i > 0$ donde $f(i) - E_c(f) = 0$, y por lo tanto, se tiene que:

$$\frac{\partial}{\partial c} q_i^*(c) \begin{cases} > 0 & \text{si } c < \hat{c}_i \\ = 0 & \text{si } c = \hat{c}_i \\ < 0 & \text{si } c > \hat{c}_i \end{cases}$$

□

El estudio de la primera derivada de dicha distribución $\boldsymbol{q}^*$ permite determinar los puntos críticos y la monotonía de dicha función. Por lo tanto, se puede ver por el apartado *a)*, que la probabilidad de encontrar una solución óptima va a aumentar a medida que decrece el valor del parámetro de control c.
Por otro lado para aquellas soluciones que no sean óptimas, $\exists\ \hat{c}_i > 0$ tal que para $c < \hat{c}_i$, la probabilidad de encontrar dicha solución decrece monótonamente a medida que decrece c.

Teorema 2.29. *Sea (S, f) una instancia de un problema de optimización combinatoria con matriz de transición asociada P(c) (dada en la Definición 2.11) para el algoritmo Simulated Annealing. Si además se satisface la siguiente condición:*

$$\forall\ i, j \in S,\ \exists\ p \geq 1,\ l_0, ..., l_p \in S \text{ siendo } l_0 = i,\ l_p = j \text{ y } G_{l_k l_{k+1}} > 0,\ k = 0, ..., p-1,$$

se tiene que la cadena de Markov tiene como distribución estacionaria la dada en la Nota 2.24.

Demostración: Para demostrar dicho resultado se va a hacer uso del Teorema 2.22 de Feller, y para ello, se debe ver que se cumplen las hipótesis del mismo, esto es, que la matriz de transición P es irreducible y aperiódica.

Irreducible: Se debe ver que $\forall\ i, j \in S,\ \exists\ p:\ 1 \leq p < \infty$ tal que $(P^p)_{ij}(c) > 0$.

$$\begin{aligned}(P^p)_{ij}(c) &= \sum_{k_1 \in S} \cdots \sum_{k_{p-1} \in S} P_{ik_1}(c) \cdots P_{k_{p-1}j}(c) \geq P_{il_1}(c) \cdots P_{l_{p-1}j}(c) \\ &\overset{(2.7)}{=} G_{il_1} A_{il_1}(c) \cdots G_{l_{p-1}j} A_{l_{p-1}j}(c) > 0,\end{aligned}$$

ya que, $A_{ij} > 0\ \ \forall\, i, j \in S$ y $G_{l_k l_{k+1}} > 0,\ k = 0, ..., p-1$ por hipótesis.

Aperiódica: Se debe ver que $\forall\ i \in S\ mcd(D_i) = 1$, y para ello, se usará el Lema 2.15, esto es, bastará probar que $\exists\ i \in S : P_{ii}(c) > 0$.

Se supone sin pérdida de generalidad que $i, j \in S$ y $f(i) < f(j)$ con $G_{ij} > 0$, lo cual, será válido dada la condición del enunciado siempre que $S_{opt} \neq S$.

$$A_{ij}(c) = exp\left(\frac{-(f(j) - f(i))^+}{c}\right) = exp\left(\frac{-(f(j) - f(i))}{c}\right) = \frac{1}{e^{\frac{f(j)-f(i)}{c}}} < 1 \quad (2.23)$$

$$P_{ii}(c) \overset{(2.7)}{=} 1 - \sum_{l\in S, l\neq i} P_{il}(c) \overset{(2.7)}{=} 1 - \sum_{l\in S, l\neq i} G_{il}A_{il}(c) = 1 - G_{ij}A_{ij}(c)$$
$$- \sum_{l\in S, l\neq i,j} G_{il}A_{il}(c) \overset{(2.23)}{>} 1 - G_{ij} - \sum_{l\in S, l\neq i,j} G_{il} = 1 - \sum_{l\in S, l\neq i} G_{il} = 0,$$

donde la última desigualdad se deduce por ser G estocástica.

Por el Teorema 2.22 de Feller, existe una única distribución estacionaria y sólo queda probar que es la dada, esto es, que $\boldsymbol{q}(c)$ es un vector estocástico y sus componentes satisfacen la expresión (2.17).

Estocástico: Se debe ver que la suma de las componentes del vector $\boldsymbol{q}$ es 1.

$$\boldsymbol{q}(c) = (q_1, ..., q_n, ...) = \left(\frac{1}{N_0(c)} e^{\frac{-f(1)}{c}}, ..., \frac{1}{N_0(c)} e^{\frac{-f(n)}{c}}, ... \right).$$

$$\text{Luego } \sum_{j\in S} q_j = \frac{1}{N_0(c)} \left(e^{\frac{-f(1)}{c}} + ... + e^{\frac{-f(n)}{c}} + ... \right) = \frac{1}{N_0(c)} \sum_{j\in S} e^{\frac{-f(j)}{c}} = 1.$$

Satisface (2.17): Sea S_i el conjunto formado por todos los estados vecinos al estado i, se tiene que $i \in S_j \Leftrightarrow j \in S_i$, luego $\chi_{S_j}(i) = \chi_{S_i}(j)$ por lo que $G_{ij} = G_{ji}$ $(*)$.
En consecuencia,

$$q_i^*(c)P_{ij}(c) = q_j^*(c)P_{ji}(c) \overset{(*)}{\equiv} q_i^*(c)A_{ij}(c) = q_j^*(c)A_{ji}(c),$$

$$\begin{aligned} q_i^*(c)A_{ij}(c) =& \frac{1}{N_0(c)} exp\left(\frac{-f(i)}{c}\right) exp\left(\frac{-(f(j)-f(i))^+}{c}\right) \\ =& \frac{1}{N_0(c)} exp\left(\frac{-f(j)}{c}\right) exp\left(-\frac{f(i)-f(j)+(f(j)-f(i))^+}{c}\right) \\ =& \frac{1}{N_0(c)} exp\left(\frac{-f(j)}{c}\right) exp\left(\frac{-(f(i)-f(j))^+}{c}\right) = q_j^*(c)A_{ji}(c). \end{aligned}$$

En la primera igualdad se han usado las expresiones (2.18) y (2.9), en la segunda se ha multiplicado la expresión por $exp\left(\frac{f(j)-f(j)}{c}\right) = 1$, y en la tercera la expresión de la segunda exponencial es 0, por lo que equivale a $exp\left(\frac{-(f(i)-f(j))^+}{c}\right)$ ya que $f(j) > f(i)$. □

En el Corolario 2.25 se vio que $\lim_{c\to 0} q_i^*(c) = \frac{1}{|S_{opt}|}\chi_{S_{opt}}(i)$, luego este resultado junto con el teorema probado, garantiza la convergencia asintótica del algoritmo *Simulated Annealing* hacia el conjunto de soluciones óptimas S_{opt}, puesto que:

$$\begin{aligned} \lim_{c\to 0}\left(\lim_{k\to\infty} \mathbb{P}(X_k \in S_{opt}) \right) &= \lim_{c\to 0}\left(\lim_{k\to\infty} \sum_{i\in S_{opt}} \mathbb{P}(X_k = i) \right) = \lim_{c\to 0} \sum_{i\in S_{opt}} q_i(c) \\ &\overset{(*)}{=} \sum_{i\in S_{opt}} \lim_{c\to 0} q_i^*(c) = \sum_{i\in S_{opt}} \frac{1}{|S_{opt}|}\chi_{S_{opt}}(i) = 1, \end{aligned}$$

ya que, sólo para $i \in S_{opt}$, $\chi_{S_{opt}}(i) = 1$. En $(*)$ se ha aplicado la unicidad de la distribución estacionaria. □

2.2.3. Convergencia usando cadenas no homogéneas

El desarrollo de esta sección se centra en demostrar la convergencia del algoritmo *Simulated Annealing* para el caso en que se modeliza mediante una única cadena de Markov no homogénea. Esto se debe a que, aunque el uso de cadenas de Markov homogéneas es más sencillo, en la práctica no se puede realizar, ya que se necesita definir infinitas cadenas de Markov homogéneas (una para cada valor descendiente del parámetro de control c).

Para hacer frente a este inconveniente, el algoritmo *Simulated Annealing* se puede modelar o describir mediante la construcción de una única cadena de Markov no homogénea cuya matriz de transición de probabilidades viene dada por la expresión (2.7). Así, se reduce la consideración de una secuencia de cadenas de Markov homogéneas infinitamente largas a una sola no homogénea de longitud infinita.

Al igual que en el caso de las cadenas homogéneas, para obtener la convergencia asintótica del algoritmo, será necesario que las matrices de generación de probabilidades y de aceptación definidas en (2.8) y (2.9) cumplan ciertas condiciones, que $\lim_{l\to\infty} c_l^{'} = 0$, etc., y por lo tanto, se comienza estableciendo la notación usada, así como una serie de definiciones y resultados previos.

Notación:

$c_l^{'}$: Valor del parámetro de control en la l-ésima cadena homogénea de Markov.

L: Longitud de las cadenas de Markov homogéneas consideradas.

c_k: Valor del parámetro de control en el paso k-ésimo del algoritmo. Como varía en cada paso se considera $c = c_k$.

$\{c_k;\ k \in \mathbb{N}^+\}$: Es la secuencia de valores que toma el parámetro de control. Se considera que:

$$c_k = c_l^{'}, \quad \text{donde} \quad lL < k \leq (l+1)L,$$

esto es, se elige constante para un número determinado de transiciones. Por lo tanto, la cadena de Markov no homogénea resultante es la combinación de un número infinito de cadenas de Markov homogéneas de longitud en este caso finita.
Además se cumple que:

$$c_{l+1}^{'} \leq c_l^{'}, \ \ l = 0, 1, ... \quad \text{y} \quad \lim_{l\to\infty} c_l^{'} = 0$$

Definición 2.30. *Dada una cadena de Markov no homogénea, se considera la matriz de transición $P(m,k)$ definida como:*

$$P(m,k) = \prod_{l=m+1}^{k} P(l), \text{ donde } P_{ij}(m,k) = \mathbb{P}(X_k = j \mid X_m = i) \ \text{ y } \ m < k. \quad (2.24)$$

Nota: Para $m = 0$, se tiene que $P(0,k) = P^k$.

Definición 2.31 (**Seneta, 1981**). *Dada $P \in \mathcal{M}_{n\times n}$ estocástica, se define el coeficiente de ergodicidad $\tau(P)$ como:*

$$\tau(P) = \frac{1}{2}\max_{i,j}\sum_{k=1}^{n}|P_{ik} - P_{jk}| = 1 - \min_{i,j}\sum_{k=1}^{n}min(P_{ik}, P_{jk}),\ 0 \leq \tau(P) \leq 1. \quad (2.25)$$

Definición 2.32 (**Seneta, 1981**). *Una cadena no homogénea finita de Markov es débilmente ergódica si:*

$$\forall\ i, j, l \in S,\ \forall\ m > 0 :\ \lim_{k\to\infty}(P_{il}(m,k) - P_{jl}(m,k)) = 0. \quad (2.26)$$

Definición 2.33 (**Seneta, 1981**). *Una cadena no homogénea finita de Markov es fuertemente ergódica si existe un vector estocástico $\boldsymbol{q}^{**}$ tal que:*

$$\forall\ i, j \in S,\ \forall\ m > 0 :\ \lim_{k\to\infty} P_{ij}(m,k) = q_j^{**}. \quad (2.27)$$

Por lo tanto, de las Definiciones 2.32 y 2.33 se deduce que la propiedad de ergodicidad débil asegura que para todo m hay pérdida de memoria, esto es, X_k pasa a ser independiente de X_m cuando $k \to \infty$, mientras que la propiedad de ergodicidad fuerte, implica la convergencia en distribución de X_k:

$$\begin{aligned}\lim_{k\to\infty}\mathbb{P}(X_k = j) &= \lim_{k\to\infty}\left(\sum_{i\in S}\mathbb{P}(X_k = j \mid X_m = i)\mathbb{P}(X_m = i)\right)\\ &\overset{(2.24)}{=} \lim_{k\to\infty}\left(\sum_{i\in S}P_{ij}(m,k)\mathbb{P}(X_m = i)\right) \overset{(2.27)}{=} q_j^{**}\sum_{i\in S}\mathbb{P}(X_m = i) = q_j^{**}.\end{aligned} \quad (2.28)$$

□

También, de las definiciones se deduce que toda cadena finita fuertemente ergódica es débilmente ergódica, ya que:

$$\lim_{k\to\infty}(P_{il}(m,k) - P_{jl}(m,k)) = \lim_{k\to\infty}P_{il}(m,k) - \lim_{k\to\infty}P_{jl}(m,k) = q_l^{**} - q_l^{**} = 0$$

□

Además, si la cadena de Markov es homogénea, débilmente ergódica y fuertemente ergódica son definiciones equivalentes ya que por el Teorema 2.22 de Feller, la distribución estacionaria para una cadena homogénea es única. Para el caso de cadenas no homogéneas, a continuación se muestran una serie de resultados para determinar qué condiciones adicionales aseguran la equivalencia anteriormente descrita.

Teorema 2.34 (**Seneta, 1981 [7]**). *Una cadena de Markov no homogénea es débilmente ergódica si y solo si existe una secuencia de números estrictamente positivos $\{k_i\},\ i = 0, 1, ...$ tal que:*

$$\sum_{i=0}^{\infty}[1 - \tau(P(k_i, k_{i+1}))] = \infty. \quad (2.29)$$

Demostración: Para probar el Teorema se deben demostrar ambas implicaciones.

Por un lado, se supone que se tiene una cadena de Markov no homogénea débilmente ergódica.

Por un resultado (Isaadson and Madsen, 1976 [8]) se sabe que ser débilmente ergódica equivale a que $\tau(P(m,k)) \overset{\text{k}\to\infty}{\longrightarrow} 0$, por lo que $\forall\, m > 0$, $1 - \tau(P(m,k)) \overset{\text{k}\to\infty}{\longrightarrow} 1$.

Se considera $m = k_0 > 0$, luego existe k_1 tal que $1 - \tau(P(k_0, k_1)) > \frac{1}{2}$. Al igual que antes existe k_2 tal que $1 - \tau(P(k_1, k_2)) > \frac{1}{2}$ y así sucesivamente. Por lo tanto, se cumple que:

$$\sum_{i=0}^{k} [1 - \tau(P(k_i, k_{i+1}))] > \frac{k+1}{2} \overset{\text{k}\to\infty}{\longrightarrow} \infty,$$

esto es, la suma es divergente. En consecuencia, se tiene una secuencia $\{k_i\}$ de números estrictamente positivos tal que $\sum_{i=0}^{\infty} [1 - \tau(P(k_i, k_{i+1}))] = \infty$, como se quería probar.

Se prueba ahora el recíproco, esto es, que si existe dicha secuencia de números estrictamente positivos cumpliendo (2.29), la cadena de Markov no homogénea es débilmente ergódica. Para ello se va a probar que $\tau(P(m,k)) \overset{\text{k}\to\infty}{\longrightarrow} 0$.

Por un resultado de análisis (Isaadson and Madsen, 1976 [8]) se sabe que si $\{\epsilon_j\}_{j=1}^{\infty}$ es una secuencia de números tales que $0 < \epsilon_j < 1$, $\forall\, j$, entonces:

$$\prod_{j=m}^{n} (1 - \epsilon_j) \overset{\text{n}\to\infty}{\longrightarrow} 0 \Leftrightarrow \sum_{j=m}^{\infty} \epsilon_j = \infty.$$

Para la secuencia $\{k_i\}$ dada se cumple (2.29) por lo que también se cumple que $\sum_{i=j}^{\infty} [1 - \tau(P(k_i, k_{i+1}))] = \infty$, $\forall\, j$.

