VNIVERSIDAD NACIONAL
AVFNºMA DE
MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

FACULTAD DE INGENIERIA

TEORÍA DE REDES:
EL PROBLEMA DE LA RUTA MÁS CORTA

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRA EN INGENIERÍA

INGENIERÍA DE SISTEMAS – INVESTIGACIÓN DE OPERACIONES
P R E S E N T A :

BIBIANA OBREGÓN QUINTANA

TUTOR:
DRA. IDALIA FLORES DE LA MOTA

MÉXICO D.F., 2005

JURADO ASIGNADO:

Presidente: Dr. RICARDO ACEVES GARCÍA

Secretario: M.I. MAYRA ELIZONDO CORTÉS

Vocal: Dra. IDALIA FLORES DE LA MOTA

1er Suplente: Dr. JUAN MANUEL ESTRADA MEDINA

2do Suplente: M.I. RUBÉN TÉLLEZ SÁNCHEZ

Lugar donde se realizó la tesis:

DEPARTAMENTO DE SISTEMAS DE LA DIVISIÓN DE INGENIERÍA MECÁNICA E INDUSTRIAL DE LA FACULTAD DE INGENIERÍA DE LA UNAM.

TUTOR DE TESIS:

Dra. IDALIA FLORES DE LA MOTA

_____

FIRMA

AGRADECIMIENTOS Y DEDICATORIAS

A DIOS:
Por darme la fuerza para vivir

A MIS PADRES Y MI HERMANO:
Gracias una vez más por estar conmigo apoyándome en todo momento para que logre realizar mis sueños

A LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO Y A LA FACULTAD DE INGENIERÍA:
Por darme la oportunidad de pertenecer a ellas, orgullo que forma parte de mí

A MI ASESORA DE TESIS DRA. IDALIA FLORES DE LA MOTA:
Gracias por su apoyo, tiempo y dedicación en la realización de este trabajo

A MIS SINODALES:
DR. RICARDO ACEVES GARCÍA
M.I. MAYRA ELIZONDO CORTÉS
DR. JUAN MANUEL ESTRADA MEDINA
M.I. RUBÉN TÉLLEZ SÁNCHEZ
Gracias por su apoyo, tiempo y sus acertados comentarios en la revisión de este trabajo

A CONACYT:
Gracias por otorgarme la beca para la realización de mis estudios

A DR. SERVIO TULIO GUILLÉN BURGUETE:
Gracias por su cátedra que siempre recordaré

A ARMANDO SANDOVAL:
Gracias por ser, estar y compartir

A BRISA JIMÉNEZ:
Gracias por tu amistad y tu apoyo

A GABRIELA VÁZQUEZ, ARTURO LÓPEZ Y URIEL GARRIDO:
Gracias por su amistad y tantos buenos momentos

Y A TODOS AQUELLOS:
Que están involucrados directa o indirectamente, gracias

ÍNDICE

INTRODUCCIÓN

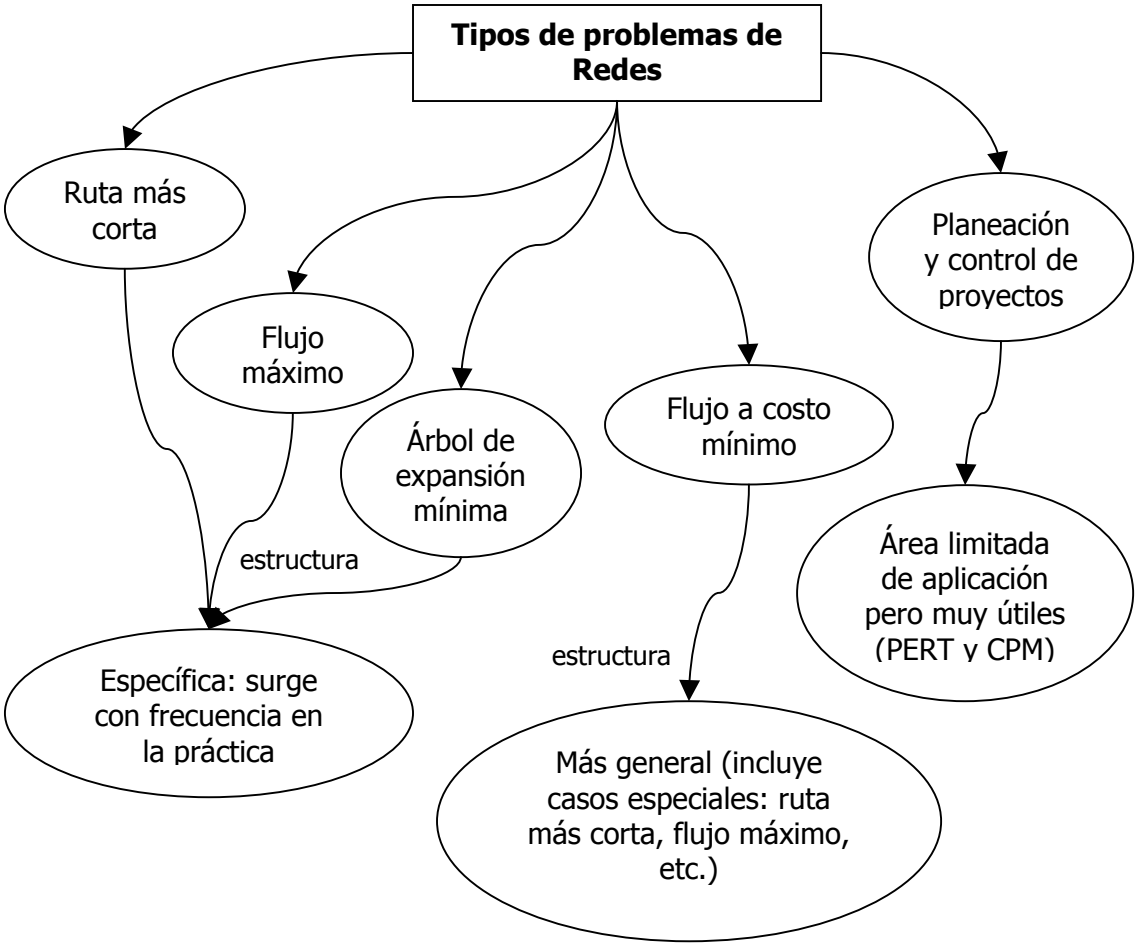
La Teoría de Redes es un área de conocimiento dentro del campo de la Investigación de Operaciones. Los problemas que estudia dicha teoría, son principalmente, de naturaleza combinatoria, es decir, relaciona rutas, cortes, árboles y otros objetos. Para obtener las soluciones de estos problemas, se requiere diseñar algoritmos. Algunos son más eficientes que otros, y su selección depende de las características del problema.

Los modelos de redes han ocupado un lugar muy importante en el progreso de la Investigación de Operaciones y de las Ciencias Administrativas. Estos modelos junto con la teoría de la Programación Lineal han mantenido una estrecha relación en su desarrollo. Lo que a su vez ha propiciado avances en el campo de la Programación Entera.

Otro aspecto es el adelanto de códigos más rápidos para los problemas de flujo en redes, lo cual favorece la relación entre la Investigación de Operaciones y las Ciencias de la Computación. Por otro lado, la investigación en modelos de redes a la par con la Ciencia de la Computación, ha propiciado la construcción de estructuras para el manejo de datos, haciendo más eficientes los algoritmos de redes.

La estructura de los problemas de redes se puede representar gráficamente. Esto ha permitido visualizar problemas en áreas como: telecomunicaciones, transporte, distribución, planeación de proyectos, localización de instalaciones, etc.

Los problemas de redes pueden clasificarse esencialmente en cinco áreas: Ruta más corta, Flujo máximo, Árbol de expansión mínima, Flujo a costo mínimo y, Planeación y control de proyectos [Hillier y Lieberman, 1991].



En esta agrupación, el Problema de la Ruta más Corta, es considerado por los investigadores como un problema central dentro del área de redes, debido a la variedad de aplicaciones prácticas, a la existencia de métodos de solución eficientes y a la aplicación de subrutinas en la búsqueda de una buena solución en problemas complejos. Sin embargo, en las asignaturas en las cuales se abordan esta clase de problemas, suele presentarse de manera simplificada y poco clara, es decir, a menudo no se reconoce o no se subraya la importancia que tiene el estudio de este tipo de problemas tanto su aspecto teórico y por la gran diversidad de aplicaciones.

Bajo estas consideraciones el presente trabajo tiene como objetivos principales, en primer lugar, integrar los aspectos fundamentales del Problema de la Ruta más Corta, como son: la teoría esencial, los algoritmos de solución y aplicaciones; y en segundo término, pueda servir como material de apoyo y referencia para personas interesadas en el estudio de este tema.

Es importante mencionar que existen softwares (LINDO, LINGO, etc.) para obtener la solución de los diversos problemas antes mencionados, pero para poder resolverlos se requiere llevarlos a una forma estándar de Programación Lineal.

Además de los paquetes anteriores, también existe paquetería especializada como TransCAD, utilizada para los estudios de transporte, la cual permite utilizar el análisis de redes, que incluye ruta más corta y genera el camino más corto, más rápido o con el menor costo entre cualquier número de pares de orígenes y destinos con cualquier número de puntos intermedios.

Para llevar a cabo esta propuesta, se desarrollaron los siguientes apartados y se proporciona una síntesis de su contenido:

Capítulo 1. Se explican las razones que hacen tan importante al problema de la ruta más corta, además se mencionan algunas de las muchas aplicaciones. La siguiente parte introduce las definiciones o conceptos del área de redes en las que se apoya el desarrollo de la propuesta. Del mismo modo, se da el planteamiento del problema y finalmente, las formas en que puede presentarse este problema, y cuando existe o no solución.

Capítulo 2. Un criterio básico para conocer la eficiencia de un algoritmo de solución es conocer su tiempo de corrida, de igual forma las mejoras en el almacenamiento de datos pueden hacer más eficiente el mismo algoritmo, por esta razón, en este capítulo se exponen los conceptos de la Teoría de la Complejidad Computacional.

Capítulo 3. Existen varios métodos de solución para este problema y dependiendo de las características de la red (p.e. costos no negativos, densidad, etc.) se puede elegir el más eficiente. Además se explican los algoritmos básicos de solución y sus modificaciones para hacerlos más eficientes. En el anexo 1, se muestra el ejemplo del algoritmo de Dijkstra utilizando el software LINGO.

Capítulo 4. Finalmente, en este capítulo se dan a conocer dos tipos de aplicaciones, la primera utiliza el Método Símplex y la segunda ilustra cómo utilizar el problema de la ruta más corta, en una subrutina en la búsqueda de una buena solución en un problema complejo, para el cual simplemente no existe un algoritmo de solución.

CAPÍTULO 1.
CONCEPTOS BÁSICOS

El problema de la ruta más corta es uno de los problemas más importantes de optimización combinatoria con muchas aplicaciones, tanto directas como subrutinas en otros algoritmos de optimización combinatoria. Los algoritmos para este tipo de problemas han sido estudiados desde la década de los 50's y continúan siendo un área activa de investigación. De hecho, ha sido el objetivo de una investigación extensiva durante muchos años y ha dado como resultado la publicación de un gran número de documentos científicos.

Encontrar la ruta más corta entre dos nodos de una red, en la cual cada arco tiene un costo (o longitud) no negativo es un problema que a menudo se presenta en cierto tipo de actividades. El objetivo es minimizar el costo (tiempo o longitud) total.

El ejemplo más sencillo para explicar el problema de la ruta más corta es tomar el viaje de una persona que quisiera ir de la Ciudad de México a la ciudad de Monterrey, Nuevo León, podría tener varias alternativas dependiendo de sus intereses, es decir, si deseara llegar más rápido (minimizando el tiempo o la distancia) o de una forma más económica (minimizando el costo), toda vez que cada carretera tiene una longitud específica (kms.) y un precio por el derecho de transitar en ella (costo). Entonces, el problema consiste en encontrar la ruta más eficiente (la ruta mínima) con base en la longitud o el costo. Este problema se representa por una red, donde las ciudades son identificadas por nodos y las carreteras por arcos.

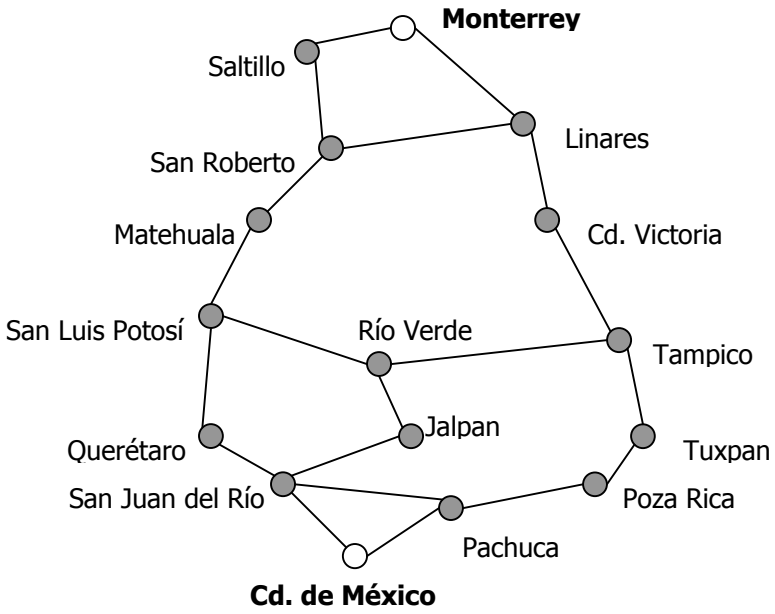


Figura 1.1. Ejemplo

1.1. IMPORTANCIA DEL PROBLEMA

El problema de la Ruta más Corta es fundamental en muchas áreas, como son: investigación de operaciones, ciencia de la computación e ingeniería. Algunas de las razones son:

i. La amplia variedad de aplicaciones prácticas como es el envío de algún material entre dos puntos específicos de la forma más eficiente, económica o rápida.

ii. Existen métodos de solución eficientes, los cuales al ser aplicados a una red con características específicas (acíclica y con costos no negativos), proveen una solución exacta a un tiempo y costo razonables.

iii. Se puede utilizar como inicio en el estudio de modelos complejos de redes, esto es, cuando no se conoce la estructura de la red se pueden aplicar algoritmos para conocer algunas características de la red (presencia de ciclos negativos).

iv. Se utiliza frecuentemente como subproblemas (subrutinas) en la solución de problemas combinatorios y redes, así en el caso de problemas para los cuales no existe un algoritmo de solución exacto (p. e. problemas NP-completos), la aplicación de algoritmos de ruta más corta, resultan auxiliares para encontrar una buena solución.

1.2. APLICACIONES

El problema de ruta más corta tiene muchas aplicaciones prácticas, algunas son: encontrar la ruta más corta o más rápida entre dos puntos en un mapa, redes eléctricas, telecomunicaciones, transporte, planeación de tráfico urbano, trasbordo, diseño de rutas de vehículos, planeación de inventarios, administración de proyectos, planeación de producción, horarios de operadores telefónicos, diseño de movimiento en robótica, redes de colaboración entre científicos, reemplazo de equipo, etc.

Además, como se mencionó anteriormente los algoritmos de solución pueden adaptarse en la búsqueda inicial de una solución aproximada de problemas complejos, esto significa que la aplicación consiste precisamente en proveer estructura para varios problemas de optimización combinatoria como: el problema de la mochila, secuencia de alineación en biología molecular (secuenciación del ADN), el problema del agente viajero, etc.

1.3. CONCEPTOS BÁSICOS

En esta sección se introducen conceptos básicos de redes que se utilizarán en el desarrollo del presente trabajo. En la literatura se observa frecuentemente el uso indistinto de *Red* o *Gráfica*.

Gráfica.

G = (V, E), es una gráfica formada por el conjunto de nodos (*vértices*) *V* y el conjunto de arcos (*aristas*) *E*.

Gráfica No Dirigida.

En una gráfica no dirigida un arco es un par no ordenado de nodos, las conexiones son bidireccionales, es decir, el orden no importa.

G = (V, E), donde:

V = {a, b, c, d, e, f}

E = {(a, b), (a, d), (b, c), (b, d), (b, e), (c, e), (c, f), (d, e), (e, f)}

Sin embargo, como el orden de los arcos no importa el arco *(a, b)* también puede considerarse como *(b, a)*, siendo lo mismo para todos los demás arcos.

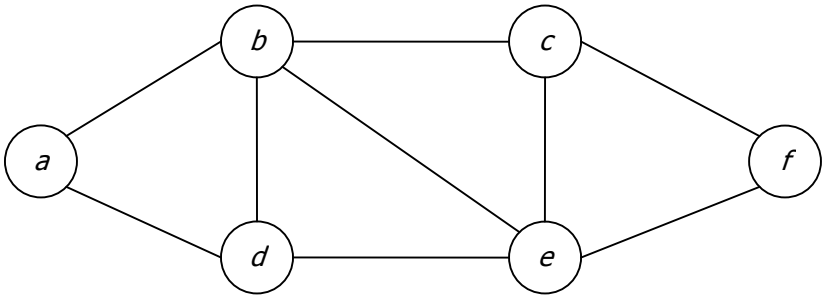


Figura 1.2. Gráfica no dirigida

Gráfica Dirigida o Digráfica.

En una gráfica dirigida un arco es un par ordenado. Esto es, si *(a, b)* es un arco dirigido, entonces *a* es el nodo inicial (el arco sale del nodo) y *b* es el nodo final (el arco entra al nodo). La conexión es únicamente del nodo inicial al nodo final.

V = {a, b, c, d, e, f}

E = {(a, b), (a, d), (b, c), (b, d), (b, e), (c, e), (c, f), (d, e), (e, f)}

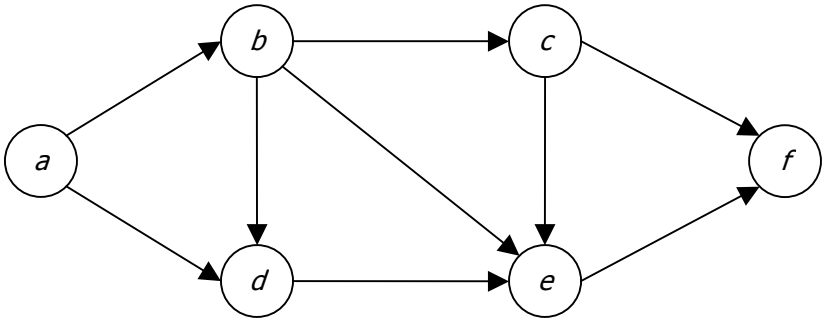


Figura 1.3. Gráfica dirigida o digráfica

Gráfica Simple.

Esta gráfica considera un nodo fuente (únicamente salen arcos) y un nodo sumidero (únicamente entran arcos). No existen arcos múltiples, es decir, dos nodos están conectados por un arco o por ninguno, tampoco existen rizos, esto es, ningún nodo está conectado a sí mismo por un arco.

Por lo general, cuando no se hace especificación se consideran gráficas simples.

Gráfica Múltiple.

Existe la posibilidad de varios arcos entre el mismo par de nodos.

Gráfica Conectada.

Todos los nodos están conectados directa o indirectamente a todos los demás nodos, esto es, existe una ruta desde cualquier nodo a cualquier otro nodo de la red.

Gráfica Bipartita.

Los nodos de la gráfica se dividen en dos conjuntos, con la característica de que todos los arcos conectan a los nodos desde un conjunto al otro.

