CICLO HAMILTONIANO ÓPTIMO EN UN GRAFO (PROBLEMA DEL VIAJANTE)

Leire Alfaro

Trabajo de fin de grado en Matemáticas

Universidad de Zaragoza

Director del trabajo: Alfredo García Olaverri

26 de Junio de 2020

PRÓLOGO

El problema del viajante, también conocido como TSP, ha sido uno de los problemas más estudiados en optimización combinatoria a lo largo de la historia. Este problema trata de determinar el camino que debe realizar un comerciante si quiere visitar n ciudades, empezando y acabando en la misma. En la vida real este problema aparece más veces de lo que cualquiera pueda imaginar. Basta con retroceder a 1930, a su origen, cuando se propuso resolver un problema de enrutamiento de un autobús escolar. Se buscaba la ruta más corta que podía hacer un autobús que, saliendo del colegio, debía pasar por todas las paradas y regresar al punto de partida.

A lo largo del trabajo, el TSP vendrá descrito mediante la teoría de grafos como una red cuyos nodos representan ciudades y cuyos enlaces tienen ciertos pesos que indican distancias entre dichas ciudades.

El trabajo contiene 3 capítulos dedicados a formulaciones y ejemplos para el problema, así como a algoritmos aproximados para resolverlo. Los diferentes algoritmos se desarrollan de forma sencilla para que los lectores puedan comprender de manera fácil qué es lo que se está haciendo. También aparece un algoritmo exacto, que podría entenderse sin tener la necesidad de leer todo el trabajo.

El Capítulo 2 es el más denso de todos. Se profundiza en su complejidad matemática ya que, a pesar de la simplicidad de su formulación, el TSP es un problema NP-Completo.

Para terminar, el último capítulo recoge resultados elementales para el TSP euclídeo, que es un caso particular del TSP general, en el que la distancia entre ciudades se calculan de acuerdo a la métrica euclídea.

RESUMEN

The study of The Traveling Salesman Problem is the main objetive of this project. This problem tries to determine the path that a salesman, who wants to visit n cities, must follow to get the tour whose distance is as small as possible.

The document consists of 5 chapters. We are going to describe briefly each of them:

Chapter 1: An overview of The Traveling Salesman Problem (TSP).

The first chapter of this project works as an introduction of the TSP study. After a short historic view, in which authors like Flood, Turker, Menger and the corporation RAND of Santa Mónica remarkably contribute, this section collects a first mathematical description of the problem, composed by an input and an output.

Most of the progress in the TSP history has been motivated by using it in quotidian facts. For this reason, one part of this chapter gather different applications based on logistics and industry above all, highlighting the problem of the job sequencing, which is explained in depth.

The characteristics of the TSP are defined by a distance matrix, so that in the last part of this first block are explained different features that can have aforementioned matrices. Moreover, this chapter contains basic results of one of the most important cases of the TSP known as Euclidean TSP.

Chapter 2: The TSP is NP-Complete.

This is one of the most difficult chapters of the project, as it tries to demonstrate clearly that the TSP is NP-complete.

At the beginning we can find new concepts like the *Big O-notation*, just as those related to the decision problems, highlighting those which are P and NP and giving examples like the graph colouring problem.

Finally, it focuses on NP-complete problems, showing that the question of the Hamiltonian cycle belongs to this kind of problems. This fact is essential to be able to conclude that the TSP is a NP-complete too.

Chapter 3: Held-Karp exact algorithm.

Once we have explained the problem we are facing up to, we need to find how to solve it. This chapter focuses on the exact algorithm, based on dynamic programming that rose up in the early 60's, known as the Held-Karp algorithm . We can find an example of how to carry out this algorithm in appendix C.

Chapter 4: Approximated algorithms.

As we are working with a complex problem, it's difficult to find an exact algorithm that allows us to solve it. There are different heuristic algorithms that give us an approximate solution. In this chapter we will study five of these heuristic algorithms, each of them paired with an explanatory example so that it will be easier to understand them.

Índice general

PRÓLOGO III

RESUMEN V

1. VISIÓN GENERAL DEL TSP 1
- 1.1. DESCRIPCIÓN DEL PROBLEMA 1
  - 1.1.1. INTRODUCCIÓN 1
  - 1.1.2. HISTORIA 1
  - 1.1.3. APLICACIONES 2
- 1.2. DESCRIPCIÓN Y FORMULACIÓN MATEMÁTICA 3
  - 1.2.1. TSP PARA TEORÍA DE GRAFOS 4

2. EL TSP ES NP-COMPLETO 7
- 2.1. COMPLEJIDAD ALGORÍTMICA 7
  - 2.1.1. NOTACIÓN O-GRANDE 7
  - 2.1.2. PROBLEMAS DE DECISIÓN 8
  - 2.1.3. PROBLEMAS DE CLASES NP-COMPLETO Y NP DURO 8
  - 2.1.4. EL TSP ES NP-COMPLETO 10

3. ALGORITMO EXACTO: HELD-KARP 15
- 3.1. ALGORITMO BRUTO 15
- 3.2. PROGRAMACIÓN DINÁMICA. HELD Y KARP (1962) 15

4. ALGORITMOS APROXIMADOS 17
- 4.1. ALGORITMOS HEURÍSTICOS 17
  - 4.1.1. MEDIDA DE CALIDAD DE UN ALGORÍTMO 17
- 4.2. MÉTODOS CONSTRUCTIVOS DEL TSP 18
  - 4.2.1. VECINO MÁS PRÓXIMO 18
  - 4.2.2. ALGORITMO DE LIN-KERNIGHAN 20
  - 4.2.3. ÁRBOL DE EXPANSIÓN MÍNIMA(MINIMUM SPANNING TREE) 22
  - 4.2.4. MATCHING DE MÍNIMO PESO O ALGORITMO DE CHRISTOFIDES 24

BIBLIOGRAFÍA 27

ANEXOS 29

A. ENSAMBLADO DE FRAGMENTOS DE ADN 29

B. TSP EUCLÍDEO 31
- B.1. RESULTADOS ELEMENTALES SOBRE EL TSP EUCLÍDEO 31
- B.2. TSP PARA UNA CLAUSURA CONVEXA Y UNA LÍNEA INTERIOR 32
- B.3. EJEMPLO VISUAL DEL TSP PARA UN CONVEXO Y UNA LÍNEA INTERIOR 34

C. EJEMPLO PARA EL ALGORITMO DE HELD-KARP **37**

D. EJEMPLOS PARA LOS ALGORITMOS APROXIMADOS **39**
D.1. ÁRBOL DE MÍNIMO PESO. 39
D.2. MATCHING DE EXPANSIÓN MÍNIMA 40

Capítulo 1

VISIÓN GENERAL DEL TSP

1.1. DESCRIPCIÓN DEL PROBLEMA

1.1.1. INTRODUCCIÓN

El problema del viajante de comercio, también conocido como TSP por sus siglas en inglés (Traveling Salesman Problem) es uno de los problemas más destacados de optimización combinatoria. Este problema responde a la siguiente pregunta:

Dada una lista de ciudades junto con la distancia que hay entre cada par de ellas, ¿cuál es la ruta más corta posible que se puede elegir para visitar cada ciudad y regresar a la ciudad origen?

Considerando el conjunto de ciudades como una red de nodos, donde la distancia entre el nodo i y el j viene dada como c_{ij}, se trata de encontrar en qué orden se recorrerán los nodos de la red, de modo que se minimice la distancia total recorrida.

El TSP remonta sus orígenes a 1930, siendo hasta hoy uno de los problemas de optimización combinatoria más estudiados. Destaca por su simplicidad en el enunciado y la dificultad de encontrar una solución válida para ese enunciado simple.

1.1.2. HISTORIA

Los orígenes del TSP no son claros, pero el primer indicio bibliográfico fue en 1832, en Alemania, en un libro titulado " El vendedor ambulante: cómo debe ser y qué debe hacer para obtener comisiones y éxito en su negocio. Por un vendedor ambulante veterano ". Se llega a la esencia del TSP en su último capítulo, indicando que con una elección adecuada en la ruta se puede ahorrar mucho tiempo y, además, da como sugerencia principal el visitar cada ciudad una única vez.

Asimismo, el libro incluye cinco rutas que recorren regiones de Alemania y Suiza, siendo una de ellas la solución óptima al TSP. En cuanto a las otras cuatro, en las que se visita alguna ciudad más de una vez, se cree que teniendo en cuenta los medios de transporte existentes en la época podrían haber sido también rutas óptimas. Sin embargo, este libro no aporta ningún teorema matemático que demuestre esto.

Aunque se tiene constancia del libro publicado en 1832, desde el punto de vista matemático los orígenes del TSP se remontan a principios de 1930. No se sabe quién introdujo el nombre de TSP en el mundo de las matemáticas; sin embargo, se sabe que Merill Flood es el mayor responsable de publicitarlo, ya que en una entrevista que le hicieron contó cómo al empezar a trabajar en la ruta de un autobús escolar en New Jersey, se empezó a interesar por el problema y afirmó haberlo conocido a través de A.W. Tucker en 1937. Además Flood relató que Tucker se lo había escuchado a Hassler Whitney en

la Universidad de Princeton, alrededor del curso 1931-32. Ello haría que Whitney fuera el pionero del TSP, pero éste negó cualquier relación con dicho problema.

Mientras tanto, en Viena, el matemático Karl Menger estudió también el problema denominándolo " el problema del mensajero", que se basaba en encontrar el camino mínimo uniendo todos los puntos de un conjunto con distancia entre ellos conocida. Esto le surgió gracias a una definición que propuso [10]: *la longitud de una curva se define como el límite superior mínimo del conjunto de todos los números que pueden ser obtenidos tomando cada conjunto finito de puntos sobre la curva*. Para Menger, este problema podía ser resuelto en un número finito de pasos pero se desconocía la existencia de reglas que permitiesen reducir este número de pasos por debajo del número de permutaciones existentes entre los puntos.

Durante los años 1950-1960, el problema fue incrementando su popularidad entre el círculo de científicos de Europa y Estados Unidos. Una notable contribución fue la de George Dantzig, Delbert Ray Fulkerson y Selmer M. Johnson de la Corporación RAND en Santa Mónica, quienes formularon el problema como un problema de Programación Lineal en Enteros y, para solucionarlo, desarrollaron el método de los planos de corte. Con este nuevo método, resolvieron de manera óptima el problema con 49 ciudades mediante la construcción de un recorrido y probando que no había un recorrido que pudiera ser más corto.

En las siguientes décadas el problema fue estudiado por muchos investigadores, matemáticos, científicos de la computación, químicos, físicos, etc.

1.1.3. APLICACIONES

El nombre de este problema: "el problema del viajante de comercio" hace pensar que tiene una aplicación muy concreta, es decir, que consiste en decidir la ruta que debe seguir un vendedor. Sin embargo, el TSP se puede emplear en cualquier situación que requiera seleccionar nodos en cierto orden de cara a reducir costes o distancias.

La mayor parte de los avances en la historia del TSP han sido motivados por aplicaciones del mismo en casos cotidianos. Hoy en día hay multitud de campos en los que el problema del viajante de comercio juega un papel importante, destacando la elección de un recorrido eficiente a la hora de transportar mercancías, pasajeros y vehículos en n ciudades distintas. Este problema básico da lugar a varias aplicaciones de enrutamiento de vehículos, de cableado de ordenadores, de recolección de basuras de manera domiciliaria, de autobuses escolares...Todos ellos siguen un mismo patrón pero con diferentes sujetos que realizan la acción.

Aparte de estos problemas basados en logística, cabe destacar el uso del TSP en la industria. Un ejemplo es "el problema de la secuencia de trabajos" en el que hay n trabajos a realizar en una máquina simple. Los trabajos se pueden llevar a cabo en cualquier orden, siendo el objetivo del problema completar todos estos trabajos en el menor tiempo posible.

Para poder empezar el trabajo j la máquina debe estar en el estado S_j, que representa las condiciones necesarias para realizarlo (la posición de rotación, el color, la presión...), mientras que el estado inicial y final de la máquina será S_0.

El tiempo requerido para completar el trabajo j directamente después del trabajo i es

$$t_{ij} = c_{ij} + p_j,$$

donde c_{ij} es el tiempo que se necesita para transformar la máquina del estado S_i al S_j, y p_j es el tiempo que tarda en realizar el trabajo j (con $p_0 = 0$). Para una permutación cíclica π de $0,\ldots,n$, el tiempo

requerido para completar todos los trabajos es:

$$\sum_{i=0}^{n}(c_{i\pi(i)} + p_{\pi(i)}) = \sum_{i=0}^{n} c_{i\pi(i)} + \sum_{j=0}^{n} p_j$$

Como la suma de todos los p_j es una constante y lo que se busca es que permutación cíclica π minimiza esa suma, el problema de la secuencia de trabajos es un TSP.