Como $0 < \tau(P(k_i, k_{i+1})) < 1$, ya que si $\tau(P) = 0$ entonces P sería una matriz constante y, por un resultado (Isaadson and Madsen, 1976 [8]) se sabe que $\tau(P(m,k)) \leq \prod_{j=m+1}^{k} \tau(P(j)) = (\frac{1}{2})^{(k-m)/2}$, luego $\tau(P(k_i, k_{i+1}) \leq (\frac{1}{2})^{(k_{i+1}-k_i)/2} < 1$. Por lo tanto, se puede aplicar el resultado anterior y concluir que:

$$\prod_{i=j}^{l} [1 - (1 - \tau(P(k_i, k_{i+1})))] = \prod_{i=j}^{l} \tau(P(k_i, k_{i+1})) \overset{\text{l}\to\infty}{\longrightarrow} 0$$

Fijado un m, se define $j = min\{i : k_i > m\}$ y $\forall\, k > m$, $l = max\{i : k_i < k\}$. Es claro que si $k \to \infty$, l también lo hará. Además por un resultado (Isaadson and Madsen, 1976 [8]) se sabe que si P y Q son dos matrices estocásticas, $\tau(QP) \leq \tau(Q)\tau(P)$, por lo que finalmente se obtiene lo buscado:

$$\tau(P(m,k)) \leq \tau(P(m,k_j)) \prod_{i=j}^{l-1} \tau(P(k_i, k_{i+1}))\tau(P(k_l, k)) \overset{\text{k}\to\infty}{\longrightarrow} 0$$

□

Teorema 2.35 (**Isaadson and Madsen, 1976** [**8**])**.** *Una cadena de Markov finita y no homogénea es fuertemente ergódica si se cumplen las siguientes condiciones:*

1) La cadena de Markov es débilmente ergódica.

2) Para todo valor de k existe un vector estocástico $\boldsymbol{q}(k)$ *que es un autovector por la izquierda de P(k) con autovalor asociado 1.*

3) Los autovectores $\boldsymbol{q}(k)$ *satisfacen que:*

$$\sum_{k=1}^{\infty} \|\boldsymbol{q}(k) - \boldsymbol{q}(k+1)\| < \infty.$$

Además se tiene que si $\boldsymbol{q} = \lim_{k\to\infty} \boldsymbol{q}(k)$*, entonces dicho vector es el dado en la Definición 2.33.*

Demostración: Para probar el Teorema es necesario enunciar unos resultados que se usarán aunque no se demostrarán. Dichos resultados se pueden encontrar en el libro indicado en el Teorema:

a) Sea P una matriz estocástica y R una matriz tal que $\sum_{k=1}^{\infty} r_{ik} = 0,\ \forall\ i$ con $\|R\| < \infty$, entonces $\|RP\| \leq \|R\|\tau(P)$.

b) Si $\sum_{k=1}^{\infty} \|\boldsymbol{q}(k) - \boldsymbol{q}(k+1)\| < \infty$ entonces es convergente, y por lo tanto, $\{\boldsymbol{q}(k)\}_{k=1}^{\infty}$ converge en norma a un vector $\boldsymbol{q}$. Como consecuencia, $\lim_{k\to\infty} \boldsymbol{q}(k) = \boldsymbol{q}$.

Teniendo en cuenta el resultado dado en *b)* se tiene que $\|\boldsymbol{q}(k) - \boldsymbol{q}\| \overset{\mathrm{k}\to\infty}{\longrightarrow} 0$. Luego, dado que $\boldsymbol{q}(k)$ es un vector estocástico, $\boldsymbol{q}$ también lo será. Para la demostración del Teorema se define la matriz estocástica constante Q cuyas filas son todas iguales al vector estocástico $\boldsymbol{q}$. Por analogía, Q_k será una matriz estocástica constante cuyas filas serán todas iguales a $\boldsymbol{q}(k)$, $\forall\ k$.

Se debe ver que si una cadena de Markov no homogénea cumple las condiciones *1)*, *2)* y *3)* entonces es fuertemente ergódica y por la Definición 2.33 se reduce a probar que:

$$\lim_{n\to\infty} P_{ij}(m,n) = q_j \equiv \|P(m,n) - Q\| \overset{\mathrm{n}\to\infty}{\longrightarrow} 0,\ \forall\ m > 0 \tag{2.30}$$

Ahora, fijado m, usando que $P(m,n) = P(m,l)P(l,n)$ y la desigualdad triangular:

$$\begin{aligned}\|P(m,n) - Q\| &\leq \|P(m,n) - Q_n\| + \|Q_n - Q\| \\ &\leq \|P(m,l)P(l,n) - Q_{l+1}P(l,n)\| + \|Q_{l+1}P(l,n) - Q_n\| + \|Q_n - Q\|\end{aligned}$$

Luego para ver que se cumple la expresión (2.30), se toma $\epsilon > 0$, y se busca un K tal que $\forall\ n \geq K$, $\|P(m,n) - Q\| < \epsilon$. Por lo tanto, para concluir basta acotar cada una de las tres desigualdades anteriores por $\frac{\epsilon}{3}$.

Se debe probar que $\|Q_{l+1}P(l,n) - Q_n\| < \frac{\epsilon}{3}$: Como consecuencia de la condición *2)*, se sabe que $Q_{l+1}P(l+1) = Q_{l+1}$, luego:

$$\begin{aligned}Q_{l+1}P(l,n) &= Q_{l+1}P(l+1,n) = Q_{l+1}P(l+1,n) - Q_{l+2}P(l+1,n) + Q_{l+2}P(l+1,n) \\ &= (Q_{l+1} - Q_{l+2})P(l+1,n) + Q_{l+2}P(l+1,n) \\ &= (Q_{l+1} - Q_{l+2})P(l+1,n) + (Q_{l+2} - Q_{l+3})P(l+2,n) + Q_{l+3}P(l+2,n)\end{aligned}$$

Haciendo esto sucesivamente y teniendo en cuenta que $Q_n P(n) = Q_n$ se deduce que:

$$Q_{l+1}P(l,n) = \sum_{k=l+1}^{n-1}(Q_k - Q_{k+1})P(k,n) + Q_n$$

Entonces aplicando la desigualdad triangular, que $0 \leq \tau(P) \leq 1$ y el resultado dado en *a)* al principio de la demostración:

$$\|Q_{l+1}P(l,n) - Q_n\| = \|\sum_{k=l+1}^{n-1}(Q_k - Q_{k+1})P(k,n)\| \leq \sum_{k=l+1}^{n-1}\|(Q_k - Q_{k+1})P(k,n)\|$$
$$\overset{a)}{\leq} \sum_{k=l+1}^{n-1}\|Q_k - Q_{k+1}\|\tau(P(k,n)) \leq \sum_{k=l+1}^{n-1}\|Q_k - Q_{k+1}\| = \sum_{k=l+1}^{n-1}\|\boldsymbol{q}(k) - \boldsymbol{q}(k+1)\|,$$

y debido a la condición *3)* se puede considerar un valor $l^* > m$ tal que $\forall\, n > l^*$,

$$\|Q_{l^*+1}P(l^*,n) - Q_n\| \leq \sum_{k=l^*+1}^{n-1}\|Q_k - Q_{k+1}\| = \sum_{k=l^*+1}^{n-1}\|\boldsymbol{q}(k) - \boldsymbol{q}(k+1)\| < \frac{\epsilon}{3}.$$

Se debe probar que $\|P(m,l)P(l,n) - Q_{l+1}P(l,n)\| < \frac{\epsilon}{3}$: Fijado l^*, como $P(m,l^*)$ y Q_{l^*+1} son matrices estocásticas, $\|P(m,l^*) - Q_{l^*+1}\| \leq 2$. Usando el resultado *a)* se tiene:

$$\|P(m,l^*)P(l^*,n) - Q_{l^*+1}P(l^*,n)\| = \|(P(m,l^*) - Q_{l^*+1})P(l^*,n)\|$$
$$\overset{a)}{\leq} \|P(m,l^*) - Q_{l^*+1}\|\tau(P(l^*,n)) \leq 2\tau(P(l^*,n))$$

Por la condición *1)*, se sabe que la cadena es débilmente ergódica luego se puede tomar un $K_1 > l^*$ tal que $\forall\, n \geq K_1$, $\tau(P(l^*,n)) < \frac{\epsilon}{6}$. Para dichos valores de n,

$$\|P(m,l^*)P(l^*,n) - Q_{l^*+1}P(l^*,n)\| \leq 2\frac{\epsilon}{6} = \frac{\epsilon}{3}$$

Se debe probar que $\|Q_n - Q\| < \frac{\epsilon}{3}$: Debido a la condición *2)*, $\{\boldsymbol{q}(k)\}_{k=1}^{\infty}$ converge en norma a un vector $\boldsymbol{q}$ cuando $k \to \infty$, $\forall\, k$ por lo que, $\|Q_n - Q\| \overset{n\to\infty}{\longrightarrow} 0$. Por lo tanto, existe un K_2 tal que $\forall\, n \geq K_2$, $\|Q_n - Q\| < \frac{\epsilon}{3}$.

Finalmente se tiene que, $\forall\, n \geq K = \text{máx}\{K_1, K_2\}$, $\|P(m,n) - Q\| < \frac{3\epsilon}{3} = \epsilon$.

□

Por lo tanto, se tendrá la convergencia en distribución del algoritmo si se prueba que la cadena de Markov no homogénea que lo modela es fuertemente ergódica, lo cual, es el objetivo del siguiente teorema.

Teorema 2.36. *Sea (S, f) una instancia de un problema de optimización combinatoria y P(k) la matriz de transición asociada al algoritmo Simulated Annealing que viene definida en (2.7), (2.8) y (2.9) cumpliéndose además que:*

1) $\forall\, i, j \in S$, $\exists\, p \geq 1$, $l_0, ..., l_p \in S$ *siendo* $l_0 = i$, $l_p = j$ *y* $G_{l_k l_{k+1}} > 0$, $k = 0, ..., p-1$.

2) La secuencia $\{c_l^{'}\}$ satisface lo siguiente:

$$c_l^{'} \geq \frac{(L+1)\Delta}{\log(l+2)}, \;\; l = 0, 1, ..., \quad donde \quad \Delta = \underset{i,j \in S}{max}\{f(j) - f(i) : j \in S_i\},$$

y L se corresponde con el máximo valor entre los valores mínimos correspondientes al número de transiciones o pasos requeridos para ir del estado j a i_{opt}, $\forall\ j \in S$. Dicho valor siempre existe debido a la condición 1).

En ese caso se tiene que la cadena de Markov converge en distribución a $\boldsymbol{q}^{**}$ con componentes:

$$q_i^{**} = \frac{1}{|S_{opt}|}\chi_{S_{opt}}(i), \;\; \forall\ i \in S. \tag{2.31}$$

Demostración: Se debe ver que se satisfacen las tres condiciones del Teorema 2.35 para concluir.

La cadena de Markov es débilmente ergódica: Para ello se va a hacer uso del Teorema 2.34 de Seneta.

Debido a la expresión dada en (2.7) se tiene que, $\forall\ k > 0,\ \forall\ i \in S,\ j \in S_i \backslash \{i\}$ (se trata del caso en que $i \neq j$), por lo que se obtiene la siguiente desigualdad:

$$P_{ij}(k) = G_{ij}A_{ij}(c_k) = \frac{\chi_{S_i}(j)}{\Theta}\ exp\left(-\frac{(f(j) - f(i))^+}{c_k}\right) \overset{\text{def } \Delta}{\geq} \frac{1}{\Theta} exp\left(\frac{-\Delta}{c_k}\right). \tag{2.32}$$

Sea $\hat{S}$ el conjunto de soluciones máximas locales y $\delta = \underset{i,j \in S}{min}^{+}\{f(j) - f(i) : j \in S_i\}$ (se coge el valor mínimo de entre aquellos que sean positivos). Entonces existe un $k_0 > 0$ tal que $\forall\ k > k_0$, $\forall\ i \in S \backslash \hat{S}$ se cumple que:

$$1 - exp\left(\frac{-\delta}{c_k}\right) > exp\left(\frac{-\Delta}{c_k}\right). \tag{2.33}$$

Por lo tanto, se tiene que $\forall\ k > k_0$, $\forall\ i \in S \backslash \hat{S}$:

$$\begin{aligned} P_{ii}(k) &= 1 - \sum_{j \in S_i \backslash \{i\}} G_{ij}A_{ij}(c_k) = 1 - \sum_{j \in S_i \backslash \{i\}} \frac{A_{ij}(c_k)}{\Theta} \geq 1 - \frac{1}{\Theta}\left(\Theta - 1 + exp\left(\frac{-\delta}{c_k}\right)\right) \\ &= \left(1 - exp\left(\frac{-\delta}{c_k}\right)\right)\frac{1}{\Theta} \overset{(2.33)}{>} \frac{1}{\Theta} exp\left(\frac{-\Delta}{c_k}\right). \end{aligned}$$

Sea $p \in S$ tal que:

$$P_{pi_{opt}}(k-1, k+L) = \underset{i \in S}{min}\left(P_{ii_{opt}}(k-1, k+L)\right). \tag{2.34}$$

Luego $\forall\ k \geq k_0$,

$$\underset{i \in S}{min}\left(P_{ii_{opt}}(k-1, k+L)\right) = \mathbb{P}(X_{k+L} = i_{opt} \mid X_{k-1} = p) =$$

$$= \sum_{j_1 \in S} \cdots \sum_{j_L \in S} P_{pj_1}(k) \cdots P_{j_L i_{opt}}(k+L) \geq P_{pl_1}(k) \cdots P_{l_L i_{opt}}(k+L)$$
$$\overset{(2.32)}{\geq} \left(\frac{1}{\Theta} \, exp\left(\frac{-\Delta}{c_k} \right) \right)^{L+1}$$

Se quiere probar (2.29) para la secuencia $\{k_l\}$, luego $\forall\ k \geq k_0$, se tiene que:

$$1 - \tau(P(k-1, k+L)) = \min_{i,j \in S} \left(\sum_{p \in S} min\{P_{ip}(k-1, k+L), P_{jp}(k-1, k+L)\} \right)$$
$$\overset{(2.34)}{\geq} \min_{i \in S} \left(P_{ii_{opt}}(k-1, k+L) \right) \geq \left(\frac{1}{\Theta} \, exp\left(\frac{-\Delta}{c_k} \right) \right)^{L+1}$$
$$= \Theta^{-L-1} exp\left(\frac{-\Delta(L+1)}{c_k} \right).$$

Usando lo anterior y considerando $k_l = lL - 1$ se obtiene que:

$$\sum_{l=0}^{\infty} [1 - \tau(P(k_l, k_{l+1}))] \geq \sum_{l=k_0}^{\infty} [1 - \tau(P(lL-1, lL+L))]$$
$$\geq \sum_{l=k_0}^{\infty} \Theta^{-L-1} exp\left(\frac{-\Delta(L+1)}{c_l'} \right) \overset{2)}{\geq} \sum_{l=k_0}^{\infty} \Theta^{-L-1} \frac{1}{l+2} = \infty,$$

ya que la serie armónica es divergente.

$\forall\ k$ existe un vector estocástico $\boldsymbol{q}$ que es autovector por la izquierda de $P(k)$ con autovalor asociado 1: Esto es consecuencia directa de la condición *1)* dada usando el Teorema 2.29, donde las componentes de $\boldsymbol{q}$ vienen dadas por la Nota 2.24. Luego se denota como $\boldsymbol{q}^*$.

Finalmente se debe probar que:

$$\sum_{k=1}^{\infty} \|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\| < \infty,$$

y, para ello se usan los resultados obtenidos en el Corolario 2.28.
Debido a dicho Corolario, se sabe que si $i \in S_{opt}$, entonces $q_i^*(c_{k+1}) - q_i^*(c_k) > 0$ y que va a existir un k_1 tal que $\forall\ k > k_1$ con $i \notin S_{opt}$, $q_i^*(c_{k+1}) - q_i^*(c_k) < 0$.

Por lo tanto, $\forall\ k \geq k_1$ se tiene que:

$$\|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\| \overset{\text{def}}{=} \sum_{i \in S} |q_i^*(c_k) - q_i^*(c_{k+1})| = \sum_{i \in S_{opt}} q_i^*(c_{k+1}) - q_i^*(c_k)$$
$$- \sum_{i \notin S_{opt}} q_i^*(c_{k+1}) - q_i^*(c_k) = 2 \left(\sum_{i \in S_{opt}} q_i^*(c_{k+1}) - q_i^*(c_k) \right),$$

ya que $\sum_{i \notin S_{opt}} q_i^*(c) = 1 - \sum_{i \in S_{opt}} q_i^*(c)$.