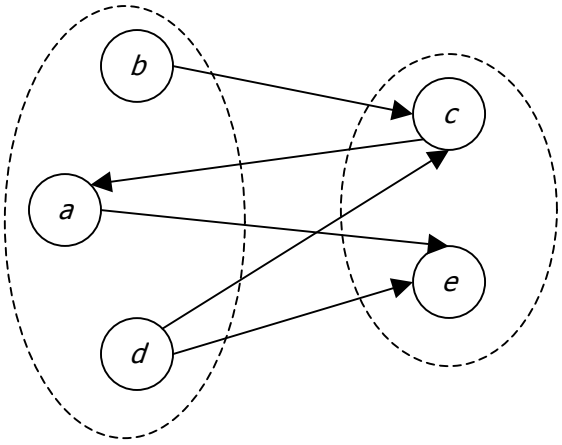


Figura 1.4. Gráfica bipartita

Grado.

Es el número de arcos incidentes en un nodo. En una digráfica existen:

- el grado interior, es el número de arcos que entran en un nodo. En la figura 1.3, el nodo *d* tiene grado interior de 2.
- y el grado exterior, es el número de arcos que salen de un nodo. En la figura 1.3, el nodo *d* tiene grado exterior de 1.

Nodos Adyacentes.

Son los nodos conectados por un arco.

Arcos Incidentes.

Un arco *e* es incidente en un nodo *v* si *v* es el extremo de *e.*

Ruta.

Una ruta en una gráfica dirigida es una secuencia de nodos y arcos, además se requiere que todos los nodos sean diferentes. En el caso de que algunos nodos o arcos se repitan en la secuencia, se conoce como camino.

En la figura 1.5. (a), se muestra la ruta del nodo *a* al nodo *f*, la cual considera los nodos *a*, *b, e* y *f*, y los arcos *(a, b)*, *(b, e)* y *(e, f)*.

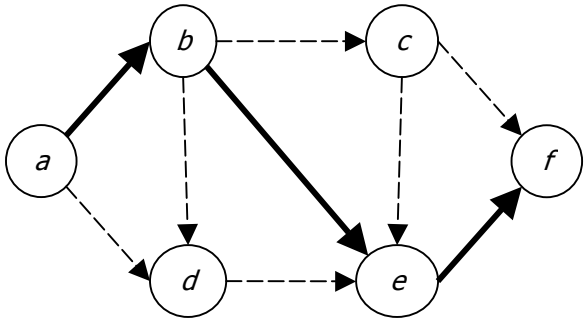


Figura 1.5. Ruta del nodo *a* al nodo *f*

Longitud de una Ruta.

La longitud de una ruta con la secuencia de arcos $a_1, a_2,..., a_n$ es la suma de las longitudes de todos los arcos de la ruta, $l(a_1) + l(a_2) +...+ l(a_n)$.

En general, la ruta más corta del nodo a_i al nodo a_j existe si y sólo si existe al menos una ruta entre ambos nodos. La distancia entre los nodos a_i y a_j será la longitud de la ruta más corta a_ia_j, y se denotará como $d(a_i, a_j)$. Si la ruta a_ia_j no existe, entonces $d(a_i, a_j)$ es infinito (∞).

Ciclo.

Es un camino donde el nodo inicial y el nodo final coinciden. Si los arcos tienen la misma dirección se conoce como circuito.

Figura 1.6. (a) Camino del nodo *c* al nodo *b*, arcos *(c, a), (a, b)*
(b) Circuito del nodo *a* al nodo *a*

Gráfica Acíclica.

Una gráfica que no contiene ciclos.

Árbol.

Un árbol es una gráfica conectada que no contiene ciclos.

Bosque.

Es una gráfica sin ciclos, se considera también como un conjunto de árboles.

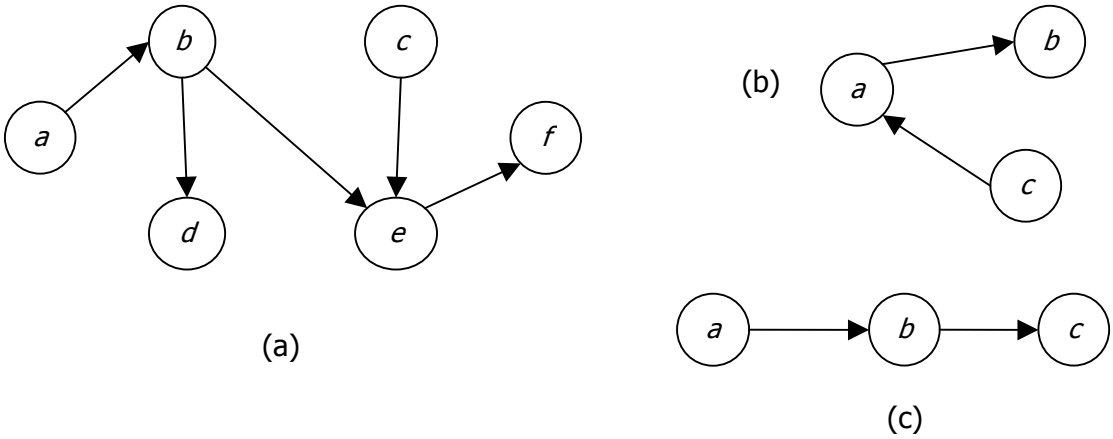


Figura 1.7. Ejemplos de árboles.
(a), (b) y (c) El conjunto de los tres forma un bosque

Arborescencia.

Es un árbol dirigido con un nodo llamado raíz.

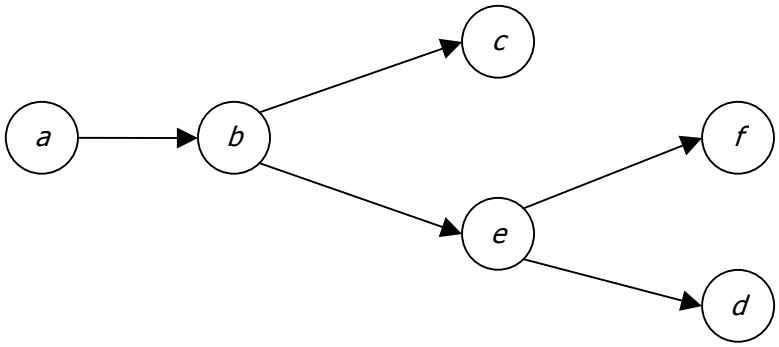


Figura 1.8. Arborescencia de raíz *a*

Subgráfica.

Es un subconjunto de nodos y arcos de una gráfica. Si un arco se incluye, los dos nodos incidentes también se incluyen.

Subgráfica Expandida.

Una subgráfica que contiene todos los nodos de la gráfica original.

Árbol de Expansión.

Una subgráfica expandida que también es un árbol, es decir, conectada y sin ciclos.

Función de Costo.

Sea *c* una función que asocia a cada elemento de *E* un costo respectivo. La función puede representar: costo, distancia, tiempo, etc.

Propiedades de una Gráfica.

⇨ Una gráfica con *n* nodos y *m* arcos:

✦ Σ grados de todos los nodos = 2m

✦ $m \leq n(n-1)/2$

⇨ Para una digráfica:

✦ Σ grados interiores = Σ grados exteriores = m

✦ $m \leq n(n-1)$

⇨ Si G es conectada, $m \geq n-1$

⇨ Si G es un árbol, $m = n-1$

⇨ Si G es un bosque, $m \leq n-1$

Representación Matricial de una Gráfica.

Una gráfica puede representarse matricialmente de las siguientes formas:

Matriz de Incidencia (nodos-arcos).

Es una matriz *nxm*, (*n*-nodos y *m*-arcos). El arco *(i, j)* tiene los elementos: -1 en el renglón correspondiente al nodo *i* ; +1 en el renglón correspondiente al nodo *j* ; 0 en otro caso.

$$e(i,j)=\begin{cases}+1 & \text{si el arco } j \text{ sale del nodo } i\\ -1 & \text{si el arco } j \text{ entra al nodo } i\\ 0 & \text{en otro caso}\end{cases}$$

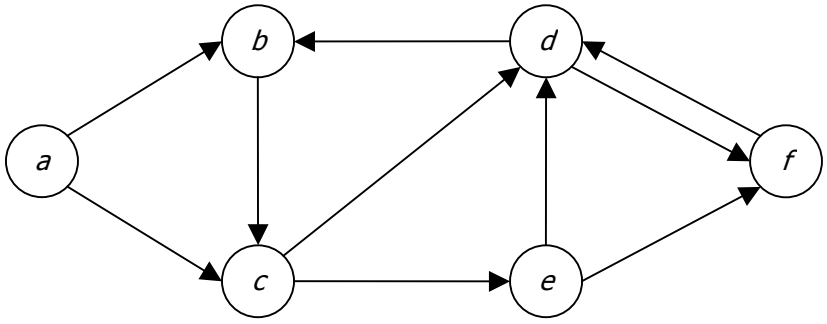


Figura 1.9. Ejemplo

Matriz de incidencia de la red representada en la figura 1.9.

$$\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\\ f\end{array}\begin{array}{c}\begin{array}{cccccccccc}(a,b) & (a,c) & (b,c) & (c,d) & (c,e) & (d,b) & (d,f) & (e,d) & (e,f) & (f,d)\end{array}\\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ -1 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0\\ 0 & -1 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & -1 & 0 & -1\\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1\end{pmatrix}\end{array}$$

Matriz de Adyacencia (nodos-nodos).

Es una matriz *nxn*. En la entrada *ij* toma el valor de 1 si existe conexión del nodo *i* al nodo *j*.

$$q(i,j)=\begin{cases}1 & \text{si existe un arco que sale de } i \text{ y llegue a } j\\ 0 & \text{en otro caso}\end{cases}$$

Matriz de adyacencia de la red representada en la figura 1.9.

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

1.4. PLANTEAMIENTO

Dada una red dirigida $G = (V, E, c)$, se denota por $(i, j) \in E$, el arco que conecta al nodo i con el nodo j, y el costo positivo asociado es c_{ij}. La red tiene dos nodos específicos: el nodo fuente s y el nodo sumidero t.

El problema consiste en encontrar la ruta (p) más corta (o de costo mínimo) iniciando en el nodo fuente y terminando en el nodo sumidero, considerando que cada arco (i, j) tiene un costo asociado c_{ij}, es decir, se busca minimizar la función:

$$c(p) = \sum_{(i,j)\in p} c(i,j)$$

1.4.1. PLANTEAMIENTO EN PROGRAMACIÓN LINEAL

El problema de la ruta más corta se puede plantear como el envío de una unidad de flujo del nodo origen 1 al nodo destino t, al mínimo costo. Esto es, $b_1 = 1$, $b_t = -1$, y $b_i = 0$, para $i \neq 1$ ó t. Entonces, el planteamiento es como sigue:

$$Minimizar \sum_{i=1}^{t}\sum_{j=1}^{t} c_{ij}\, x_{ij}$$

sujeto a:

$$\sum_{j=1}^{t} x_{ij} - \sum_{k=1}^{t} x_{ki} = \begin{cases} 1 & \text{si} \quad i = 1 \\ 0 & \text{si} \quad i \neq 1 \text{ ó } t \\ -1 & \text{si} \quad i = t \end{cases}$$

$x_{ij} = 0$ ó 1 $\quad i, j = 1, 2, \ldots, t$

Sin embargo, como las ecuaciones de conservación de flujo son unimodulares, es decir, si existe una solución óptima el método simplex obtendrá valores 1, 0. Por esta razón la última restricción puede plantearse como:

$x_{ij} \geq 0 \qquad i, j = 1, 2, \ldots, t$

1.5 VARIACIONES DEL PROBLEMA

Las diferentes formas que puede presentar el problema de la ruta más corta son:

- Del nodo fuente *s* al nodo sumidero *t*. Para que exista solución se debe cumplir:
  i. Existe al menos una trayectoria entre *s* y *t.*
  ii. No existen circuitos negativos tales que haya una ruta de *s* a algún nodo del circuito y otra de algún nodo del circuito a *t*.

- Del nodo fuente *s* a todo nodo de la red *i*. Para que exista solución se debe cumplir:
  i. Existen rutas de *s* a *i.*
  ii. No existen circuitos negativos en la red.

- Entre todo par de nodos. Para que exista solución se debe cumplir:
  i. Existe, al menos, una trayectoria entre todo par de nodos.
  ii. No existen circuitos negativos en la red.

1.6 TIPOS DE PROBLEMAS

Las rutas a encontrar pueden ser:

a. formadas por un simple arco, *(s, t)*, o
b. la ruta del nodo *s* al nodo *t* que atraviesa por otros nodos.

Además de forma independiente de la variación del problema, se pueden tener los siguientes casos:

- ⇨ El tipo más sencillo del problema de la ruta más corta es cuando la longitud de cada arco es 1. Esto significa que la longitud de la ruta es exactamente el número de arcos que contiene.

- ⇨ Cuando todos los arcos tienen distancias (costos) no negativas y no existen circuitos en la red.

- ⇨ Cuando no existen ciclos dirigidos.

- ⇨ Cuando no existen ciclos dirigidos con longitudes negativas.

- ⇨ Ruta más corta en gráficas no dirigidas con longitudes de los arcos no negativas. En este caso, se reemplaza cada arco no dirigido *uv* por dos arcos dirigidos *uv* y *vu*, con la misma longitud que *uv*.

- ⇨ Cuando existen arcos con costos negativos. Actualmente no se conoce un algoritmo que resuelva este tipo de problema polinomialmente, y la teoría de la complejidad computacional parece indicar que no existe un algoritmo.

- ⇨ En gráficas no dirigidas con al menos un arco con longitud negativa. En este caso, al reemplazar el arco *uv* por *uv* y *vu*, juntos forman un ciclo

con longitud negativa. Este tipo de problema es igual que cuando existen arcos con longitudes negativas, es decir, no existe un algoritmo de tiempo polinomial para resolverlo.

La teoría de la Complejidad Computacional sirve para evaluar la eficiencia de los algoritmos, pero también ayuda a clasificar los problemas de tal manera que puede saberse previamente si la búsqueda de un algoritmo es posible o no, evitando así trabajo innecesario.

En el siguiente capítulo se explica esta teoría y los conceptos en que se basa para hacer las clasificaciones.

CAPÍTULO 2.
COMPLEJIDAD COMPUTACIONAL

La complejidad computacional estudia la eficiencia de los algoritmos estableciendo su efectividad de acuerdo al tiempo de corrida y al espacio requerido en la computadora o almacenamiento de datos, ayudando a evaluar la viabilidad de la implementación práctica en tiempo y costo.

Por otra parte, provee herramientas para clasificar la dificultad inherente de un problema, de esta manera se puede conocer previamente si la búsqueda de un algoritmo eficiente para la solución de dicho problema es posible o no.

Existen problemas que únicamente pueden resolverse utilizando un algoritmo de tiempo exponencial o incluso, puede ser que no exista algoritmo alguno, por lo cual, al tener este tipo de información puede optarse por la búsqueda o aplicación de técnicas heurísticas existentes, que si bien no garantizan una solución óptima, si pueden proporcionar una buena solución o una aproximada.

2.1. DEFINICIÓN DE ALGORITMO

Un algoritmo es un procedimiento computacional bien definido, el cual considera un valor o conjunto de valores como valores de entrada y a través de una secuencia de instrucciones organizadas produce un valor o conjunto de valores de salida, en un tiempo determinado con la finalidad de dar solución a un problema específico.

Las características principales de un algoritmo son:

- ⇨ Preciso y Definido. Cada paso debe ser definido en forma precisa e indicar el orden de realización.

- ⇨ Datos de entrada (input). El algoritmo recibe datos iniciales antes de su ejecución.

- ⇨ Datos de salida (output). El algoritmo tiene una o más salidas, es decir, datos que tienen una relación específica respecto a los datos de entrada.

- ⇨ Generalidad. Independientemente de las veces que se siga un algoritmo, se debe obtener el mismo resultado.

- ⇨ Finito. Un algoritmo debe terminar siempre después de un número finito de pasos.

Al analizar un algoritmo se considera el tiempo de corrida y consiste en el número de operaciones elementales que ejecuta en cada paso. Dicho análisis se concentra generalmente en encontrar el peor caso de tiempo de corrida, es decir, el mayor tiempo que tardaría el algoritmo en obtener los valores de salida.

2.2. ANÁLISIS ASINTÓTICO DE LOS ALGORITMOS

El análisis asintótico permite conocer la eficiencia de un algoritmo con base en el tiempo de corrida cuando el tamaño de los datos de entrada es suficientemente grande de tal forma que las constantes y los términos de menor orden no afectan.

Para expresar la tasa de crecimiento de una función se toma en cuenta:

- El término dominante con respecto a n ; y
- Se ignoran las constantes.

Ejemplo:

1) $k_1n + k_2 \sim n$
2) $k_1nlogn \sim nlogn$
3) $k_1n^2 + k_2n + k_3 \sim n^2$

Este tipo de análisis permite facilitar la elección del mejor algoritmo entre varios y, por supuesto, es más conveniente aplicar medidas para la eficiencia que implementarlo y medir la eficacia después de cada corrida.

2.2.1. NOTACIÓN ASINTÓTICA

Las notaciones para el tiempo de corrida asintótico de un algoritmo se definen en términos de funciones cuyo dominio es el conjunto de números naturales. Las notaciones se utilizan para describir la función del peor caso del tiempo de corrida, $T(n)$, normalmente definidas en tamaños de entrada enteros.

i. **Notación Θ**. Dada una función $g(n)$, se denota por $\Theta(g(n))$ al conjunto de funciones:
$\Theta(g(n)) = \{f(n)$: existen las constantes positivas c_1, c_2, y n_0 tales que $0 \leq c_1g(n) \leq f(n) \leq c_2g(n)$ para toda $n \geq n_0\}$

La definición de $\Theta(g(n))$ implica que cada miembro de $\Theta(g(n))$ sea asintóticamente no negativo, es decir, que $f(n)$ es no negativa cuando n sea suficientemente grande.

Entonces si:
$f(n) = \Theta(g(n))$ sí y solo sí $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$

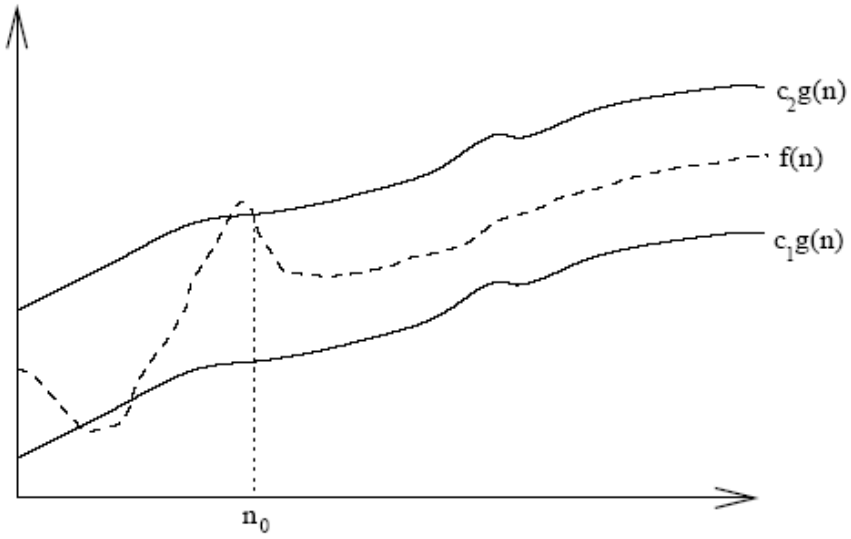


Figura 2.1. *f(n) = Θ(g(n))*
Fuente: [Misra, 2005]

Ejemplo:

1) $k_1n^2 + k_2n + k_3 \in \Theta(n^2)$

2) $n^2 / 3 - 3n \in \Theta(n^2)$

ii. **Notación *O***. Representa una cota asintótica superior. Sea una función *g(n)*, se denota por *O(g(n))* el conjunto de funciones:
O(g(n)) = {f(n) : existen constantes positivas *c* y n_0 tales que $0 \leq f(n) \leq cg(n)$ para toda $n \geq n_0$*}*

La notación *O* se utiliza para acotar el peor caso del tiempo de corrida de un algoritmo.