El problema del TSP, además de usarse en los múltiples ámbitos ya mencionados, ha llegado a ser utilizado en laboratorios con partículas muy pequeñas, siendo uno de los ejemplos más destacados el ensamblado de ADN que podemos ver descrito en el *Anexo A*.

En la práctica es muy común encontrarse con problemas del viajante que presenten pequeñas variaciones respecto del problema original, como puede ser el TSP con vendedores múltiples, con condiciones de repetición sobre algunas ciudades, etc.

A lo largo del trabajo se estudiarán de manera matemática algunos ejemplos.

1.2. DESCRIPCIÓN Y FORMULACIÓN MATEMÁTICA

Una vez se tiene una idea general de qué es el TSP y de cuál es el objetivo del problema, es necesario plantearlo de manera matemática.

Una descripción genérica del problema del viajante está compuesta por una entrada y una pregunta relacionadas entre sí, es decir, la pregunta se debe plantear según la información que se dé en la entrada.

TSP
ENTRADA : Entero $n \geq 3$ y una matriz nxn, denotada $C = (c_{ij})$, donde c_{ij} son enteros no negativos.
PREGUNTA: ¿Qué permutación cíclica π de enteros del 1 al n minimiza la suma $\sum_{i=0}^{n} c_{i\pi(i)}$?

Notar que, en esta definición, se restringe la distancia entre ciudades para que sean enteros. Se quiere conseguir un algoritmo que dada una entrada nos genere el recorrido de mínima distancia. Un algoritmo trivial sería generar todos los recorridos, evaluar cada uno de ellos y elegir el mejor. Aunque en la práctica no es útil ya que si la entrada del problema es $n = 30$ el número de recorridos posibles es 30! que es aproximadamente 2,65x10^{32} y esto implica un tiempo computacional muy grande e imposibilita llevarlo a cabo.

Al igual que existen diversas variaciones, existen también varias formulaciones del TSP y para cada una de ellas existirá un método de resolución distinto. El tipo de TSP asociado a un problema se distingue según su matriz de costes (o distancias), es decir, según las propiedades que tenga C. Por ejemplo:

Si la matriz es simétrica:

El TSP tendrá una matriz simétrica asociada a él si la distancia entre ciudades es la misma se tome el sentido que se tome. Por el contrario, si la distancia puede ser diferente a la ida como a la vuelta e incluso no existir una de ellas se llamará matriz asimétrica.

Si la matriz cumple la propiedad triangular:

Matriz cuyas distancias forman una métrica que satisfacen la desigualdad triangular $\equiv d_{AB} \leq d_{AC} + d_{CA}$, es decir, que la conexión directa entre dos nodos A y B nunca sea mayor que hacer el camino pasando por C.

Si la matriz es euclídea:

Matriz de costes donde el conjunto de las n ciudades dadas están en un espacio euclídeo, donde la distancia entre cada par de ciudades se representa como $d(x_1, x_2) = ||x_1 - x_2||_2$. Como estas distancias forman una métrica, concretamente la métrica euclídea, satisfacen la desigualdad triangular.

1.2.1. TSP PARA TEORÍA DE GRAFOS

La matriz de costes del problema es la que permite definir el grafo asociado a dicho problema o viceversa, es decir, un grafo siempre va ligado a una matriz de costes. Definamos pues el TSP para teoría de grafos, pero antes de ello veamos algunas definiciones previas:

Definiciones.

- Grafo: conjunto de objetos llamados vértices o nodos, unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.
- Grafo completo: grafo no orientado que tiene entre cualquier par de sus nodos al menos una arista que los une.
- Grafo dirigido: grafo en el cual las aristas tienen un sentido definido.
- Camino hamiltoniano: camino que recorre todos los nodos del grafo una y sólo una vez.

Contrucción de la matriz de distancias asociada a un grafo.

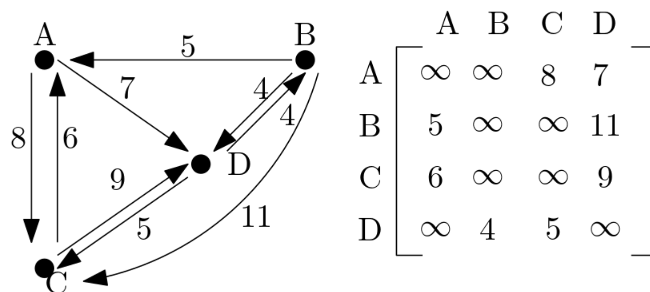
Sea $G = (V, A)$ un grafo completo siendo $V = 1, ..., n$ el conjunto de vértices y A el conjunto de arcos. Los vértices representan las ciudades que se deben visitar, siendo v_1 la ciudad de origen y final. Cada arco (i, j) tendrá asociado un valor no negativo c_{ij} que recoge la distancia que hay entre las ciudades i y j.

A partir de esta información se construye la matriz de costes asociada al grafo G, donde las filas de la matriz representan los nodos i y las columnas los nodos j, es decir, la matriz tendrá la siguiente forma: $\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{pmatrix}$.

Hay veces que dentro de un grafo no todos los nodos están conectados, es decir, faltan conexiones para construir la matriz. En ese caso, como el TSP consiste en conseguir un camino de mínima distancia o coste, suponemos que aquellas conexiones que no aparecen en el grafo tendrán un coste muy alto, indicándolo como $c_{ij} = \infty$ para que dichas conexiones nunca formen parte del recorrido óptimo. Además, en este problema no es posible quedarse en una misma ciudad más de una iteración, por lo que por la misma razón todas las componentes c_{ii} de la matriz se representarán con el valor ∞.

A modo de resumen se puede concluir que el problema del viajante se define sobre un grafo (generalmente completo) con costes asociados a los arcos y consiste en encontrar el camino hamiltoniano de coste mínimo.

***Ejemplo* 1.1.** Matriz de costes asociada a un grafo:



Como la matriz de costes C siempre va relacionada con un grafo G, las propiedades que pueden tener las matrices anteriormente mencionadas también influyen en las características del grafo. Notar que, en el caso en el que la matriz de costes C que recoge las distancias es asimétrica , es decir, $c_{ij} \neq c_{ji}$ G será un grafo dirigido, dando lugar al problema asimétrico del TSP (ATSP). Mientras que, si la matriz de costes es simétrica $c_{ij} = c_{ji}$ para todo (i, j) que pertenece a A, G será no dirigido y se denotará como STSP.

Como todo grafo no dirigido puede transformarse en uno dirigido duplicando las aristas de manera que haya una en cada sentido, el STSP puede verse como un caso especial del ATSP, siendo uno de los ejemplos más destacados para el TSP asimétrico el ensamblado de ADN explicado en el Anexo A.

Si la matriz de costes C es euclídea se tiene el TSP euclídeo de n-ciudades que es un importante caso particular del TSP, donde cada ciudad i viene representada como un punto $p_i = (x_i, y_i)$, x_i, $y_i \in \mathbb{R}$ en el plano y las distancias $c(i, j)$ entre cada par de ciudades i, j se calculan de acuerdo a la métrica euclídea, $i, j = 1, ..., n$. Para leer mas información sobre este caso de TSP ir al Anexo B.

Capítulo 2

EL TSP ES NP-COMPLETO

En este capítulo se va a estudiar la complejidad del TSP. Para ello se van a enunciar algunas de las cuestiones más estudiadas sobre la teoría de la complejidad como es, por ejemplo, determinar si un problema pertenece a la clase P o NP de problemas de decisión en función de si se conoce un algoritmo eficiente para el mismo (se verán con detenimiento las definiciones correspondientes para las clases P y NP). Después se definirán los conceptos NP-Duro y NP-Completo para así poder demostrar que el problema del que se viene hablando durante todo el trabajo pertenece a esta segunda clase, es decir, es NP-Completo.

2.1. COMPLEJIDAD ALGORÍTMICA

2.1.1. NOTACIÓN O-GRANDE

En análisis de algoritmos una cota superior asintótica es una función que sirve de cota superior de otra función cuando el argumento tiende a infinito. Usualmente se utiliza la notación de Landau:$O(g(n))$, Orden de $g(n)$, coloquialmente llamada notación *O-Grande*, para referirse a las funciones acotadas superiormente por la función $g(n)$.

***Definición* 2.1.** Sea $f : Z^+ \to R^+$ y $g : Z^+ \to R^+$ funciones del conjunto de los enteros positivos al conjunto de los número reales positivos. La función $f(n)$ está en la familia $O(g(n))$ si hay constantes c y N mayores que 0 tales que

$$f(n) \leq cg(n) \quad \text{para todo} \quad n > N$$

Si $f(n)$ está en $O(g(n))$, se dice que $f(n)$ está asintóticamente dominado por $g(n)$. En otras palabras, para una n y una constante c suficientemente grandes, $cg(n)$ es mayor que $f(n)$.

Nota: Se sigue de la definición que $f(n)$ está en $O(g(n))$ si y solo si existe una constante c tal que $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq c$.

***Ejemplo* 2.1.** La función $f(n) = \frac{n^2+n}{2}$ está en $O(n^2)$. Para ver esto, observamos que para $n > 0$.

$$f(n) = \frac{n^2+n}{2} < n^2 + n \leq n^2 + n^2 = 2n^2$$

Satisface la ecuación para $c = 2$ y $N = 0$. Es verdad que $f(n)$ está en $O(n^r)$ para todo $r \geq 2$, pero $O(n^2)$ es lo más preciso. Además, tomando $c = 2$ y $N = 0$, la siguiente cadena de desigualdades muestra que n^2 está en $O(f(n))$.

$$n^2 < n^2 + n = 2f(n)$$

Las funciones $f(n)$ y $g(n)$ son equivalentes *O-grande* si y solo si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ para alguna constante no nula. Dos funciones polinómicas son *O-grande* equivalentes si y solo si ellas tienen el mismo grado.

***Definición* 2.2.** El tiempo de ejecución $T(n)$ de un algoritmo se dice que es $O(g(n))$ si existe una constante positiva c y un número entero N tales que $\forall n \geq N$ se tiene $T(n) \leq cg(n)$.

***Definición* 2.3.** Un algoritmo de tiempo polinomial es un algoritmo cuyo tiempo de ejecución está en $O(n^k)$ para algún valor de k.

2.1.2. PROBLEMAS DE DECISIÓN

***Definición* 2.4.** Un problema de decisión es un problema que requiere solo una respuesta de *si* o *no*.

***Definición* 2.5.** Un problema de decisión pertenece a la clase P si hay un algoritmo de tiempo polinomial que resuelve el problema.

***Definición* 2.6.** Un problema de decisión pertenece a la clase NP si una respuesta que sea "sí" puede ser confirmada por un algoritmo de tiempo polinomial.

Nota: La letra P proviene de "*polynomial*" mientras que las letras NP vienen de "*nondeterministic polynomial*".

***Ejemplo* 2.2.** El problema de decisión "*¿ Es G bipartito ?*" dado un grafo G pertenece a la clase P.

***Ejemplo* 2.3.** No se sabe si el problema de decisión *"¿Pueden colorearse los vértices de un grafo con tres colores distintos tal que los vértices adyacentes sean de colores diferentes?"* está en la clase P. Primero se debe dar una respuesta para la pregunta planteada, la cual sea "sí" o "no". Imaginando que la respuesta que se ha dado es "sí", se eligen 3 colores para los vértices. Una vez se tienen elegidos, hay que comprobar que utilizando sólo estos 3 colores se pueden colorear todos los vértices cumpliendo los requisitos. Para ello se asigna un color a cada uno de los n vértices, lo cual se puede realizar en un tiempo lineal. Y después se revisa que cada par de vértices adyacentes posea dos colores diferentes, para esto se necesitará un tiempo cuadrático ya que el número de ejes es cuadrático con respecto al número de vértices.
Por tanto, una vez dada la respuesta "sí" esta se puede comprobar en un tiempo polinomial por lo que el problema pertenece a la clase NP.

Puede parecer que la clase de problemas que podrían resolverse en tiempo polinomial con un ordenador no determinista deberían ser más grandes que los que pueden resolverse mediante algoritmos de tiempo polinomial en un ordenador secuencial. Sin embargo, nadie ha podido encontrar un problema en la clase NP que no esté en la clase P. Determinar si existe tal problema de decisión es uno de los principales problemas no resueltos en la informática teórica.

2.1.3. PROBLEMAS DE CLASES NP-COMPLETO Y NP DURO

***Definición* 2.7.** Un problema de decisión R es polinomialmente reducible a un problema de decisión Q si hay una trasformación en tiempo polinomial de cada ejemplo I_R del problema R a un ejemplo I_Q del problema Q, donde los ejemplos o casos de I_R y I_Q tienen la misma respuesta (sí o no).

***Definición* 2.8.** Un problema de decisión C es NP-completo si:

1. C está contenido en el conjunto NP.

2. Todo problema de NP es reducible a C en tiempo polinomial.

Esto implica que si encontramos un algoritmo de resolución polinomial para un problema NP-completo, podríamos resolver en tiempo polinomial cualquier problema de NP a través de su transformación y estaría demostrado que P=NP.