Finalmente se obtiene lo siguiente:

$$\sum_{k=1}^{\infty} \|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\| = \sum_{k=1}^{k_1} \|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\| + \sum_{k=k_1+1}^{\infty} \|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\|$$

$$= \sum_{k=1}^{k_1} \|\boldsymbol{q}^*(k) - \boldsymbol{q}^*(k+1)\| + 2\left(\sum_{i \in S_{opt}} q_i^*(c_\infty) - q_i^*(c_{k_1+1}) \right) \leq 2k_1 + 2 < \infty.$$

Como consecuencia, la cadena de Markov no homogénea converge en distribución hacia $\boldsymbol{q}^{**} = \lim_{k\to\infty} \boldsymbol{q}^*$, y por lo tanto, se concluye que:

$$\lim_{k\to\infty} \mathbb{P}(X_k \in S_{opt}) = \lim_{k\to\infty} \sum_{i \in S_{opt}} \mathbb{P}(X_k = i) \overset{(2.28)}{=} \sum_{i \in S_{opt}} q_i^{**} = \sum_{i \in S_{opt}} \frac{\chi_{S_{opt}}(i)}{|S_{opt}|} = 1$$

□

Así, se concluye que el algoritmo *Simulated Annealing* (modelado en este caso como una única cadena de Markov no homogénea) converge en distribución al conjunto de soluciones óptimas, siempre que la disminución del parámetro de control sea lenta (determinado por la segunda hipótesis de dicho teorema).

2.3. Elección de los parámetros (caso finito)

Las secciones anteriores se han desarrollado para probar que el algoritmo *Simulated Annealing* converge hacia el conjunto de soluciones óptimas tras un número infinito de pasos o transiciones realizadas. Sin embargo, en la mayoría de los casos va a interesar que dicho tiempo sea mucho menor, aunque con ello, sólo se logre obtener soluciones próximas a la óptima.

Por ello, esta sección se centra en estudiar cuál debe ser la elección correcta de los parámetros que intervienen en el algoritmo, de manera que se obtengan soluciones tan cercanas a la óptima como sea posible.

En este caso, se considera la implementación del algoritmo como una secuencia finita de cadenas homogéneas de Markov, también de longitud finita. Para probar la convergencia, se deben estudiar los parámetros que influyen en la misma, para así, elegirlos de la manera más adecuada posible.

En el recocido simulado o *Simulated Annealing* hay que tener en cuenta:

Parámetro de control c_k: Se debe considerar un valor inicial c_0, que debe ir decrementándose lentamente hasta obtener un valor final, mediante un determinado criterio de parada.

Transiciones realizadas por cada valor del parámetro de control: Será necesario realizar un número finito de pasos o transiciones para cada valor determinado del parámetro de control, para lo cual, las cadenas de Markov homogéneas consideradas tendrán longitud finita. L_k será la longitud correspondiente a la k-ésima cadena de Markov de la secuencia.

Para poder determinar el valor de dichos parámetros de la manera más adecuada, es necesario conocer lo que es el cuasi-equilibrio, ya que es lo que se pretende obtener.

Definición 2.37. *Dada una secuencia de cadenas de Markov homogéneas de longitud finita, se dice que se logra obtener el cuasi-equilibrio si:*

$$\|\mathbf{a}(L_k, c_k) - \boldsymbol{q}(c_k)\| < \epsilon, \quad \textit{donde } \epsilon > 0, \tag{2.35}$$

siendo $\mathbf{a}(L_k, c_k)$ la distribución de probabilidad de las soluciones tras L_k transiciones en la k-ésima cadena de Markov y $\boldsymbol{q}(c_k)$ la distribución estacionaria en c_k que fue definida en la Nota 2.24.

Esta condición es demasiado estricta y llevaría a considerar, como en las secciones anteriores, muchas transiciones para lograr alcanzar la distribución estacionaria. Para ello, el tiempo requerido sería de orden exponencial por lo que se establecerán condiciones menos rígidas que la dada, logrando obtener soluciones lo más próximas posibles a la óptima en tiempo polinomial.

Todo esto, ha dado lugar a considerar diversos enfoques a la hora de determinar el valor de dichos parámetros, siendo unas más sencillas que otras.

Estas consideraciones suelen realizar decrementos pequeños del parámetro de control c_k para así poder usar cadenas de Markov de menor longitud, pues en caso de que estos decrementos fueran más grandes, su longitud aumentaría añadiendo mayor complejidad.

A continuación, se describen dos propuestas. La primera fue realizada por Kirkpatrick, Gelatt y Vecchi (1982-1983). La segunda fue propuesta por Aarts y Van Laarhoven (1985) y, aunque garantiza que la ejecución se realiza en tiempo polinomial, tiene el inconveniente de que no establece la diferencia existente entre la solución obtenida y la óptima.

2.3.1. Valor inicial del parámetro de control

Primera propuesta: Debe ser lo suficientemente alto para permitir que al comienzo todas las transiciones sean aceptadas, de manera que se pueda explorar todo el espacio de estados S. Para ello, $exp\left(\frac{f(i)-f(j)}{c_0}\right) \approx 1$ para casi todos los estados i, j. Un método para lograrlo consiste en tomar como c_0 un valor grande y fijar una tasa de aceptación inicial χ_0 (suele ser 0.8). Tras ello, se realizan una serie de transiciones y si la tasa de aceptación

$$\chi = \frac{número\ de\ transiciones\ aceptadas}{número\ total\ de\ transiciones} < \chi_0,$$

entonces $c_0(l+1) = 2 \cdot c_0(l)$ siendo l la iteración en la que se encuentra el proceso que calcula el valor inicial del parámetro de control. Dicho proceso es iterativo y continua hasta que $\chi > \chi_0$.

Segunda propuesta: La elección de c_0, al igual que en el caso anterior, se basa en que al comienzo todos los posibles estados sean aceptados con la misma probabilidad. Para lograrlo se sigue un método más riguroso que el anterior.

Se considera una serie de transiciones m_0 con parámetro de control c y sean:

m_1: Número de transiciones propuestas para ir del estado i al j siendo $f(j) \leq f(i)$.

m_2: Número de transiciones propuestas para ir del estado i al j siendo $f(j) > f(i)$.

$\overline{\Delta f}^+$: Media de la diferencia del incremento en coste debido al aumento de coste de las transiciones de m_2.

Se define entonces el ratio de convergencia χ mediante la aproximación al valor

$$\chi \approx \frac{m_1 + m_2 \cdot exp\left(\frac{-\overline{\Delta f}^+}{c}\right)}{m_1 + m_2} \implies c = \frac{\overline{\Delta f}^+}{\ln\left(\frac{m_2}{m_2 \cdot \chi - m_1(1-\chi)}\right)} \tag{2.36}$$

Para calcular el valor inicial del parámetro de control primero se fija $c_0(0) = 0$ y, se generan m_0 transiciones. Tras cada transición o paso se genera un nuevo valor usando la ecuación (2.36) donde $\chi = \chi_0$ es la tasa de aceptación inicial (≈ 0.8).

Los valores de m_1 y m_2 se corresponden con el número de transiciones que contribuyen a reducir o aumentar el costo respectivamente hasta el momento considerado, siendo $m_0 = m_1 + m_2$. Gracias a experimentos numéricos, se sabe que aplicando este procedimiento, se obtiene la convergencia a un valor final adecuado siempre y cuando los estados del conjunto de estados S sean fácilmente alcanzables en cada transición, esto es, la distancia entre ellos no debe ser muy grande ni deben existir entornos en los que haya una gran concentración de estados. Finalmente, el valor obtenido es el valor inicial del parámetro de control considerado.

2.3.2. Disminución del parámetro de control

Primera propuesta: Suelen considerarse disminuciones del parámetro de control pequeñas tomando funciones lineales de la forma $c_{k+1} = \alpha \cdot c_k$, donde $k \in \mathbb{N}^+$ se refiere al paso k-ésimo del algoritmo, y α es el *ratio de enfriamiento*. Dicho parámetro suele tomar un valor próximo a 1, esto es, $\alpha \in [0.8, 0.99]$.

Segunda propuesta: En este caso se relaja la condición del cuasi-equilibrio dada en la Definición 2.37 y se sustituye por la siguiente:

$$\| \boldsymbol{q}(c_k) - \boldsymbol{q}(c_{k+1}) \| < \epsilon, \qquad \forall\, k \geq 0 \; y \; \epsilon > 0 \; (pequeño),$$

esto es, se busca que

$$\frac{1}{1+\delta} < \frac{q_i(c_k)}{q_i(c_{k+1})} < 1 + \delta \qquad \forall\, k \geq 0, \;\; \forall\, i \in S \;\; y \; \delta > 0 \; (pequeño). \tag{2.37}$$

Así, se estará cada vez más cerca de la distribución estacionaria en cada transición o paso entre valores distintos del parámetro de control (se logra el cuasi-equilibrio para cada cadena considerada). Dicha condición también debe cumplirla el valor inicial del parámetro de control c_0, y de hecho, fue elegido para verificarla.

Para garantizar que se cumple la expresión (2.37), es necesario que se den ciertas condiciones, que se estudian a continuación.

Teorema 2.38. *Sea $\boldsymbol{q}(c_k)$ la distribución estacionaria para una cadena de Markov homogénea asociada al Simulated Annealing cuyas componentes vienen dadas por la expresión (2.18) y tal que $c_k > c_{k+1}$ (valores sucesivos del parámetro de control). Si se cumple que:*

$$\frac{exp\,\left(\frac{-\delta_i}{c_k}\right)}{exp\,\left(\frac{-\delta_i}{c_{k+1}}\right)} < 1 + \delta \qquad \forall\, k \geq 0, \; \forall\, i \in S, \tag{2.38}$$

siendo $\delta_i = f(i) - f_{opt}$ entonces se cumple la expresión dada en (2.37).

Demostración: Se sabe por las expresiones dadas en (2.18) que:

$$q_i(c) = \frac{1}{N_0(c)} exp\left(\frac{-f(i)}{c}\right), \;\; \forall\, i \in S \quad \text{donde} \quad N_0(c) = \sum_{j \in S} exp\left(\frac{-f(j)}{c}\right).$$

Multiplicando las dos expresiones anteriores por $exp\left(\frac{f_{opt}}{c}\right)$ y sustituyendo c por c_k se obtiene lo siguiente:

$$q_i(c_k) = \frac{1}{M_0(c_k)} exp\left(\frac{-\delta_i}{c_k}\right), \quad \forall\, i \in S \quad \text{donde} \quad M_0(c_k) = \sum_{j \in S} exp\left(\frac{-\delta_j}{c_k}\right).$$

Como por hipótesis $c_k > c_{k+1}$ se tiene que $M_0(c_k) \geq M_0(c_{k+1})$ $(*)$, y en consecuencia, se obtienen las desigualdades dadas en (2.37), $\forall\, i \in S$ pues:

$$q_i(c_k) = \frac{exp\left(\frac{-\delta_i}{c_k}\right)}{M_0(c_k)} \overset{(*)}{\leq} \frac{exp\left(\frac{-\delta_i}{c_k}\right)}{M_0(c_{k+1})} \overset{(2.38)}{<} (1+\delta)\, \frac{exp\left(\frac{-\delta_i}{c_{k+1}}\right)}{M_0(c_{k+1})} = (1+\delta)\, q_i(c_{k+1})$$

$$q_i(c_k) = \frac{exp\left(\frac{-\delta_i}{c_k}\right)}{M_0(c_k)} \geq \frac{exp\left(\frac{-\delta_i}{c_{k+1}}\right)}{M_0(c_k)} \overset{(2.38)}{>} \frac{exp\left(\frac{-\delta_i}{c_{k+1}}\right)}{\sum_{j \in S}(1+\delta) exp\left(\frac{-\delta j}{c_{k+1}}\right)} = \frac{q_i(c_{k+1})}{(1+\delta)}$$

□

Como consecuencia de este teorema, se puede saber cuál debe ser el valor del parámetro de control c_{k+1} en función del de c_k, ya que basta con despejar su valor de la expresión (2.38) y realizar las sustituciones oportunas. Se obtiene que:

$$c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{f(i) - f_{opt}}}, \quad \forall\, i \in S\ y\ \forall\, k \geq 0 \tag{2.39}$$

Sin embargo, esta expresión puede simplificarse algo más si se restringe la condición dada en (2.38) al conjunto de soluciones que ocurren con mayor probabilidad durante la generación de la k-ésima cadena de Markov. Para ello, es necesario introducir el concepto de coste medio y varianza de costes considerados en la Definición 2.26 y considerar el conjunto $S_{c_k} = \{i \in S : f(i) - f_{opt} \leq E_{c_k}(f) - f_{opt} + 3\sigma_{c_k}\}$.
Gracias a estudios realizados sobre el comportamiento de la distribución de probabilidades de los valores de coste de las soluciones, se sabe que la probabilidad de que una solución obtenida en la k-ésima cadena de Markov pertenezca a ese conjunto es próxima a 1. Por ello, teniendo en cuenta lo anterior y realizando las sustituciones oportunas, se tiene que:

$$c_{k+1} > \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{E_{c_k}(f) - f_{opt} + 3\sigma_{c_k}}}, \quad k \geq 0 \tag{2.40}$$

Como además, en la mayoría de las ocasiones el valor de f_{opt} es desconocido y el comportamiento frente al parámetro de control de la media y la varianza es similar, se puede sustituir $E_{c_k}(f) - f_{opt} + 3\sigma_{c_k}$ por $3\sigma_{c_k}$. Para que la eliminación del término $E_{c_k}(f) - f_{opt}$ no afecte en la correcta elección habrá que considerar valores muy pequeños de δ, de manera que la disminución del valor del parámetro de control vendrá determinado por la elección de dicho valor, conocido como *parámetro de distancia.*

$$c_{k+1} = \frac{c_k}{1 + \frac{c_k \ln(1+\delta)}{3\sigma_{c_k}}}, \ \text{siendo}\ \ c_{k+1} > c_k, \ \ k \geq 0 \tag{2.41}$$

Por lo tanto, si se toman valores pequeños de δ se tendrá que los decrementos del parámetro de control serán más pequeños (es lo recomendable), mientras que para valores grandes, el decrecimiento será más rápido.

2.3.3. Valor final del parámetro de control

Primera propuesta: Se considera como criterio de parada, que el coste de la función objetivo obtenido en el último paso de una de las cadenas de Markov, no se vea modificado para un determinado número de cadenas consecutivas.

Segunda propuesta: En este caso se hace uso de la extrapolación del valor medio del coste $E_{c_k}(f)$ cuando $c_k \overset{\text{k}\to\infty}{\longrightarrow} 0$.
Para ello, se va a tener en cuenta una consideración sobre la que no se va a profundizar que es la siguiente: Usando el Teorema Central del Límite y la Ley de los Grandes Números se sabe que el coste esperado $E_c(f)$ y la desviación estándar de coste σ_c pueden ser aproximadas por $\bar{f}(c)$ y $\sigma(c)$ donde,

$$\bar{f}(c) = \frac{1}{L}\sum_{i=1}^{L} f_i(c) \quad \text{y} \quad \sigma(c) = \left(\frac{1}{L}\sum_{i=1}^{L}(f_i(c) - \bar{f}(c))^2\right)^{\frac{1}{2}} \tag{2.42}$$

siendo L el número de soluciones generadas para un valor determinado de c.