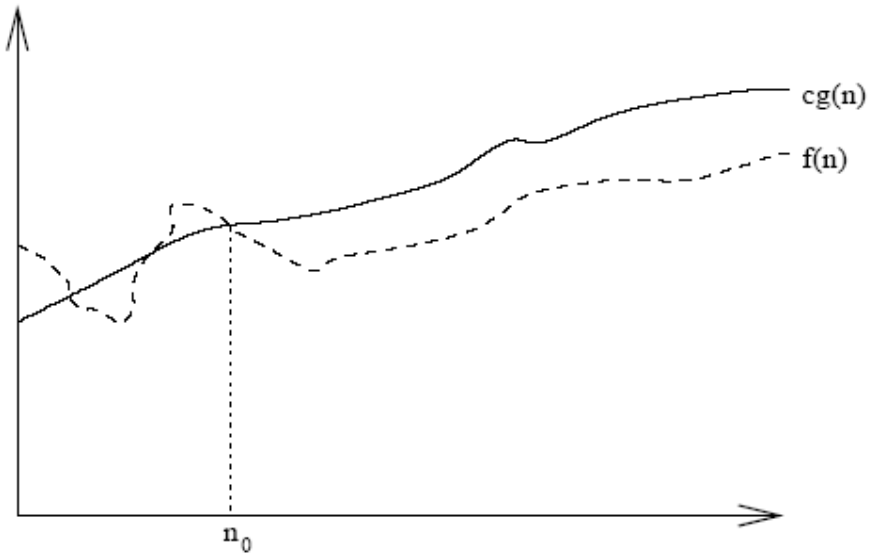


Figura 2.2. *f(n) = O(g(n))*
Fuente: [Misra, 2005]

Ejemplo:

1) $f(n) = an^2 + bn + c$, $g(n) = n^2$

$f(n) \in O(g(n)) = O(n^2)$ $\Rightarrow$ $f(n)$ es de orden $O(n^2)$

$g(n) \in O(f(n)) = O(n^2)$ $\Rightarrow$ $g(n)$ es de orden $O(n^2)$

2) $k_1n^3 + k_2n^2 + k_3 \in O(n^3)$

iii. **Notación Ω**. Representa una cota asintótica inferior. Sea una función $g(n)$, se denota por $\Omega(g(n))$ el conjunto de funciones:
$\Omega(g(n)) = \{f(n) :$ existen constantes positivas c y n_0 tales que $0 \leq cg(n) \leq f(n)$ para toda $n \geq n_0\}$

La notación Ω se utiliza para acotar el mejor caso del tiempo de corrida de un algoritmo.

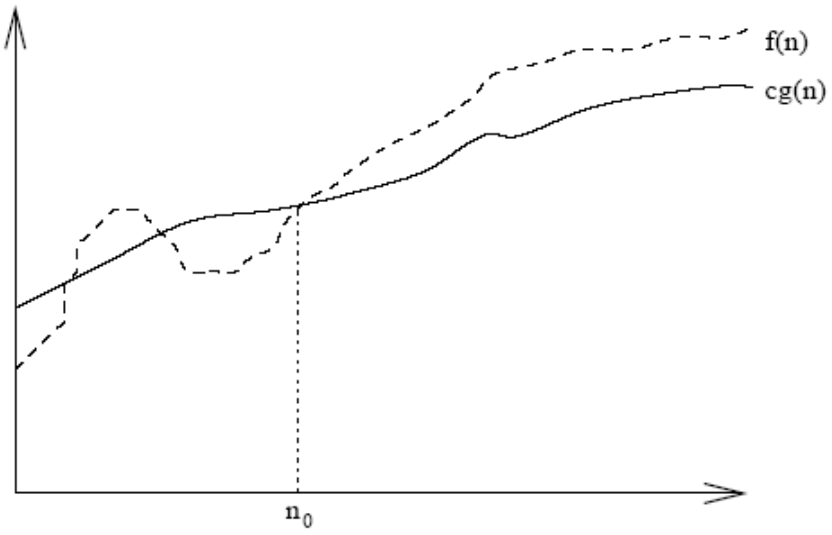


Figura 2.3. *f(n) = Ω(g(n))*
Fuente: [Misra, 2005]

Ejemplo:

1) $k_1n^2 + k_2n + k_3 \in \Omega(n)$

2) $f(n) = n$, $g(n) = n^2 \Rightarrow g(n) \in \Omega(f(n))$

Regla de suma y producto del análisis asintótico.

Sean $T_1(n)$ y $T_2(n)$ los tiempos de corrida de dos algoritmos, entonces con base en la notación asintótica:

Si $T_1(n) = O(f(n))$ y $T_2(n) = O(g(n))$

Suma: $T_1(n) + T_2(n) = max \{ O(f(n)), O(g(n)) \}$

Producto: $T_1(n) * T_2(n) = O(f(n) * g(n))$

2.3. ANÁLISIS DEL PEOR CASO DEL TIEMPO DE CORRIDA

Los tiempos de corrida de un algoritmo pueden dividirse en: el mejor, el probabilístico y, el peor. Sin embargo, para analizar un algoritmo normalmente se considera el peor caso, es decir, el tiempo más largo para cualquier entrada de tamaño *n*. Las razones son las siguientes:

- El mejor caso no es representativo porque se analiza una entrada para la cual el problema es trivial.

- El caso probabilístico sería una muy buena opción, no obstante, el problema principal es que matemáticamente puede resultar muy difícil de medir.

- El peor caso es muy práctico, debido a que, en algunos casos es muy cercano al probabilístico e incluso a observaciones experimentales. De hecho, es muy frecuente que el caso promedio sea tan malo como el peor caso.

- El peor tiempo de corrida de un algoritmo es una cota superior del tiempo de corrida para cualquier entrada, garantizando que el algoritmo no tardará más.

- Para algunos algoritmos, el peor caso ocurre muy rara vez.

2.4. PROBLEMAS DE DECISIÓN

Los problemas de decisión son utilizados para la clasificación de la complejidad de los problemas, y se refieren a una subclase de problemas donde la solución es un conjunto $S = \{ sí, no \}$ únicamente.

Cada problema de optimización tiene un problema de decisión correspondiente, por ejemplo para el problema de la ruta más corta, se tiene:

Problema de Optimización. Dada una gráfica *G = (V, E)* y dos vértices *v* y *w*, encontrar la ruta más corta desde *v* a *w*.

Problema de Decisión. Dada una gráfica *G = (V, E)* y dos vértices *v* y *w*, y un entero *k*; decidir si existe una ruta desde *v* a *w* de longitud máxima *k*.

La importancia de este tipo de problemas radica en que si es posible proveer evidencia de que un cierto problema de decisión es improbable que tenga un algoritmo de tiempo polinomial, entonces el problema de optimización correspondiente, es igualmente improbable que tenga un algoritmo de tiempo polinomial.

2.5. COMPLEJIDAD DE LOS PROBLEMAS

La clasificación de la complejidad de los problemas es en cuatro clases [Flores de la Mota, 2002]:

1. Problemas Indecidibles. Son problemas para los cuales no se puede escribir un algoritmo para su solución, por lo tanto son los problemas de complejidad más alta.

2. Problemas Intratables. Son los problemas para los cuales no se puede desarrollar un algoritmo de tiempo polinomial, únicamente algoritmos exponenciales.

3. Problemas NP (Polinomial no determinístico). Son los problemas para los cuales la factibilidad del problema utilizando el correspondiente problema de decisión, puede ser verificada en tiempo polinomial, sin embargo, el problema solo puede resolverse con algoritmos no determinísticos.

Un algoritmo no determinístico es un modelo teórico de computación donde la computadora debe *adivinar* que paso seguir, y en caso de existir un conjunto de *adivinanzas* que la computadora debiera resolver, entonces se supone que *adivina* correctamente. Obviamente este tipo de modelo es imposible de implementar.

4. Problemas P. Si un problema está en la clase P, se dice que es polinomial y significa que existe un algoritmo de tiempo polinomial para su solución.

2.6. ESPACIO COMPUTACIONAL

La complejidad de espacio de un algoritmo indica la cantidad de almacenamiento temporal requerido para correr un algoritmo, esto es, el acceso directo a la memoria de la computadora requerido para representar los datos necesarios para llevar a cabo el procedimiento.

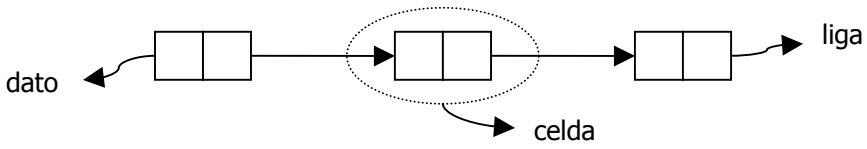
Los esquemas de almacenamiento y manipulación de datos dentro de la memoria son indispensables para el desempeño de los algoritmos, haciendo en ocasiones la diferencia en la elección entre varios algoritmos. Incluso, al mejorar estos aspectos se puede mejorar la complejidad del peor caso, obteniendo así un algoritmo más eficiente.

La forma del almacenamiento de datos se relaciona con el tipo de operaciones que se pueden realizar, porque dependiendo de la manera en que se almacenen los datos se podrán efectuar cierto tipo de operaciones y también se relaciona con rapidez para efectuarlas.

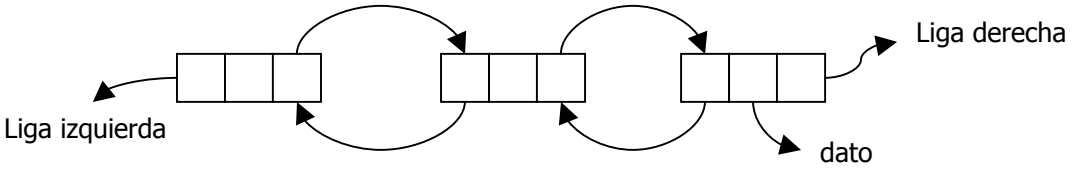
Formas de almacenamiento [Ahuja, Magnanti y Orlin, 1993]

⇨ Arrays. Es la estructura más simple, con un número fijo de datos almacenados en forma ordenada y se accesan por un índice.

⇨ Listas ligadas sencillas. El conjunto de datos está organizado secuencialmente y para el acceso se requiere información adicional. Los datos se encuentran en celdas con dos campos: el dato o elemento del conjunto y la liga.



⇨ Listas ligadas dobles. La diferencia con las listas sencillas es que cada celda tiene dos ligas, una a la celda predecesora y la otra a la sucesora. Debido a su estructura permite realizar operaciones en ambas direcciones.



⇨ Stacks (pila). Es una lista ordenada de datos o elementos, la característica es que los datos se insertan y se eliminan en la parte superior.

⇨ Colas. Este tipo de lista tiene la característica de que los datos se insertan en un extremo y se eliminan en el otro.

⇨ d-heaps. Este tipo de estructura permite almacenar y manipular un conjunto de elementos eficientemente y cada elemento tiene asociado un número real.

⇨ Fibonacci heaps. Es la estructura más reciente y permite realizar las operaciones más eficientemente que en d-heaps. Utiliza las propiedades de los números Fibonacci.

➲ *Números Fibonacci*

Se definen por la siguiente recurrencia:

$F_o = 0$

$F_1 = 1$

$F_i = F_{i-1} + F_{i-2}$ para $i \geq 2$

Cada número Fibonacci es la suma de los dos números previos, dando la secuencia:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,…

Los algoritmos utilizados para la solución de problemas de redes requieren manipular los datos, como son por ejemplo las estructuras para representar la red gráficamente, por lo que un mejor almacenamiento conlleva a un procedimiento más eficiente.

Como se explica en este capítulo, el análisis asintótico sirve para conocer la eficiencia de los algoritmos, el cual se utiliza en el siguiente capítulo para conocer porque los métodos de solución que se explican son considerados eficientes.

El almacenamiento de datos, ha servido para realizar mejoras en los algoritmos existentes, y de igual forma en el siguiente capítulo, puede verse como utilizando este concepto se ha logrado hacerlos más eficientes.

CAPÍTULO 3.
MÉTODOS DE SOLUCIÓN

Para solucionar el problema de la ruta más corta existen varios algoritmos eficientes, con los cuales dependiendo de las características de la red (p. e. con costos no negativos), se puede obtener una solución exacta en un tiempo razonable.

En el capítulo anterior se explica la complejidad computacional y los conceptos que se utilizan para evaluar la eficiencia de un algoritmo, en este capítulo se aplica a cada método, con el fin de conocer cual es más eficiente. Por otra parte, los esquemas de almacenamiento de datos se utilizan para dar a conocer las mejoras que se han realizado a los algoritmos existentes, esta explicación es la parte final de este capítulo.

A continuación se describen dichos algoritmos, especificando las condiciones requeridas de la red para aplicarlos.

3.1. ALGORITMO DE DIJKSTRA

Este algoritmo fue publicado por primera vez por Edsger Dijkstra en 1959, con la finalidad de solucionar el problema de la ruta más corta desde el nodo fuente (*s*) a todos los demás nodos de la red (*i*), siempre y cuando las distancias de los arcos sean no negativas.

Es considerado el algoritmo más eficiente para solucionar este tipo de problemas. Se basa en la asignación temporal de etiquetas a los nodos:

⇨ La etiqueta inicial es una cota superior de la longitud de la ruta del nodo *s* al nodo *i*.

⇨ Como el proceso es iterativo las etiquetas se reducen en cada iteración (etiquetas temporales), hasta obtener la mínima longitud de la ruta de *s* a *i* (etiquetas permanentes).

Sea *G = (V, E, d)* una gráfica dirigida y con distancias no negativas en sus arcos, y sea *s* el nodo fuente (raíz) de *G*. Para cada nodo *v* en *G* se busca encontrar la ruta desde *s* a *v*, *P(s,v)*, talque la suma de las distancias de los arcos en *P(s,v)* sea mínima. El algoritmo de Dijkstra obtiene la arborescencia de rutas más cortas del nodo fuente *s* a todo nodo de la red *i*. El procedimiento consiste en elegir el arco para agregar en cada paso considerando que es el único que *minimiza d(s,k) + longitud(k,i)*.

Algoritmo [Flores de la Mota, 1999]
Datos de entrada: una gráfica dirigida *G = (V, E, d)*, donde cada arco *(i, j)* tiene una longitud no negativa, *d(i, j)*.

Paso 1 **Iniciación de etiquetas**

Etiqueta permanente:	$d(s) = 0$	
Etiquetas temporales:	$d(i) = \infty$	para toda $i \neq s$
	$a(i) = s$	donde s es el predecesor de i
	$p = s$	el nodo s tiene etiqueta permanente

Paso 2 **Actualización de etiquetas**

Sea E^+ el conjunto de arcos con trayectoria positiva

Para toda $i \in E^+(p)$ con etiqueta temporal, actualizar de la siguiente forma:

$$d(i) = \min \{d(i), d(p) + d(p,i)\}$$

Si d(i) se modificó, entonces a(i) = p

Sea i* tal que d(i*) = min {d(i) | d(i) es temporal}

Si d(i*) = ∞ terminar, no existe arborescencia de raíz s

En otro caso d(i*) es etiqueta permanente y hacer p = i*

Paso 3

Si todos los nodos tienen etiqueta permanente, terminar.

En otro caso, ir al paso 2.

Nota:

Si la ruta es de s a k, entonces el algoritmo se modifica en el paso 3 como sigue:

Si p = k terminar, y la distancia del nodo s al nodo i es d(i).

Si p ≠ k ir al paso 2.

Justificación del algoritmo de Dijkstra

El algoritmo de Dijkstra es válido, esto es, si dada una gráfica dirigida con distancias no negativas en sus arcos, *G = (V, E)*, y que tiene un nodo fuente (*s*) y un nodo sumidero (*f*), se puede encontrar la ruta más corta de *s* a *f*.

Para verificar lo anterior, primero se consideran los siguientes supuestos:

i. El conjunto *U* formado por el nodo *s* y los *k* nodos más cercanos a *s*
ii. Para cada $u \in U$, *d(u)* es la distancia de la ruta más corta en *G* desde *s* hasta *u*.
iii. Para cada $v \notin U$, *d(v)* es la distancia de la ruta más corta U_k a *v*, donde *d(v)* = ∞ si no existe tal ruta. (Nota: la ruta U_k a *v* inicia en *s* y tiene nodos intermedios en U_k solamente).

Por inducción, se tiene:

Para el paso 0, P(0):

i. El conjunto *U = {s}*, entonces *U* contiene a *s* y en este caso ningún otro nodo es cercano a *s*.

ii. $d(s) = 0$, lo cual es cierto, porque la distancia de s a si mismo es 0 (debido a que no se permite la existencia de rizos).
iii. La única ruta U_0 a v en este punto son los arcos desde s y para todos ellos $d(v) = d(s, v)$ cuando $(s, v) \in E$. Para todos los demás v, la condición es válida porque se tiene $d(v) = \infty$.

Por la hipótesis de inducción, es válido para el paso k, $P(k)$. Y falta probar para el paso $k+1$, $P(k+1)$. Entonces, como el siguiente nodo más cercano es u_{k+1}, se tiene que la distancia desde s en G como la longitud de la ruta más corta U_k a u_{k+1}, y por el supuesto (iii) para $P(k)$ esas longitudes son los valores de $d(v)$ al final del paso k. Entonces, u_{k+1} es en el inicio del paso $k+1$, el nodo $v \notin U_k$ para el cual $d(v)$ es mínima. Por lo tanto, cuando u_{k+1} es agregada a U en ese paso los supuestos (i) y (ii) son válidos.

Sin embargo, si $u_{k+1} = f$, entonces por el supuesto (ii), se tiene la ruta más corta de s a f. Por otra parte, si $u_{k+1} \neq f$ entonces, la única forma de que $d(v)$ pueda disminuir es si la ruta más corta desde s a u_{k+1}, agregando el arco (u_{k+1}, v) en caso de existir, se obtiene una distancia menor que el valor anterior de $d(v)$.

Tiempo de corrida del algoritmo

Sea una gráfica dirigida con n nodos, se tiene $n(n-1)$ arcos, uno para cada par ordenado de nodos. La inicialización toma $O(n)$ pasos, por la asignación para cada nodo. Este paso se repite máximo $n-1$ veces. Para verificar si el nodo $v \notin U$ se requieren máximo n veces. Para verificar si el nodo v es adyacente a u y no pertenece a U se requieren n veces. Por lo tanto, el algoritmo completo es $O(n^2)$ en el peor caso, que es cuando el nodo f es el más lejano al nodo s.