***Definición* 2.9.** Un problema de decisión C es NP-duro cuando cada problema en NP se puede reducir en tiempo polinomial a C.

Hay una larga colección de problemas NP-completos, incluyendo el problema del viajante (cuando se formula como un problema de decisión), para los que no se han encontrado algoritmos con tiempo polinomial a pesar del considerable esfuerzo realizado en las últimas décadas, por lo que queda en el aire la cuestión de si existen dichos algoritmos para estos problemas.

El primer problema NP-completo fue el llamado problema de satisfacibilidad (SAT) de expresiones booleanas.

***Definición* 2.10.** Sea $U = \{u_1, u_2, ..., u_m\}$ un conjunto de variables booleanas. Se llama *correspondencia de verdad para* U, a una función $T : U \to \{true, false\}$.

Si $T(u) = true$ entonces decimos que u es verdad con respecto a T. En caso contrario, decimos que u es falsa.

***Definición* 2.11.** Sea $u \in U$, entonces u, $\bar{u}$ son literales de u. El literal u es cierto con respecto a T si y solo si la variable u es cierta con respecto a T (es decir, $T(u) = true$) y $\bar{u}$ es cierto si y solo si $T(u) = false$.

***Definición* 2.12.** Una *cláusula* sobre U es un conjunto de literales de U tal que representa la disyunción de los literales, es decir, es satisfecha por una correspondencia de verdad T si y sólo si al menos uno de sus miembros es cierto bajo T.

***Definición* 2.13.** Una colección de cláusulas C sobre U es satisfacible si y sólo si existe alguna correspondencia de verdad para U que satisface todas y cada una de las cláusulas de C.

***Definición* 2.14.** Sea $X = x_1, x_2, ..., x_n$ un conjunto de n variables booleanas. El *problema de satisfacibilidad* es el problema de saber si, dada una expresión booleana con variables y sin cuantificadores, hay alguna asignación de valores para sus variables que hace que la expresión sea verdadera.

Problema de Satisfacibilidad (SAT):

Sea U un conjunto de variables, y C una colección de claúsulas sobre U ¿Existe alguna correspondencia de verdad que satisfaga C?

***Teorema* 2.1.** (Teorema de Cook) El problema de Satisfacibilidad (SAT) es NP-Completo.

Problema de la 3-Satisfacibilidad (3-SAT):

Es una restricción del problema SAT donde toda entrada tiene que tener exactamente 3 literales en cada claúsula.

***Teorema* 2.2.** El problema de la 3-satisfabilidad (3-SAT) es NP-Completo.

2.1.4. EL TSP ES NP-COMPLETO

En esta sección primero se demostrará que el problema del ciclo hamiltoniano es NP-Completo, de ahí se pasará a demostrar lo mismo para el TSP.

***Definición* 2.15.** Un ciclo hamiltoniano en un grafo es un camino cerrado que pasa una sola vez por todos los nodos (vértices) del grafo. A su vez, un grafo hamiltoniano es aquel que contiene un ciclo hamiltoniano.

Nota: Cuando hablamos de grafos que poseen costes en sus aristas, el ciclo hamiltoniano de menor coste es también la solución del TSP.

***Definición* 2.16.** La versión de decisión del TSP viene dada por:
Entrada- Matriz de distancias asociada al problema y un entero B.
Pregunta- ¿Existe un camino de coste (longitud) menor o igual que B?

***Teorema* 2.3.** El problema de decidir si un grafo es hamiltoniano es NP.

Demostración. Dada una entrada I para el problema, es decir, un grafo G. ¿Se podrá comprobar que G es hamiltoniano en un tiempo polinomial? Por definición sabemos que un grafo es hamiltoniano si contiene un ciclo que pase por todos sus vértices, por lo que se puede obtener un certificado de la respuesta mediante una permutación de sus vértices. Suponiendo, por ejemplo, que los vértices que componen dicho grafo son $v_6, v_3, v_1, v_7, v_5, v_8, \dots v_9, v_2$, se puede comprobar que G es un grafo hamiltoniano averiguando en tiempo $O(n^2)$ si existen las aristas $v_6v_3, v_3v_1, v_1v_7, \dots, v_9v_2, v_2v_6$ en el grafo. De este modo se comprueba la afirmación en tiempo polinomial, y por tanto el problema de decisión pertenece a la clase NP. □

***Teorema* 2.4.** El problema del circuito hamiltoniano es NP-Completo.

Demostración. En la proposición anterior se acaba de demostrar que el problema del ciclo hamiltoniano es NP. Siguiendo la definición, para poder ver que este problema es NP-completo se necesitará comprobar que todo problema de NP es reducible a él en tiempo polinomial. Por ello, vamos a tomar el problema 3-SAT que es NP-completo y a comprobar que se transforma polinomialmente en dicho problema.

Dada una colección de cláusulas de tres literales $Z = \{Z_1, ..., Z_M\}$ sobre un conjunto de variables $U = \{u_1, ..., u_n\}$ como entrada del problema 3-SAT, se quiere construir un grafo G que cumpla que: Z es satisfacible si y solo si G contiene un ciclo hamiltoniano.

Construcción del grafo.

Para la contrucción del grafo se crean dos estructuras distintas:

1. La figura 5.2(a) corresponde a la primera de las estructuras que se van a utilizar. Es un subgrafo A de G donde ningún vértice de A excepto u, u', v, v' inciden con otra arista de G que no esté en A. Por tanto, si se quiere buscar un ciclo hamiltoniano en A se podrá hacer de dos formas distintas como se muestra en la Figura 5.2 (b),(c).

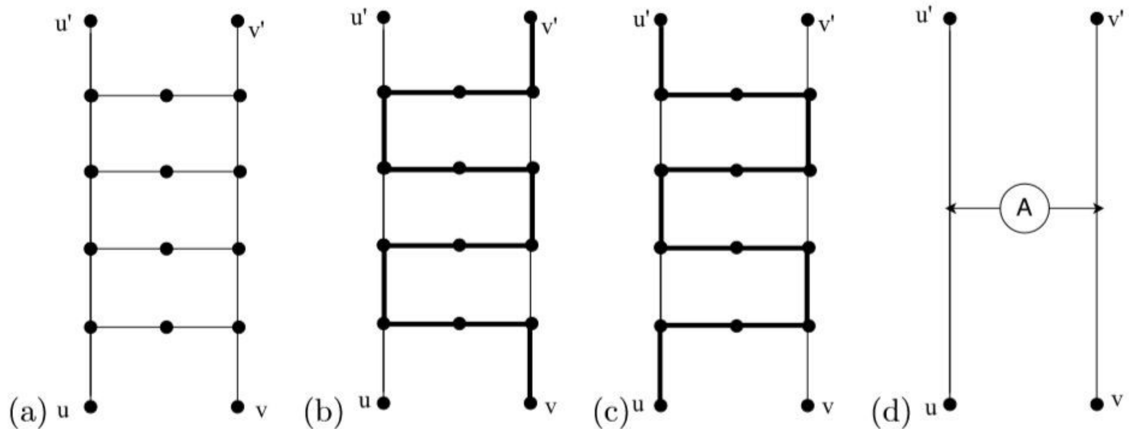


Figura 5.2: Estructura A.

Si se consideran únicamente las dos aristas $u-u'$,$v-v'$, éstas se unirán mediante A cuando entre ellas se cree una estructura igual a la que aparece en la Figura 5.2(a). Para simplificar el resultado, se describe la unión de las dos aristas mediante A como se puede ver en la Figura 5.2(d).

Se puede asegurar que si en un grafo hay una estructura de tipo A, cualquier ciclo hamiltoniano del grafo debe contener exactamente una de las aristas $u-u'$,$v-v'$. Para poder entenderlo mejor nos podemos fijar otra vez en la Figura 5.2 (b),(c). En el caso de (b) se puede apreciar que el ciclo hamiltoniano pasa tanto por el vértice v como por el vértice v' por lo que contiene a la arista $v-v'$. Sin embargo, en la (c) son los vértices u y u' los que están contenidos en el ciclo hamiltoniano, por lo que en este caso la arista contenida es $u-u'$.

2. La segunda estructura necesaria para la construcción es la que representa el grafo B de la Figura 5.3(a). B es también subgrafo de G donde los únicos vértices incidentes con otras aristas de G son u, u'. En este caso ningún circuito hamiltonano de G podrá atravesar a las tres aristas $\{e_1,e_2,e_3\}$. Se representará B para simplificar la estructura como la Figura 5.3(b).

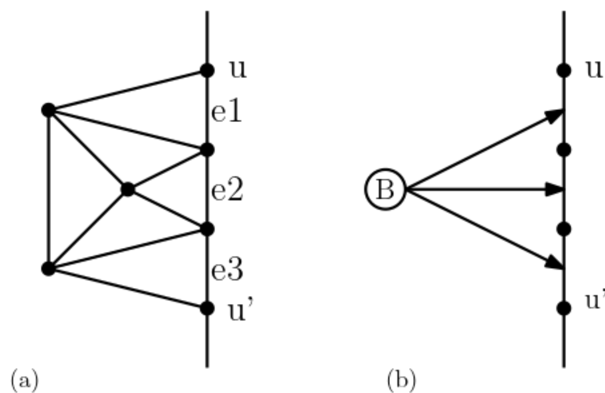


Figura 5.3: Estructura B

Con estas dos estructuras ya se puede construir desde Z el grafo G que se buscaba. Para cada cláusula $Z_{i=1,...,m}$ se pone una copia de B tras otra en una misma columna, y en una columna paralela se pone una sucesión de vértices, dos por cada variable $u_j(j=1,...n)$ que exista en las cláusulas. La primera y la última copia de B se unen con esta sucesión de vértices. Estos vértices están a su vez unidos entre ellos. La arista que une cada par de vértices correspondería a cada variable u_j, esta se dobla para representar a los literales u_j y $\bar{u}_j$. En la Figura 5.4 se puede ver esto gráficamente.

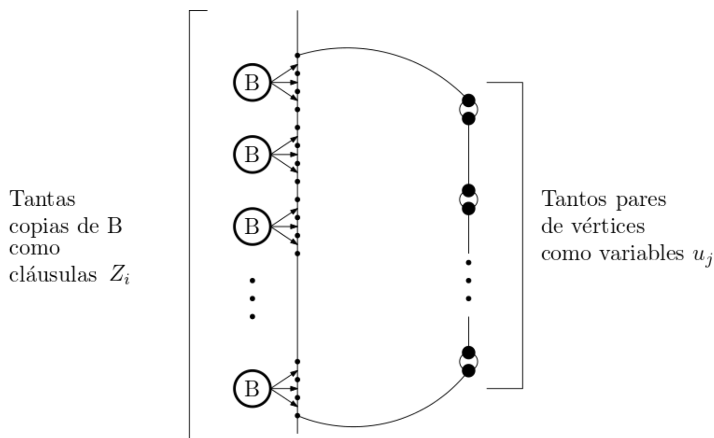


Figura 5.4: Ejemplo

Ahora cada copia de B se une mediante copias de A con la sucesión de vértices, es decir, las aristas e_1, e_2, e_3 de cada B estarán unidas con el primer, segundo y tercer literal de la cláusula correspondiente. Notar que al estar unidos por A cualquier ciclo hamiltoniano de G debe contener o bien a e_i o bien al literal relacionado mediante A con esa arista. La arista correspondiente a un literal puede formar parte de más de una copia de A.

A continuación se da un ejemplo concreto para la construcción de este grafo. Para ello, vamos a tomar como entrada del problema 3-SAT un conjunto de cláusulas concreto, sea $U = \{\{x_1, \bar{x_2}, \bar{x_3}\}, \{\bar{x_1}, x_2, \bar{x_3}\}, \{\bar{x_1}, \bar{x_2}, x_3\}\}$, el grafo correspondiente para U siguiendo lo anteriormente explicado es el que se representa en la Figura 5.5(a). Además, en la Figura 5.5(b) se puede ver la primera parte del grafo ampliada, es decir, la primera estructura de tipo A y B que aparece en el ejemplo anterior, para que ayude a entender mejor lo que se está intentando representar.

Cada una de las estructuras de tipo A de la Figura 5.5(a) se representaría de esta manera. Hay que tener en cuenta que si hay dos cláusulas que tienen el mismo literal, en la arista correspondiente a ese literal aparecerán puntos intermedios independientes para cada una de las dos estructuras A.