Si se considera $\triangle E_c(f) = E_c(f) - f_{opt}$, de manera teórica es claro que el algoritmo parará cuando dicho valor sea pequeño comparado con $E_{c_0}(f)$ (coste esperado en c_0) y, por el Corolario 2.27 se sabe que cuando c_0 es grande, $E_{c_0}(f) \approx E_\infty(f)$. Para valores de c mucho menores que 1, se puede considerar la siguiente aproximación:

$$\triangle E_c(f) \approx c\,\frac{\partial E_c(f)}{\partial c} = c\frac{\sigma_c^2}{c^2} = \frac{\sigma_c^2}{c}.$$

Por lo tanto, de manera teórica, el criterio de parada de dicho algoritmo se basa en encontrar un k tal que:

$$\frac{c_k}{E_\infty(f)}\frac{\partial E_c(f)}{\partial c}\bigg|_{c=c_k} < \epsilon_s, \quad \text{siendo } \epsilon_s > 0 \text{ y pequeño.} \tag{2.43}$$

ϵ_s suele denominarse *parámetro de parada*.

Sin embargo, dado que los valores anteriores no son fáciles de determinar, en la práctica se consideran las aproximaciones dadas en (2.42). En este caso el valor $\bar{f}(c_k)$ usado para aproximar $E_{c_k}(f)$ puede variar mucho, sobretodo cuando el valor del parámetro de control c_k es grande. Esto, puede provocar que se satisfaga muy pronto el criterio de parada dado en (2.43), provocando que el algoritmo pare antes de haber logrado converger al conjunto de soluciones óptimas S_{opt}.
Para evitar este inconveniente, se suele suavizar el valor de $\bar{f}(c_k)$ sustituyendo cada par de datos $(c_k, \bar{f}(c_k))$ por $(c_k, f_s(c_k))$, siendo $f_s(c_k)$ la media para un conjunto de valores consecutivos de $\bar{f}$ en torno a c_k.

2.3.4. Longitud de las cadenas de Markov

Primera propuesta: Su elección se basa en lograr que para cada valor c_k considerado se logre recuperar el cuasi-equilibrio. Por ello, si para una determinada cadena

el valor de c_k disminuye bastante, provocará que en la siguiente sean necesarias más transiciones para lograr el estado de cuasi-equilibrio con c_{k+1}, y por lo tanto, llevará más tiempo. En consecuencia, se tendrá que $L_k \overset{k\to\infty}{\longrightarrow} \infty$ cuando $c_k \overset{k\to\infty}{\longrightarrow} 0$, por lo que la longitud de las cadenas suele limitarse por una constante, para evitar así que crezcan mucho cuando el parámetro de control se aproxime a cero.

Segunda propuesta: Se sabe que si la disminución del valor del parámetro de control es la considerada con anterioridad, con un número de transiciones relativamente pequeño, la aproximación a la distribución estacionaria será mayor para el siguiente valor del parámetro de control. Por lo tanto, en este caso, se trata de encontrar el valor de k (número de transiciones) necesario para visitar la mayor parte de los estados vecinos de un determinado estado o solución.

Teorema 2.39. *Dado el conjunto* $\mathbb{S}$ *de cardinalidad S, se tiene que la cantidad esperada de elementos diferentes elegidos de* $\mathbb{S}$ *en N muestras aleatorias con repetición de elementos de* $\mathbb{S}$ *para valores de S y N grandes, es aproximadamente* $1 - exp(-\frac{N}{S})$.

Demostración: Se sabe que la cantidad esperada de elementos diferentes elegidos de $\mathbb{S}$ en N muestras equivale a la probabilidad de elegir un determinado elemento de $\mathbb{S}$ en N muestras.
Por lo tanto, dicha probabilidad vendrá dada por: $\mathbb{P} = \left[1 - \left(1 - \frac{1}{S}\right)^N\right]$

y, tomando límites en infinito para N y S se tiene que:

$$\lim_{N\to\infty}\left(\lim_{S\to\infty} 1 - \left(1 - \frac{1}{S}\right)^N\right) = 1 - exp\left(\lim_{N\to\infty}\frac{-N}{S}\right) = 1 - e^{-x}, \text{ con } x = \frac{N}{S}.$$

□

Se considera en este caso, que $L_k = L = \Theta \ \forall\, k$, esto es, la longitud de las cadenas de Markov va a ser igual al tamaño del entorno de vecinos que se considera constante. Se considera que el *Simulated Annealing* comienza generando una cadena de Markov de longitud L a partir de un estado i, y que durante la generación de la cadena de Markov no se acepta ninguna transición. Por lo tanto, según el Teorema 2.39, siendo $S = \Theta$ y $N = L = \Theta$, se tiene que las soluciones que siendo entorno de i serán visitadas es $1 - e^{-1}$. En las sucesivas cadenas de Markov, este valor se aproximará a 1, de manera que prácticamente todas las soluciones del entorno serán visitadas.

Por lo tanto, la segunda propuesta está determinada por la elección que se realice de los parámetros: χ_0, δ y ϵ_s.

Para finalizar, el siguiente teorema garantiza que el número de transiciones necesarias para realizar esta segunda propuesta, está limitado por $\mathcal{O}(\ln|S|)$.

Teorema 2.40. *Se considera la función de disminución para el parámetro de control como:*

$$c_{k+1} = \frac{c_k}{1 + \alpha_k c_k}, \ \ \forall\, k \ \ donde \ \ \alpha_k = \frac{\ln(1+\delta)}{3\sigma_{c_k}}, \ \ \forall\, k \tag{2.44}$$

y sea K el primer entero para el cual se satisface el criterio de parada (expresión (2.43)). Entonces se tiene que $K = \mathcal{O}(\ln|S|)$.

Demostración: Primero se expresa el número de transiciones K como una función de c_K (valor final del parámetro de control).

Por inducción se tiene que:

$$c_k \leq \frac{c_0}{1 + k\alpha c_0}, \quad \forall\, k, \text{ donde } \alpha = \min_k \alpha_k. \tag{2.45}$$

Es claro que se cumple para $k = 0$. Se supone que se cumple para k y se debe ver que es cierto para $k + 1$.

$$c_{k+1} = \frac{1}{\frac{1}{c_k} + \alpha_k} \overset{(2.45)}{\leq} \frac{1}{\frac{1+k\alpha c_0}{c_0} + \alpha} = \frac{c_0}{1 + (k+1)\alpha c_0}$$

De esta manera, ya quedaría probada dicha desigualdad $\forall\, k$. Como consecuencia, se obtiene que:

$$K \leq \frac{c_0 - c_K}{\alpha c_0 c_K} < \frac{1}{\alpha c_K} \overset{(2.44)}{=} \frac{3\, \max_k(\sigma_{c_k})}{\ln(1+\delta) c_K} \approx \frac{3\, \sigma_\infty}{\ln(1+\delta) c_K} \tag{2.46}$$

Ahora se debe probar que $K = \mathcal{O}(\ln |S|)$: Para ello se hará uso de la Definición 2.26 y el Corolario 2.27 pues se sabe que:

$$\frac{\partial}{\partial c} E_c(f) = \frac{\sigma_c^2}{c^2} \;\; ; \;\; \frac{\partial}{\partial c} \mathbb{S}_c = \frac{\sigma_c^2}{c^3} \implies \frac{\partial}{\partial c} E_c(f) = c \frac{\partial}{\partial c} \mathbb{S}_c \tag{2.47}$$

También se sabe que $\mathbb{S}_\infty = \ln |S|$ y $\mathbb{S}_0 = \ln |S_{opt}|$. Por la expresión dada en (2.43) se tiene que:

$$\exists\, \epsilon' \in (0, \epsilon] : \epsilon' = \frac{c_K}{E_\infty(f)} \frac{\partial E_c(f)}{\partial c}\bigg|_{c=c_K} \overset{(2.47)}{\implies} \epsilon' = \frac{c_K^2}{E_\infty(f)} \frac{\partial \mathbb{S}_c}{\partial c}\bigg|_{c=c_K}$$

Como c_K es mucho menor que 1,

$$\frac{\partial \mathbb{S}_c}{\partial c}\bigg|_{c=c_K} \approx \frac{\mathbb{S}_K - \mathbb{S}_0}{c_K}, \text{ por lo que:} \tag{2.48}$$

$$\epsilon' \overset{(2.48)}{=} \frac{c_K^2}{E_\infty(f)} \frac{\mathbb{S}_K - \ln |S_{opt}|}{c_K} < \frac{c_K \ln |S|}{E_\infty(f)} \implies c_K > \frac{E_\infty(f) \epsilon'}{\ln |S|}$$

Sustituyendo en (2.46) se obtiene que:

$$K < \frac{\ln |S|}{\alpha E_\infty(f) \epsilon'} \overset{(2.44)}{\approx} \frac{3\, \sigma_\infty \ln |S|}{\ln(1+\delta) E_\infty(f) \epsilon'}$$

Como consecuencia, el número total de transiciones necesarias es del orden de $\mathcal{O}(\ln |S|)$, pues los otros términos son constantes. □

Por lo tanto, si se realizan las elecciones de los parámetros según lo descrito, el tiempo de ejecución del algoritmo es del orden de $\mathcal{O}(\tau \cdot L \cdot \ln |S|)$, donde:

- L: Es la longitud de las cadenas de Markov.
- τ: Es el tiempo para realizar una transición.
- $\ln |S|$: Es el límite superior de cadenas de Markov consideradas.

L y τ pueden elegirse de forma polinomial en el problema considerado, por lo tanto, el tiempo de ejecución será polinomial si $\ln |S|$ se elige para que lo sea.

Conclusiones:

Podemos decir que la primera alternativa es más sencilla de realizar que la segunda, donde la principal diferencia viene dada por la manera en la que se realiza la disminución del parámetro de control c y por cómo se determina la longitud de las cadenas de Markov consideradas. Mientras que para la primera alternativa la longitud de las cadenas de Markov es variable (depende del momento en el que se encuentre la ejecución del algoritmo) y el decremento del parámetro de control es fijo, para la segunda, dicho decremento es variable pero la longitud de las cadenas es fija.

Por otro lado, cabe destacar la diferencia existente entre el valor final del parámetro de control en cada alternativa, ya que, mientras que en la primera, la ejecución del algoritmo termina si el coste de la función objetivo no mejora para un cierto número de cadenas de Markov consecutivas, en la segunda se basa en realizar extrapolaciones del coste medio de los diferentes estados a lo largo de una serie de cadenas de Markov consecutivas.

Por todo ello, la primera alternativa es adecuada cuando se requiere sencillez pues se basa en la aplicación de una serie de reglas empíricas bastante simples. Por otro lado, la segunda alternativa es algo más elaborada y compleja ya que sigue un proceso más teórico, y en consecuencia, más difícil de realizar.
Ambas propuestas se realizan en tiempo de ejecución finito, y además, se sabe que la segunda se realiza en tiempo polinomial del orden $\mathcal{O}(\tau \cdot L \cdot \ln |S|)$. Sin embargo, aunque la segunda alternativa sigue un procedimiento más elaborado que la primera, presenta el inconveniente de no poder determinar con certeza la distancia existente entre la solución obtenida tras su aplicación y la solución óptima.

Gracias a estudios realizados, se sabe que los resultados obtenidos siguiendo ambas alternativas son bastante buenos, siendo para el caso de la segunda alternativa más cercanos a la solución óptima, lo cual, se podía esperar dado que el procedimiento en el que se basa esta última es más elaborado.

Capítulo 3

Simulated Annealing para la resolución de ejemplos generales

En el presente capítulo se pretende explicar y mostrar de manera visual algunas de las aplicaciones que tiene el algoritmo *Simulated Annealing*, ya que, además de resolver el problema del TSP se aplica a la resolución de muchos más problemas. Su amplia aplicación se debe a que es un método general de optimización combinatoria que se basa en optimizar una determinada función objetivo (tanto maximizarla como minimizarla) sujeto a una serie de restricciones.

En el caso en que dichas restricciones deban ser satisfechas por una determinada solución es recomendable dividir el conjunto de soluciones S en dos: el conjunto de soluciones viables, esto es, aquellas que satisfacen las restricciones y el conjunto que no las satisface. El conjunto de soluciones viables se denota por $S^{'}$.
En un problema de minimización el objetivo será encontrar una solución óptima $i_{opt} \in S^{'}$ tal que $f(i_{opt}) \leq f(i),\ \forall\, i \in S^{'}$.

A continuación, se van a considerar algunas aplicaciones de dicho método, de manera que unas serán descritas de forma teórica, y otras se ilustrarán mediante ejemplos prácticos. La máquina usada para realizar las distintas pruebas es un ordenador portátil ACER Aspire 5 A515-51G-751G con un procesador Intel Core i7-7500U, cuya velocidad va de 2.7 GHz a 3.5 GHz, con memoria RAM de 8 GB y Sistema Operativo Windows 10 Home.

3.1. Problema del corte máximo

Es un problema NP-Duro cuyo objetivo es buscar a partir de un grafo $G = (V, A)$ con valores positivos en las aristas, una partición disjunta de V en V_0 y V_1 tal que la suma de los pesos de las aristas de A que tienen un vértice en V_0 y otro en V_1 sea máxima.

En este caso se tiene que:

$S = \{V_0, V_1 : V = V_0 \dot{\bigcup} V_1\}$, esto es, todas las posibles particiones disjuntas del conjunto V en los conjuntos V_0 y V_1.

La función de coste u objetivo f a maximizar viene dada por:

$$f(V_0, V_1) = \sum_{(v_0,v_1) \in \delta(V_0,V_1)} w(a_{01}),$$

donde $w(a_{01})$ es el valor de la arista que une el vértice $v_0 \in V_0$ con $v_1 \in V_1$ y $\delta(V_0, V_1)$ el corte de la partición de V en V_0 y V_1, esto es,
$\delta(V_0, V_1) = \{a_{01} \in A : v_0 \in V_0 \; y \; v_1 \in V_1\}$

El mecanismo para generar nuevos estados o soluciones consiste en elegir de manera arbitraria un vértice $v_i \in V$ y moverlo de V_0 a V_1 si $v_i \in V_0$ o viceversa.
Por lo tanto, la diferencia de coste viene dada por:

$$\Delta f = \sum_{(v_i,v_j)\in A\setminus\delta(V_0,V_1)} w(a_{ij}) - \sum_{(v_i,v_j)\in\delta(V_0,V_1)} w(a_{ij})$$

A continuación, se muestra en la Figura 3.1 un ejemplo del problema descrito donde $V_0 = \{v_0, v_2, v_4, v_6\}$ y $V_1 = \{v_1, v_3, v_5\}$ siendo 1 el peso de todas las aristas. En este caso, dicha partición es máxima siendo el valor de la función objetivo 9 pues es el número de aristas en común entre V_0 y V_1 ya que el peso de todas ellas es 1.

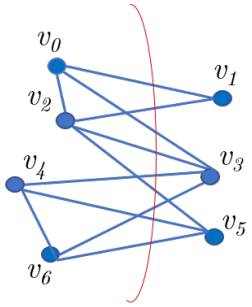


Figura 3.1: Corte máximo

3.2. Problema del conjunto independiente

Es un problema NP-Duro cuyo objetivo es buscar a partir de un grafo $G = (V, A)$ el mayor subconjunto $V^{'} \subset V$ tal que $\forall\, v_i, v_j \in V^{'}, a_{ij} \notin A$.

En este caso se tiene que:

$S = \{V^{'} : V = V^{'} \dot{\bigcup} V \setminus V^{'}\}$, esto es, el conjunto de todas las posibles particiones disjuntas de V en $V^{'}$ y $V \setminus V^{'}$ siendo $S^{'}$ el conjunto de todas las particiones tal que $\forall\, v_i, v_j \in V^{'}, a_{ij} \notin A$.

La función de coste u objetivo f que hay que maximizar viene dada por

$$f(V^{'}) = |V^{'}| - \alpha|A^{'}|$$

siendo $A^{'} = \{a_{ij} \in A : v_i, v_j \in V^{'}\}$ y α un factor de ponderación mayor que 1.

Se puede observar de la expresión anterior, que las soluciones viables sólo contribuyen en el primer término de la función de coste, mientras que las inviables (no cumplen las restricciones) también contribuyen al segundo término, lo cual, reduce el coste de las particiones que tengan la misma cardinalidad. Se consideran ambos tipos de soluciones ya que ayuda a que el algoritmo converja hacia el conjunto de soluciones óptimas S_{opt} permitiendo escapar de óptimos locales.