Ejemplo:

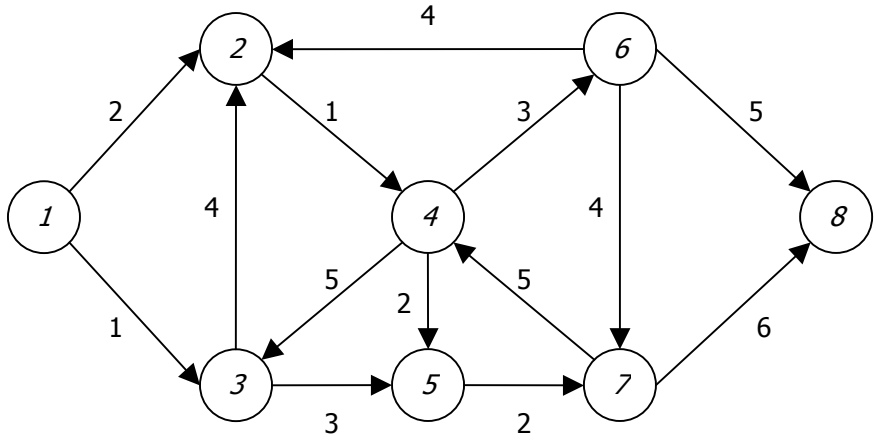


Figura 3.1. Ejemplo del algoritmo de Dijkstra

Iteración 1

Paso 1 Iniciación de etiquetas

d(1) = 0 etiqueta permanente
d(x) = ∞ para toda x ∈ V, tal que x ≠ s etiquetas temporales
p = 1

Paso 2 Actualización de etiquetas

$E^+(p = 1) = (2, 3)$
d(2) = min {∞, 2} = 2 a(2) = 1
d(3) = min {∞, 1} = 1 a(3) = 1 etiqueta temporal
el nodo 3 tiene la mínima etiqueta temporal
i*= 3
d(3) = 1 etiqueta permanente
p = 3
existen etiquetas temporales, continuar

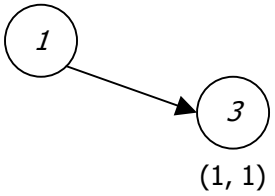


Figura 3.2. Iteración 1, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 2

Actualización de etiquetas

$E^+(p = 3) = (2, 5)$

d(2) = min {2, 1 + 4} = 2 a(2) = 1
d(5) = min {∞, 1 + 3} = 4 a(5) = 3 etiqueta temporal
el nodo 2 tiene la mínima etiqueta temporal
i*= 2
d(2) = 2 etiqueta permanente
p = 2
existen etiquetas temporales, continuar

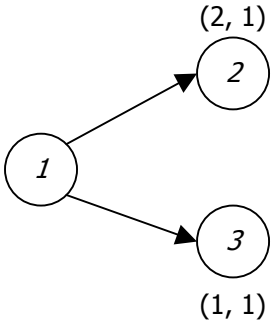


Figura 3.3. Iteración 2, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 3

Actualización de etiquetas

$E^+(p = 2) = (4)$
d(4) = min {∞, 2+1} = 3 a(4) = 2
el nodo 4 tiene la mínima etiqueta temporal
i*= 4
d(4) = 3 etiqueta permanente
p = 4
existen etiquetas temporales, continuar

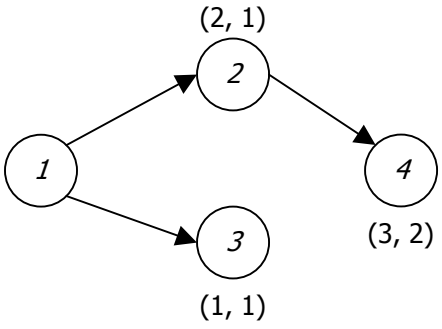


Figura 3.4. Iteración 3, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 4

Actualización de etiquetas

$E^+(p = 4) = (5, 6)$

d(5) = min {4, 3 + 2} = 4 a(5) = 3
d(6) = min {∞, 3 + 3} = 6 a(6) = 4 etiqueta temporal
el nodo 5 tiene la mínima etiqueta temporal
i*= 5
d(5) = 4 etiqueta permanente
p = 5
existen etiquetas temporales, continuar

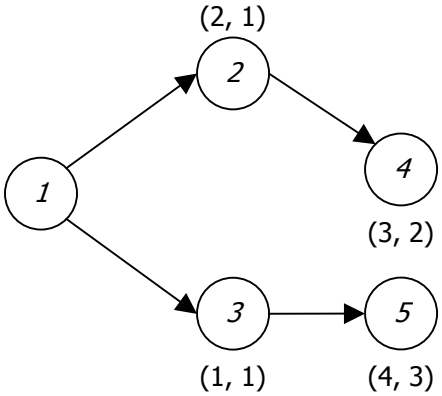


Figura 3.5. Iteración 4, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 5

Actualización de etiquetas

$E^+(p = 5) = (7)$
d(7) = min {∞, 4 + 2} = 6 a(7) = 5 etiqueta temporal
el nodo 6 tiene la mínima etiqueta temporal
i*= 6
d(6) = 6 etiqueta permanente
p = 6
existen etiquetas temporales, continuar

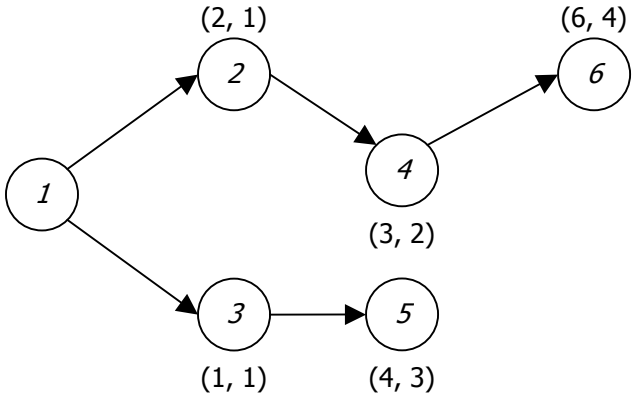


Figura 3.6. Iteración 5, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 6

Actualización de etiquetas

$E^+(p = 6) = (7, 8)$
$d(7) = \min \{6, 6 + 4\} = 6$ $a(7) = 5$
$d(8) = \min \{\infty, 6 + 5\} = 11$ $a(8) = 6$ etiqueta temporal
el nodo 7 tiene la mínima etiqueta temporal
$i^* = 7$
$d(7) = 6$ etiqueta permanente
$p = 7$
existen etiquetas temporales, continuar

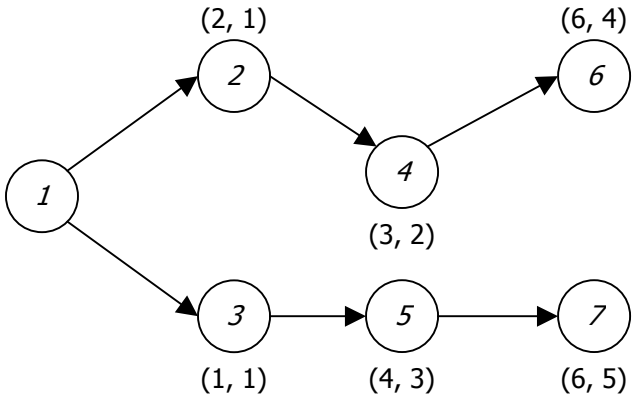


Figura 3.7. Iteración 6, algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Iteración 7

Actualización de etiquetas

$E^+(p = 7) = (8)$
$d(8) = \min \{11, 6 + 6\} = 11$ $a(8) = 6$
el nodo 8 tiene la mínima etiqueta temporal
$i^* = 8$
$d(8) = 11$ etiqueta permanente
$p = 8$
todos los nodos tienen etiquetas permanentes, por lo que se terminan las iteraciones

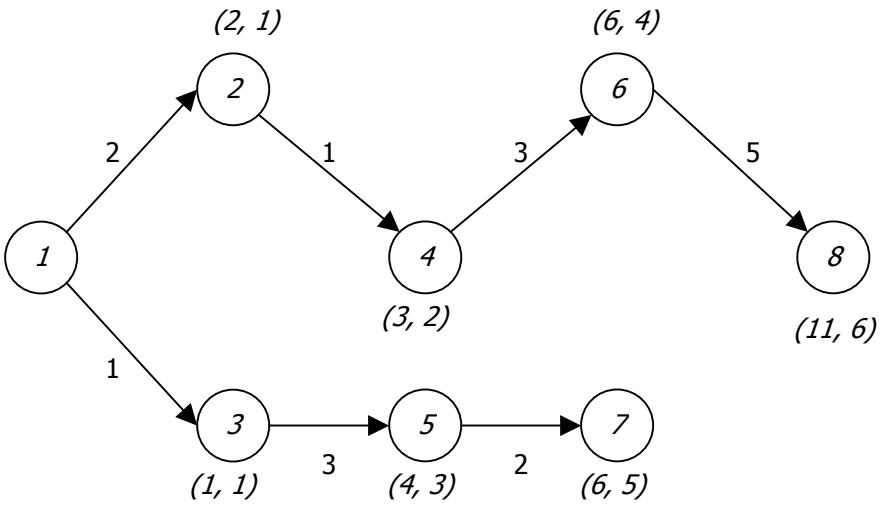


Figura 3.8. Arborescencia final del ejemplo del algoritmo de Dijkstra
Etiqueta (d(i), a(i))

Observaciones:

⇨ Este algoritmo trabaja igualmente bien en gráficas no dirigidas siempre y cuando se considere para cada arco *(u, v)* otro arco *(v, u)* con el mismo costo que *(u, v)*.

⇨ Si todas las distancias tienen el valor de 1, este algoritmo encuentra la ruta con el mínimo número de nodos.

⇨ El algoritmo calcula la distancia a todos los nodos cuya distancia se encuentra antes del nodo destino (o nodo sumidero). Por esta razón, este algoritmo se puede adaptar fácilmente para encontrar la distancia a todos los nodos.

⇨ El algoritmo no trabaja en gráficas con costos negativos en los arcos.

3.2. ALGORITMO DE FLOYD Y WARSHALL

El algoritmo de Floyd y Warshall resuelve el problema de la ruta más corta entre todo par de nodos. Fue desarrollado por Robert W. Floyd en 1962, y se basa en el teorema de Stephen Warshall sobre matrices boleanas (1960). El algoritmo fue desarrollado para solucionar el problema de la ruta más corta desde el nodo origen al nodo destino, pero utiliza los nodos intermedios para encontrar todas las rutas más cortas como regla general.

Este método emplea la técnica de la programación dinámica, y utiliza eficientemente la matriz de adyacencia de la red.

Entonces, sea $G = (V, E, d)$ una red, suponiendo que los nodos están numerados desde 1 hasta v, $V = \{v_1, v_2, \ldots, v_v\}$. Además, sea $V_k = \{v_1, v_2, \ldots, v_k\}$ el subconjunto de V que contiene a los primeros k nodos de V, $0 < k < V$.

Sea $P_k(v, w)$ la ruta más corta desde el nodo v al w que pasa a través de los vértices en V_k, si dicha ruta existe, $P_k(v, w) = \{v, \ldots, w\}$.

Sea $D_k(v, w)$ la longitud de la ruta $P_k(v,w)$

$$D_k(v,w) = \begin{cases} |P_k(v,w)| & P_k(v,w) \text{ existe} \\ \infty & \text{en otro caso} \end{cases}$$

como $V_0 = 0$, la ruta P_0 corresponde a los arcos de G

$$P_0(v,w) = \begin{cases} \{v,w\} & (v,w) \in E \\ \text{indefinida} & \text{en otro caso} \end{cases}$$

Entonces, la longitud de la ruta, D_0, corresponde a las distancias en los arcos de G

$$D_0(v,w) = \begin{cases} c(v,w) & (v,w) \in E \\ \infty & \text{en otro caso} \end{cases}$$

El algoritmo de Floyd y Warshall utiliza la secuencia de matrices $D_0, D_1, \ldots, D_v$. Las longitudes en D_i representan las rutas con nodos intermedios en V_i. Si V_{i+1}

$= V_i \cup \{v_{i+1}\}$, es posible obtener las distancias en D_{i+1} desde D_i considerando únicamente las rutas que pasan a través del nodo v_{i+1}.

Para cada par de nodos (v, w), se compara la distancia a $D_i(v,w)$, la cual representa la ruta más corta desde v a w que no pasa a través de v_{i+1}, con la suma $D_i(v, v_{i+1}) + D_i(v_{i+1}, w)$, que representa la ruta más corta desde v a w que pasa a través de v_{i+1}. Entonces, D_{i+1} se calcula:

$D_{i+1}(v, w) = min \{D_i(v, v_{i+1}) + D_i(v_{i+1},w), D_i(v,w)\}$

Algoritmo [Flores de la Mota, 1999]

Datos de entrada: una gráfica $G = (V, E, d)$, con n nodos.

Paso 1

Construir la matriz de adyacencia (nxn), haciendo las entradas d_{ij} de la siguiente forma:

Paso 2

Hacer k = k+1 para toda i ≠ k y j ≠ k, talque $d_{ij} \neq \infty$ y $d_{kj} \neq \infty$, hacer:

$$d_{ij} = \begin{cases} 0 & si\ \ i = j \\ \infty & si\ \ (i,j) \notin A \\ d(i,j) & si\ \ (i,j) \in A \end{cases}$$

k = 0

$d_{ij} = \min \{d_{ij}, d_{ik} + d_{kj}\}$

Paso 3

i. Si $d_{ii} < 0$ para alguna i, terminar. No existe solución.
ii. Si $d_{ii} \geq 0$ para toda i y k = n, terminar
    d_{ij} es la distancia más corta del nodo i al nodo j.
iii. Si $d_{ii} \geq 0$ para toda i y k < n, ir al paso 2.

Recuperación de rutas

Además de obtener la matriz de las distancias más cortas, se puede obtener la matriz que indica la ruta más corta. Si se construye una matriz *A (nxn)*, donde la entrada a_{ij} identifica el predecesor del nodo *j* en la ruta de *i* a *j* en cada iteración. Entonces, las entradas son $a_{ij} = i$ para todo par de nodos $i, j \in V$.
La matriz se actualiza en el paso 2 del algoritmo, para la k-ésima iteración se tiene:

$$a_{ij}^{(k)} = \begin{cases} a_{kj}^{(k-1)} & si\ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ a_{ij}^{(k-1)} & si\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Detección de ciclos negativos

El algoritmo de Floyd y Warshall tiene una característica muy importante que es la detección de ciclos negativos en una red. La manera en que lo hace es muy sencilla y es al actualizar el valor de *d(i, i)* para algún nodo *i*, si este valor es negativo, indica que existe un ciclo negativo.

Justificación

Por inducción:
Suponemos que previo a la iteración *k* se tiene que para $i, j \in v$, *d(i, j)* es la distancia de la ruta más corta *P(i, j)*, desde *i* a *j* en *G* que contiene únicamente los nodos en el conjunto *{1,…, k-1}* y que *a(i, j)* es el predecesor inmediato de *j* en *P*. Esto es cierto para *k = 0* (después de la inicialización).

En la iteración *k*, la longitud de *P* se compara con la longitud de una ruta *R* compuesta de dos subrutas, R_1 y R_2. R_1 es la ruta desde *i* hasta *k* y contiene solamente los nodos intermedios en *{1,…, k-1}* y R_2 es la ruta desde *k* hasta *j*, con los nodos intermedios en *{1,…, k-1}*. La ruta más corta de las dos es la que se elige.

La ruta más corta desde *i* hasta *j* en *G* contiene únicamente los nodos en el conjunto *{1,…, k}* y se tiene una de las dos opciones:

i. Si *P(i, j)* tiene como nodos intermedios al conjunto *{1,…, k}* y el nodo *k* es nodo intermedio, entonces *P* es de la forma $i \rightarrow k \rightarrow j$, donde R_1

(respectivamente para R_2) es la ruta más corta de *i* a *k* (respectivamente de *k* a *j*), entonces el conjunto de nodos intermedios es *{1,…, k-1}*, y ambas rutas fueron agregadas en la iteración *k-1*; o

ii. Si *P(i, j)* tiene como nodos intermedios al conjunto *{1,…, k}* y el nodo *k* no está contenido, entonces la ruta más corta *P(i, j)* tiene como nodos intermedios al conjunto *{1,…, k-1}*. Entonces, el nodo *k* se agrega en la iteración *k-1*.

Tiempo de corrida del algoritmo

Sea *d(u, v, r)* la longitud de la ruta más corta desde el nodo *u* al nodo *v*, donde todos los nodos intermedios (si existen) están numerados *r* o menos. Así, se tienen tres variables que son recurrentes *(u, v,* y *r)*, cada una de las cuales puede tomar valores en *V*, por lo tanto la complejidad de este algoritmo es de orden $\Theta(V^3)$, donde *V* es el conjunto de nodos de la red.

Ejemplo

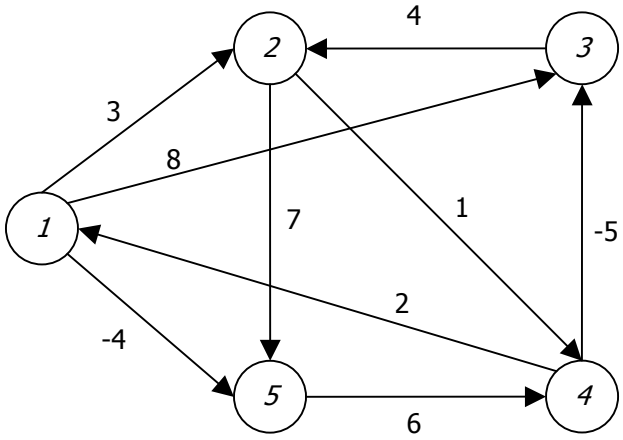


Figura 3.9. Ejemplo del algoritmo de Floyd-Warshall

<u>Iteración 1</u>

<u>Paso 1</u> Construir la matriz de adyacencia (nodos-nodos)

$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \end{array}$$

Paso 2
k = 1
Se actualizan valores para la matriz de adyacencia y se obtiene el nodo antecesor en la ruta:

d_{42} = min {∞, 2 + 3} = 5 a_{42} = 1
d_{43} = min {-5, 8 + 2} = -5 no se modifica
d_{45} = min {∞, 2 - 4} = -2 a_{45} = 1

$$D^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(1)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & \mathbf{1} & 3 & 4 & \mathbf{1} \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \end{array}$$

Paso 3
$d_{ii} \geq 0$, para toda i; y k = 1 < n = 5; ir al paso 2

Iteración 2

Paso 2
k = 2
Se actualizan valores para la matriz de adyacencia y se obtiene el nodo antecesor en la ruta:

d_{14} = min {∞, 3 + 1} = 4 a_{14} = 2
d_{34} = min {∞, 1 + 4} = 5 a_{34} = 2
d_{35} = min {∞, 7 + 4} = 11 a_{35} = 2

$$D^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & \mathbf{4} & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \mathbf{5} & \mathbf{11} \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(2)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & \mathbf{2} & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & \mathbf{2} & \mathbf{2} \\ 1 & 1 & 3 & 4 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \end{array}$$

Paso 3
$d_{ii} \geq 0$, para toda i; y k = 2 < n = 5; ir al paso 2

Iteración 3

Paso 2
k = 3
Se actualizan valores para la matriz de adyacencia y se obtiene el nodo antecesor en la ruta:
$d_{42} = \min \{5, 4 - 5\} = -1$ $a_{42} = 3$

$$D^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & \mathbf{-1} & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(3)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 2 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 2 & 2 \\ 1 & \mathbf{3} & 3 & 4 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix} \end{array}$$

Paso 3
$d_{ii} \geq 0$, para toda i; y k = 3 < n = 5; ir al paso 2

Iteración 4

Paso 2
k = 4
Se actualizan valores para la matriz de adyacencia y se obtiene el nodo antecesor en la ruta:
$d_{13} = \min \{8, 4 - 5\} = -1$ $a_{13} = 4$
$d_{21} = \min \{\infty, 1 + 2\} = 3$ $a_{21} = 4$
$d_{23} = \min \{\infty, 1 - 5\} = -4$ $a_{23} = 4$
$d_{25} = \min \{7, 1 - 2\} = -1$ $a_{25} = 4$
$d_{31} = \min \{\infty, 5 + 2\} = 7$ $a_{31} = 4$
$d_{35} = \min \{11, 5 - 2\} = 3$ $a_{35} = 4$
$d_{51} = \min \{\infty, 2 + 6\} = 8$ $a_{51} = 4$
$d_{52} = \min \{\infty, -1 + 6\} = 5$ $a_{52} = 4$
$d_{53} = \min \{\infty, -5 + 6\} = 1$ $a_{53} = 4$

$$D^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 3 & \mathbf{-1} & 4 & -4 \\ \mathbf{3} & 0 & \mathbf{-4} & 1 & \mathbf{-1} \\ \mathbf{7} & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \mathbf{8} & \mathbf{5} & \mathbf{1} & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(4)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & 2 & \mathbf{4} & 2 & 5 \\ \mathbf{4} & 2 & \mathbf{4} & 4 & \mathbf{4} \\ \mathbf{4} & 2 & 3 & 2 & \mathbf{4} \\ 1 & 3 & 3 & 4 & 1 \\ \mathbf{4} & \mathbf{4} & \mathbf{4} & 4 & 5 \end{pmatrix} \end{array}$$

Paso 3
$d_{ii} \geq 0$, para toda i; y k = 4 < n = 5; ir al paso 2

Iteración 5

Paso 2
k = 5
Se actualizan valores para la matriz de adyacencia y se obtiene el nodo antecesor en la ruta:

d_{12} = min {3, -4 + 5} = 1 $\qquad a_{12} = 5$
d_{13} = min {-1, -4 + 1} = -3 $\qquad a_{13} = 5$
d_{14} = min {4, -4 + 6} = 2 $\qquad a_{14} = 5$

$$D^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & \mathbf{1} & \mathbf{-3} & \mathbf{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \end{array} \qquad A^{(5)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 1 & \mathbf{5} & \mathbf{5} & \mathbf{5} & 5 \\ 4 & 2 & 4 & 4 & 4 \\ 4 & 2 & 3 & 2 & 4 \\ 1 & 3 & 3 & 4 & 1 \\ 4 & 4 & 4 & 4 & 5 \end{pmatrix} \end{array}$$

Paso 3
$d_{ii} \geq 0$, para toda i; y k = 5 = n = 5; terminar

En la matriz $D^{(5)}$ se tiene la longitud de la ruta, y en la matriz $A^{(5)}$ la secuencia de los nodos. Entonces, si por ejemplo se quisiera la ruta del nodo 1 al nodo 2, primero d(1, 2) = 1 y la ruta se obtiene siguiendo los valores en la matriz $A^{(5)}$:

a_{12} = 5, esto significa que el nodo 5 es nodo intermedio en la ruta.
a_{15} = 5, esto significa que la ruta del nodo 1 al nodo 5 es directa.
a_{52} = 4, esto significa que el nodo 4 es nodo intermedio en la ruta.
a_{54} = 4, esto significa que la ruta del nodo 5 al nodo 4 es directa.
a_{42} = 3, esto significa que el nodo 3 es nodo intermedio en la ruta.
a_{43} = 3, esto significa que la ruta del nodo 4 al nodo 3 es directa.
a_{32} = 2, esto significa que la ruta del nodo 3 al nodo 2 es directa.