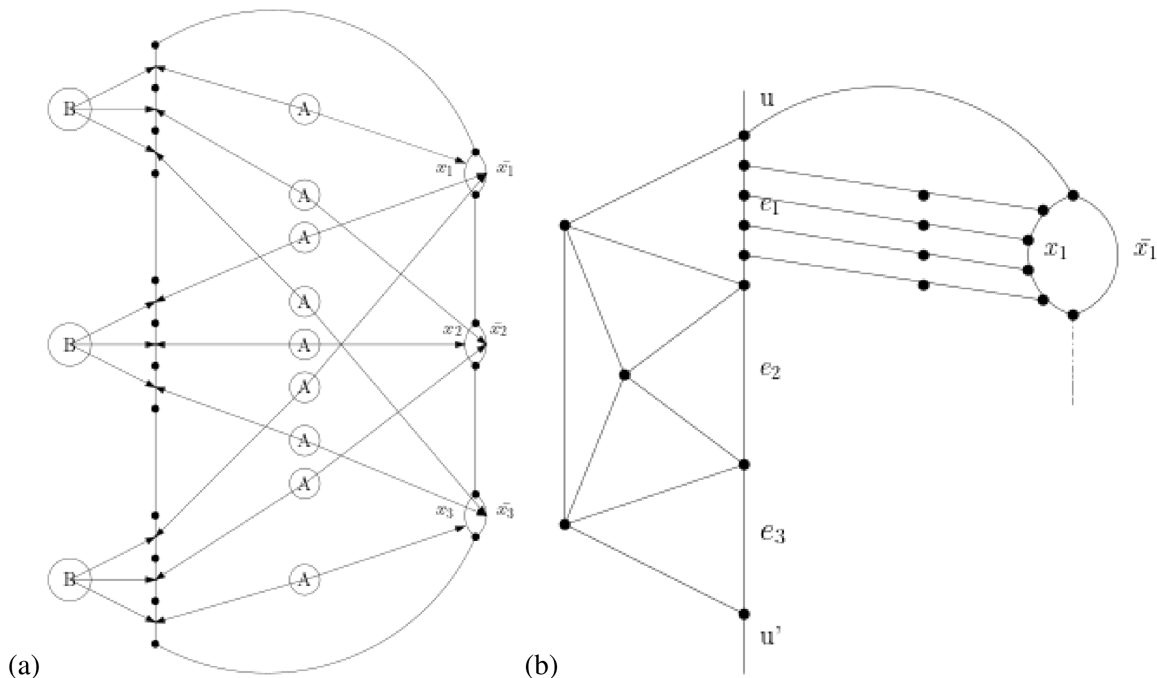


Figura 5.5: Ejemplo

Una vez se ha construido el grafo G, se debe probar:

G es hamiltoniano $\leftrightarrow$ z es satisfacible y la construcción es polinomial.

$\rightarrow$ | Sea C un ciclo hamiltoniano, definimos una correspondencia de verdad T que verifique que $T(x) = true$, y que x literal si y solo si C contiene a la arista correspondiente. Por la propiedad de B, un ciclo hamiltoniano no puede tener a la vez a las aristas e_1, e_2, e_3 correspondientes a una misma cláusula y por la propiedad de A, si el ciclo hamiltoniano no contiene a uno de esas tres aristas, entonces contiene a la arista correspondiente al literal unido mediante A, por lo que toda cláusula contiene un literal tal que la arista correspondiente a ese literal pertenece al ciclo. Luego existe una correspondencia de verdad para U y por tanto z es satisfacible.

$\leftarrow$ | Por otro lado, si Z es satisfacible , entonces cualquier correspondencia de verdad que satisfaga Z define un conjunto de aristas que corresponden a los literales que son verdad. Como cada cláusula contiene al menos un literal cierto (pues Z es satisfacible), entonces este conjunto de aristas puede ser completado hasta un ciclo en G, usando la construcción explicada anteriormente.

Se demuestra así que un problema NP-completo, como es 3-SAT, puede reducirse al problema del ciclo Hamiltoniano, por lo que éste también es NP-completo.

$\square$

***Teorema* 2.5.** El problema de decisión de ciclos Hamiltonianos (HCP) se transforma polinómicamente en el TSP.

Demostración. Se debe encontrar una función f que a cada problema de decisión HCP le corresponda uno del TSP, y que satisfaga las propiedades de transformación polinomial.

Sea $G = (V,E)$ entrada del HCP con $|V| = m$. Su entrada correspondiente en el TSP tiene un conjunto C de ciudades idéntico a V y para cada par de ciudades $v_i, v_j \in C$ su distancia será:

$$d(v_i, v_j)=1 \text{ si } v_i, v_j \in E$$
$$d(v_i, v_j) = 2 \text{ en otro caso.}$$

Se sabe que la longitud máxima del recorrido entre ciudades siguiendo las aristas pertenecientes al grafo es m. Se llamará a esta longitud B.

Ahora bien, se necesita transformar también la pregunta del problema de decisión del HCP a la correspondiente al TSP, es decir, se necesita ver:

G contiene un ciclo hamiltoniano $\leftrightarrow$ hay un camino que pasa por todas las ciudades de $f(G)$ cuyo recorrido no es mayor que B.

Primero $\Rightarrow$

Sea $\{v_1, v_2, ..., v_m\}$ un circuito hamiltoniano de G. Entonces $v_1, v_2, ..., v_m$ es también un tour en $f(G)$ de longitud $m = B$ pues cada una de las aristas visitadas es un elemento de E con longiutd 1.

Ahora $\Leftarrow$

Sea $\{v_1, v_2, ..., v_m\}$ tour en $f(G)$ con longitud total menor o igual que B. Como las distancias entre las ciudades son o bien 2, o bien 1, y se recorren en total m ciudades (m sumandos), entonces la distancia entre las ciudades visitadas es exactamente 1, es decir, por definición de f, $\{v_i, v_{i+1}\}$,$\{v_m, v_1\} \in E$
$\forall$ i=1,...,m-1, luego $\{v_1, v_2, ..., v_m\}$ es un circuito hamiltoniano de G.

$\square$

Por una parte, se ha demostrado que el problema de decisión del ciclo hamiltoniano es NP-Completo; por la otra, se ha visto que dicho problema se puede transformar polinómicamente en el TSP. Por tanto, se puede concluir que el TSP también es un problema NP-Completo.

Capítulo 3

ALGORITMO EXACTO: HELD-KARP

Una vez estudiado qué es el TSP, en qué consiste y su complejidad, es necesario buscar un algoritmo que lo resuelva. Este capítulo se centrará sobre todo en un método exacto para resolver el problema del viajante, conocido como el método de Held-Karp basado en programación dinámica. Cabe destacar que a lo largo de la historia, además de este método, se han planteado muchos otros como es el de los planos de corte u otros métodos basados en algoritmos de ramificación y acotación (*Branch and Bound*).

3.1. ALGORITMO BRUTO

Uno de los algoritmos más básicos propuestos para el TSP consiste en generar todas las permutaciones posibles de los vértices dados y ver cuál es el camino de mínimo coste que une todos ellos. Para esto se siguen los siguientes pasos:

1. Se denota como ciudad 1 aquella que sea el punto de inicio y final del camino.
2. Se generan todas las $(n-1)!$ permutaciones de las ciudades.
3. Se calcula el coste de todas las permutaciones y se elige el camino que corresponde a la permutación de mínimo coste.
4. Se vuelve a la ciudad 1 siguiendo el camino elegido.

El tiempo computacional para este algoritmo es $O(n!)$. Es un tiempo computacional muy grande, por lo que no es un algoritmo útil.

3.2. PROGRAMACIÓN DINÁMICA. HELD Y KARP (1962)

El algoritmo de resolución para el TSP recurriendo a la teoría de programación dinámica surgió a principio de los años sesenta. Held y Karp propusieron la siguiente idea [2]: *en un circuito óptimo tras haber recorrido una serie de ciudades, el camino que atraviese las restantes ciudades debe ser también óptimo*. Este hecho permite construir el camino paso a paso: a partir de una lista de los caminos de mínimo coste entre las ciudades de todos los subconjuntos de tamaño k, especificando en cada caso el origen y el destino, se puede crear una lista de todos esos caminos para conjuntos de k ciudades.

El procedimiento que sigue el algoritmo de Held y Karp es el siguiente:

Sea $\{1,2,...,n\}$ el conjunto de vértices (ciudades) dado, donde 1 es la ciudad de salida y llegada. Para cada vértice i (excluyendo el vértice 1) se debe encontrar el camino de mínimo coste con 1 como punto de inicio e i como vértice final, donde todos los vértices del conjunto aparecen una única vez.

Se denota el coste de estos caminos $cost(i)$, por lo que el coste total del ciclo viene dado por $cost(i) + d(i,1)$ siendo $d(i,1)$ la distancia que hay para llegar desde el vértice i al inicial. De esta manera, se escoge como camino óptimo el ciclo cuyo coste asociado sea menor, es decir, el $min_{i=2,\dots n}[cost(i) + d(i,1)]$. Parece un procedimiento simple pero, ¿cómo obtenemos $cost(i)$? Para poder calcular el $cost(i)$ con programación dinámica, se debe tener una relación recursiva en términos de subproblemas.

Primero, para cada vértice i, se generan todos los posibles subconjuntos con los vértices dados (excluyendo a 1 y a dicho vértice i) , desde el conjunto vacío hasta conseguir el conjunto total, es decir, para cada ciudad i se tendrá una lista de subconjuntos de cada tamaño. Por ejemplo, el grupo de los subconjuntos de tamaño uno estará compuesto por los vértices de manera individual, los de tamaños dos serán conjuntos con pares de vértices y así sucesivamente.

Ahora se define $C(i,S)$ como el coste total del camino de coste mínimo que visita todas las ciudades del conjunto S empezando en 1 y terminando en i. Y se calcula este coste para todos los subconjutos S de manera recursiva empezando por los subconjuntos de tamaño más pequeño.

Es decir, se tiene:

- Si el tamaño de S es 0, S debe ser el conjunto vacío.
  $C(i,S)$ será el coste del camino que parte de 1 y llega hasta i sin tener que pasar por ninguna ciudad.
  $$C(i,S) = C(i,\emptyset) = dist(1,i).$$
- Si el tamaño de S es 1, S debe ser $\{j\}$ donde $j \neq i$ y $j \neq 1$
  $C(i,S)$ será el coste del camino que empieza en 1 y acaba en i pasando por la ciudad j.
  $$C(i,S) = C(i,\{j\}) = dist(i,j) + C(j,\emptyset).$$
- Si el tamaño de S es mayor que 1. $C(i,S)$ es el mínimo coste de los costes de todos los caminos que empiezan en 1 y acaban en i recorriendo todas las ciudades de S.
  $$C(i,S) = min\{dist(i,j) + C(j,S-j)\} \text{ donde } j \in S,\ j \neq i \text{ y } j \neq 1.$$

Para llevar esto a cabo se generan 2^n subconjuntos y para cada subconjunto se pasa por cada vértice. Mientras se pasa por cada vértice se verifica cual debería ser el vértice anterior, luego se está haciendo un trabajo de n^2 para cada subconjunto. Así, el tiempo total de ejecución de este método es $O(n^2 2^n)$ que es mucho menor que $O(n!)$ conseguido mediante el otro método. El principal inconveniente de este método es la cantidad de tiempo computacional que requiere incluso con la tecnología actual. Por eso se restringe su uso a pequeños problemas.

***Ejemplo* 3.1.** Se puede ver un ejemplo para este algoritmo en el Anexo C.

Capítulo 4

ALGORITMOS APROXIMADOS

Normalmente para resolver el problema del viajante se utilizan algoritmos aproximados ya que los algoritmos exactos requieren un tiempo computacional muy grande y no son útiles. Los algoritmos aproximados más frecuentes están basados en métodos heurísticos, es decir, algoritmos que no garantizan una solución óptima pero que las soluciones que encuentran son aproximadas al óptimo.

4.1. ALGORITMOS HEURÍSTICOS

Antes de comenzar con la descripción de los algoritmos heurísticos, que proporcionan una buena aproximación para la solución del TSP, se debe saber qué es un algoritmo heurístico y cuáles son sus propiedades.

***Definición* 4.1.** En programación se dice que un *algoritmo es heurístico* cuando la solución no se determina de forma directa, sino mediante ensayos, pruebas y reensayos.

Todos los algoritmos heurísticos siguen un método similar. Primero se generan los candidatos de soluciones posibles de acuerdo a un patrón dado. Después, estos candidatos son sometidos a pruebas de acuerdo a un criterio que caracteriza la solución. Si un candidato no es aceptado se genera otro y los pasos dados con el candidato anterior no se consideran. Por esta razón, este tipo de algoritmos también se denominan “ algoritmos con vuelta atrás ”.

4.1.1. MEDIDA DE CALIDAD DE UN ALGORÍTMO

Un buen algoritmo heurístico se caracteriza por ser eficiente, robusto y bueno, es decir, debe tener un esfuerzo computacional realista para obtener la solución, siendo muy baja la probabilidad de obtener una solución lejana a la óptima.