El mecanismo de generación de nuevas soluciones se basa en elegir de manera arbitraria un vértice $v_i \in V$ y moverlo de V' a $V \setminus V'$ si $v_i \in V'$ o viceversa.
Por lo tanto, la diferencia de coste viene dada por:

$$\Delta f = (\chi_{V \setminus V'}(v_i) - \chi_{V'}(v_i)) \left(1 - \alpha \sum_{(v_i, v_j) \in A, v_j \in V'} 1 \right)$$

En la Figura 3.2 se muestra para el caso del cubo 6 conjuntos independientes (los dos primeros máximos), donde V' está formado por los vértices en color amarillo.

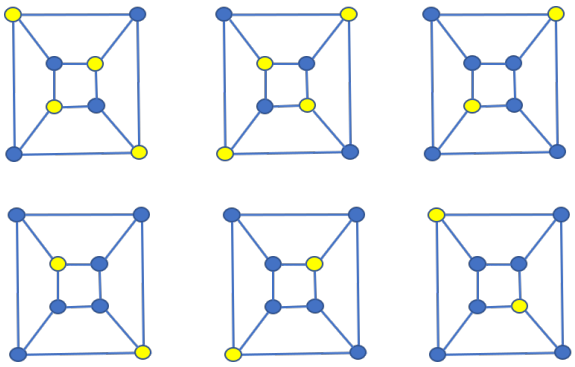


Figura 3.2: Conjuntos independientes del cubo

3.3. Píxeles de una imagen

Se trata de un modelo donde la función de energía considera como parámetro de entrada una imagen en blanco y negro. La energía viene definida de manera local para cada píxel de la imagen que se representa mediante un cuadrado que puede ser negro o blanco. La función de energía global es la suma de la energía de cada uno de los píxeles. En este caso, el objetivo es minimizar la energía global del sistema teniendo en cuenta que esta se minimiza cuando para cada píxel, el color del píxel superior e inferior es el mismo pero distinto del situado a su izquierda y su derecha.

La imagen inicial que representa el estado de partida contiene la misma cantidad de píxeles blancos y negros, siendo estos dispuestos de manera aleatoria. Por lo tanto, es claro que un estado óptimo consiste en obtener líneas alternas verticales de color blanco y negro. Para buscar la solución óptima se hace uso del algoritmo *Simulated Annealing* mediante el control (principalmente) de la temperatura ya que va a determinar el número de cambios a realizar y si se aceptan los estados generados o no

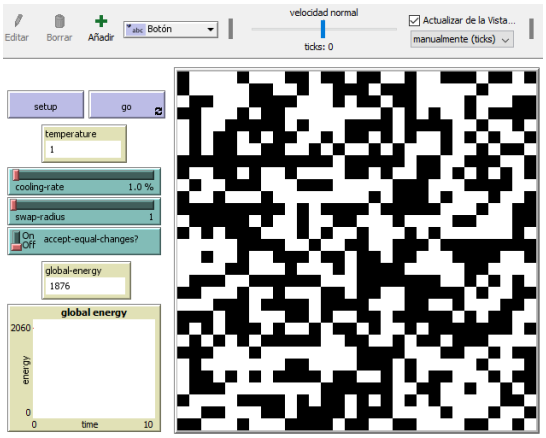
(tal como se explicó en el Capítulo 2).

A continuación, se ilustra mediante imágenes lo explicado con anterioridad, donde para ello, se ha hecho uso del programa *NetLogo*.
En las siguientes gráficas se ve que en función del valor de los parámetros considerados, la solución obtenida es diferente. Se tienen diversos parámetros a controlar:

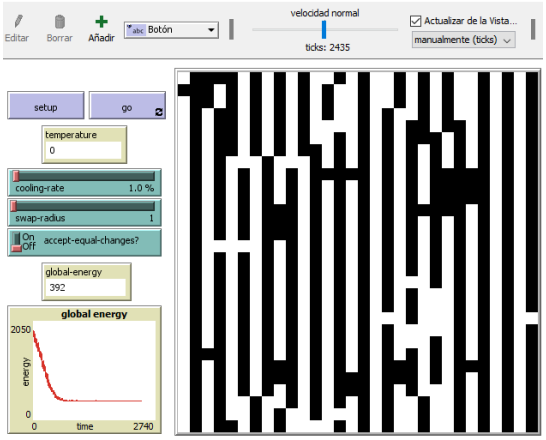
cooling-rate: Permite ajustar la velocidad con la que disminuye la temperatura. En todos los casos se fija al mismo valor para que dicha disminución sea lenta pues es una condición necesaria para garantizar la convergencia del algoritmo.

swap-radius: Permite controlar lo lejos que puede estar un píxel de la imagen de otro para poder ser intercambiado con él.

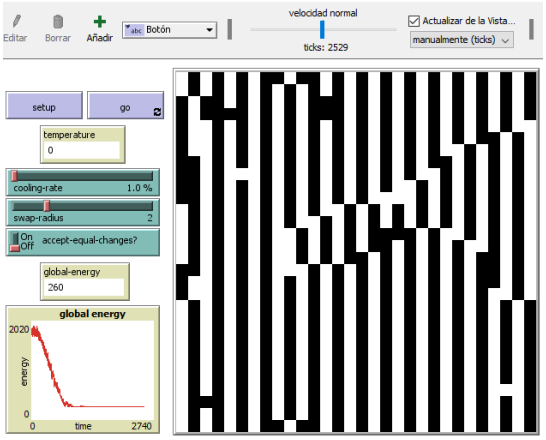
accept-equal-changes?: Puede tomar el valor ON/OFF. En el caso de que esté activo (valor ON) se aceptan nuevos estados aunque la energía del sistema no varíe al pasar de un estado a otro.



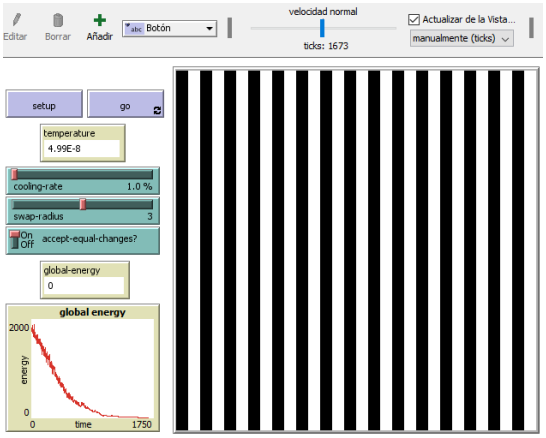
(a) Estado inicial



(b) Estado final con swap-radius = 1



(c) Estado final con swap-radius = 2



(d) Estado final con swap-radius = 3

Se puede observar que se obtienen mejores resultados cuando se aceptan estados que tienen la misma energía que el actual, así como cuando el valor de swap-radius es mayor ya que se realiza un mayor número de intercambios, esto es, se explora todo el espacio de soluciones S.

3.4. Aplicación al problema del viajante o TSP

Como ya se ha comentado en otras ocasiones, el algoritmo *Simulated Annealing* es aplicado para resolver el problema del viajante conocido como TSP. Por ello, para finalizar este capítulo se ha realizado un estudio del comportamiento de la mayoría de los algoritmos descritos en la Sección 1.6 del Capítulo 1 junto con el *Simulated Annealing*. Además, se llevará a cabo una comparativa de los mismos en función de los resultados obtenidos en cuanto a distancia recorrida y tiempo de ejecución.

Para ello, se ha utilizado el programa *R-Studio*, así como una serie de paquetes y comandos necesarios, que no serán detallados en el trabajo.
Conviene mencionar que esta parte del trabajo se ha realizado de manera conjunta con Juan Manuel Velasco Heras, esto es, ambos hemos considerado los mismos datasets para realizar las pruebas, que han sido realizadas en la máquina cuyas características fueron comentadas al comienzo del capítulo. Por lo tanto, los resultados obtenidos son los mismos en ambos trabajos, y las conclusiones de cada uno similares.

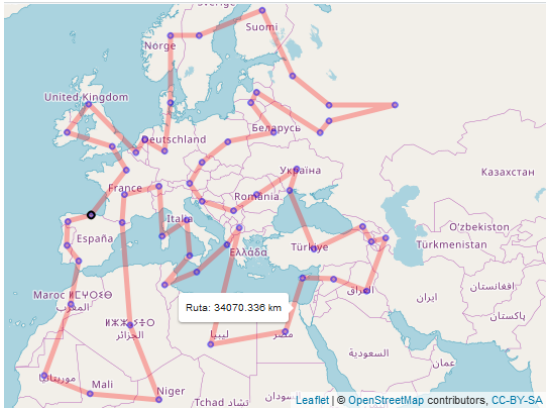
Primero, partimos de un dataset denominado *Ciudades* que consta de cuatro columnas (una es el nombre de la ciudad, otra es un ID con los tres primeros caracteres de la ciudad, la tercera es la longitud y la cuarta la latitud de la misma) y 55 filas, esto es, tenemos 55 ciudades.

Además, se ha considerado a la hora de realizar las diversas implementaciones, como distancia entre cada par de ciudades consideradas, la distancia del haversiano que ha sido calculada a través de la fórmula de Haversine. Esta fórmula sirve para calcular la distancia de círculo máximo entre dos puntos de la esfera sabiendo su longitud y su latitud.

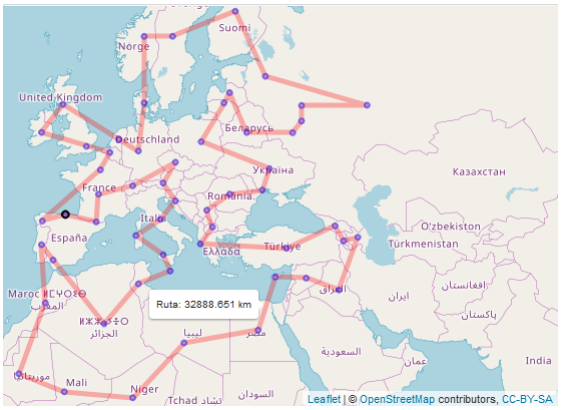
Tras ello, a dicho conjunto de datos, se le ha aplicado la resolución del TSP usando una serie de métodos y considerando como ciudad de partida (siempre que el método considerado permita dejarla fija) la primera del dataset, esto es, Santander. Los métodos considerados son los siguientes: *Inserción más cercana, más lejana, más barata y aleatoria, Vecinos más próximo y Vecinos más próximo repetitivo, 2-opt, Lin-Kernighan, Concorde* y *Simulated Annealing*.

Todos los métodos mencionados con anterioridad han sido descritos o considerados en el Capítulo 1 a excepción del método *Vecinos más próximo repetitivo*. Dicho método aplica el método vecinos más próximo partiendo de cada una de las diferentes ciudades, para así, quedarse con el recorrido que da el mejor resultado.
También conviene decir, que el método *Concorde* es un algoritmo bastante avanzado y preciso para resolver el TSP y se basa en técnicas de ramificación y corte. Por ello, es de esperar obtener muy buenos resultados al ser aplicado.
Todos estos métodos se encuentran dentro del paquete de *R-Studio* denominado *TSP* a excepción del algoritmo *Simulated Annealing*, por lo que ha sido necesario hacer uso del método *optim()* que se encuentra en el paquete *stats*, e ir variando los valores de las iteraciones y la temperatura elegida para ver con cuáles se obtenían mejores resultados.

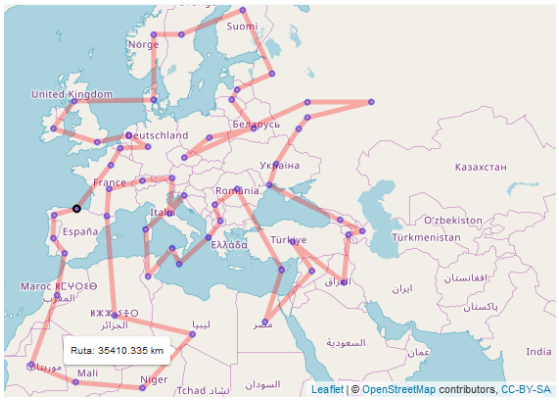
A continuación, se muestran una serie de gráficos que ilustran el camino seguido para resolver el TSP, para cada uno de los métodos considerados.
La ciudad de partida (que también es la de fin) viene marcada con un círculo negro, a diferencia del resto, que tienen color azul. También se muestra en dichas imágenes cuál es el recorrido total de la ruta en km para cada uno de los métodos usados.



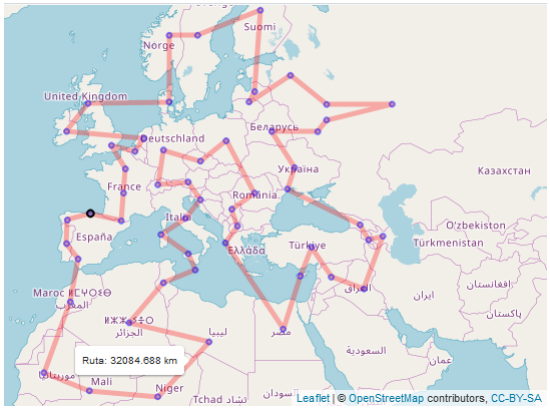
(e) *Inserción más cercana* (34070.34 km)



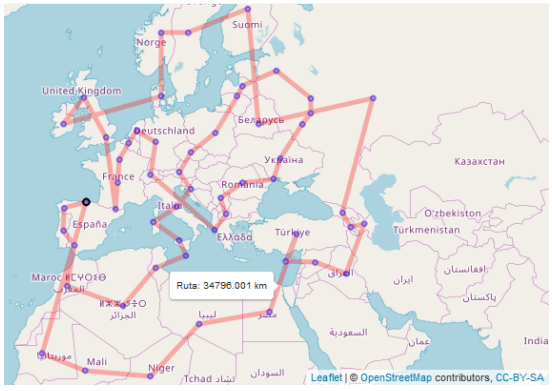
(f) *Inserción más lejana* (32888.65 km)



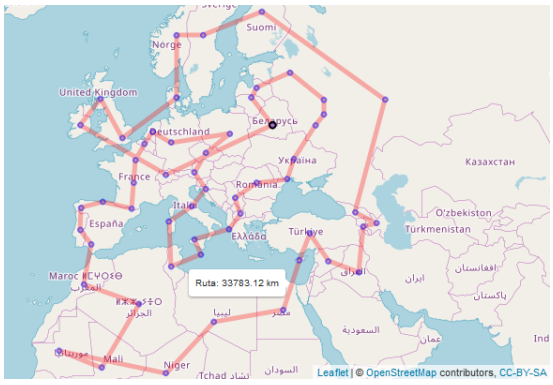
(g) *Inserción más barata* (35410.33 km)



(h) *Inserción aleatoria* (32084.69 km)

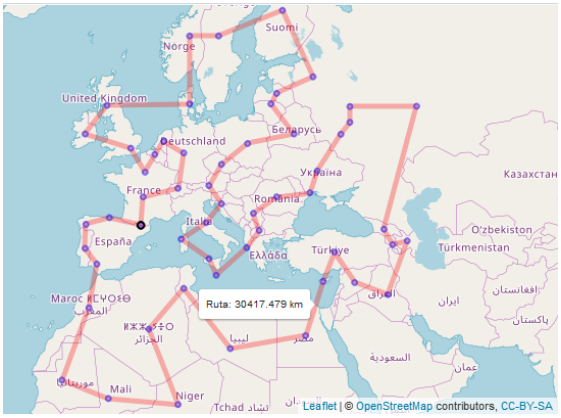


(i) *Vecinos más próximo* (34796.00 km)

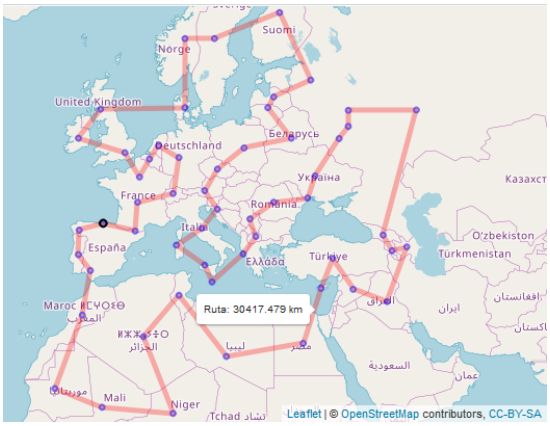


(j) *Vecinos más próximo repetitivo* (33783.12 km)