Por lo tanto, la ruta es: $1 \rightarrow 5 \rightarrow 4 \rightarrow 3 \rightarrow 2$

Observaciones

⇨ El algoritmo de Floyd-Warshall no funciona con ciclos negativos, sin embargo, la detección de estos ciclos es posible al examinar la diagonal de la matriz (presencia de algún valor negativo).

⇨ Permite encontrar la ruta más corta entre todo par de nodos, sin importar si la red es dirigida o no, con la posibilidad de la existencia de costos negativos en los arcos.

⇨ Al ejecutar ambas matrices $D^{(k)}$ y $A^{(k)}$, se obtiene al mismo tiempo la longitud de la ruta y la secuencia de los nodos en la misma.

⇨ Este método es eficiente en redes densas.

3.3. ALGORITMO DE BELLMAN-FORD (MOORE)

Este algoritmo fue publicado por primera vez por Moore en 1957, y después de forma independiente por Bellman en 1958, quien usó la idea de la relajación de arcos, que había sido propuesta por Ford en 1956. Este algoritmo es conocido normalmente por Bellman-Ford. El algoritmo calcula la ruta más corta del nodo fuente *s* a todo nodo de la red *i*, además es eficiente incluso si en la red existen arcos con distancias negativas. También puede detectar la presencia de ciclos negativos.

Este método es iterativo y se basa en la etiquetación de nodos, donde en la iteración *k*, la etiqueta representa el costo de la ruta más corta del nodo fuente *s* a todo nodo *i*, la cual contiene *k+1* o menos arcos.

Algoritmo [Christofides, 1975]

Datos de entrada: una gráfica *G = (V, E, d)*

Paso 1 Iniciación de etiquetas

Sea E^+ el conjunto de arcos con trayectoria positiva.

Sea $S = E^+_s(s)$, $k = 1$, $d^1(i) = d(s, i)$, para toda $i \in E^+(s)$ y $d^1(i) = \infty$, para todas las demás i.

Paso 2 Actualización de etiquetas

Para todo nodo $i \in E^+_s(s)$, $(i \neq s)$, actualizar de acuerdo a:

$$d^{k+1}(i) = \min\left[d^k(i), \min_{j \in T_i}\left\{d^k(j) + d(j,i)\right\}\right]$$

donde $T_i = E^-(i) \cap S$

si $i \notin E^+(s)$, sea $d^{k+1}(i) = d^k(i)$

Paso 3

a) Si $k \leq n-1$ y $d^{k+1}(i) = d^k(i)$ para toda i, se ha obtenido la solución óptima y las etiquetas son las longitudes de las rutas más cortas, terminar.
b) Si $k < n-1$ pero $d^{k+1}(i) \neq d^k(i)$ para alguna i, ir al paso 4.
c) Si $k = 1$ y $d^{k+1}(i) \neq d^k(i)$ para alguna i, entonces existe un circuito negativo en la red y no existe solución, terminar.

Paso 4

Actualizar el conjunto S:

$S = \{i \mid d^{k+1}(i) \neq d^k(i)\}$

Paso 5

Sea $k = k+1$ ir al paso 2

Nota:

Cuando el arco *(i, j)* no tiene dirección se reemplaza por los arcos *(i, j)* y *(j, i)* con el mismo costo.

Recuperación de rutas

Para obtener las rutas solo es necesario realizar lo siguiente en el desarrollo del algoritmo:

⇨ En el paso 1 del algoritmo se hace a(i) =s, donde s es el predecesor de i.

⇨ Y en el paso 2, si $d^{k+1}(i)$ se modificó, entonces a(i) = j.

Justificación

Demostración por inducción.

Si la ruta más corta al nodo *j* consiste de *k*, entonces después de *k* iteraciones *d(j)* es correcto.

Para *k = 1*

Si la ruta más corta a *j* tiene solamente un arco, entonces es el arco *(s, j)*. Durante la iteración *k = 1* se revisa este arco y se actualiza la distancia *d(j)* correctamente.

Suponemos por hipótesis de inducción, que después de *k – 1* iteraciones, los nodos que utilizan a lo más *k – 1* arcos en la ruta más corta, están etiquetados correctamente. Si se considera un nodo *j* el cual en su ruta más corta tiene exactamente *k* arcos, sea $s \rightarrow i_1 \rightarrow i_2 \ldots i_{k-1} \rightarrow i_k = j$, la ruta más corta a *j*. Por inducción, $d(i_{k-1}, j)$ es válido, cuando el arco (i_{k-1}, j) se revisa durante la k-ésima iteración, se actualiza $d(j) = d(i_{k-1}) + c_{ik-1,j}$. Entonces, *d(j)* es válido después de *k* iteraciones.

Tiempo de corrida

Como *d(s)= 0* y *d(s, i)= ∞*, para toda *i*, al aplicar $min\ \{d^{k-1}(i), [d^{k-1}(j) + d(j, i)]\ \}$, se puede calcular la ruta más corta desde *s* a *t* que no use más de *n-1* arcos. Suponiendo que, la gráfica tiene *n* nodos, en cada iteración $min\ \{d^{k-1}(i), [d^{k-1}(j) + d(j, i)]\}$, se tienen que calcular *n-1* nuevos valores siendo el mínimo de a lo más *n* valores. En conclusión, el algoritmo requiere $O(n^3)$ pasos en total.

Ejemplo

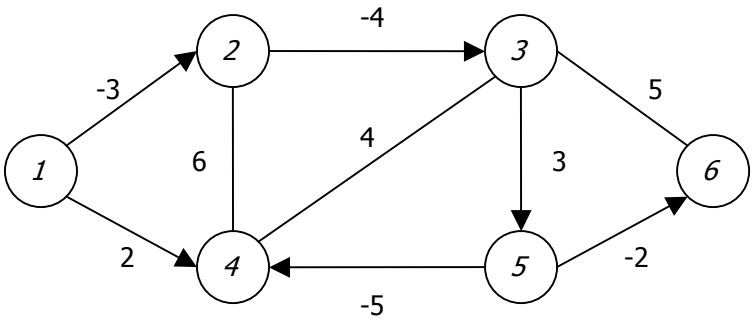


Figura 3.10. Ejemplo del algoritmo de Bellman-Ford

Iteración 1

Paso 1 Iniciación de etiquetas

s = 1, S = {2, 4}, $d^1(1) = 0$, $d^1(2) = -3$, a(2) =1, $d^1(4) = 2$, a(4) = 1

$d^1(i) = \infty$ para todas las demás i. Sea k = 1

Paso 2 Actualización de etiquetas

$E^+(S) = \{2, 3, 4\}$

Para 2: $T_2 = \{1, 4\} \cap \{2, 4\} = \{4\}$

$d^2(2) = \min \{-3, [d^1(4) + d(4, 2)]\}$

$= \min \{-3, (2 + 6)\}$

$= -3$ a(2) = 1

Para 3: $T_3 = \{2, 4, 6\} \cap \{2, 4\} = \{2, 4\}$

$d^2(3) = \min \{\infty, [d^1(2) + d(2, 3), d^1(4) + d(4, 3)]\}$

$= \min \{\infty, (-3 - 4), (2 + 4)\}$

$= -7$ a(3) = 2

Para 4: $T_4 = \{1, 2, 3, 5\} \cap \{2, 4\} = \{2\}$

$d^2(4) = \min \{2, [d^1(2) + d(2, 4)]\}$

$= \min \{2, (-3 + 6)\}$

$= 2$ a(4) = 1

$d^2(i) : [0, -3, -7, 2, \infty, \infty]$

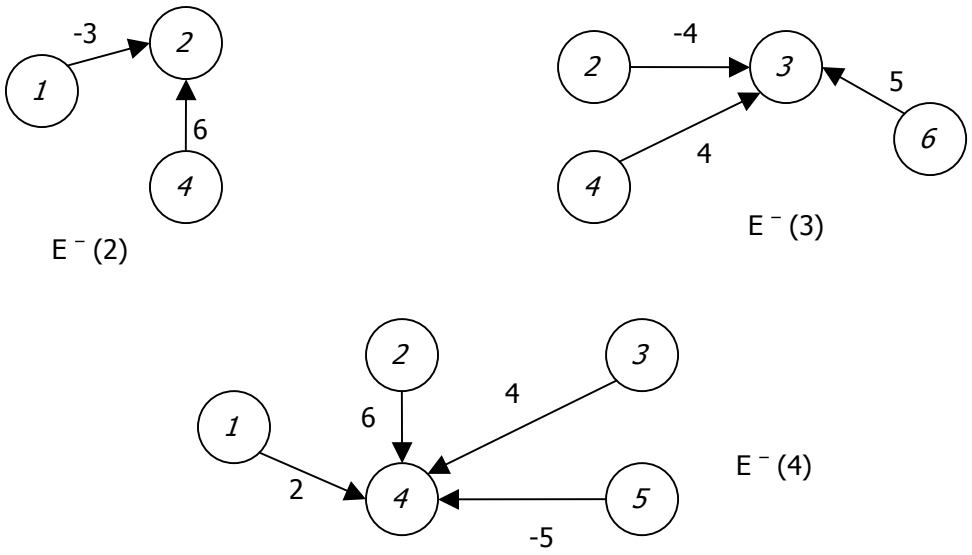


Figura 3.11. Iteración 1, paso 2, algoritmo de Bellman-Ford
$E^-(2)$, $E^-(3)$ y $E^-(4)$

Paso 3 (b)

Ir al paso 4

Paso 4

S = {3}

Paso 5

k = 2, ir al paso 2

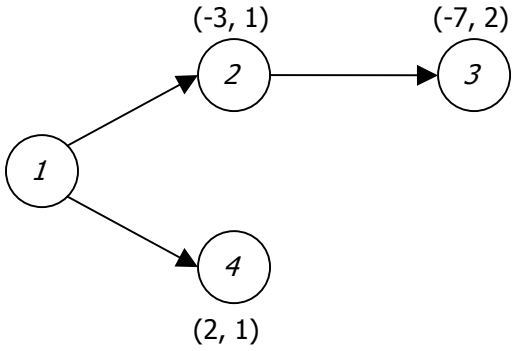


Figura 3.12. Iteración 1, algoritmo de Bellman-Ford
Etiqueta (d(i), a(i))

Iteración 2

Paso 2 Actualización de etiquetas

$E^+(S) = \{4, 5, 6\}$

Para 4: $T_4 = \{1, 2, 3, 5\} \cap \{3\} = \{3\}$
$d^3(4) = \min \{2, [d^2(3) + d(3, 4)]\}$
$= \min \{2, (-7 + 4)\}$
$= -3$ a(4) = 3

Para 5: $T_5 = \{3\} \cap \{3\} = \{3\}$
$d^3(5) = \min \{\infty, [d^2(3) + d(3, 5)]\}$
$= \min \{\infty, (-7 + 3)\}$
$= -4$ a(5) = 3

Para 6: $T_6 = \{3, 5\} \cap \{3\} = \{3\}$
$d^3(6) = \min \{\infty, [d^2(3) + d(2, 6)]\}$
$= \min \{\infty, (-7 + 5)\}$
$= -2$ a(6) = 3

$d^3(i)$: [0, -3, -7, -3, -4, -2]

Paso 3 (b)

Ir al paso 4

Paso 4

S = {4, 5, 6}

Paso 5

k = 3, ir al paso 2

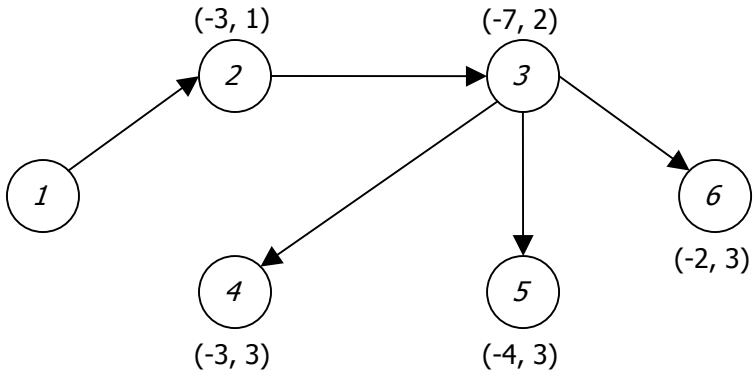


Figura 3.13. Iteración 2, algoritmo de Bellman-Ford
Etiqueta (d(i), a(i))

Iteración 3

Paso 2 Actualización de etiquetas

$E^+(S) = \{2, 3, 4, 6\}$

Para 2: $T_2 = \{1, 4\} \cap \{4, 5, 6\} = \{4\}$
$d^4(2) = \min \{-3, [d^3(4) + d(4, 2)]\}$
$= \min \{-3, (-3 + 6)\}$
$= -3$ $a(2) = 1$

Para 3: $T_3 = \{2, 4, 6\} \cap \{4, 5, 6\} = \{4, 6\}$
$d^4(3) = \min \{-7, [d^3(4) + d(4, 3), d^3(6) + d(6, 3)]\}$
$= \min \{-7, (-3 + 4), (-2 + 5)\}$
$= -7$ $a(3) = 2$

Para 4: $T_4 = \{1, 2, 3, 5\} \cap \{4, 5, 6\} = \{5\}$
$d^4(3) = \min \{-3, [d^3(5) + d(5, 4)]\}$
$= \min \{-3, (-4 - 5) \}$
$= -9$ $a(4) = 5$

Para 6: $T_6 = \{3, 5\} \cap \{4, 5, 6\} = \{5\}$
$d^4(6) = \min \{-2, [d^3(5) + d(5, 6)]\}$
$= \min \{-2, (-4 - 2)\}$
$= -6$ $a(6) = 5$

$d^4(i)$: [0, -3, -7, -9, -4, -6]

Paso 3 (b)

Ir al paso 4

Paso 4

S = {4, 6}

Paso 5

k = 4, ir al paso 2

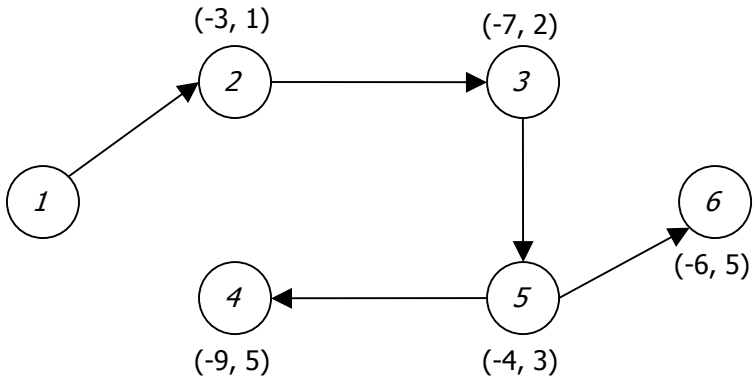


Figura 3.14. Iteración 3, algoritmo de Bellman-Ford
Etiqueta (d(i), a(i))

Iteración 4

Paso 2 Actualización de etiquetas

$E^+(S) = \{2, 3\}$

Para 2: $T_2 = \{1, 4\} \cap \{2, 3\} = \phi$

Para 3: $T_3 = \{2, 4, 6\} \cap \{2, 3\} = \{2\}$

$d^5(3) = \min \{-7, [d^4(2) + d(2, 3)]\}$

$= \min \{-7, (-3 - 4)\}$

$= -7$ a(3) = 1

$d^5(i) : [0, -3, -7, -9, -4, -6]$

Paso 3 (a)

Terminar, $d^4(i) = d^5(i)$, se han obtenido la ruta más corta

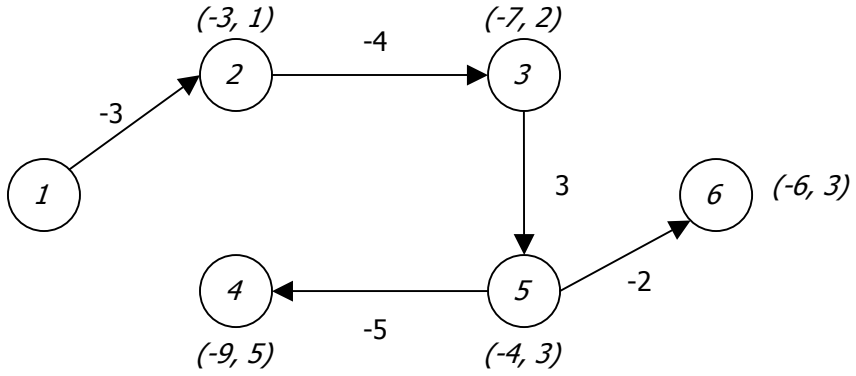


Figura 3.15. Arborescencia del ejemplo del algoritmo de Bellman-Ford
Etiqueta (d(i), a(i))

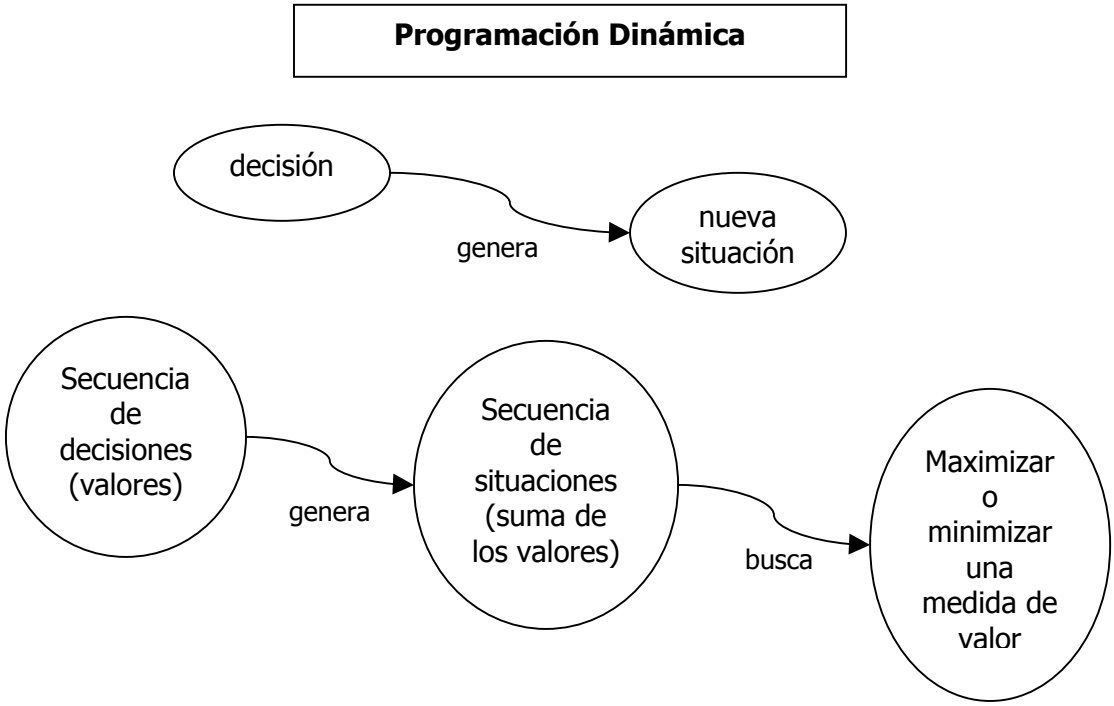
Observaciones

⇨ Detecta la presencia de ciclos negativos.