Para medir la calidad de un heurístico existen diversos procedimientos, entre los que se encuentran los siguientes:

- ***Comparación con la solución óptima:*** para cada uno de los ejemplos se mide la desviación porcentual de la solución heurística frente a la óptima, calculando después el promedio de estas desviaciones.
- ***Comparación con una cota:*** muchas veces no podemos obtener un algoritmo óptimo aunque se cuente con un conjunto reducido de ejemplos. En estos casos, podemos evaluar la calidad de nuestro heurístico con una cota del problema.
- ***Comparación con otros heurísticos:*** es uno de los métodos más empleados. Se usan algoritmos heurísticos ya conocidos y se compara con el nuevo. Para ello, se somete a comparación su tiempo de ejecución, su facilidad de implementación, su flexibilidad e incluso su simplicidad.

4.2. MÉTODOS CONSTRUCTIVOS DEL TSP

Los métodos constructivos son métodos iterativos que se basan en tomar un subtour inicial, normalmente elegido de manera aleatoria, y a partir de ahí en cada paso se añade un elemento más hasta completar la solución. En el caso del TSP en cada iteración se añade una ciudad nueva hasta completar el recorrido óptimo. Para ello, en cada paso, se debe decidir entre qué ciudad se selecciona para insertarla en el subciclo correspondiente y dónde se insertará la ciudad.

4.2.1. VECINO MÁS PRÓXIMO

El método del vecino más próximo es un algoritmo heurístico de los más sencillos. Consiste en construir un ciclo Hamiltoniano de bajo coste basándose en el vértice más cercano a uno dado. Para llevarlo a cabo se siguen los siguientes pasos:

1. Elección de un vértice inicial arbitrario v_1.
2. Búsqueda del vértice más cercano a v_1 e inserción de éste en el recorrido. Este nuevo vértice pasará a estar marcado como vértice visitado.
3. Si el subtour actual está formado por los vértices $v_1, v_2, ..., v_k$, se busca el vértice más próximo a v_k, que será v_{k+1}, pasando éste a formar parte del tour.
4. Si todos los vértices del conjunto han sido visitados, cerramos el algoritmo y se vuelve al vértice inicial v_1. Si no, se vuelve al paso 3.

Se puede ver el algoritmo enunciado de manera matemática:

Algoritmo del vecino más próximo:

Inicialización:

- Seleccionar un vértice j al azar.
- Hacer $t = j$ y $W = V \backslash \{j\}$ donde t es el vértice actual en cada iteración y V los vértices del conjunto.

Mientras ($W \neq \emptyset$): Tomar nuevo j de W tal que $c_{tj} = min\{c_{ti}$ con i en $W\}$.

Para llevar a cabo este método se realiza un número de operaciones de orden $O(n^2)$. Si se lleva a cabo el proceso para un ejemplo dado, se podrá observar que al principio comienza muy bien, seleccionando aristas de coste bajo. Pero en las últimas iteraciones del proceso probablemente queden vértices cuya conexión obligará a introducir caminos más costosos.

Este algoritmo puede codificarse en pocas líneas. Sin embargo, una implementación directa será muy lenta para ejemplos de gran tamaño (1000 vértices), por lo que hasta para un heurístico sencillo como éste es importante tener en cuenta la eficiencia y la velocidad del algoritmo.

Se pueden incluir diversas mejoras al algoritmo tales como:

- Al seleccionar el vértice j que se va a unir a t (tour en construcción), en lugar de examinar todas las conexiones posibles con otros vértices, se tienen en cuenta solo los adyacentes a t en el subgrafo candidato. En caso de que todos los adyacentes ya formen parte del tour se examinarán todos los posibles.

- Cuando un vértice queda conectado al tour, es decir, tiene dos enlaces, se eliminan del subgrafo candidato las aristas incidentes a él.
- Se especifica un número $s < k$. Si un vértice no está en el tour o está conectado únicamente a un número de aristas del subgrafo menor o igual que s, entonces se considerará que se está quedando aislado y por ello se insertará inmediatamente en el tour. Como punto de inserción se toma el mejor de los k vértices más cercanos presentes en el tour.

En el peor de los casos, el algoritmo da como resultado un recorrido que es mucho más largo que el recorrido óptimo. Para ser precisos, para cada constante r hay una entrada del problema del viajante, de modo que la longitud calculada con el algoritmo del vecino más próximo es mayor que r veces la duración del recorrido óptimo. Esto se recoge en el siguiente teorema:

***Teorema* 4.1.** Para todo $r > 1$ existe una entrada de n-ciudades para el TSP, que cumple la desigualdad triangular y donde n es un número arbitrario grande tal que:

$$\text{NN(I)} \geq r\ \text{OPT(I)}$$

donde NN representa la longitud calculada con el algoritmo del vecino más próximo.

Demostración. Para $i > 0$ se construye una entrada F_i para la cual el algoritmo del vecino más próximo funcione mal, encontrando un ciclo cuya longitud sea al menos $(i+2)/6$ veces la longitud óptima.

Teniendo en cuenta que esta relación va aumentando conforme i tiende a infinito, las entradas F_i se construyen de manera recursiva. En la Figura 4.1(a) se puede ver F_1. Al igual que todos los F_i posteriores, F_1 tiene tres vértices distinguidos (en el caso de F_1, no tiene más que esos tres): un punto final a la izquierda A, un punto medio B y un punto final a la derecha C.

En la Figura 4.1 (b) se puede ver F_2 y F_3. La entrada F_i, $i > 1$, se construye a partir de dos copias de F_{i-1} junto con tres ciudades adicionales, como muestra la Figura 5.2(c). La ciudad E es el punto medio de F_i, y las ciudades A y C' son los puntos finales izquierdo y derecho, respectivamente. Teniendo en cuenta que en la figura solo se especifican algunas de las distancias c_{ij}, si c_{ij} no se especifica en la figura, se toma como la longitud del camino más corto entre las ciudades i y j a través de los ejes que sí que poseen una distancia específica en el grafo. Esta elección de longitudes garantizará la desigualdad triangular.

Si llamamos f_i al número de ciudades en la etapa i, por construcción se verifica que $f_i = 2f_{i-1} + 3$ y la solución de esta recurrencia con f_1=3 es $f_i = 3(2^i - 1)$. Un posible ciclo puede ser aquel que atraviese las ciudades en orden, de izquierda a derecha, y después volver directamente al punto de inicio. Se tiene OPT(I)$\leq 6(2^i)$-8, el término derecho de la desigualdad lo extraemos de recorrer dos veces las conexiones que unen las ciudades, es decir, $2(3(2^i - 1) - 1) = 6(2^i) - 8$ que es el camino que se realiza al seguir las distancias conocidas.

Por otro lado, el algoritmo del vecino próximo podría, dada una secuencia desfavorable de opciones donde se enfrenta con “lazos” entre posibles vecinos más próximos, encontrar un ciclo de longitud $(i+2)2^i - 3$, como vemos para F_3 en la Figura 4.1(d). Es necesario ver de dónde sale esta longitud:

- Para empezar se parte de la figura F_1 que empieza en el punto A, sigue en C con costo $AC = 1$ y termina en B con $BC = 1$, por lo que la longitud de este camino es $l_1 = 2$.
- Si ahora se tienen dos copias de la figura F_{i-1}, cada figura de manera independiente representa un camino que empieza en A y termina en el centro, es decir, en B, y como se ha visto tendrá longitud l_{i-1}.

- Éstas se combinan como se ve en la Figura 4.1(c) y se añaden 3 puntos intermedios con distancia 1, estos puntos son D, E y D'. Siguiendo el camino de la copia izquierda, se une el punto B con D' con un arco $BD' = 2^{i-1}$, de D' vamos al comienzo de la copia derecha de F_{i-1} con $D'A'$ de costo 1, se recorre la segunda copia como hemos hecho para la primera, llegando así a B'. Y se conecta B' con D mediante el arco $B'D = 2^{i-1}$ y de D se va al punto central E y así se define el camino F_i, que empieza en su primer extremo A y acaba en su centro E.

Por tanto, la longitud l_i viene dada por $l_i = 2 * l^{i-1} + 2 * 2^{i-1} + 2$ (dos copias de F_{i-1}, más dos arcos de longitud 2^{i-1}, más dos arcos de longitud 1). Esta recurrencia con longitud inicial $l_1 = 2$ da como resultado la longitud $(i+1)2^i - 2$ y añadiendo el arco EA (arco de vuelta al nodo de origen) de costo 2^{i-1} se consigue la longitud que se buscaba.

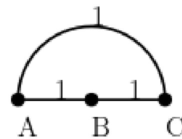


FIGURA 4.1 (a) - F_1

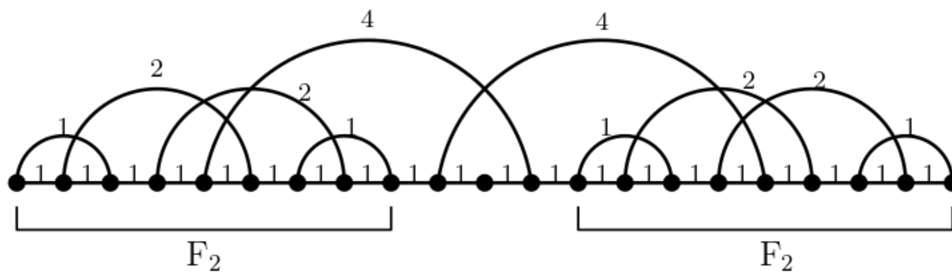


FIGURA 4.1 (b) - F_3:Número de ciudades 21.

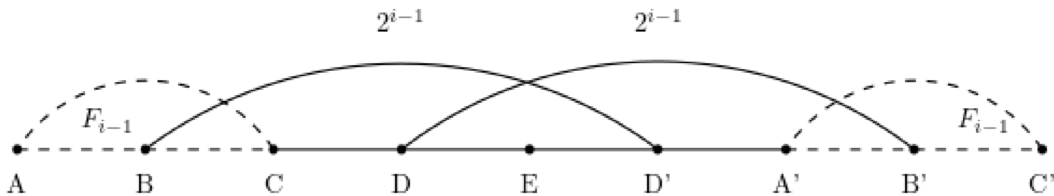


FIGURA 4.1 (c) - Contrucción recursiva F_i, $i > 0$.

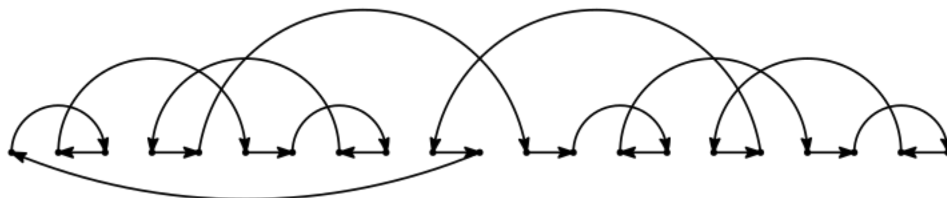


FIGURA 4.1 (d) - Tour conseguido mediante el método del vecino más próximo aplicado a F_3.
Longitud: $(i+2)2-3 = 37$.

□

4.2.2. ALGORITMO DE LIN-KERNIGHAN

La heurística de Lin-Kernighan para el problema del viajante (TSP) data del año 1973, pero es considerada todavía la mejor heurística disponible para este problema. Consiste en que k ejes de un recorrido factible se intercambien por k ejes que no están en ese camino, siempre y cuando el nuevo recorrido siga siendo solución y su longitud sea menor que la del recorrido anterior. Aunque aquí lo encontremos junto los heurísticos constructivos también se puede clasificar como un algoritmo de búsqueda local.

***Definición* 4.2.** Dado un ciclo hamiltoniano con dos aristas $\{i, j\}$ y $\{k,l\}$ incidentes en cuatro nodos distintos y tales que el camino visita los nodos de la siguiente manera i,j,k,l; un 2-intercambio asociado con las aristas $\{i, j\}$ y $\{k,l\}$ consiste en reemplazarlas por las aristas $\{i,k\}$ y $\{j,l\}$.

***Ejemplo* 4.1.**

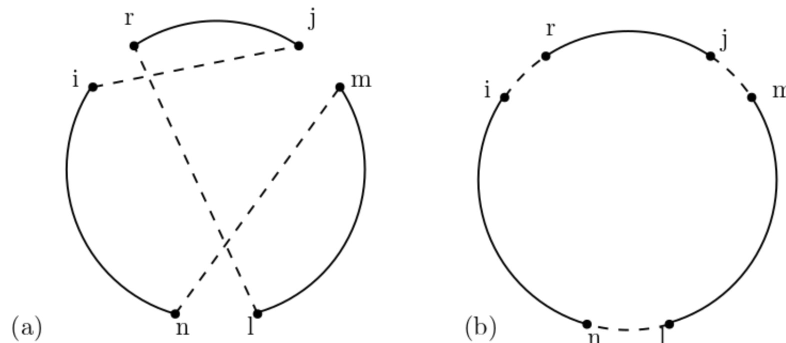


Figura 5.4: ejemplo 3-opt.
(a) Camino inicial. (b) Camino después de los intercambios.

Nota: El realizar un 2-intercambio en un ciclo hamiltoniano da lugar a un nuevo ciclo hamiltoniano, donde la diferencia de costes entre el ciclo original y el nuevo es la diferencia de costes entre las aristas intercambiadas. A partir de la definición de 2-intercambio se define de la misma forma el concepto k-intercambios.