(k) *2-opt* (36857.00 km)

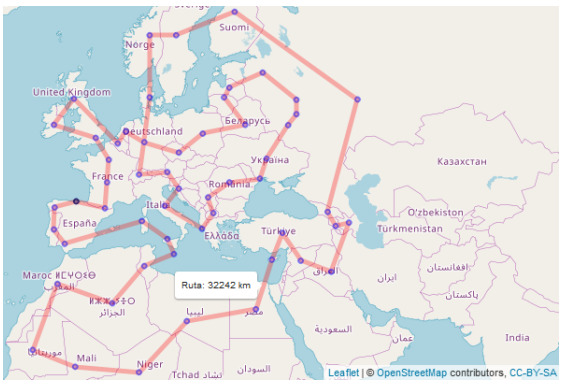


(l) *Lin-Kernighan* (30417.48 km)

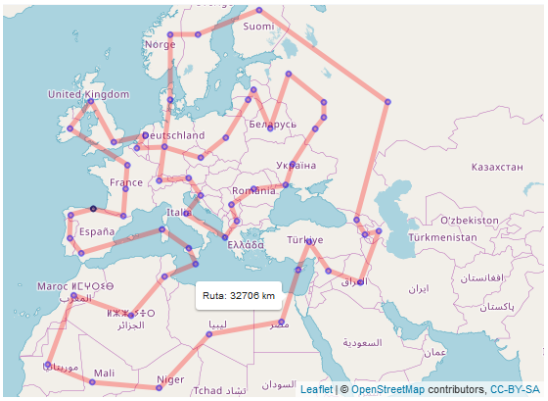


(m) *Concorde* (30417.48 km)

(n) *SA* con iter = 1000000 (32230.00 km)



(ñ) *SA* con iter = 100000 (32242.00 km)



(o) *SA* con iter = 40000 (32706.00 km)

Figura 3.3: Ruta ordenada tras aplicar el algoritmo de ejecución indicado

Para el caso del algoritmo *Simulated Annealing* se muestran tres imágenes diferentes que se corresponden con un número de iteraciones de 1000000, descendiendo a 100000, y finalmente, fijándolo en 40000.
Se puede observar que a mayor número de iteraciones realizadas, el valor del recorrido total de la ruta es menor y más próximo al valor óptimo. Sin embargo, el tiempo de ejecución aumenta considerablemente, por lo que, dado que en el caso de 100000

iteraciones sólo se obtienen 12 km a mayores con respecto a las 1000000 iteraciones, y el tiempo es mucho más reducido sería una mejor opción a considerar.

Tras ello, se han considerado otros dos datasets. Uno de ellos denominado *Ciudades1* que consta de 502 ciudades y otro denominado *Ciudades2* que consta de 1200 ciudades. Esto nos ha permitido comparar el funcionamiento de dichos métodos cuando se aumenta el número de ciudades consideradas para su resolución.

A continuación, se muestra una tabla comparativa de todos los algoritmos, teniendo en cuenta la distancia recorrida y el tiempo de ejecución para el caso de 55, 502 y 1200 ciudades.

Algoritmo de resolución	**55 ciudades**		**502 ciudades**		**1200 ciudades**	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**	**Distancia**	**Tiempo**
Inserción más cercana	34070.34	0.00	120688.81	0.79	181704.24	9.59
Inserción más lejana	32888.65	0.02	105201.13	0.89	160878.27	7.47
Inserción más barata	35410.33	0.03	115215.36	0.58	179368.30	6.42
Inserción aleatoria	32084.69	0.00	107423.11	0.02	162013.16	0.02
Vecinos más próximo	34796.00	0.00	130530.73	0.00	187372.14	0.05
Vecinos más próximo repetitivo	33783.12	0.07	118415.83	8.23	185668.79	84.14
2-opt	36857.00	0.00	108908.00	0.44	160585.00	8.37
Lin-Kernighan	30417.48	0.19	100692.08	3.37	148663.53	17.92
Concorde	30417.48	0.50	98438.31	13.74	147638.86	61.59
***SA* (40000 iteraciones)**	32706.00	2.70	105113.00	4.91	160878.00	13.54
***SA* (100000 iteraciones)**	32242.00	9.32	104987.00	10.60	160846.00	27.64
***SA* (1000000 iteraciones)**	32230.00	70.83	104512.00	105.89	160479.00	164.34

Tabla 3.1: Comparativa de los algoritmos en distancia (km) y tiempo de resolución (sg)

Según los resultados obtenidos, se puede decir que los algoritmos que mejor resultado obtienen en cuanto a distancia recorrida son el *Concorde* y el *Lin-Kernighan*. Además, si se tiene en cuenta el tiempo de ejecución de ambos, el mejor de ellos sería el *Lin-Kernighan* pues difiere muy poco de los resultados obtenidos con *Concorde* siendo su tiempo de ejecución bastante menor. Obtener resultados tan buenos para estos dos métodos era de esperar dado que se han diseñado para resolver este tipo de problemas.

Por otro lado, se ve que los algoritmos *Inserción más cercana*, *Inserción más barata* y *Vecinos más próximo* aunque tienen un buen tiempo de ejecución, el resultado obtenido sobre la distancia difiere bastante de los obtenidos con *Lin-Kernighan* y *Concorde*. Como es lógico, debido al procedimiento seguido por el algoritmo *Vecinos más próximo repetitivo*, los resultados obtenidos son mejores que los del algoritmo *Vecinos más próximo*, sin embargo, para ello se necesita un tiempo de ejecución bastante mayor. Otros como *Inserción más lejana* o *Inserción aleatoria* obtienen resultados medios y tienen la ventaja de que el tiempo de ejecución no es elevado.

Para el caso del algoritmo *2-opt*, los resultados obtenidos no son tan buenos como los de *Concorde* o *Lin-Kernighan*, y es debido principalmente a la disposición de las ciudades consideradas, pues en otros casos (como veremos en el Capítulo 4), dicho algoritmo obtiene muy buenos resultados (iguales o mejores que los otros dos) y tiene la ventaja de que el tiempo de ejecución es mucho menor. Por lo tanto, podemos

decir que mientras que los algoritmos *Lin-Kernighan* y *Concorde* se adaptan bien tanto a situaciones de partida buenas o malas, el algoritmo *2-opt* funciona bien pero para ello es necesario que la ruta de partida considerada no sea muy mala.

Para el algoritmo *Simulated Annealing* se han realizado pruebas para distintas iteraciones. Además, para el caso de 55 ciudades se ha usado una temperatura de 1500, mientras que, para los otros dos casos se ha aplicado primero el algoritmo *Inserción más lejana*, y tras ello, el *Simulated Annealing* con una temperatura de 20. El haber aplicado primero dicho algoritmo, se debe a que la relación que presenta en términos de distancia y tiempo de ejecución es relativamente buena, y así, al partir de una solución próxima a la óptima, *Simulated Annealing* logra mejorarla, pues sin aplicarlo, el resultado obtenido es mucho peor y se necesitarían muchas más iteraciones para lograr lo mismo.

En conclusión, centrándonos en el algoritmo *Simulated Annealing* que es nuestro método de estudio, se puede decir que es adecuado para resolver el problema del viajante, ya que en muchas ocasiones, logra mejorar la solución de partida aproximándose a la óptima (siempre y cuando la disminución del parámetro de control sea lo suficientemente lenta).

Sin embargo, como se ha podido ver existen otros algoritmos que son más adecuados para resolver este tipo de problema, y se debe principalmente, a que han sido creados específicamente para ello. Pero el *Simulated Annealing*, a diferencia del resto, puede ser aplicado para resolver múltiples problemas, algunos de los cuales fueron comentados al principio de esta sección, ya que, partiendo de un espacio de estados inicial, una función a optimizar, una función generadora de estados y una medida de la diferencia de coste entre soluciones o estados, dicho algoritmo logra optimizar cualquier función objetivo dada.

Capítulo 4

Comportamiento del *Simulated Annealing* y otras heurísticas frente al problema planteado

Antes de realizar la aplicación de los distintos algoritmos a las trayectorias de los vuelos proporcionados, se va a realizar una breve introducción del proyecto "AIRPORTS" con el que se está colaborando en la realización del presente trabajo, así como del problema que se presenta.

4.1. Problema planteado sobre la reordenación de trayectorias de vuelo

El presente trabajo busca resolver el problema sobre el alineamiento temporal de las trayectorias que describen las distintas aeronaves. Se enmarca dentro del contexto de "AIRPORTS" que es un proyecto liderado por Boeing Research and Technology Europe (BR&T-E) en el que participan varias instituciones españolas (como es el caso de la Universidad de Valladolid).

El objetivo de dicho proyecto es mejorar la eficiencia del sistema de vuelos mediante el análisis de las trayectorias que estos describen. Dichas trayectorias se construyen gracias a los distintos mensajes ADS-B que son transmitidos por las aeronaves durante el vuelo, los cuales, provienen de diferentes fuentes o proveedores.
Una vez transmitidos dichos mensajes, la entidad receptora se encarga de captarlos y asignarlos el tiempo de recepción, conocido como *timestamp*. Como consecuencia del retardo en el tiempo de recepción de dichos mensajes y debido a que los relojes de tiempo de los dispositivos receptores no se encuentran sincronizados, los tiempos asignados pueden no ser los correctos. Debido a ello, dichos mensajes se registran con un orden inadecuado, esto es, no es el mismo orden con el que fueron emitidos. Esta situación provoca que cuando se construye la trayectoria a partir de los distintos mensajes obtenidos, sea errónea.

Por ello, el propósito perseguido con la aplicación de las distintas heurísticas des-

critas en el Capítulo 1 al conjunto de vuelos de estudio, es poder obtener el camino hamiltoniano de mínima distancia para cada una de las trayectorias proporcionadas y realizar la nueva asignación de tiempos para aquellos puntos que han sido alterados tras la ordenación.

4.2. Aplicación de las distintas heurísticas

En esta sección, se van a aplicar las heurísticas descritas en el Capítulo 1 a ciertas trayectorias con el objetivo de mejorarlas encontrando la ruta de mínima distancia.

Para ello, se persigue resolver el problema de los caminos hamiltonianos de mínima distancia. Luego, si tenemos un conjunto de n datos, la resolución de dicho problema consiste en encontrar el recorrido más corto que una el punto o estado 1 con el punto o estado n (siendo estos fijados al principio), de manera que sólo se pase por cada estado en una única ocasión. Como la formulación de este problema no coincide exactamente con la del problema del TSP descrita en el Capítulo 1, es necesario realizar una adaptación del mismo. Para ello, se ha considerado la adición de un estado o punto auxiliar en la trayectoria que tiene distancia cero con el resto de estados considerados, y por lo tanto, permite obtener un camino cerrado.

Lo anteriormente descrito se puede visualizar en la Figura 4.1 que se muestra a continuación. Así, mediante la adición de dicha ciudad abstracta (marcada en color amarillo), se puede formar un circuito hamiltoniano de coste mínimo que es el objetivo para resolver el problema del viajante o TSP. En ella, los datos aparecen ordenados en función del tiempo de recepción de los mismos y viene indicado justo encima de cada punto.

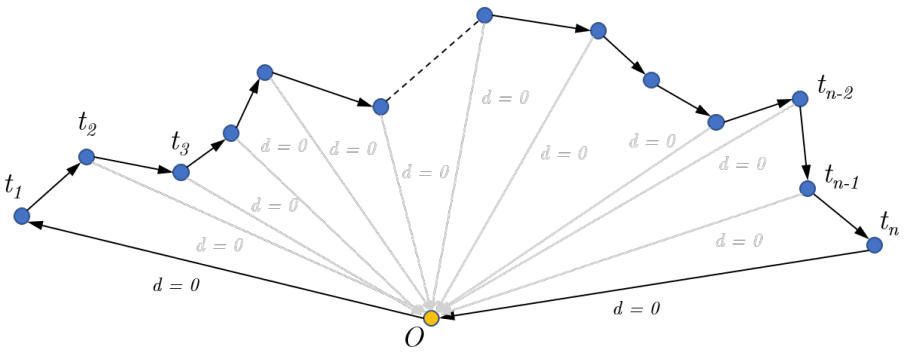


Figura 4.1: Recorrido en el que se añade una ciudad abstracta

Así, resolver el problema del camino hamiltoniano de mínima distancia entre el estado inicial 1 y el estado final n (considerando un conjunto de n datos y siendo estos estados fijados al comienzo), equivale a encontrar el circuito hamiltoniano de mínima distancia considerando la adición del estado auxiliar, siendo este el objetivo del propio problema TSP. Para que quede más claro, supongamos que se tiene el conjunto $D = \{d_i : i \in \{1, ..., n\}\}$, y que se añade el punto auxiliar denotado por O. Como la distancia entre dicho punto y el resto de puntos del conjunto D es 0, esto

es, $dist(d_i, O) = 0,\ \forall\ i \in \{1, ...n\}$, se puede aplicar el problema del TSP a dicho conjunto buscando encontrar el circuito hamiltoniano de coste mínimo.
Así, si se fija como punto inicial d_1 y como final d_n, se obtendrá un circuito de mínima distancia formado por $n+1$ puntos, por ejemplo vendrá dado según la secuencia de puntos siguiente: $d_1d_4d_2d_3....d_iOd_{i+2}...d_{n-2}d_n$. Ahora, para encontrar el camino hamiltoniano de mínima distancia entre los puntos inicial y final fijados, basta eliminar la arista que une los puntos d_i con O y la que une los puntos O con d_{i+2}, pues al ser la distancia entre ellos 0, el coste de la solución no se verá alterado. Luego, en este caso se obtendrá el camino hamiltoniano siguiente: $d_1d_4d_2d_3....d_id_{i+2}...d_{n-2}d_n$ cuya distancia será la misma que en el caso anterior. Este camino es la solución al problema del camino hamiltoniano de mínima distancia.

Para encontrar el camino hamiltoniano de coste mínimo de una serie de vuelos proporcionados, se ha usado al igual que en la Sección 3.4, el paquete *TSP* y el método *optim()* del paquete *stats*. Su uso en *R-Studio* permite disponer de una serie de algoritmos, que han sido aplicados a las distintas trayectorias de los vuelos obtenidas, y son los siguientes: *Inserción más cercana, más lejana, más barata y aleatoria, Vecinos más próximo y Vecinos más próximo repetitivo, 2-opt, Lin-Kernighan, Concorde* y *Simulated Annealing*. En este caso, a mayores de lo realizado en la Sección 3.4, para llevar a cabo las distintas pruebas y ejecuciones se plantean dos posibilidades diferentes: aplicar los algoritmos usando ventanas de tiempo o sin usarlas.

El motivo de poder elegir la opción de ejecutar los algoritmos usando ventanas de tiempo se debe a que al reducir el número de datos a los cuales se aplican los algoritmos, se mejoran los resultados obtenidos en distancia recorrida y tiempo de ejecución. Para el caso del *Simulated Annealing* como el número de iteraciones para cada ventana es fijo, el tiempo de ejecución es algo mayor cuando se consideran ventanas que cuando no se hace. Sin embargo, este aumento de tiempo no es relevante con respecto a la mejora en distancia obtenida. Además, se considera la posibilidad de elegir un solapamiento entre las distintas ventanas para así evitar aislar los datos entre pares de ventanas consecutivas. Dicho solapamiento se realiza entre los últimos elementos de una ventana y los primeros de la siguiente. También, es posible elegir tamaños de ventana y de solapamiento diferentes en las zonas del aeropuerto y en la zona central del vuelo, lo cual, se debe a que en las zonas del aeropuerto las trayectorias no son tan rectas como en la zona central, presentando diversos quiebros y zig-zag. Por lo tanto, en ocasiones, es conveniente reducir el tamaño de las ventanas en el aeropuerto y el número de datos de solapamiento para obtener mejores resultados.