3.4. PROGRAMACIÓN DINÁMICA

La técnica de Programación Dinámica [Dreyfus y Law, 1977] es un procedimiento matemático aplicable a problemas que requieren una serie de decisiones interrelacionadas, con la finalidad de determinar la combinación de decisiones que optimiza la efectividad total. Es decir, el problema completo se

divide en subproblemas para los cuales la toma de decisiones es más fácil y a su vez generan una nueva situación. Además, como los cálculos son en etapas y se enlazan en forma recursiva, al final se obtiene la solución al problema completo.



Características

a. El problema se divide en etapas (decisiones interrelacionadas) y se requiere una política de decisión en cada una de ellas. En cada etapa existe cierto número de estados, que son las distintas condiciones posibles en las que se puede encontrar el sistema en ese momento.

   La política de decisión selecciona la mejor alternativa entre todos los posibles estados, generando una relación (recursiva) para la siguiente etapa.

b. La relación recursiva utiliza el valor óptimo dado por la política de decisión en la etapa *n* para la etapa *n+1* (o *n-1*), dependiendo del tipo de procedimiento utilizado:

   ⇨ Procedimiento de avance (forward). Los cálculos avanzan de la primera etapa a la última.

⇨ Procedimiento de retroceso (backward). Los cálculos avanzan de la última etapa a la primera.

La elección del tipo de procedimiento depende de la estructura del problema y el tipo de información de que se disponga, los cuales implican la formulación de las etapas y los estados buscando la manera más lógica y directa.

c. **Principio de Optimalidad.** Es la teoría unificadora fundamental y se refiere a que las decisiones tomadas en las etapas anteriores son independientes de las decisiones para las etapas restantes. Este concepto es básico debido a que, como los cálculos son recursivos, los valores óptimos de las etapas anteriores están implícitos en la decisión de la etapa actual, de esta manera al seleccionar el mejor valor en cada una de las siguientes etapas, se puede llegar a resolver el problema completo óptimamente.

d. Terminología.

i. Función de valor óptimo, $f_n(s_n)$.

Regla para asignar los valores en las distintas etapas.

s_n, indica que el sistema se encuentra en la etapa n.

ii. Función de estrategia óptima, $P_n(s_n)$.

La primera mejor decisión asociada a cada subproblema.

iii. Relación de recurrencia

El principio de optimalidad genera una fórmula o conjunto de fórmulas relacionando varios valores de s.

$f_n(s_n) = \max_{x_n}\{f_n(s_n, x_n)\}$ o $f_n(s_n) = \min_{x_n}\{f_n(s_n, x_n)\}$

x_n, variable de decisión para la etapa n.

iv. Condiciones frontera, $f_0(s_n)$.

El valor de la función de valor óptimo, $f_n(s_n)$, para ciertos subproblemas se considera obvia tanto para la definición del problema como para la definición de la función.

Planteamiento

Sea $G = (V, E, d)$ una gráfica y cada arco $(i, j) \in E$, que conecta a los nodos i y j, tiene asociada una distancia (costo), d_{ij}. Suponemos que $d_{ii} = 0$, sin embargo, no necesariamente $d_{ij} = d_{ji}$ o se cumple la desigualdad $d_{ij} + d_{jk} \geq d_{ik}$.

Entonces, el problema consiste en encontrar la ruta más corta desde el nodo fuente (origen) *1 ,* a todo nodo de la red *i*.

Gráfica acíclica

En una gráfica acíclica dirigida existe la posibilidad de enumerar los nodos considerando la dirección de los arcos, es decir, para el arco (i, j) $\in$ E, implica que i < j. Además, el nodo 1 (origen) es el nodo que únicamente tiene arcos que salen de él.

La formulación en programación dinámica en procedimiento de avance (forward), quedaría de la siguiente manera:

Función de valor óptimo:

f_i = la longitud de la ruta más corta desde el nodo 1 (nodo fuente) al nodo *i*

Relación de recurrencia:

$$f_i = \min_{j<i}(f_j + d_{ji}) \qquad (i = 2,\ldots,V)$$

$d_{ij} = \infty$ si $(j, i) \notin E$

Condiciones frontera:

$f_1 = 0$

Función de estrategia óptima:

$P_i = j$, j es el nodo procedente al nodo i y minimiza el lado derecho en f_i

Justificación

Para demostrar que la relación de recurrencia está definida correctamente se utiliza inducción.

Para $i = 2$, $f_2 = d_{12}$, porque la única ruta del nodo 1 la nodo 2 es utilizando el arco (1, 2).

Se supone f_j es válido, $j = 1,\ldots, i-1$

Por demostrar f_i, como los arcos están numerados por ser una gráfica acíclica, cualquier arco que entre en el nodo i, tiene como nodo precedente uno menor, puede ser el nodo j. Por hipótesis se tiene que la longitud de la ruta más corta del nodo 1 al nodo i, tiene al nodo j como el inmediato precedente y la longitud está dada por $f_j + d_{ji}$. Entonces al elegir el nodo j óptimamente, f_i está definida correctamente.

Tiempo de corrida

El número de operaciones requeridas en la formulación anterior es de: etapa 2: 1 suma, etapa 3: 2 sumas y 1 comparación,…, etapa n: (n-1) sumas y (n-2) comparaciones; por lo que el total de operaciones es de: n(n-1)/2 sumas y (n-1)(n-2)/2 comparaciones, donde n es el número de nodos de la red, más el tiempo requerido en la enumeración de nodos inicial.

Ejemplo

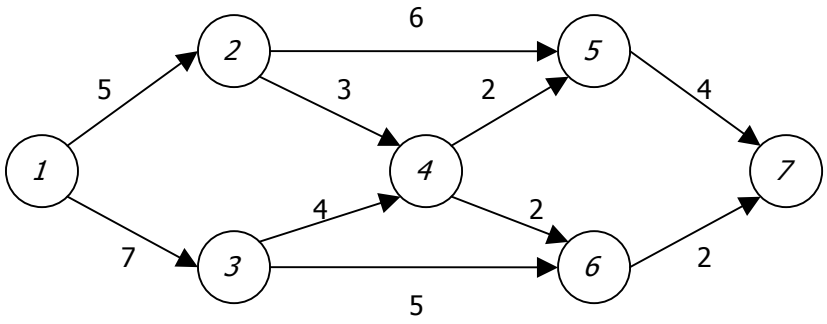


Figura 3.16. Ejemplo de Programación dinámica

$f_1 = 0$

$f_2 = f_1 + d_{12} = 0 + 5 = 5$ $\quad P_2 = 1$

$f_3 = \min (f_1 + d_{13}, f_2 + d_{23}) = (0 + 7, 5 + \infty) = 7$ $\quad P_3 = 1$

$f_4 = \min (f_1 + d_{14}, f_2 + d_{24}, f_3 + d_{34}) = (0 + \infty, 5 + 3, 7 + 4) = 8$ $\quad P_4 = 2$

$f_5 = \min (f_1 + d_{15}, f_2 + d_{25}, f_3 + d_{35}, f_4 + d_{45}) = (0 + \infty, 5 + 6, 7 + \infty, 8 + 2) = 10$ $P_5 = 4$
$f_6 = \min (f_1 + d_{16}, f_2 + d_{26}, f_3 + d_{36}, f_4 + d_{46}, f_5 + d_{56}) = (0 + \infty, 5 + \infty, 7 + 5, 8 + 2, 10 + \infty) = 10$ $P_6 = 4$
$f_7 = \min (f_1 + d_{17}, f_2 + d_{27}, f_3 + d_{37}, f_4 + d_{47}, f_5 + d_{57}, f_6 + d_{67}) = (0 + \infty, 5 + \infty, 7 + \infty, 8 + \infty, 10 + 4, 10 + 2) = 12$ $P_7 = 6$

La ruta se obtiene siguiendo de forma inversa la función de estrategia óptima:

$P_7 = 6 \rightarrow P_6 = 4 \rightarrow P_4 = 2 \rightarrow P_2 = 1$

Entonces, para este ejemplo la ruta del nodo 1 al nodo 7 es:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 7$, y la distancia es de 12.

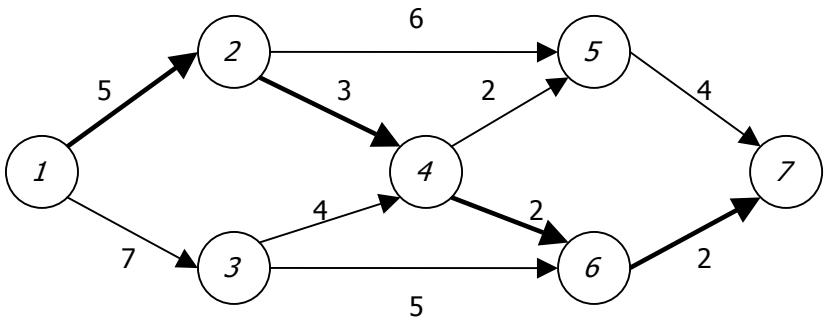


Figura 3.17. Solución del ejemplo de Programación dinámica

Gráfica cíclica

En una gráfica que tenga la presencia de uno o más ciclos, no es posible enumerar los nodos en orden creciente como se hizo para el caso de una gráfica acíclica, por esta razón, la formulación anterior no funciona. Como se menciona en el capítulo 1, (punto 1.5), existen tres formas en que se puede presentar el problema de la ruta más corta:

a. Del nodo origen *s* al nodo destino *t*.
b. Del nodo origen *s* a todo nodo de la red *i*.
c. Entre todo par de nodos.

Y como se ha visto en el desarrollo de este capítulo, existen algoritmos eficientes para su solución. Sin embargo, los algoritmos anteriores pueden ser

formulados con la técnica de programación dinámica, así, si se utiliza el algoritmo de Dijkstra, se puede resolver el problema para los dos primeros casos, y si se quiere obtener la ruta entre todo par de nodos, se utiliza el algoritmo de Floyd. La forma de plantear los algoritmos es posible consultarla en libros de programación dinámica [Dreyfus y Law, 1977].

Observaciones

⇨ La técnica de programación dinámica tiene enfoque general, es decir, no existe una formulación estándar, es necesario desarrollar las ecuaciones específicas para cada problema particular.

⇨ Si se compara con la enumeración exhaustiva, proporciona grandes ahorros computacionales, al reducir considerablemente las combinaciones de decisiones, en especial en la solución de problemas grandes.

3.5. MODIFICACIONES Y MEJORAS DE LOS ALGORITMOS

Los algoritmos para resolver el problema de ruta más corta han tenido modificaciones y mejoras debido a que, son utilizados frecuentemente en subrutinas para la solución de problemas complejos. Como se menciona en el capítulo anterior en el punto 2.6., las diferentes formas en el almacenamiento de datos ayudan a que un algoritmo sea más eficiente. A continuación se explican algunas mejoras que se han realizado a los algoritmos descritos en el desarrollo de este capítulo:

- ✓ El algoritmo de Dijkstra:
  - ✧ Si utiliza para el almacenamiento de nodos a Fibonacci heap, el tiempo total de corrida es: $O(E + V \log V)$, asumiendo que no hay arcos con costos negativos.

- ✧ Para obtener la ruta entre todo par de nodos, utilizando el algoritmo del nodo fuente a todo nodo *V* veces y se utiliza un array unidimensional, el tiempo es de: $O(VE + V^2 \log V)$, considerando que todos los arcos tienen costos positivos.

- ✧ Si se utilizan heaps binarios, el tiempo es de: $O((E + V) \log V)$.

- ✧ Si se utiliza una cola, el tiempo es: $O(E + V^2)$.

✓ El algoritmo de Bellman-Ford:

- ✧ Si se utiliza una cola, el tiempo es de: $O(VE)$.

- ✧ Para obtener la ruta entre todo par de nodos, utilizando el algoritmo del nodo fuente al todo nodo *V* veces y se utiliza un array unidimensional, el tiempo es de: $\Theta(V^2E) = O(V^4)$.

✓ El algoritmo símplex para redes:

- ✧ Si se utiliza la regla de pivoteo del primer arco, el tiempo es de: $O(V^2E)$.

- ✧ Si se utiliza la regla de pivoteo de Dantzig, el tiempo es de: $O(V^2E \log (VC))$.

3.5.1. ALGORITMO DE JOHNSON

El algoritmo de Johnson es un buen ejemplo de las modificaciones que pueden hacerse con los algoritmos existentes, es decir, este método encuentra la ruta más corta entre todo par de nodos, utilizando Dijkstra y Bellman-Ford mejorados como subrutinas.

Este método consiste en ejecutar el algoritmo de Dijkstra para cada nodo de la red. Pero considerando primeramente una reponderación de los arcos para que todos sean positivos. Se supone que la red tiene un nodo *s*, el cual tiene una ruta a todo nodo de la red, entonces el algoritmo encuentra la ruta más corta desde el nodo *s* a todo nodo utilizando Bellman-Ford. Sin embargo, si en la red no existe un nodo *s* con rutas a todos los demás nodos, entonces se añade un nuevo nodo (nodo artificial) *s*, con costos cero en sus arcos, que van desde el nodo *s* al todo nodo de la red, pero ningún arco entra en el nodo *s*. Esta adición no altera las rutas más cortas entre todo par de nodos debido a que, no hay rutas que entren a *s*.

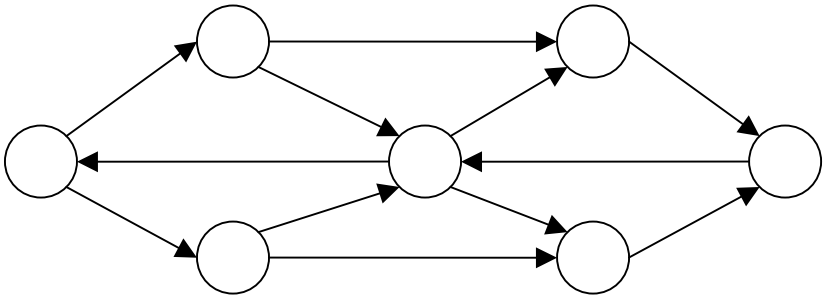


Figura 3.18. Ejemplo de red sin nodo fuente

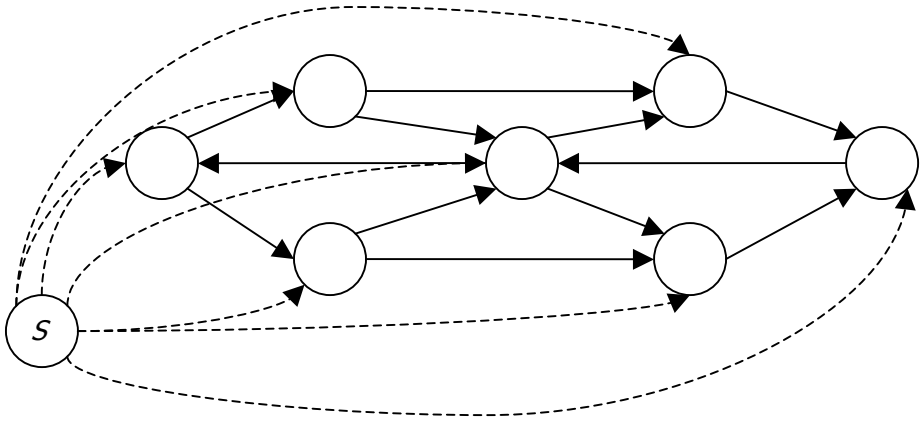


Figura 3.19. Ejemplo después de añadir el nodo artificial *s*

Reponderación

Para una red dirigida ponderada *G = (V, E)* y su función de ponderación *w: E* $\rightarrow$ ***R***, sea una función *h: V* $\rightarrow$ ***R***. La función de ponderación *w′(i, j) = w(i, j) + h(i) – h(j)* genera las mismas rutas más cortas que la ponderación original *w*.

Justificación

Sea *h: V* $\rightarrow$ ***R*** una función que asocia un nodo en V con un número real.

Se define *w′ (i, j) = w(i, j) + h(i) – h(j)*

Ahora se prueba que la ruta más corta *p* usando *w* es la misma que usando *w′* y viceversa:

Considerando cualquier ruta $p = (v_0, \ldots, v_k)$ su costo *w′* es:

$$w'(p) = \sum_{i=1,\ldots,k} w'(v_{i-1}, v_i) = \sum_{i=1,\ldots,k} \left[w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i) \right] = w(p) + h(v_0) - h(v_k)$$

Asumiendo que una ruta *p* entre *i* y *j* es la más corta con un costo *w(p)*, pero si se tiene una ruta más corta diferente *q* en *w′*, esto es, de costo *w′ (q) < w′ (p)*, entonces, hay una contradicción:

w(q) + h(i) – h(j) = w′ (q) < w′ (p) = w(p) + h(i) – h(j)

Simplificando tenemos, *w(q) < w(p)*, lo cual contradice la elección de *p* como la ruta más corta en *w* entre *i* y *j*.

Incluso, los ciclos no cambian su costo al utilizar *w′* en lugar de *w*:

w′ (c) = w(c) + h(i) – h(j) = w(c), porque en un ciclo el primer nodo y el último son el mismo.

Algoritmo

El algoritmo utiliza otros algoritmos vistos previamente (Dijkstra y Bellman-Ford), y considera seis etapas:

1. aumentar el nodo v_0
2. ejecutar Bellman-Ford
3. renombrar los nodos
4. reponderación de los arcos
5. ejecutar Dijkstra para cada nodo
6. restablecer las distancias de origen

Tiempo de corrida

Para modificar la gráfica G añadiendo el nodo artificial, requiere *O(V)*, el tiempo de corrida de Bellman-Ford es *O(VE)*, para la reponderación de la gráfica *O(E)* y para ejecutar *V* veces Dijkstra con Fibonacci heaps $O(VE + V^2 \log V)$. Por lo tanto, el tiempo total de corrida es de $O(VE + V^2 \log V)$.

Observaciones

⇨ El algoritmo de Johnson es una solución mejor para trabajar con gráficas poco densas.