Los procedimientos de intercambio son denominados procedimientos k-opt, donde k es el número de ejes intercambiados en cada iteración. En un algoritmo k-opt, todos los intercambios de k ejes se prueban hasta que no hay intercambio factible que mejore la solución actual. Esta solución se conocerá como k-óptimo.

En general, cuanto mayor sea el valor de k mayor probabilidad habrá de que la solución final sea óptima, pero el número de operaciones necesarias para probar todos los intercambios k aumenta rápidamente a medida que incrementa el número de ciudades. Por ello, los valores $k = 2$ y $k = 3$ son los más comunes. El planteamiento es el siguiente:

El algoritmo de Lin-Kernighan decide de manera dinámica en cada iteración cuántos ejes hay que intercambiar, es decir, se decide el valor de k. Para ello, se empieza eligiendo un número s de intercambios donde $s < k$ y se realizan una serie de pruebas para determinar si se deben considerar los intercambios de $s+1$ aristas. Esto continúa hasta que se cumplen las condiciones de parada obteniendo así el valor k deseado.

Los pasos del algoritmo de Lin-Kerninghan son:

Algoritmo de Lin-Kernighan:

Dada la figura 5.4, se va a ilustrar como funciona el algoritmo. Sea G_p^* la mejora o el ahorro que se puede obtener si los p ejes considerados se intercambian para generar un nuevo recorrido. En términos generales, el procedimiento funciona de la siguiente manera.

Paso 1 Elegir un recorrido inicial.

Paso 2 Tomar G_p^*=0. Elegir algún nodo como origen (e.g nodo l) y uno de los ejes que sale de este vértice y que además pertenezca al recorrido (e.g $\{l,r\}$). Poner $p = 1$.

Paso 3 Desde el otro punto final de este eje (nodo r) elegir otro eje que no esté en el recorrido donde $g_1 = c_{lr} - c_{ri} > 0$. El eje se elige de manera que g_1 se maximice (g_i es la ganancia al intercambiar una arista por otra).

Paso 4 Habiendo elegido $\{l, r\}$ para salir y $\{r, i\}$ para entrar en la solución de la iteración anterior, el eje que abandona en la $p-$ésima iteración está determinado. Debe de ser uno de los dos ejes adyacentes del nodo i de manera que al quitarlo siga el recorrido conectado. En este caso, $\{i, j\}$ debe salir de la solución. Notando que al añadir $\{j, l\}$ se reconstruye el camino, sería un candidato para entrar. Ahora $G_1^* = g_1 + c_{ij} - c_{jl}$. Suponiendo que $g_1 > G_1^* > 0$. Incrementar p en 1.

Paso 5 El eje $\{j, l\}$ no tiene porque ser el eje que entre en esta iteración. Es decir, nosotros buscamos un nodo q que maximice $g_p = c_{ij} - c_{jq}$ si en vez de tomar el nodo l tomamos $q = m$ observamos que este eje aún nos produce una mejora mayor por lo que éste es el eje que entra $\{j, m\}$. Una vez elegidas las conexiones que cambian se debe calcular $G_p = \sum_{s=1}^{p} g_s$ y luego $G_p^* = G_p + c_{mn} - c_{nl}$. Siendo $G^* = max\{G_0^*, G_1^*, ..., G_p^*\}$.
Incrementamos p y se repite el Paso 5 a no ser que:

(a) No existan más intercambios factibles.

(b) La configuración actual ya es un recorrido.

(c) $G_p \leq 0$.

(d) $G_p \leq G^*$.

Si una de estas condiciones se cumple, se construye el tour asociado con el mejor de $\{G_1^*, G_2^*, ..., G_p^*\}$. Todo el proceso se repite hasta que todo vértice es usado como punto de origen y no se hayan encontrado mejoras adicionales.

4.2.3. ÁRBOL DE EXPANSIÓN MÍNIMA(MINIMUM SPANNING TREE)

***Definición* 4.3.** Un árbol de expansión (spanning tree) para un conjunto de n ciudades es una colección de $n - 1$ aristas que une todas las ciudades en una única componente.

La búsqueda del árbol de expansión mínima consistirá pues en encontrar el árbol de expansión que menor coste tenga. Este problema puede ser resuelto de manera eficiente, ya que si un problema viene dado por una matriz de distancias C, ésta tendrá un tamaño proporcional a n^2 y por tanto se podrá calcular el árbol de expansión mínima en un tiempo $O(n^2)$.

Además, esta solución proporciona un límite inferior para el recorrido óptimo, ya que al eliminar cualquier eje del ciclo óptimo del problema se consigue un árbol de extensión, es decir, un camino simple que pasa por todas las ciudades. Por tanto, el recorrido óptimo para el problema del viajante debe ser estrictamente mayor que la longitud del árbol de expansión mínima.

Ahora bien, si lo que se quiere es conseguir un recorrido que visite todas las ciudades dadas de un conjunto a partir del árbol de expansión mínima se hace lo siguiente: se duplican las aristas del grafo (ramas del árbol) y se aplica el algoritmo de Euler para encontrar un ciclo que contenga a todos los ejes una sola vez. Al duplicar las aristas del grafo se consigue que todos los vértices tengan grado par y por tanto el grafo será euleriano (*Teorema de Euler:* Un grafo conexo es euleriano si y solo si todos los vértices del grafo tienen grado par) y así se puede asegurar que se puede construir un camino que pase por todos los vértices, donde el vértice inicial y el final sea el mismo y que todas las ramas del árbol sean recorridas.

Nota: En teoría de grafos el grado de un vértice es el número de aristas incidentes al vértice.

Así pues, la distancia que se ha recorrido siguiendo el proceso es, por la propiedad triangular, menor o igual que dos veces la del árbol mínimo. Pero con este método algunas ciudades son visitadas más de una vez, cosa que no se permite en el problema del TSP. Para que el recorrido únicamente visite todos los nodos una vez se recurre a la desigualdad triangular, ya que se puede evitar repetir ciudades introduciendo ”atajos”, es decir, se sigue el procedimiento anterior pero cada vez que se llegue a un vértice ya visitado, este se salta y se va directamente al próximo que no haya sido visitado todavía. Y si todos los vértices ya han sido visitados, volvemos al inicial.

Como el recorrido obtenido de este modo aún es más corto que el obtenido anteriormente, este recorrido estará acotado superiormente por dos veces la longitud del árbol de extensión mínima y, por tanto, según las observaciones anteriores sobre los límites inferiores, como máximo será el doble del óptimo del problema del viajante.

Se tiene entonces, un algoritmo de tiempo polinomial que proporciona cierta garantía. Se puede ver el algoritmo matemático del árbol de expansión mínima a continuación.

Algoritmo del árbol de expansión mínima:

1. *Buscar un árbol de extensión mínima A para el conjunto de vértices dados -* $O(n^2)$.
2. *Construir un primer recorrido de A, duplicando las aristas y aplicando el algoritmo de Euler -* $O(n)$.
3. *Meter atajos en el recorrido anterior para obtener el tour buscado -* $O(n)$.

El tiempo de ejecución del algoritmo está dominado por el tiempo de búsqueda del árbol de extensión mínima en el *Paso 1*, por lo que el tiempo de ejecución es $O(n^2)$.

***Teorema* 4.2.** Para todo TSP con entrada I que obedece a la desigualdad triangular, si $\overline{MST(I)}$ es la longitud de un tour construido por el algortimo del árbol de expansión mínima aplicado a I, se tiene:

$$\overline{MST(I)} \leq 2OPT(I)$$

Demostración. Se considera $MST(I)$ como la longitud del árbol de expansión mínima, y $\overline{MST(I)}$ tal y como se describe en el enunciado del teorema. Según lo anteriormente explicado se tiene que, por una parte, el recorrido óptimo para el problema del viajante debe ser estrictamente mayor que la longitud del árbol de expansión mínima, es decir, $MST(I) < OPT(I)$. Por otra parte, se tiene que la longitud del camino generado a través del algoritmo está acotada superiormente por dos veces la longitud de expansión mínima, es decir, $\overline{MST(I)} \leq 2MST(I)$.
Y juntando estas dos expresiones se llega a lo que se quería demostrar:

$$\overline{MST(I)} \leq 2MST(I) \leq 2OPT(I).$$

□

***Ejemplo* 4.2.** Se puede ver un ejemplo para este algoritmo en el Anexo D.1.

4.2.4. MATCHING DE MÍNIMO PESO O ALGORITMO DE CHRISTOFIDES

Christofides propuso una nueva visión para el problema del TSP, que provenía del concepto de grafo euleriano ya mencionado en el apartado anterior.

La idea general es parecida a la del método del árbol de expansión mínima, donde se empieza con un árbol de mínima expansión, se duplican sus ramas para obtener un grafo euleriano y se busca el camino más corto o de menor coste usando "atajos". Sin embargo, este nuevo método, en vez de conseguir el grafo euleriano duplicando las ramas del árbol de mínima expansión, plantea encontrar una mejor manera de generar un grafo euleriano que conecte las ciudades consiguiendo así mejores rutas para el TSP. Fue en 1976 cuando Christofides introdujo el concepto de matching de mínimo peso.

***Definición* 4.4.** Dado un número par de vértices , un *matching* es una colección M de ejes, de modo que cada vértice es el extremo de un único eje en M. Y un matching de mínimo peso es un matching para el cual la longitud total de sus ejes es mínima. Estos matchings de mínimo peso se pueden encontrar en un tiempo $O(n^3)$.

En un árbol de expansión mínima T para un TSP, no todos los vértices tienen el mismo grado. Algunos son de grado par y otros, por el contrario, de grado impar. Además, hay que tener en cuenta que por definición la suma de todos los vértices de un grafo euleriano es par, por lo que debe haber un número par de vértices impares.

Ahora bien, debemos enfrentamos a la búsqueda de la solución del problema. Como ya se ha mencionado, se empieza con un árbol de expansión mínima que contenga todos los vértices del conjunto dado y hay que convertir ese árbol en un grafo euleriano. En vez de duplicar todas las ramas del árbol, con el algoritmo de Christofides los únicos vértices que habrá que modificar son aquellos que tengan grado impar, añadiéndoles a éstos un matching. Esto incrementará en una unidad los grados de los vértices impares, quedando ahora todos con un número par de conexiones y por tanto, si se suma a T el matching de mínimo peso para los vértices impares, se obtiene un grafo euleriano con longitud mínima conteniendo a T, que es lo que se buscaba. Una vez obtenido esto se buscará, dentro de este nuevo grafo, el camino óptimo de la misma forma que se hacía con el método anterior.

En resumen:

Algoritmo de Christofides:

Suponemos un grafo F asociado al TSP.

a) *Crear un "mininimum spanning tree" T de F - $O(n^2)$.*

b) *Obtener el conjunto de vértices de grado impar en el árbol T - $O(n)$.*

c) *Encontrar un matching de mínimo peso (refiriéndonos a peso como a la distancia o coste total del recorrido) para los vértices de grado impar y añadirlo a T - $O(n^3)$.*
Consiguiendo un grafo con todos sus vértices de orden par P, el cual será euleriano.

d) *Buscar un tour euleriano para P - $O(n)$.*

e) *Aplicar los " atajos" para evitar repeticiones - $O(n)$.*

El tiempo de ejecución del algoritmo está dominado por el tiempo de búsqueda del matching en el paso c, por lo que el tiempo de ejecución es $O(n^3)$.

***Teorema* 4.3.** Para toda entrada I del TSP que cumpla la desigualdad triangular, se cumple:

$$\mathrm{C(I)}<\tfrac{3}{2}\mathrm{OPT(I)}$$

Demostración. Por definición del algoritmo de Chistofides, se tiene que la longitud del camino conseguido por este método es igual a la suma de la longitud del árbol de expansión mínima y la longitud del matching, es decir, $C(I) = MST(I) + M(I)$.
Sabemos que $MST(I) < OPT(I)$. Además, el ciclo óptimo produce dos matchings M y M' entre los nodos impares, cuya suma de longitudes es menor o igual que la longitud del ciclo óptimo, es decir, $M(I) + M'(I) \leq OPT(I)$. Esto se debe a que cada conexión del matching satisface la desigualdad triangular y, por tanto, uno de los matchings siempre tendrá menor longitud que el otro implicando $M(I) \leq M'(I) \leq OPT(I)/2$.
Juntando todas las condiciones se llega a que

$$C(I) =< MST(I) + M(I) < OPT(I) + OPT(I)/2 = \frac{3}{2}OPT(I).$$

□

Nota: Siendo en este caso $MST(I)$ la longitud del árbol de expansión mínima conteniendo a I, C la longitud del camino conseguido mediante el algoritmo de Chistofides y $OPT(I)$ la solución óptima del TSP.