A continuación, se explican dichas posibilidades de manera más formal:

1ª) Sin usar ventanas de tiempo: Esta posibilidad, permite resolver el problema considerado con el algoritmo indicado, el cual, es aplicado a todo el conjunto de datos del dataset.

2ª) Usando ventanas de tiempo: Con esta posibilidad, se consideran subconjuntos del conjunto de datos original, a los cuales, se les aplica el algoritmo indicado.

Para ello, será necesario considerar ventanas de tiempo de un cierto tamaño y con un determinado solapamiento entre ellas.

Formalizándolo en términos matemáticos: Sea $D = \{d_i : i \in \mathbb{N}\}$ un conjunto tal que d_i es la información asociada al mensaje ADS-B i recibido de un vuelo, de manera que $d_1 < d_2 < ... < d_n < ...$, esto es, los mensajes están ordenados de manera creciente en función del tiempo de recepción (*timestamp*). Se entiende por ventana de tiempo de tamaño t a un subconjunto de datos $D' \subset D$ tal que $|D'| = t < |D|$ (considerando que hay más de una ventana de tiempo pues sino $|D'| = t = |D|$).
Si se tienen n ventanas de tiempo consecutivas de tamaño t con solapamiento s entre ellas siendo $s \leq t$, entonces se tiene que: $D_1, ..., D_n \subset D$ de manera que $|D_1| = ... = |D_n| = t$ y $|D_i \cap D_{i+1}| = s$, $\forall\, i = 1, ..., n$, esto es, coinciden los s últimos elementos de la ventana D_i con los s primeros de D_{i+1}. Luego, si $D_i = \{d_{(i-1)t}, ..., d_{it-1}\}$ y $D_{i+1} = \{d_{it}, ..., d_{(i+1)t-1}\}$, entonces $d_{(it-1)-s} = d_{it}, ..., d_{it-1} = d_{it+s}$.
En la Figura 4.2 se detalla de manera gráfica la explicación anterior.

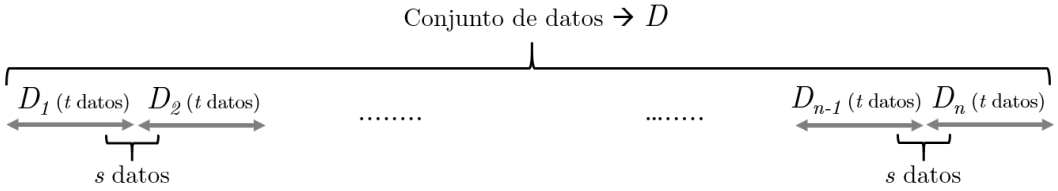


Figura 4.2: Usando ventanas de tiempo

Cuando se usen ventanas de tiempo se van a considerar dos posibles casos:

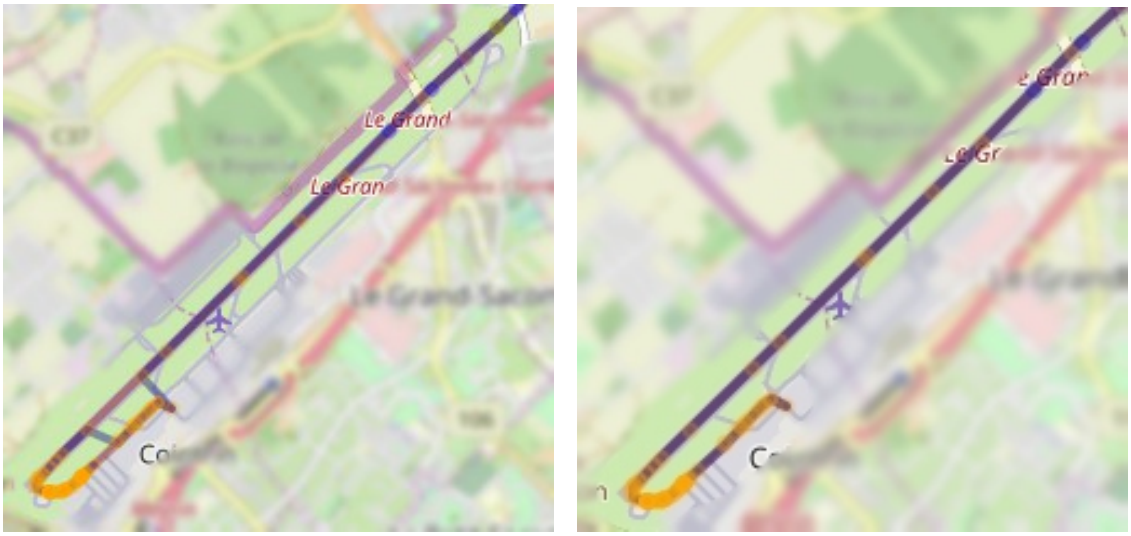
· El caso en que todas las ventanas son del mismo tamaño t con solapamiento s. En dicho caso, la aplicación del algoritmo debe realizarse sobre cada uno de las ventanas consideradas, donde el número de ejecuciones vendrá dado por:

$$f(t, s) = 1 + \left\lfloor \frac{|D| - s - 1}{t - s} \right\rfloor$$

· El caso en que se consideran diferentes tamaños de ventana. En este caso sólo se van a tener en cuenta dos tamaños diferentes, donde uno se corresponderá con las zonas de los aeropuertos, y el otro con la zona central del vuelo, siendo el motivo de esta diferenciación el comentado anteriormente. Se denota por t_v el tamaño de ventana en vuelo, t_a el tamaño de ventana en el aeropuerto, s_v el solapamiento en vuelo y s_a el solapamiento en el aeropuerto. Así, el algoritmo se aplica a cada uno de los diferentes trozos: en los dos trozos del aeropuerto, se aplica $f(t_a, s_a)$ veces y, en el trozo del vuelo se aplica $f(t_v, s_v)$ veces. Por lo tanto, se obtiene que el número de veces que se aplica el algoritmo es $2f(t_a, s_a) + f(t_v, s_v)$.

Dependiendo de si un vuelo presenta más problemas debido al fallo en el alineamiento temporal o menos, será recomendable aplicar una alternativa u otra.
A continuación, se muestran los resultados obtenidos de una serie de ejemplos realizados con diferentes vuelos y consideraciones.

Las dos imágenes siguientes 4.3a y 4.3b, sirven para ilustrar como en determinados vuelos cuyas trayectorias cerca de los aeropuertos presentan ciertos quiebros y zigzag es útil usar ventanas de tiempo. La presencia de dichas irregularidades se debe a que en las fases del aterrizaje y el despegue las aeronaves suelen realizar más maniobras que en la zona del vuelo, siendo en el vuelo las trayectorias más rectas. Se observa como en la imagen 4.3b se corrige la desviación presente en la trayectoria de la imagen 4.3a.



(a) Sin usar ventanas (b) Usando ventanas con $t_a = 10$ y $s_a = 2$

Figura 4.3: Trayectoria ordenada sin usar ventanas de tiempo o usándolas

Estas correcciones aunque resulten mínimas son muy importantes, pues gracias a ello, en vuelos largos se logra reducir la distancia total obtenida.
Este hecho se puede observar cuando se aplica la heurística de mejora *Simulated Annealing* al vuelo *IBE05DK* (temperatura 1 y 100000 iteraciones) cuya ruta sin aplicar ningún algoritmo es de 602.47 km. En el caso en que no se usan ventanas de tiempo se reduce dicha distancia a 590.95 km, y si se usan ventanas en la zona del aeropuerto (para aquellos datos con altura inferior a 6000) de tamaño 20 y solapamiento 5, se reduce a 582.79 km. Por lo tanto, se produce una disminución de la distancia obtenida en el caso en que se usan ventanas de tiempo en la zona del aeropuerto, llegando a reducirse en 8.16 kilómetros. Esto nos lleva a concluir que el uso de ventanas de tiempo resulta ventajoso y es aconsejable utilizarlo ya que se consiguen resultados más próximos a la solución óptima, esto es, la de mínima distancia.

A continuación, usando el ordenador cuyas características fueron detalladas en el Capítulo 3, se muestran dos tablas donde se realiza una comparativa de los distintos algoritmos comentados en términos de distancia recorrida y tiempo de ejecución. Conviene mencionar que para estas ejecuciones no se han usado ventanas de tiempo. Además, cabe destacar que al igual que en el Capítulo 3, dichas tablas se han realizado de manera conjunta con Juan Manuel Velasco Heras y son las mismas en ambos trabajos, pues se han usado los mismos vuelos y métodos para su realización.

Los vuelos considerados para realizar dicho estudio son 5 y son los siguientes:

IBE04HT: Vuelo de Madrid a Asturias que tiene 1090 mensajes ADS-B.

IBE04NL: Vuelo de Asturias a Madrid que tiene 1151 mensajes ADS-B.

IBE0519: Vuelo de A Coruña a Madrid que tiene 1105 mensajes ADS-B.

IBE05DK: Vuelo de Madrid a A Coruña que tiene 1079 mensajes ADS-B.

RYR9KY_4CA97C: Vuelo de Ibiza a Barcelona que tiene 533 mensajes ADS-B.

Algoritmo de resolución	***IBE04HT***		***IBE04NL***		***IBE0519***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	511.99	-	659.99	-	682.84	-
Inserción más cercana	453.22	5.46	588.89	7.24	639.72	5.60
Inserción más lejana	533.07	5.41	570.51	6.14	600.42	5.97
Inserción más barata	452.95	3.76	570.42	4.03	600.38	3.89
Inserción aleatoria	529.76	0.05	576.98	0.05	873.01	0.04
Vecinos más próximo	518.98	0.05	888.53	0.05	1227.06	0.05
Vecinos más próximo repetitivo	453.49	66.22	570.76	67.54	602.57	55.41
2-opt	452.95	0.42	570.42	0.63	600.38	0.50
Lin-Kernighan	452.97	20.85	570.39	18.58	600.38	22.14
Concorde	452.97	8.08	570.39	10.44	600.38	8.05
***SA* (40000 iteraciones)**	505.91	5.04	657.75	5.08	677.75	5.22
***SA* (100000 iteraciones)**	503.99	12.20	656.83	14.06	664.79	12.56
***SA* (1000000 iteraciones)**	482.97	141.86	611.94	133.66	641.93	120.99

Tabla 4.1: Comparativa de los algoritmos en distancia (km) y tiempo de resolución (sg)

Algoritmo de resolución	***IBE05DK***		***RYR9KY_4CA97C***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	602.47	-	346.98	-
Inserción más cercana	573.73	5.59	300.85	0.78
Inserción más lejana	801.13	4.97	360.63	0.80
Inserción más barata	558.69	3.04	299.99	0.49
Inserción aleatoria	653.80	0.05	330.80	0.02
Vecinos más próximo	913.47	0.04	332.44	0.01
Vecinos más próximo repetitivo	558.77	51.36	310.66	6.75
2-opt	558.69	0.28	299.99	0.03
Lin-Kernighan	558.69	16.97	299.99	3.62
Concorde	558.69	8.40	299.99	2.49
***SA* (40000 iteraciones)**	601.86	5.59	340.84	3.65
***SA* (100000 iteraciones)**	590.95	11.98	327.98	9.17
***SA* (1000000 iteraciones)**	578.84	120.21	309.50	89.64

Tabla 4.2: Comparativa de los algoritmos en distancia (km) y tiempo de resolución (sg)

Cabe destacar que en el caso de la Sección 3.4 la ruta inicial de partida se consideró de manera aleatoria, mientras que en el presente estudio se espera que esta difiera poco de la solución óptima, por lo que ciertos algoritmos que se comportaban mal podrían mejorar su comportamiento al ser aplicados para realizar la reordenación

de las trayectorias de los vuelos antes mencionados.

Al igual que en la Sección 4 del Capítulo 3 observamos que los mejores métodos en cuanto a distancia recorrida son el *Lin-Kernighan* y el *Concorde*. Además, observamos como el *2-opt* arroja muy buenos resultados, y como ya comentamos, es debido a que en este caso la solución de partida no es muy lejana a la óptima, esto es, las trayectorias no son muy malas. Con respecto a los algoritmos que obtienen buenos resultados, el método *2-opt* presenta la ventaja de tener un tiempo de ejecución bastante menor. También se observa que otros algoritmos como *Inserción más barata* y *Vecinos más próximo repetitivo* arrojan muy buenos resultados.

Por otro lado, para algoritmos como *Inserción más cercana, más lejana* y *Vecinos más próximos* vemos que el resultado depende de la ejecución realizada, de manera que las soluciones mostradas en las Tablas 4.1 y 4.2 podrían mejorarse si se realizan una serie de ejecuciones y se elige la mejor de ellas. Este es uno de los inconvenientes que presentan dichos algoritmos, pues en función del dato de partida considerado se obtienen mejores o peores resultados, aunque para el caso del algoritmo *Vecinos más próximos*, la distancia obtenida suele estar en la mayoría de las ocasiones lejana de la óptima (incluso llega a empeorarla). Además, para el algoritmo *Inserción aleatoria*, como la inserción de los nodos se realiza al azar, habrá diferencias en los resultados dependiendo de la ejecución realizada pues cada vez será diferente, arrojando en algunos casos mejores resultados que en otros.

Por último vemos como el *Simulated Annealing* también obtiene buenos resultados, logrando aproximarse bastante a la solución óptima, pero para ello, es necesario realizar unas cuantas iteraciones lo que provoca un aumento del tiempo de ejecución. En este caso para realizar las ejecuciones del algoritmo *Simulated Annealing* no se ha usado un algoritmo previo de resolución.

A continuación, se muestran dos gráficas en las Figuras 4.4 y 4.5, donde se representa para cada uno de los algoritmos considerados en las Tablas 4.1 y 4.2 la diferencia entre la distancia original de cada uno de los vuelos considerados con respecto a la distancia tras aplicar un determinado algoritmo de resolución.

En la primera gráfica (ver Figura 4.4) se puede ver como para algunos algoritmos se obtienen diferencias negativas, ya que en ocasiones, la ejecución de dichos algoritmos arroja resultados peores que los de partida.

Por otro lado, en la segunda gráfica (ver Figura 4.5), se muestra la representación asociada a las heurísticas no constructivas basados en realizar intercambios (*2-opt* y *Lin-Kernighan*) o en el azar (*Simulated Annealing*) junto con el algoritmo *Concorde*. Se puede observar como para el caso del *Simulated Annealing* si se usa un adecuado número de iteraciones la diferencia con respecto a los otros algoritmos no es muy elevada. Además, como se ha podido comprobar tras realizar diferentes pruebas, los resultados de dicho algoritmo se aproximan mucho más a la solución óptima cuando se considera el uso de ventanas de tiempo, lo cual, no fue considerado en este caso.