3.5.2. TABLA RESUMEN DE LOS TIEMPOS DE CORRIDA

La siguiente tabla muestra los tiempos de corrida de los algoritmos explicados en el desarrollo del capítulo, especificando el tipo de problema en que pueden aplicarse, y también, los tiempos que se obtienen en la aplicación de las mejoras.

Desde el nodo fuente a todo nodo de la red	
Algoritmo de Dijkstra ⇨ Trabaja con distancias no negativas ⇨ Gráficas dirigidas y no dirigidas	O (V^2)
Algoritmo de Bellman-Ford ⇨ Trabaja con costos negativos ⇨ Detecta ciclos negativos	O (V^3)
Método Símplex	Pseudopolinomial
Algoritmo de Dijkstra utilizando Fibonacci heaps	O (E + V log V)
Algoritmo de Dijkstra utilizando heaps binarios	O ((E +V) log V)
Algoritmo de Dijkstra utilizando cola	O (E + V^2)
Algoritmo de Bellman-Ford utilizando cola	O (VE)
Método Símplex utilizando pivoteo de primer arco	O (V^2E)

Método Símplex utilizando pivoteo de Dantzig	O (V^2E log VC)
Entre todo par de nodos de la red	
Algoritmo de Floyd y Warshall ⇨ Trabaja con costos negativos ⇨ Gráficas dirigidas y no dirigidas ⇨ Detecta ciclos negativos ⇨ Eficiente en redes densas	Θ (V^3)
Algoritmo de Dijkstra utilizando array unidimensional	O (VE + V^2 log V)
Algoritmo de Bellman-Ford utilizando array unidimensional	O (V^4)
Algoritmo de Johnson ⇨ Eficiente en redes poco densas	O (VE + V^2 log V)

Tabla 3.1. Tiempos de corrida. *G = (V, E, C)*, gráfica formada por *V*, conjunto de nodos, *E*, conjunto de arcos y *C*, función de costo.
Elaboración propia

CAPÍTULO 4.
APLICACIONES

Como se menciona en el capítulo 1, parte de la importancia del problema de la ruta más corta radica en su aplicación como subrutina para problemas complejos. En este capítulo se muestran dos aplicaciones, la primera consiste en la utilización del método símplex, para la cual se requiere hacer una modificación en el planteamiento como un problema de flujo a costo mínimo.

En la segunda aplicación se explica cómo en un problema NP-completo se utiliza el problema de la ruta más corta como subrutina con la finalidad de obtener una buena solución para dicho problema.

4.1. MÉTODO SÍMPLEX

El método símplex para redes detecta la presencia de ciclos negativos, en el caso de que la red no tenga ciclos negativos, entonces es posible determinar la distancia de la ruta más corta desde el nodo fuente *s* a todo nodo de la red *i*. Los pasos a seguir son los mismos que en el método símplex: encontrar la variable básica entrante, determinar la variable básica que sale y obtener la nueva solución básica factible [Ahuja, Magnanti y Orlin, 1993].

En el capítulo 1, (punto 1.4.1.), puede verse como el problema de la ruta más corta se plantea como un problema de flujo, esto es, sea $G = (V, E)$ una red con n nodos, si se considera un nodo, s como el nodo raíz, y un suministro de $n - 1$ en ese nodo, es decir $b_s = n - 1$, y una demanda de una unidad en cada

uno de los demás nodos, $b_1 = \ldots = b_{n-1} = -1$, de tal manera que el suministro y la demanda sean iguales. Sea el costo c_{ij} del arco (i, j) dado por su longitud, entonces se resuelve este problema de flujo a costo mínimo utilizando el algoritmo símplex para redes. La longitud de la ruta es la suma de las distancias de los arcos asociados y, la ruta más corta desde el nodo *s* a todo nodo *i* se obtiene siguiendo los arcos del árbol de expansión desde *s* a *i*.

El concepto básico de este método consiste en encontrar un árbol de expansión de solución factible y en cada iteración se introduce un nuevo arco que no pertenece al árbol en el lugar de otro que sí pertenece, de esta manera se va mejorando la estructura hasta encontrar el árbol óptimo. Este proceso recibe el nombre de pivoteo.

La diferencia esencial entre la aplicación del método símplex para redes a un problema de flujo a costo mínimo y un problema de ruta más corta consiste en, la selección del arco que se introduce en el árbol de expansión, es decir, al introducir un nuevo arco se forma un ciclo, sin embargo, como no existen cotas superiores de flujo, no es necesario determinar el arco que sale por el valor del flujo, en vez de esto, el arco que sale es el arco predecesor incidente en dicho nodo, debido a que no hay necesidad de mantener el flujo de los arcos, lo cual facilita la aplicación del algoritmo.

Algoritmo

Datos de entrada: una gráfica G = (V, E, c)

Paso 1 Iniciación

Encontrar un árbol de expansión que sea solución factible de raíz s.
Sea B la matriz de incidencia del árbol base.
Calcular los flujos básicos x_{ij} y las variables duales w asociadas con el árbol base.

Paso 2 Principal

Calcular $\delta_{ij} = z_{ij} - c_{ij} = w_i - w_j - c_{ij}$, tal que (i, j) es no básico

a) Si $z_{ij} - c_{ij} \leq 0$, terminar. La solución actual es óptima

b) Si $z_{ij} - c_{ij} > 0$, pivotear. Introducir el arco (i, j) en el árbol base y eliminar el arco predecesor incidente en el nodo j.

   Actualizar B de acuerdo al nuevo árbol base y calcular las variables duales w asociadas con el nuevo árbol.

Repetir el paso 2.

Reglas para pivotear

Existen algunas reglas para pivotear, es decir, para elegir el arco que entra en el árbol, las cuales mejoran el comportamiento de los algoritmos, como por ejemplo:

1. Primer arco.

   Si se elige el arco *(i, j)* se actualiza la distancia del nodo *j*, además se actualizan todas las distancias de los nodos descendientes del nodo *j*.

2. Regla de Dantzig.

   Se elige el arco con la máxima violación, $\delta_{ij} = \max \{ z_{ij} - c_{ij} = w_i - w_j - c_{ij} : (i, j) \text{ es no básico}\}$.

Si se desean conocer otras reglas puede consultarse bibliografía especializada [Ahuja, Magnanti y Orlin, 1993; Bazaara, Jarvis y Sherali, 1990] para su aplicación al problema de flujo a costo mínimo.

Tiempo de corrida

El algoritmo símplex para redes corre en tiempo pseudopolinomial, sin embargo, dependiendo de la forma de seleccionar los arcos que entran en el árbol de expansión, el tiempo de corrida mejora a tiempos polinomiales.

Ejemplo

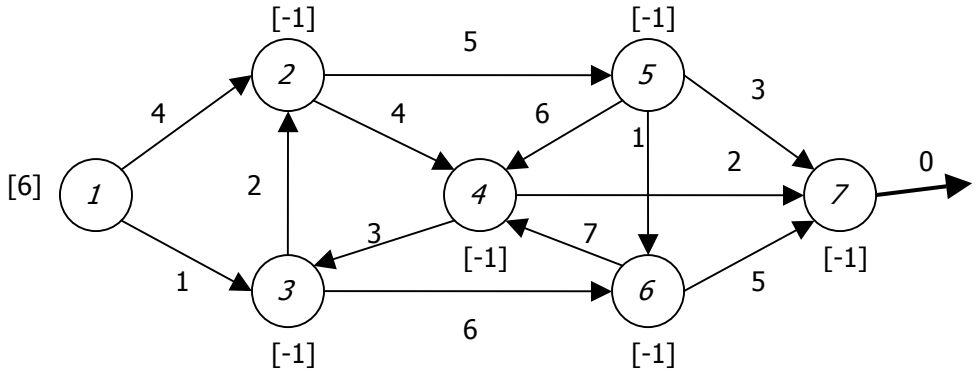


Figura 4.1. Ejemplo de Método Símplex

Iteración 1

Paso 1

Árbol de expansión, solución factible de raíz 1

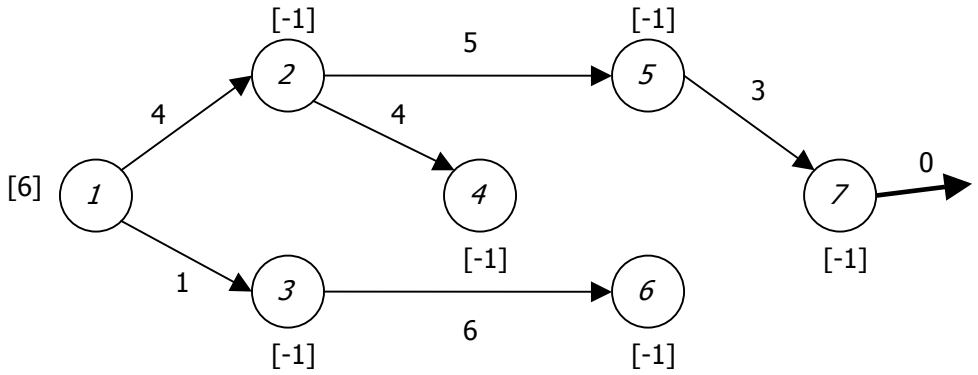


Figura 4.2. Árbol inicial del ejemplo de Método Símplex

Matriz de incidencia (nodos-arcos) del árbol inicial

$$B_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,2) & (1,3) & (2,4) & (2,5) & (3,6) & (5,7) & (7,?) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

Cálculo de flujos básicos

$[B_1]\ [x_{12}, x_{13}, x_{24}, x_{25}, x_{36}, x_{57}, x_7] = [6, -1, -1, -1, -1, -1, -1]$

$x_{12} + x_{13} = 6$ $\Rightarrow$ $x_{12} = 4$
$- x_{12} + x_{24} + x_{25} = -1$ $x_{13} = 2$
$- x_{13} + x_{36} = -1$ $x_{24} = 1$
$- x_{24} = -1$ $x_{25} = 2$
$- x_{25} + x_{57} = -1$ $x_{36} = 1$
$- x_{36} = -1$ $x_{57} = 1$
$- x_{57} + x_7 = -1$ $x_7 = 0$

Cálculo de variables duales

$[w_1, w_2, w_3, w_4, w_5, w_6, w_7]\ [B_1] = [4, 1, 4, 5, 6, 3, 0]$

$w_1 - w_2 = 4$ $\Rightarrow$ $w_1 = 12$
$w_1 - w_3 = 1$ $w_2 = 8$
$w_2 - w_4 = 4$ $w_3 = 11$
$w_2 - w_5 = 5$ $w_4 = 4$
$w_3 - w_6 = 6$ $w_5 = 3$
$w_5 - w_7 = 3$ $w_6 = 5$
$w_7 = 0$ $w_7 = 0$

Paso 2

Se calcula δ_{ij} para los arcos no básicos:

$\delta_{32} = w_3 - w_2 - c_{32} = 11 - 8 - 2 = 1$
$\delta_{43} = w_4 - w_3 - c_{43} = 4 - 11 - 3 = -10$
$\delta_{47} = w_4 - w_7 - c_{47} = 4 - 0 - 2 = 2$
$\delta_{54} = w_5 - w_4 - c_{54} = 3 - 4 - 6 = -7$
$\delta_{56} = w_5 - w_6 - c_{56} = 3 - 5 - 1 = -3$
$\delta_{64} = w_6 - w_4 - c_{64} = 5 - 4 - 7 = -6$
$\delta_{67} = w_6 - w_7 - c_{67} = 5 - 0 - 5 = 0$

Paso 2 b)

δ_{47} es el valor con la máxima violación, entonces el arco (4, 7) entra a la base y como el arco (5, 7) es el predecesor incidente en el nodo 7, sale de la base.

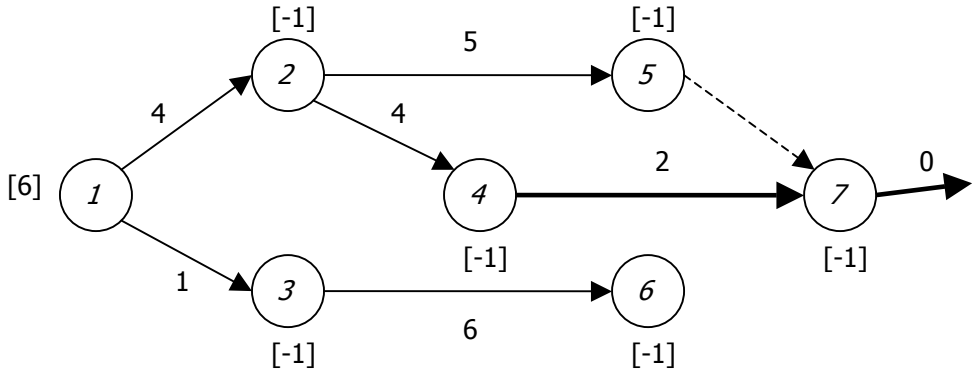


Figura 4.3. Segundo árbol del ejemplo de Método Símplex

Iteración 2

Matriz de incidencia (nodos-arcos) del segundo árbol

$$B_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,2) & (1,3) & (2,4) & (2,5) & (3,6) & (4,7) & (7,?) \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

Cálculo de variables duales

$[w_1, w_2, w_3, w_4, w_5, w_6, w_7]\ [B_2] = [4, 1, 4, 5, 6, 2, 0]$

$w_1 - w_2 = 4$
$w_1 - w_3 = 1$
$w_2 - w_4 = 4$
$w_2 - w_5 = 5$
$w_3 - w_6 = 6$
$w_4 - w_7 = 2$
$w_7 = 0$

$\Rightarrow$

$w_1 = 10$
$w_2 = 6$
$w_3 = 9$
$w_4 = 2$
$w_5 = 1$
$w_6 = 3$
$w_7 = 0$

Paso 2

Se calcula δ_{ij} para los arcos no básicos:

$\delta_{32} = w_3 - w_2 - c_{32} = 9 - 6 - 2 = 1$
$\delta_{43} = w_4 - w_3 - c_{43} = 2 - 9 - 3 = -10$
$\delta_{54} = w_5 - w_4 - c_{54} = 1 - 2 - 6 = -7$
$\delta_{56} = w_5 - w_6 - c_{56} = 1 - 3 - 1 = -3$
$\delta_{57} = w_5 - w_7 - c_{57} = 1 - 0 - 3 = -2$
$\delta_{64} = w_6 - w_4 - c_{64} = 3 - 2 - 7 = -6$

$\delta_{67} = w_6 - w_7 - c_{67} = 3 - 0 - 5 = -2$

Paso 2 b)

δ_{32} es el valor con la máxima violación, entonces el arco (3, 2) entra a la base y como el arco (1, 2) es el predecesor incidente en el nodo 2, sale de la base.

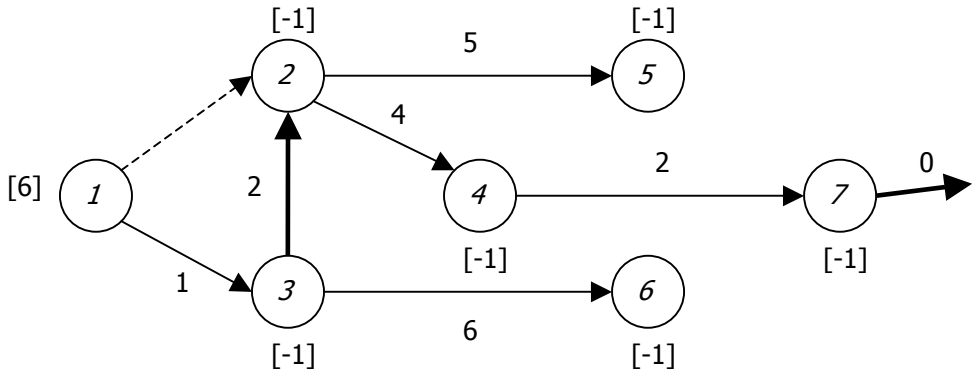


Figura 4.4. Tercer árbol del ejemplo de Método Símplex

Iteración 3

Matriz de incidencia (nodos-arcos) del tercer árbol

$$B_3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} (1,3) & (2,4) & (2,5) & (3,2) & (3,6) & (4,7) & (7,?) \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}$$

Cálculo de variables duales

$[w_1, w_2, w_3, w_4, w_5, w_6, w_7]\ [B_3] = [1, 4, 5, 2, 6, 2, 0]$

$w_1 - w_3 = 1$
$w_2 - w_4 = 4$
$w_2 - w_5 = 5$
$w_3 - w_2 = 2$
$w_3 - w_6 = 6$
$w_4 - w_7 = 2$
$w_7 = 0$

$\Rightarrow$

$w_1 = 9$
$w_2 = 6$
$w_3 = 8$
$w_4 = 2$
$w_5 = 1$
$w_6 = 2$
$w_7 = 0$

Paso 2

Se calcula δ_{ij} para los arcos no básicos:

$\delta_{12} = w_1 - w_2 - c_{12} = 9 - 6 - 4 = -1$
$\delta_{43} = w_4 - w_3 - c_{43} = 2 - 8 - 3 = -9$
$\delta_{54} = w_5 - w_4 - c_{54} = 1 - 2 - 6 = -7$
$\delta_{56} = w_5 - w_6 - c_{56} = 1 - 2 - 1 = -2$
$\delta_{57} = w_5 - w_7 - c_{57} = 1 - 0 - 3 = -2$
$\delta_{64} = w_6 - w_4 - c_{64} = 2 - 2 - 7 = -7$
$\delta_{67} = w_6 - w_7 - c_{67} = 2 - 0 - 5 = -3$

Paso 2 a)

Como los valores de $\delta_{ij} \leq 0$, se ha encontrado el árbol óptimo

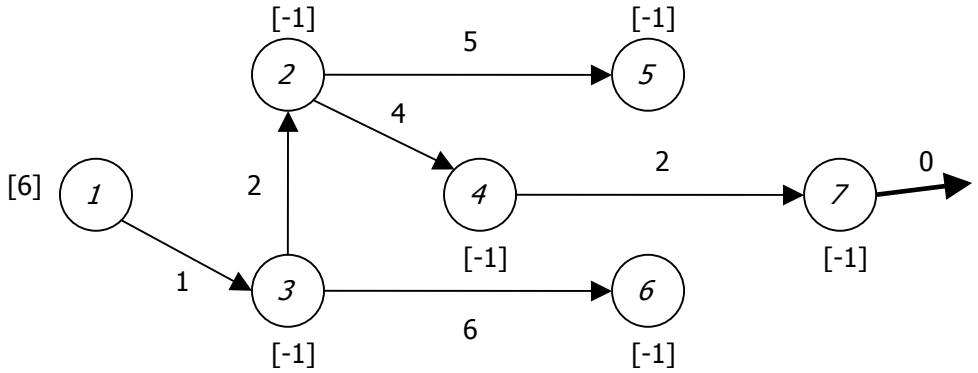


Figura 4.5. Árbol óptimo del ejemplo de Método Símplex

Observaciones

⇨ El algoritmo símplex para redes es una adaptación del método símplex y en la aplicación no hay necesidad de la tabla símplex.

4.2. EL PROBLEMA PRIZE-COLLECTING STEINER TREE (PCST)

Para explicar en qué consiste el problema del árbol de Steiner, se tiene el siguiente ejemplo:

Si se considera una compañía de paquetería que desea enviar los paquetes en el transcurso de la noche entre varios pares de ciudades, la compañía puede construir rutas que conecten entre ciudades, de tal manera que mediante el

horario de los mensajeros la compañía tenga la posibilidad de enviar los paquetes durante la noche entre los pares de las ciudades conectadas. Entonces, se asume que el costo de conexión de la ciudad *i* a la ciudad *j* es c_{ij} y los costos son simétricos. Además la compañía tiene la opción de contratar los servicios de otras compañías para los envíos a algunos pares de ciudades *(i, j)* con un costo de α_{ij} de tal forma que no tiene que preocuparse por los envíos de la otra compañía entre las ciudades *i* y *j* durante la noche. Lo que se busca es construir algunas rutas y contratar otra compañía para las demás rutas de manera que la compañía de mensajería haga las entregas durante la noche al mínimo costo.