***Ejemplo* 4.3.** Se puede ver un ejemplo de este algoritmo en el Anexo D.2.

Hasta ahora todos los algoritmos heurísticos estudiados han sido constructivos. Sin embargo, existen otros como son los heurísticos de búsqueda local en el problema del viajante o métodos combinados que no se tratarán en este trabajo.

BIBLIOGRAFÍA

[1] E.L.LAWLER,J.K.LENSTRA, A.H.G.RINNOOY KAN Y D.B.SHMOYS , *The Traveling Salesman Problem*, fecha de publicación original 1985.

[2] M.HELD,R.M.KARP, *A dynamic programming approach to sequencing problems*, Journal of the Society of Industrial an Applied Mathematics b10 (1962) 196-210.

[3] JONATHAN L. GROSS, JAY YELLEN, MARK ANDERSON , *Graph Theory and Its Applications*, fecha de publicación original 1998.

[4] WIKIPEDIA CONTRIBUTORS, *Travelling Salesman Problem — Wikipedia, The Free Encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Travelling_salesman_problem&oldid=962245585`.

[5] FLOOD, M.M., *The Travelling Salesman Problem*, Operations Research 4 (1956) 61-75.

[6] PAPADIMITRIOU, C.H., *The Euclidean traveling salesman problem is NP-complete*, Theoretical Compuler Science 4 (1977) 237-244.

[7] FLOOD, M.M , *Merrill Flood (with Albert Tucker), Interview of Merrill Flood in San Francisco on 14 May 1984*, The princeton Mathematics Community in the 1930s, Transcript number 11 (PMC11). Princeton University (1984).

[8] GEEKSFORGEEKS, *Travelling Salesman Problem*, `https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/`.

[9] AMEDEO R.ODONI, *Networks:Lecture 2*, `https://ocw.mit.edu/courses/civil-and-environmental-engineering/1-203j-logistical-and-transportation-planning-methods-fall-2006/lecture-notes/lec15.pdf`.

[10] KARL MENGER, *Bericht über ein mathematisches Kolloquium. Monatshefte für Mathematik und Physik 38* (1931), 17-38.

ANEXO A

ENSAMBLADO DE FRAGMENTOS DE ADN

Hoy en día crear una secuencia de ADN es bastante automático. Especificando la secuencia de pares de bases que se necesitan, una máquina puede crear el ADN por nosotros. La síntesis automática de ADN funciona mediante el uso de la reacción en cadena de la polimerasa. En este proceso, pequeñas secciones de ADN se unen y se reproducen hasta que haya una cantidad utilizable. Esto funciona bien, pero si el ADN deseado contiene secciones repetidas, el proceso falla porque los fragmentos de ADN pueden ensamblarse de múltiples formas.

La proteína que codifica el ADN es una secuencia de aminoácidos, y cada codón (tres letras de ADN), especifica un solo aminoácido. Hay 61 codones que producen solo 20 aminoácidos, por lo que una secuencia de ADN se puede cambiar sustituyendo los codones por otros equivalentes y aún así generar con ella la misma proteína. El poder sustituir codones equivalentes es la clave para resolver el problema de secuencia repetida. Todo lo que tenemos que hacer es cambiar los codones para eliminar las repeticiones en el ADN mientras se conservan las repeticiones en la proteína producida. Esto es lo que los científicos de la Universidad de Duke han hecho usando una analogía con el problema del viajante. La tarea es encontrar la secuencia menos repetitiva que todavía produce la misma proteína.

Para ello partimos de un problema simple (SSP):

PROBLEMA: *Dado un conjunto de cadenas, encontrar la cadena más corta que contenga a todas estas.*
ENTRADA: Cadenas $s_1, s_2, \ldots, s_n$
PREGUNTA: Cadena s que contiene a todas las cadenas $s_1, s_2, \ldots, s_n$*, tal que la longitud de s sea mínima.*

Y lo transformamos en un TSP, para esto se debe:

- Definir *superposición* (s_i, s_j) como la longitud del prefijo más largo de s_j que coincide con un sufijo de s_i. Por ejemplo si:

  s_i : *aaaggcatcaaatct***aaaggcatcaaa**
  s_j : **aaaggcatcaaa***tctaaaggcatcaaa*

  La superposición (s_i, s_j)=12

- Construir un grafo de n vértices representando las n cadenas $s_1, s_2, \ldots, s_n$.
- Insertar ejes entre los vértices s_i y s_j cuya longitud de superposición es (s_i, s_j).

- Encontrar el camino más corto que visite todos los vértices una vez.

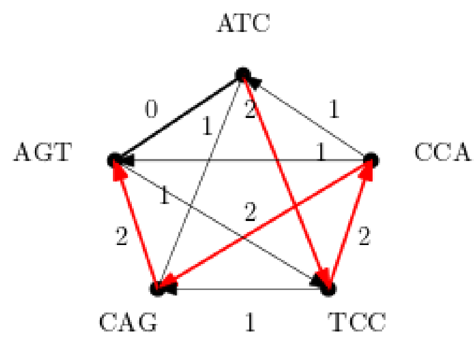
Siguiendo todos estos pasos se llega al problema del viajante (TSP), que es lo que se deseaba.

***Ejemplo* A.1.** Dado un conjunto S={ATC, CCA, CAG, TCC, AGT }
Tanto con el SPP y TSP se llega al mismo resultado.

SPP

Buscando la mínima secuencia que contiene a todas las cadenas: ATCCAGT.

TSP



Se consigue: ATCCAGT.

ANEXO B

TSP EUCLÍDEO

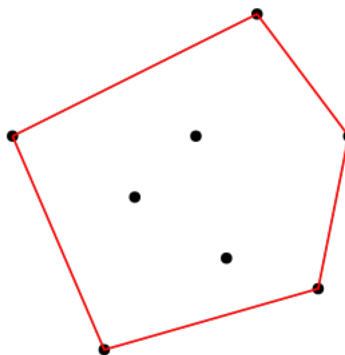
Descripción genérica:

TSP EUCLÍDEO.
ENTRADA: n puntos en $\mathbb{R}^2$ con distancias euclídeas, es decir, $d(x,y) = \|x-y\|_2$.
PREGUNTA: encontrar el camino más corto que visite todos los puntos.

A lo largo de este Anexo se presentarán una serie de resultados elementales sobre el TSP Euclídeo y después se dará el boceto de un algoritmo para resolver un caso especial del TSP Euclídeo de n-ciudades.

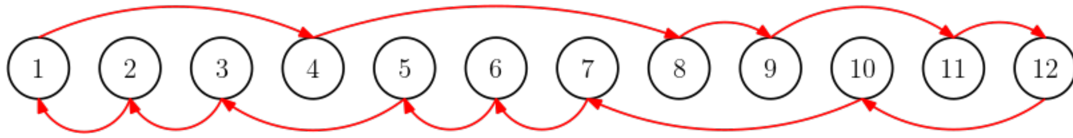
B.1. RESULTADOS ELEMENTALES SOBRE EL TSP EUCLÍDEO

***Definición* B.1.** Se define la *clausura convexa* de un conjunto de puntos X de dimensión n como la intersección de todos los conjuntos convexos que contienen a X.



Clausura convexa de un conjunto
de 8 puntos en el plano.

***Definición* B.2.** Un recorrido que pasa por n ciudades se llama *piramidal* si tiene la forma $(1 = i_0, i_1 \ldots, i_\gamma, n, j_1, \ldots, j_{n-r-2})$ con $i_i < i_2 < \ldots i_\gamma$ y $j_1 > j_2 > \ldots > j_{n-r-2}$, es decir, si empieza en la ciudad 1 y después visita las ciudades en orden creciente hasta llegar a la ciudad n, y finalmente regresa a través de las ciudades restantes en orden decreciente hasta la ciudad de inicio.



Ejemplo de ciclo piramidal $< 1,4,8,9,11,12,10,7,6,5,3 >$

A lo largo de la historia algunos matemáticos han formulado condiciones para la matriz de distancias bajo la condición de que un recorrido óptimo es al menos tan largo como el camino piramidal más corto. Estas matrices son denotadas *piramidal solucionables*, para cada matriz en esta clase hay un recorrido óptimo que es piramidal . Aunque el número de recorridos piramidales en n ciudades es exponencial, un recorrido piramidal de costo mínimo puede determinarse en tiempo $O(n^2)$ mediante un enfoque de programación dinámica como se puede ver en el capítulo 4 de [1].

Un resultado muy conocido sobre el TSP Euclídeo fue dado por Flood que decía: *"En el plano euclídeo el recorrido mínimo (u óptimo) no se cruza consigo mismo"*.

Si se considera los nodos de una entrada para el TSP y se asume que no todas las ciudades se encuentran en una misma línea, un recorrido óptimo visitará en su orden cíclico a las ciudades en la frontera de la clausura convexa ya que si no fuera así habría una intersección y eso no es posible. Por el contrario, si todas las ciudades n se encuentran en una línea y están etiquetadas según su orden en la línea, por definición un camino es óptimo si y solo si es piramidal y este podrá conseguirse en un tiempo $O(n^2)$. Se debe tener en cuenta que el caso en el que todas las ciudades se encuentran en 2 líneas paralelas corresponde al caso en el que todas las ciudades están en la frontera de la clausura convexa.

Culter dio, por ejemplo, un algoritmo de programación dinámica con tiempo $O(n^3)$ para resolver el llamado TSP con puntos en 3 líneas, es decir, el TSP euclídeo en el que todos los puntos están en tres líneas paralelas en el plano. Los resultados se extendieron al TSP con puntos en N líneas, es decir, al TSP euclídeo en el que todos los puntos están en N líneas paralelas, con N un entero pequeño.

A continuación se va a ver un caso especial del TSP euclídeo, este es extensión del TSP con puntos en 3 líneas. Es conocido como el TSP para una clausura convexa y una línea interior.

B.2. TSP PARA UNA CLAUSURA CONVEXA Y UNA LÍNEA INTERIOR

En esta sección se va a dar un boceto de un algoritmo de tiempo $O(mn)$ para el caso especial del TSP euclídeo de n ciudades, donde $n-m$ ciudades se encuentran en la frontera de la clausura convexa de las n ciudades, y las otras m ciudades se encuentran en un segmento de línea dentro de esta clausura convexa.

Los puntos del segmento que se encuentra en el interior de la clausura convexa son denotados como $g_1, g_2, ..., g_m$ asumiendo que $m \geq 1$ y el conjunto de todos esos puntos es llamado G. Los puntos que se encuentran en la parte de arriba o incluso a la altura de la línea en la clausura convexa se denotan como $u_1, u_2, ..., u_p$. Por el contrario, los puntos que se encuentran en la parte de abajo son $l_1, l_2, ..., l_q$. El conjunto de todos los puntos en dicha clausura es denotado como B.

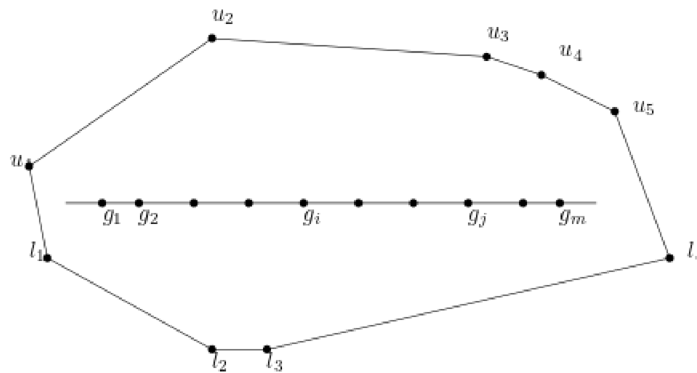


Figura 5.1: entrada del TSP para una clausura convexa y una línea interior.

Como ya se ha dicho las ciudades de B deberán ser recorridas en su orden cíclico, de lo contrario habrá una intersección. Por lo tanto, como se quiere encontrar un camino que una todos los puntos, para cada ciudad $g_i \in G$, hay que determinar entre qué dos ciudades adyacentes en B se visita.

El siguiente lema da una condición necesaria para un recorrido óptimo para este tipo de TSP.

***Lema* B.1.** Sea $g_i, g_j \in G$ y sean v y w dos ciudades adyacentes en B. Si en un recorrido óptimo τ ambas g_i y g_j han sido visitadas entre v y w, todas las ciudades que estén entre g_i y g_j en G habrán sido visitadas también entre v y w.

Como consecuencia de este lema y de que la ciudades de B son visitadas en su orden cíclico se obtiene:

***Lema* B.2.** Un recorrido óptimo puede ser obtenido dividiendo el conjunto de puntos G en $k+1$ segmentos

$$\{g_1, g_2, ..., g_{i_1}\}, \{g_{i_1+1}, ..., g_{i_2}\}, ..., \{g_{i_k+1}, ..., g_m\}$$

para $0 \leq k < m$, $0 = i_0 < i_1 < ... < i_k < m$, y metiendo estos segmentos entre dos puntos adyacentes de B.