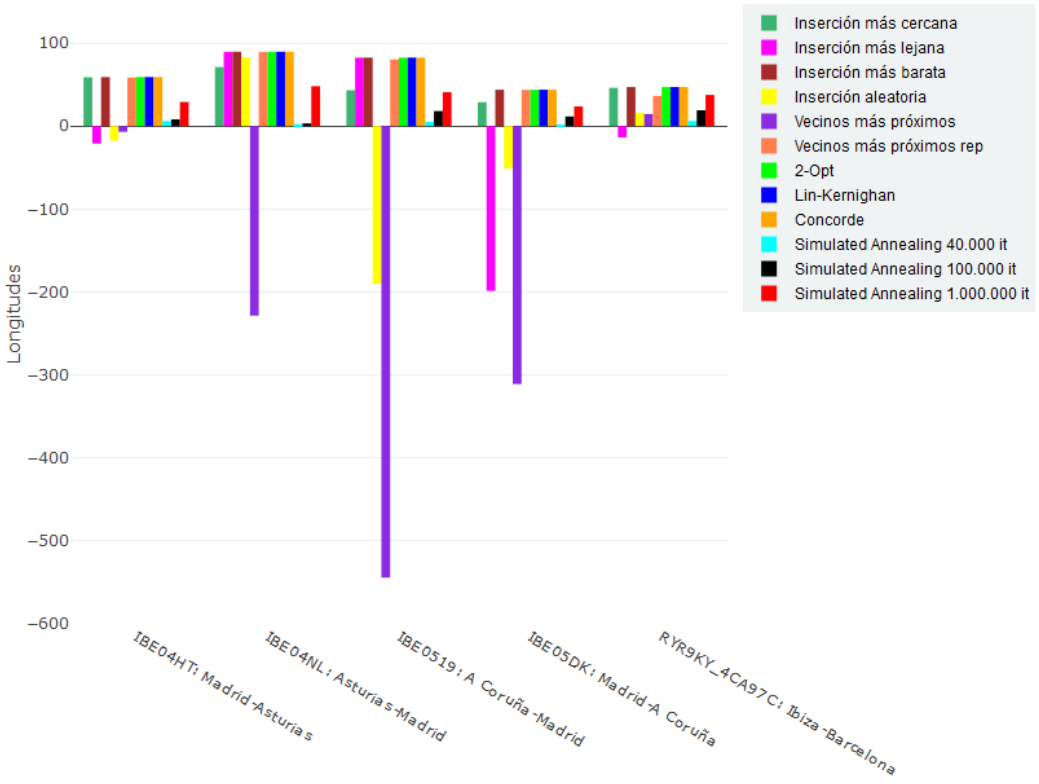


Figura 4.4: Comparativa de todos los algoritmos de resolución

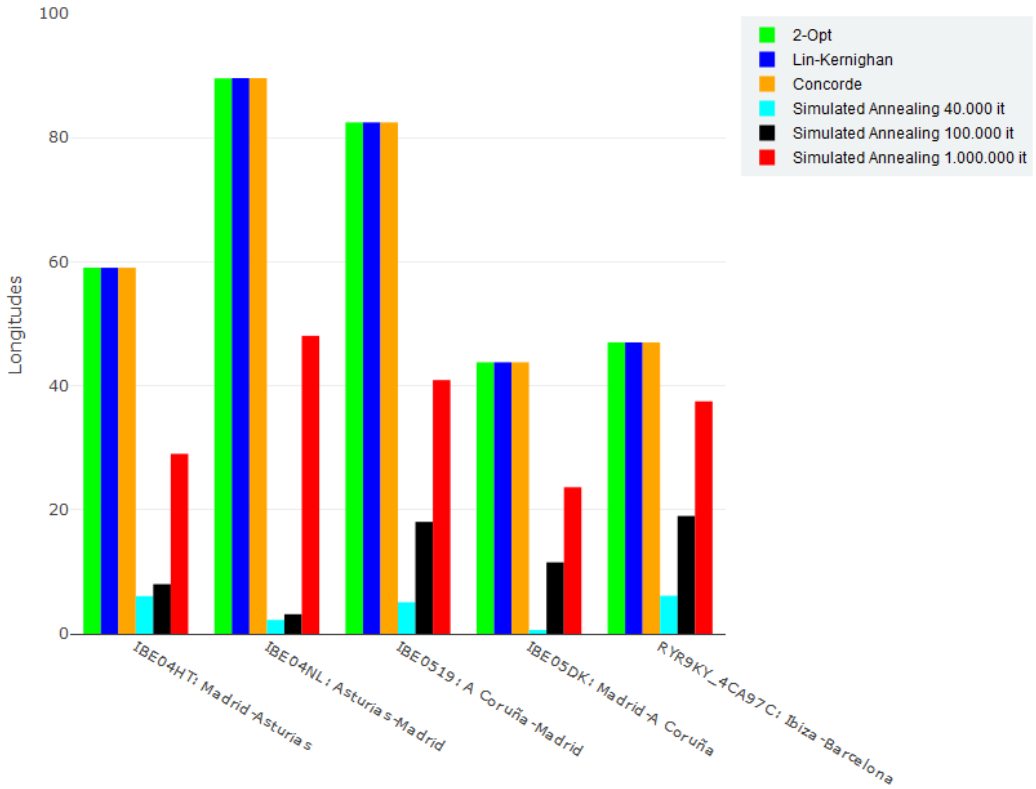


Figura 4.5: Comparativa de algunos algoritmos de resolución

Para ilustrar la mejora del algoritmo *Simulated Annealing* cuando se usan ventanas de tiempo, se ha realizado la ejecución de los vuelos mostrados en las Tablas 4.1 y 4.2. Para ello, se ha usado un tamaño de ventana de 100, un solapamiento de 20 (igual para la zona del vuelo que para las zonas del aeropuerto) y una temperatura de 1. Los resultados obtenidos se muestran en las siguientes tablas.

***Simulated Annealing* con 40000 iteraciones**:

Algoritmo de resolución	***IBE04HT***		***IBE04NL***		***IBE0519***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	511.99 km	-	659.99 km	-	682.84 km	-
Sin ventanas	505.91 km	5.04 sg	657.75 km	5.08 sg	677.75 km	5.22 sg
Con ventanas	467.14 km	45.30 sg	587.62 km	44.93 sg	619.90 km	53.21 sg

Tabla 4.3: Comparativa entre el uso o no de ventanas de tiempo (40000 iteraciones)

Algoritmo de resolución	***IBE05DK***		***RYR9KY_4CA97C***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	602.47 km	-	346.98 km	-
Sin ventanas	601.86 km	5.59 sg	340.84 km	3.65 sg
Con ventanas	570.51 km	47.68 sg	307.80 km	18.47 sg

Tabla 4.4: Comparativa entre el uso o no de ventanas de tiempo (40000 iteraciones)

***Simulated Annealing* con 100000 iteraciones**:

Algoritmo de resolución	***IBE04HT***		***IBE04NL***		***IBE0519***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	511.99 km	-	659.99 km	-	682.84 km	-
Sin ventanas	503.99 km	12.20 sg	656.83 km	14.06 sg	664.79 km	12.56 sg
Con ventanas	463.05 km	126.93 sg	580.08 km	142.59 sg	614.23 km	134.98 sg

Tabla 4.5: Comparativa entre el uso o no de ventanas de tiempo (100000 iteraciones)

Algoritmo de resolución	***IBE05DK***		***RYR9KY_4CA97C***	
	Distancia	**Tiempo**	**Distancia**	**Tiempo**
Sin aplicar algoritmo	602.47 km	-	346.98 km	-
Sin ventanas	590.95 km	11.98 sg	327.98 km	9.17 sg
Con ventanas	568.21 km	127.81 sg	305.93 km	46.64 sg

Tabla 4.6: Comparativa entre el uso o no de ventanas de tiempo (100000 iteraciones)

No se ha considerado el caso de 1000000 iteraciones dado que la mejora obtenida con respecto al caso de 100000 iteraciones sería mínima y el tiempo de ejecución aumentaría mucho, no siendo recomendable. Además, en las tablas se puede observar que la mejora es mucho mayor que cuando no se usan ventanas de tiempo, aunque es cierto que con su uso aumenta el tiempo de resolución, y como ya se comentó,

es debido a que el número de iteraciones para cada ventana es fijo. Sin embargo, como la mejora en términos de distancia es muy significativa y el tiempo tampoco es demasiado elevado, se puede concluir que el uso de ventanas resulta adecuado y recomendable para este algoritmo.

Para finalizar el presente capítulo, en las imagénes de la Figura 4.6 se muestra un trozo de la ruta de partida del vuelo *IBE04HT* y el resultado obtenido al ejecutar el algoritmo *Simulated Annealing* (con temperatura 1 y 100000 iteraciones) a dicha trayectoria de vuelo.

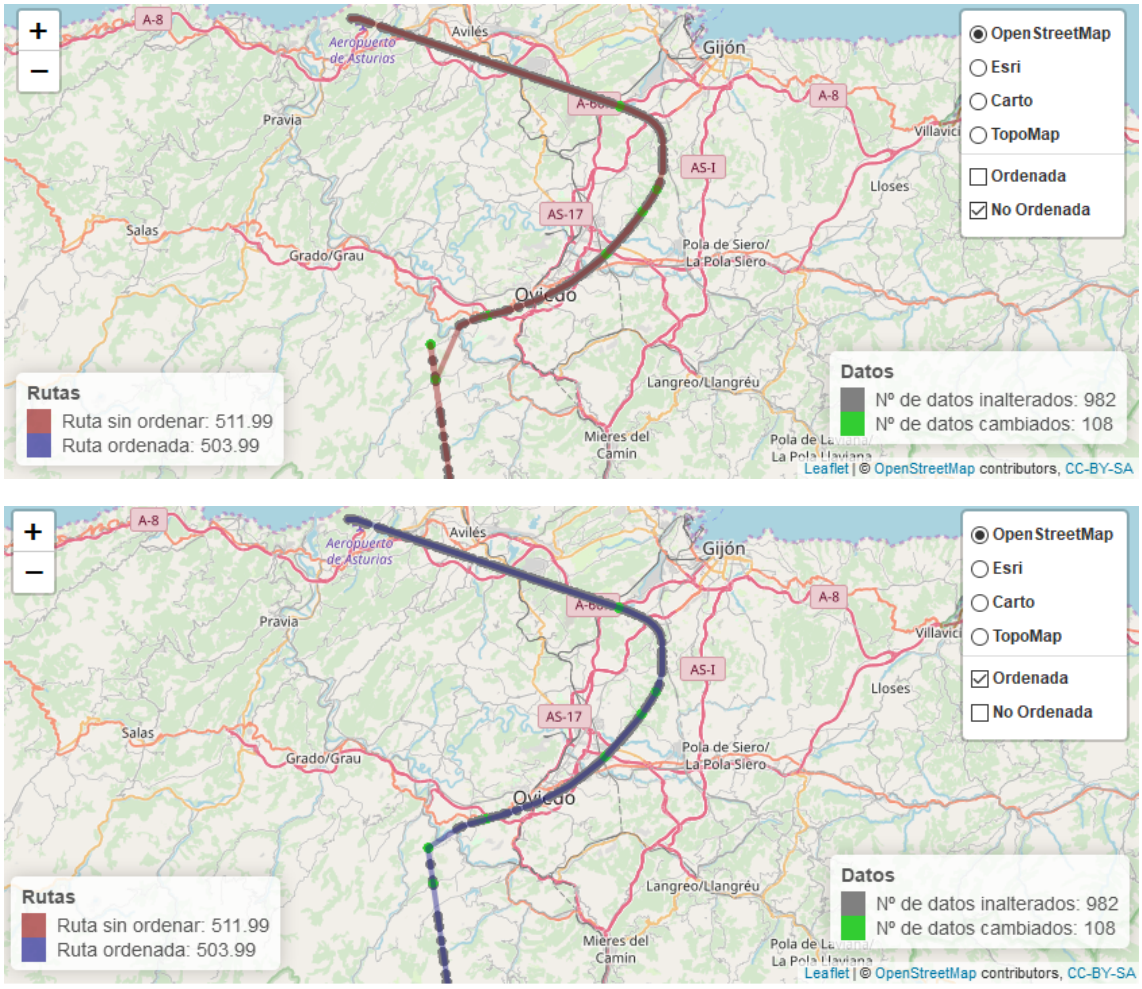


Figura 4.6: Trozo no ordenado y ordenado del vuelo *IBE04HT*

Se puede observar como para el caso de la ruta ordenada la distancia recorrida es menor. Esto se debe a que la ejecución del algoritmo va intercambiando diversos pares de estados o puntos quedándose con aquellos que logran reducir la distancia de la ruta actual (en este caso un total de 108 puntos han sido alterados). Además, en las imágenes se observa una de las mejoras realizadas en dicho vuelo. Estas pequeñas mejoras hacen que la distancia vaya disminuyendo poco a poco hasta acercarse lo más posible a la solución óptima, esto es, a la ruta de mínima distancia.

Capítulo 5

Conclusiones

Una vez realizado el desarrollo del presente trabajo, se puede concluir que han sido alcanzados con éxito los objetivos planteados al comienzo del mismo. Así, gracias al estudio y desarrollo de los distintos algoritmos presentados a lo largo del mismo, en particular centrándonos en el *Simulated Annealing*, se ha podido obtener una solución eficiente y escalable que permite corregir aquellas trayectorias que presentan alguna anomalía (asociada principalmente a una mala asignación de los *timestamps*) y realizar la correspondiente reasignación de *timestamps* para los puntos que han sido alterados durante la realización de la reordenación. Por lo tanto, se ha podido observar como los fundamentos teóricos de dicho algoritmo y las conclusiones de los mismos que fueron descritas en el Capítulo 2 eran certeras, ya que ha sido posible su visualización práctica.

Para ello, ha sido de gran utilidad el dashboard realizado usando el entorno de programación *R-Studio*. El dashboard creado ha sido el pilar fundamental para la realización del Trabajo de Fin de Grado de Informática, y además, ha sido usado en el presente trabajo para poder ver el funcionamiento de los diferentes algoritmos y realizar la comparativa mostrada en el Capítulo 4. Gracias a ello, se ha podido ilustrar cómo al aplicar alguno de dichos algoritmos a los distintos vuelos proporcionados, se logra obtener trayectorias cuyas distancias reducen de manera considerable la de la trayectoria de partida.

Todo esto, permite concluir que el estudio realizado puede ser llevado a cabo para su aplicación en los sistemas actuales de gestión del tráfico aéreo. Para el caso concreto del *Simulated Annealing*, se ha realizado un proyecto en Java usando el modelo de programación *MapReduce*, que permite obtener las rutas ordenadas de un conjunto dado de vuelos de manera eficiente y escalable, así como los nuevos tiempos obtenidos para cada punto tras la reordenación. El propósito de su realización es que pueda ser incorporado al tratamiento de los datos que se hace en las plataformas Big Data que tratan con las trayectorias y que permita mejorar y avanzar en lo relativo a la gestión del tráfico aéreo. Esto, es una mínima parte de todo lo que es necesario y queda por hacer para lograr optimizar el uso del espacio aéreo, pero es un buen punto de partida con el que se puede seguir trabajando e investigando.

Bibliografía

[1] William J. Cook, *In Pursuit of the Travelling Salesman*, Princeton University Press, 1957.

[2] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan y D.B. Shmoys, *The Travelling Salesman*, Princeton University y AT & T Bell Laboratories, New Jersey 1985.

[3] P.J.M. van Laarhoven y E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, Springer Science+Business Media Dordrecht, 1987.

[4] Emile Aarts y Jan Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons Lid, 1989.

[5] Wikipedia, *Teoría de grafos*, URL: https://es.wikipedia.org/wiki/Teoria_de_grafos, 2018. Visitado por última vez el 18 de Enero de 2019.

[6] Feller, W. [1950], *An Introduction to Probability Theory and Its Applications 1*, Wiley, New York.

[7] Seneta, E. [1950], *Non-negative Matrices and Markov Chains*, Springer-Verlag, New York.

[8] Isaacson, D. and R. Madsen [1976], *Markov Chains*, Wiley, New York.

[9] Alfredo Caicedo, Graciela Wagner de García y Rosa María Méndez, *Introducción a la teoría de grafos*, Ediciones Elizcom, 2010.

[10] Lowell W.Beineke and Robin J.Wilson, *Selected topics in graph theory 3*, England, 1988.

[11] Kai Lai Chung, *Markov Chains*, Springer, NewYork, 1960.

[12] J.C. Rosales, P.A. García-Sánchez, *Numerical Semigroups*, Springer, NewYork, 2009.

[13] Martínez Prieto, Miguel A., Bregón Bregón, A., García Miranda, I., Álvarez Esteban, Pedro C. y Díaz, F. (2017), *Towards a Scalable Architecture for Flight Data Management*, *SCITEPRESS (Science and Technology Publications, Lda.)*, 263-268. DOI: 10.5220/0006473402630268.

[14] Alonso Isla, A., Martínez Prieto, Miguel A., Bregón Bregón, A., García Miranda, I., Álvarez Esteban, Pedro C., Díaz, F. y Gordaliza, P. (2017), *Airports: Análisis de Eficiencia Operacional basado en Trayectorias de Vuelo*, URL: https://biblioteca.sistedes.es/submissions/descargas/2018/JISBD/2018-JISBD-063.pdf.