El algoritmo primero determinará para cada segmento posible $\{g_i, g_{i+1}, ..., g_{j-1}, g_j\}$,$1 \geq i < j \geq m$, la manera más barata de insertarlo entre dos puntos adyacentes de B, y después determinará la mejor manera de dividir $\{g_1, g_2, ..., g_m\}$ en segmentos.

En principio, la inserción de un segmento $\{g_i, g_{i+1}, ..., g_{j-1}, g_j\}$ entre dos puntos adyacentes v y w se puede hacer de dos maneras. Sin embargo, en casi todos los casos, se debe descartar una forma porque produce una intersección. Además, el insertar un segmento entre u_1 y l_1 también puede dar lugar a una intersección en el recorrido, como se ve en el siguiente lema.

***Lema* B.3.** Para cualquier recorrido óptimo, el segmento $\{g_i, g_{i+1}, ..., g_{j-1}, g_j\}$ no se pueden insertar entre u_1 y l_1 a menos que $i = 1$. Del mismo modo, el segmento $\{g_i, g_{i+1}, ..., g_{j-1}, g_j\}$ no se puede insertar entre u_p y l_q a no ser que $j = m$.

Resumiendo:

Dos puntos adyacentes v y w en B se denominarán admisibles para un segmento $\{g_i, g_{i+1}, ..., g_{j-1}, g_j\}$, $1 \leq i < j \leq m$, si:

1. v y w se encuentran en el mismo lado de la línea G.
2. $\{v, w\} = \{u_p, l_q\}$ y $j = m$.

3. $\{v,w\} = \{u_1,l_1\}$ y $i = 1$.

Las posibles divisiones de $\{g_1,...,g_m\}$ pueden asociarse con caminos en un diagrama acíclico D con vértices $\{0,1,...m\}$ (donde el vértice adicional 0 actúa como fuente) y los arcos (i,j) con el coste d_{ij} para todo $0 \leq i < j \leq m$, donde d_{ij} es el mínimo coste de insertar el segmento $\{g_{i+1},g_{i+2},...,g_j\}$ entre dos puntos admisibles en B.

Si se asocia con el arco (i,j) el segmento $\{g_{i+1},g_{i+2},...,g_j\}$, entonces hay una correspondencia uno a uno entre las divisiones de $\{g_1,...,g_m\}$ en segmentos y caminos en D desde 0 hasta m. Por ejemplo, la división $\{g_1,g_2,g_3\},\{g_4,g_5\},\{g_6\},\{g_7,g_8,g_9,g_{10}\}$ se corresponde con el camino 0,3,5,6,10.
El camino más corto desde 0 hasta m en D determina un recorrido óptimo para el TSP de clausura convexa y línea interior. Esto se podría enunciar como un teorema principal para este caso especial de TSP euclídeo.

***Teorema* B.1.** Sea σ el subtour inicial para un TSP de clausura convexa y línea interior que visita sólo las ciudades en el límite de la clausura convexa de manera cíclica. Luego un recorrido τ es óptimo si y solo si se puede obtener insertando los puntos de B en σ de tal manera que la ruta correspondiente en el diagrama D tenga longitud mínima. Como consecuencia, la longitud de un recorrido óptimo es la longitud del subtour inicial σ más la longitud de la ruta más corta en D.

En la primera parte del algoritmo se calcula el coste de la menor ruta del diagrama acíclico mediante programación dinámica y en la segunda parte usamos este recorrido para construir el óptimo.
Ya se tiene todo lo que se necesita para construir el método. Se debe destacar que, para un valor fijo de j y para todo i, $0 \leq i < j$, el coste de insertar $\{g_{i+1},g_{i+2},...,g_j\}$ entre u_k y u_{k+1} puede conseguirse en un tiempo $O(mn)$. Con todo esto ya tenemos el teorema principal:

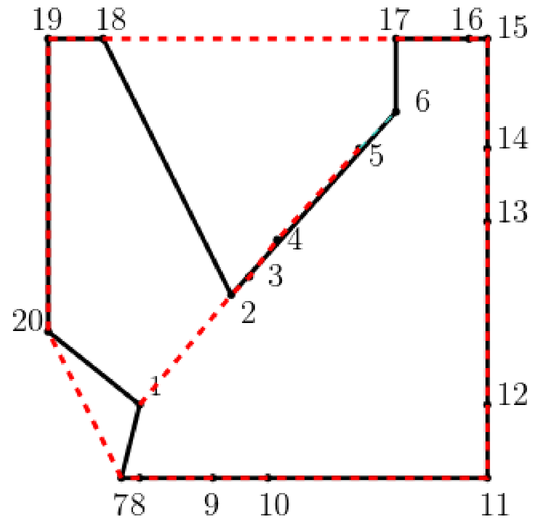
***Teorema* B.2.** El TSP para una clausura convexa y una línea interior, es decir, el TSP Euclídeo de n ciudades donde $n-m$ son las ciudades que se encuentran en el límite de la clausura convexa del conjunto de todas las ciudades y las otras m ciudades se encuentran en una misma línea dentro de esta clausura puede ser resuelto en un tiempo $O(mn)$.

B.3. EJEMPLO VISUAL DEL TSP PARA UNA CLAUSURA CONVEXA Y UNA LÍNEA INTERIOR

***Ejemplo* B.1.** Sea $n = 20$ y sean las coordenadas de cada punto las que vienen recogidas en la Tabla 1. Hay $m = 6$ puntos que se encuentran en linea dentro de la clausura convexa que son 1,2,3,4,5 y 6. El subtour inicial es (7,8,9,10,11,12,13,14,15,16,17,18,19,20). El camino más corto de 0 hasta 20 es 0,1,20 y la longitud del recorrido es 43 + 801=844. Esto quiere decir que para obtener un recorrido óptimo se tiene que insertar el punto 1 y el segmento {2,3,4,5,6} en el subtour inicial. El punto 1 se mete entre 7 y 20 y el segmento mencionado se mete entre 17 y 18. Quedando el tour óptimo:

$$\{1,7,8,9,10,11,12,13,14,15,16,17,6,5,4,3,2,18,19,10\}.$$

1	(177,177)	11	(1000,32)
2	(355,355)	12	(1000,268)
3	(381,381)	13	(1000,681)
4	(457,457)	14	(1000,822)
5	(632,632)	15	(1000,992)
6	(789,789)	16	(993,993)
7	(164,0)	17	(794,1000)
8	(171,0)	18	(57,1000)
9	(387,0)	19	(0,1000)
10	(409,0)	20	(0,329)



Subtour inicial en el conjunto convexo y línea inicial dentro de dicho conjunto.

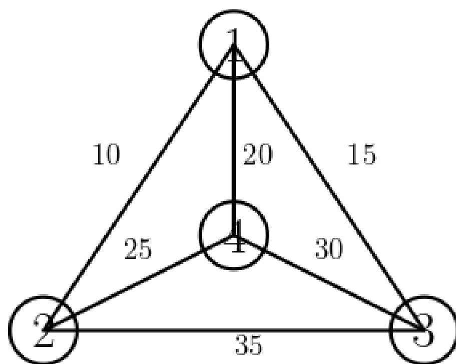
Recorrido óptimo.

ANEXO C

EJEMPLO PARA EL ALGORITMO DE HELD-KARP

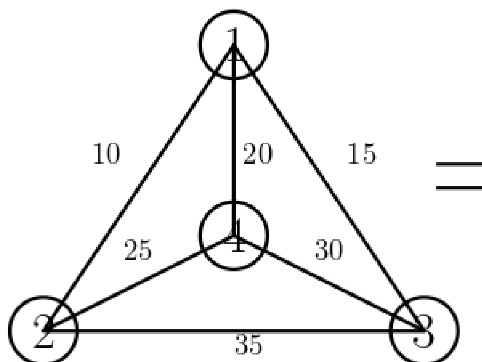
Dada el siguiente grafo para el problema del viajante, buscaremos el camino de mínimo coste.

Aplicando el algoritmo bruto:



PERMUTACIONES	COSTE
1-2-3-4-1	95
1-2-4-3-1	80
1-3-2-4-1	95
1-3-4-2-1	80
1-4-3-2-1	95
1-4-2-3-1	95

Aplicando el algoritmo de Held-Karp:



$$= \begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$$

S- Subconjunto del gráfico que aún no se ha recorrido.

$dist(i, 1)$: distancia de i a 1

Siendo el nodo 1 inicial y final

$C(4,\phi)=20$	$C(3,\phi)=15$	$C(2,\phi)=10$

$C(2,\{3\})=d(2,3) + C(3,\phi)=50$	$C(2,\{4\})=d(2,4) + C(4,\phi)=45$
$C(3,\{2\})=d(3,2) + C(2,\phi)=45$	$C(3,\{4\})=d(3,4) + C(4,\phi)=50$
$C(4,\{2\})=d(4,2) + C(2,\phi)=35$	$C(4,\{3\})=d(4,3) + C(3,\phi)=45$

$C(2,\{3,4\})=\min(d(2,3) + C(3,\{4\}),d(2,4) + C(4,\{3\}))=\min(85,80)=80$
$C(3,\{2,4\})=\min(d(3,2) + C(2,\{4\}),d(3,4) + C(4,\{2\}))=\min(80,65)=65$
$C(4,\{2,3\})=\min(d(4,2) + C(2,\{3\}),d(4,3) + C(3,\{2\}))=\min(75,75)=75$

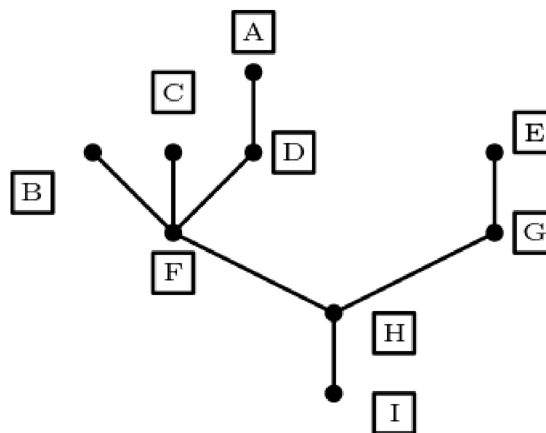
$C(1,\{2,3,4\})=\min(d(1,2)+C(2,\{3,4\}),\ d(1,3) + C(3,\{2,4\}),\ d(1,4) + C(4,\ \{2,3\})=\min(90,80,95)=80$

ANEXO D

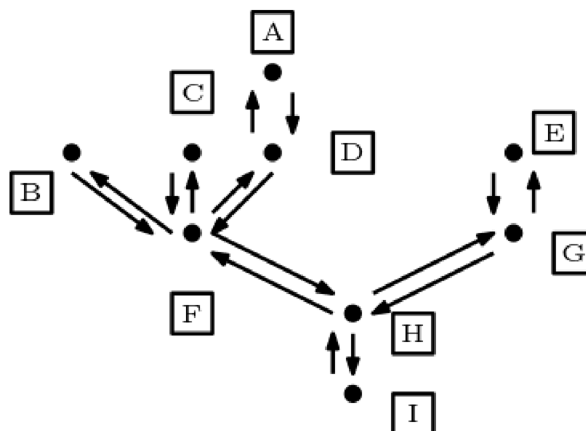
EJEMPLOS PARA LOS ALGORITMOS APROXIMADOS

D.1. ÁRBOL DE MÍNIMO PESO.

***Ejemplo* D.1.** 1.Sea (a) el árbol de extensión mínima asociado a un problema del TSP.

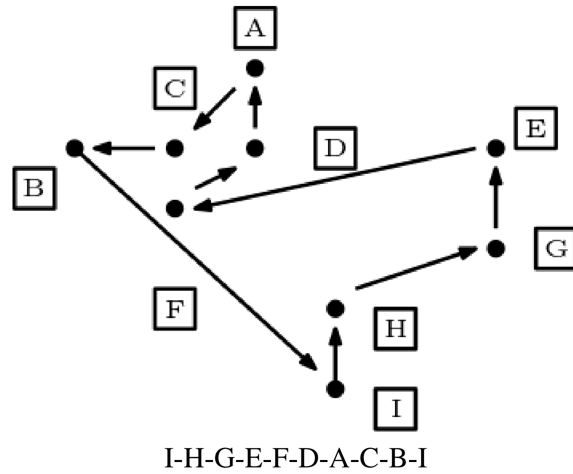


2. Recorrido conseguido al duplicar las aristas y buscar el ciclo Euleriano. (a)



I-H-G-E-G-H-F-D-A-D-F-C-F-B-F-H-I

3.Recorrido conseguido si el grafo cumple la propiedad triangular y se busca el camino con atajos.



I-H-G-E-F-D-A-C-B-I

D.2. MATCHING DE EXPANSIÓN MÍNIMA

***Ejemplo* D.2.** En estos 5 grafos se puede ver el proceso que lleva a cabo el algoritmo de Christofides.

(a)

Nodos con grado impar

(b)

Nodos con grado impar

Ejes nuevos para los nodos

(c)

(d)

(e)