Con base en lo anterior, sea $G = (V, E)$ una gráfica con costos en sus arcos c_e, en los arcos $e \in E$, y con un subconjunto de los nodos T llamados nodos terminales. El objetivo es encontrar un árbol de peso mínimo de G que se expande en todos los nodos en T. El sub-árbol puede o no incluir algunos de los otros nodos $S = V \mid T$, a los cuales se les conoce como nodos Steiner. En la siguiente figura se muestra un ejemplo de un árbol reducido de Steiner, donde los nodos terminales son cuadrados y los nodos redondos son los llamados nodos Steiner. Las conexiones corresponden a las rutas más cortas de la gráfica original.

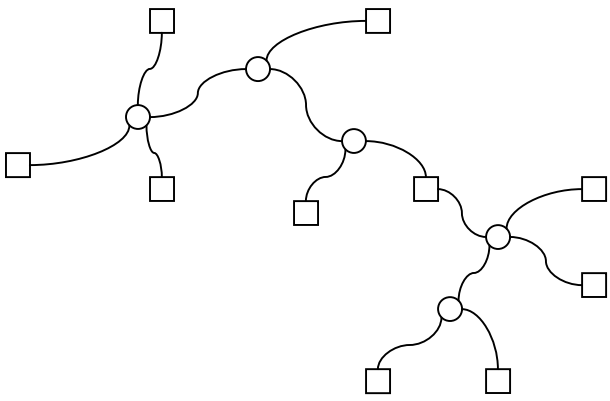


Figura 4.6. Árbol reducido de Steiner.

Este problema tiene varias aplicaciones en el diseño de redes de comunicaciones o de distribución de servicios.

Para solucionar este problema que es NP-completo y por tanto se entiende que no existe un algoritmo exacto, entonces un paso importante es encontrar una buena representación en programación lineal del problema. Empezando con una formulación en programación entera o programación entera binaria, la meta es añadir nuevas restricciones válidas o nuevas variables al modelo talque el resultado de programación lineal provea una frontera ajustada en el valor de la solución óptima.

Cabe mencionar que se han desarrollado diversos algoritmos para obtener una aproximación: estudios poliédricos, relajación Lagrangeana, planos de corte, algoritmos basados en búsqueda local.

4.2.1. PLANTEAMIENTO GENERAL DEL PROBLEMA PCST

Dada una gráfica $G = (V, E)$ con costos en los arcos w_e para $e \in E$, se designa un nodo raíz r, el cual será el único nodo terminal (i.e. $T = \{r\}$). Para todos los demás nodos $j \in V \mid \{r\}$ se obtiene una utilidad d_j si el árbol de Steiner contiene al nodo j [Ball, Magnanti, Monma y Nemhauser, 1995].
Sea:

$$\begin{cases} z_j = 1 & \text{si el nodo } j \text{ está en el árbol de Steiner} \\ z_j = 0 & \text{en otro caso} \end{cases}$$

$$\begin{cases} x_e = 1 & \text{si el arco } e \text{ está en el árbol de Steiner} \\ x_e = 0 & \text{en otro caso} \end{cases}$$

entonces,

$$min \sum_{e \in E} w_e x_e - \sum_{i \in V} d_i z_i$$

sujeto a:

$$\sum_{e \in E(U)} x_e \leq \sum_{i \in U/\{k\}} z_i \qquad \forall U \subset V \text{ y } \forall k \in U \qquad (1)$$

$$\sum_{e \in E} x_e = \sum_{i \in V/\{r\}} z_i \qquad (2)$$

$z_r = 1$

$0 \leq x_e \leq 1,\ 0 \leq z_i \leq 1$

$x, z \in Z$

donde:

(1) este conjunto de restricciones implica que la solución no contiene ciclos en la subgráfica de G determinada por los nodos seleccionados, es decir, aquellos con zi = 1

(2) este conjunto de restricciones asegura que la solución se expande al conjunto de nodos seleccionados y entonces se define un árbol de Steiner

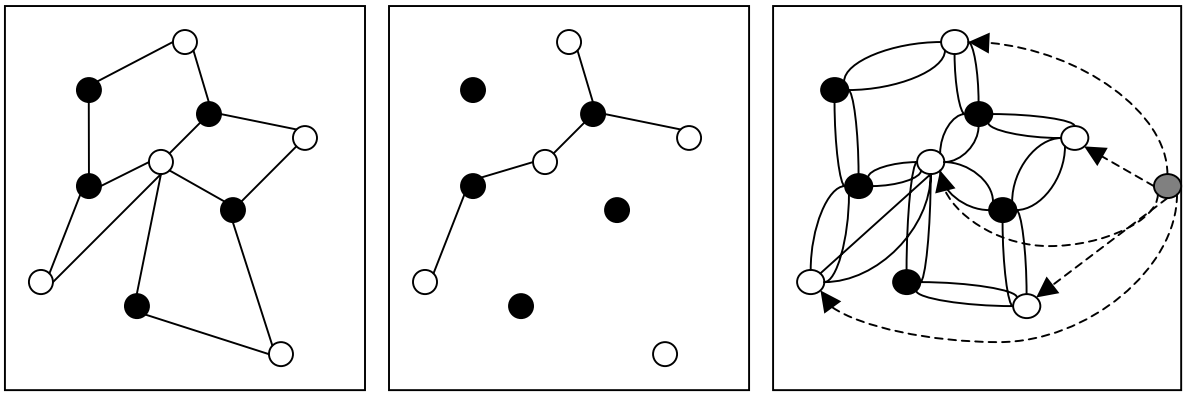
4.2.2. DEFINICIÓN DEL PROBLEMA PCST [Klau, Ljubic, Moser, Mutzel, Neuner, Pferschy, Raidl, Weiskircher, 2004]

Sea una gráfica conectada no dirigida $G = (V, E, c, p)$, con $p : V \rightarrow \boldsymbol{R} \geq 0$ una función de utilidades en los nodos, y $c : E \rightarrow \boldsymbol{R} \geq 0$ una función de costos en los arcos. El problema consiste en encontrar una subgráfica conectada $T = (V_T, E_T)$ de G, $V_T \subseteq V$, $E_T \subseteq E$, que minimice:

$$c(T) = \sum_{v \notin V_T} p(v) + \sum_{e \in E_T} c(e)$$

Cuando se encuentra una subgráfica T que maximiza la suma de las utilidades de los nodos en T menos el costo de los arcos en T, cada solución óptima es una solución óptima para el problema de minimización y viceversa:

$$utilidades(T) = \sum_{v \in V_T} p(v) - \sum_{e \in E_T} c(e)$$



(a) (b) (c)

Figura 4.7. (a) Ejemplo de una instancia del problema PCST, donde los nodos en blanco y los negros representan nodos positivos y negativos respectivamente; (b) se muestra una solución factible; (c) se muestra la transformación en un problema de arborescencia de Steiner.

4.2.3. ESTRUCTURA DEL ALGORITMO MEMÉTICO CON PROGRAMACIÓN ENTERA

Este algoritmo se desarrolló para resolver el problema PCST [Klau, Ljubic, Moser, Mutzel, Neuner, Pferschy, Raidl, Weiskircher, 2004], obteniendo un acercamiento significativamente más rápido que trabajos realizados anteriormente por otros investigadores. Consiste de tres partes:

1. Una extensa fase de preprocessing, la cual reduce la gráfica sin cambiar la estructura de la solución óptima.
La gráfica original $G = (V, E, c, p)$ se reduce a $G' = (V', E', c', p')$ mediante la aplicación de los siguientes pasos:

⇨ **Prueba de costo mínimo**. Sea d_{ij} la distancia más corta entre los nodos i y j, desde V (se consideran solo los costos de los arcos). Si existe $e = (i, j)$ talque $d_{ij} < c_{ij}$ entonces el arco e se elimina.

⇨ **Prueba de grado *l***. Sea un nodo v con grado $l \geq 3$, conectado a los nodos $Adj(v) = \{v_1, v_2, \ldots, v_l\}$, entonces para cualquier subconjunto $K \subset V$, denotar con $MST_d(K)$ al árbol de expansión mínima de K con distancias d_{ij}.

$$MST_d(k) \leq \sum_{w \in K} c_{vw}, \qquad \forall K \subseteq Adj(v), \qquad |K| \geq 3$$

entonces el grado de v en una solución óptima debe ser cero o dos. Por lo tanto, es posible eliminar v reemplazando cada par de arcos (v_i, v), (v, v_j) con (v_i, v_j) tanto si se añade un nuevo arco $e = (v_i, v_j)$ de costo $c_e = c_{vi,v} + c_{v,vj} - p_v$ o en caso de que exista e, entonces se define $c_e = min\{c_e, c_{vi,v} + c_{v,vj} - p_v\}$.

⇨ **Prueba de adyacencia mínima (o reducción)**. Si existen nodos adyacentes $i, j \in \boldsymbol{R}$ tales que:

$$\min\{p_i, p_j\} - c_{ij} > 0 \qquad \text{y} \quad c_{ij} = \min_{it \in E} c_{it}$$

entonces i y j se pueden unir en un nodo de peso $p_i + p_j - c_{ij}$.

Las pruebas anteriores se aplican iterativamente, y el número de iteraciones está acotado por el número de arcos en G. La complejidad del preprocessing es $O(|E|^2|V| + |E||V|^2 \log |V|)$ y la complejidad del espacio no excede $O(|E|^2)$.

2. La parte central considera el algoritmo memético (AM) basado en un algoritmo de estado estable evolutivo combinado con un algoritmo exacto para el problema de árboles.

En cada iteración se hace una selección con reemplazo en orden para seleccionar dos soluciones paternas. La nueva solución es creada por la combinación de sus padres, haciendo una mutación con probabilidad $p_{mut} \in [0, 1]$, realizando una poda del árbol obtenido óptimamente.

✧ **Mejora local**. Dado un árbol $T' = (V_{T'}, E_{T'}, p', c')$ creado por el AM, un subárbol de T'es óptimo si no existe otro subárbol de T'con costos menores

que $c(T')$. Este algoritmo maximiza la suma de las utilidades de los nodos en T' menos la suma de los costos de los arcos en T'.

1. Sea $l_v = p'_v$, para toda $v \in V_{T'}$;
2. Para todas las salidas $u \in V_{T'}$: (a) si $c'_{uv} \leq l_u$, reducir u y v en un nodo y hacer $l_v = l_v + l_u - c'_{uv}$; (b) eliminar u;
3. Regresar al paso 2. hasta tener un solo nodo.

El algoritmo corre en un tiempo de $O(|V'|)$.

✧ **Agrupamiento**. Para cada nodo positivo z $\in$ R', se define un conjunto N(z):

$$N(z) := \{v \in V' \mid R' \mid \forall c \in R' : d'_{uz} \leq d'_{vc}\} \cup \{z\}$$

donde d'_{vz} es la ruta de longitud más corta entre v y z. Esto es, cada nodo no positivo v es asignado al grupo de sus nodos más cercanos positivos $z = base(v)$.

El algoritmo de agrupación corre en un tiempo de $O(|V'|log|V'| + |E'|)$.

✧ **Codificación del conjunto de arcos**. Las soluciones de los arcos se almacenan en tablas hash, requiriendo espacio de $O(|V'|)$.

✧ **Inicialización**. Dada una gráfica de entrada $G' = (V', E', c', p')$ y el conjunto de nodos positivos R', la distancia de la red $G_D(R', E_D, c_D)$ es una gráfica no dirigida completa cuyos costos de los arcos $c_D(u, v)$ están dados por la ruta de longitud más corta entre u y v en G'. Para generar las soluciones iniciales se utiliza la siguiente modificación de la heurística de la distancia de la red para el problema del árbol de Steiner:

1. Seleccionar aleatoriamente un subconjunto $V'_{init} \subset R'$ de tamaño $\lceil p_{init} |R'| \rceil$, $p_{init} \in (0,1)$;

2. Construir el árbol de expansión mínima (MST) T'_{init} en la subgráfica de G_D inducida por V'_{init};
3. Reemplazar dada arco de T'_{init} por su correspondiente ruta más corta en G' para obtener $G'_r = (V'_r, E'_r)$;
4. Encontrar el MST T'_r en la subgráfica de G' inducida por V'_r;
5. Aplicar el algoritmo exacto para árboles para resolver T'_r óptimamente.

✧ **Recombinación**. El operador de recombinación busca adoptar las propiedades de la estructura de dos soluciones padres. Si las dos soluciones que se combinan se conjuntan en un nodo, se construye un árbol de expansión sobre la unión del conjunto de sus arcos. Debido a la naturaleza determinística de la subrutina de mejora local, se construye un árbol de expansión aleatorio de la unión de los arcos padres para evitar convergencia prematura.

Cuando las soluciones padres son disjuntas, se elige aleatoriamente un nodo fuera de cada solución, entonces se busca la ruta más corta entre esos dos nodos y se añade para cada nodo v a lo largo de la ruta todos los arcos que corresponden al grupo *N(base (v))*. Finalmente, se construye un árbol de expansión aleatorio sobre todos los arcos y se aplica la mejora local.

✧ **Mutación**. El operador de mutación realiza pequeños cambios en la solución actual llevando a cabo la conexión de un conjunto a la solución. El algoritmo elige aleatoriamente un nodo orilla v, es decir, un nodo adyacente a al menos un nodo fuera de la solución actual. Se incorporan los nodos del grupo *N(base(v'))* en la solución y se busca un grupo vecino que preferiblemente no tenga al nodo base v' como elemento de la solución actual; los nodos de *N(base(v'))* se añaden a la solución. Finalmente, se construye un árbol de expansión mínima y se aplica mejora local.

El tiempo de corrida de los operadores de inicialización y variación es $O(|E'|\ \alpha(|E'|, |V'|))$.

3. La población de solución del algoritmo memético provee un excelente estado inicial para la post-optimización mediante la solución relajada del modelo de programación entera, construido en base al modelo para encontrar la arborescencia mínima de Steiner en una gráfica dirigida.

La gráfica $G_{IP} = (V_{IP}, E_{IP}, c', p')$ que resulta de la aplicación del AM se transforma en la gráfica dirigida $G'_{IP} = (V_{IP} \cup \{r\}, A_{IP}, c'')$.

Además de los nodos de la gráfica de entrada G_{IP}, el conjunto de nodos de la gráfica transformada contiene una base artificial r. El conjunto de arcos A_{IP} contiene dos arcos dirigidos (v, w) y (w, v) para cada arco $(v, w) \in E_{IP}$ además un conjunto de arcos desde la base r a los nodos positivos $\{v \in V_{IP} \mid p_v > 0\}$. El vector de costos c'' se define como:

$$c''_{vw} = c'_{vw} - p'_w \ \forall (v, w) \in A_{IP},\ v \neq r \text{ y } c''_{rv} = -p'_v \quad \forall (r, v) \in A_{IP}$$

Una subgráfica T_{IP} de G'_{IP} que forma un árbol dirigido con base en r se conoce como una *arborescencia de Steiner*. Una subgráfica correspondiente a una solución del problema PCST si tiene el grado de $r = 1$ en G'_{IP} es una arborescencia factible. En particular, una arborescencia factible con mínimo costo total de los arcos corresponde a un óptimo PCST.

El problema consiste en encontrar una arborescencia mínima de Steiner T_{IP} mediante un modelo de programación entera, entonces el vector x se define como:

$$x_{vw} = \begin{cases} 1 & (v,w) \in T_{IP} \\ 0 & e.o.c. \end{cases} \quad \forall (v,w) \in A_{IP}, \quad x_{UV} = \begin{cases} 1 & v \notin T_{IP} \\ 0 & e.o.c. \end{cases} \quad \forall v \in V_{IP} \setminus \{r\}$$

El modelo es:

$$\min \sum_{a \in A_{IP}} c''_a x_a \qquad (1)$$

sujeto a :

$$x(\delta^-(\{v\})) + x_{vv} = 1 \qquad \forall v \in V_{IP} - \{r\} \qquad (2)$$

$$x(\delta^-(S)) \geq 1 - x_{vv} \qquad v \in S, r \notin S, \forall S \subset V_{IP} \qquad (3)$$

$$\sum_{(r,v) \in A_{IP}} x_{rv} \leq 1 \qquad (4)$$

$$x_{vw}, x_{vv} \in \{0,1\} \qquad \forall (v,w) \in A_{IP}, \forall v \in V_{IP} \qquad (5)$$

donde $\delta^-(S) = \{(u,v) \in A_{IP} \mid u \notin S, v \in S\}$

Interpretación de las restricciones:

- (2) significa que cada nodo que forma parte la de solución debe tener al menos un arco de entrada.
- (3) para cada nodo *v* en la solución, debe existir una ruta directa de *r* a *v*.
- (4) considera que máximo se elige un arco que sale de la base artificial *r*.

El número de restricciones del tipo (3) es exponencial sin embargo, no es necesario insertar todas en el inicio sino de forma separada durante el proceso de optimización, esto es, únicamente se añaden las restricciones violadas por la solución actual del problema entero relajado. Dichas restricciones violadas se pueden encontrar eficientemente utilizando un algoritmo de flujo máximo en la gráfica con capacidades en los arcos obtenidas en la solución actual.

4.2.4. RESULTADOS COMPUTACIONALES

A continuación se presenta la tabla comparativa 4.1., de las aproximaciones realizadas a 60 instancias. El tamaño de la serie C es de 500 nodos y la serie D 1,000 nodos. La explicación de las columnas es la siguiente:

- Nombre de la instancia.

- Número de arcos *(|E|)*.

- El tamaño de la gráfica después de las reducciones de preprocessing *(|V'|, |E'|)* y el tiempo requerido *(t_p [s])*.

- El algoritmo memético se ejecutó 30 veces en cada instancia. El promedio de los costos *($c(T)_{avg}$)*, su desviación estándar *(σ©)*, el tiempo promedio de CPU *(t[s])*, el tiempo promedio de la evaluación de las soluciones hasta que se encontró la mejor *(evals)*, el porcentaje de instancias para las cuales se pudo obtener soluciones óptimas *(sr[%])*.

- Comparación de la aproximación realizada por la combinación del AM y programación entera, donde una corrida del AM (con un valor fijo) fue post-optimizado con el método de programación entera. El valor de la solución obtenida *(c(T))* y el tiempo en segundos del CPU de la post-optimización *(t[s])*.

- Comparación de los resultados con los obtenidos anteriormente (CRR) que utilizaron búsqueda local multi-inicial con perturbaciones y búsqueda de vecindades. El valor de la solución obtenida *(c(T))* y el tiempo total de corrida en segundos *(t[s])*. Sin embargo, para la comparación de los tiempos de corrida se deben dividir los valores de CRR entre 10.

- Finalmente, los valores de las soluciones enteras obtenidas *(OPT)*. Si no se obtuvo una solución entera, o si el algoritmo basado en programación entera terminó anormalmente (por el consumo de la memoria), los valores que se muestran son los obtenidos por Lucena ($^{+}$) o por Resende (*). La última columna es el tiempo adicional necesario de CPU para calcular una solución óptima probable *(t[s])*.

	Orig.	Preprocessing			MA					MA+ILP		CRR		OPT-ILP	
Instance	$\|E\|$	$\|V'\|$	$\|E'\|$	t_p [s]	$c(T)_{avg}$	$\sigma(c)$	t [s]	*evals*	*sr* [%]	$c(T)$	t [s]	$c(T)$	t [s]	*OPT*	t [s]
C11-A	2500	489	2143	9.4	18.0	0.0	6.1	500	100.0	18	0.4	18	128	18	0.2
C11-B	2500	489	2143	9.5	32.0	0.0	9.1	1103	100.0	32	0.4	32	140	32	4.7
C12-A	2500	484	2186	6.8	38.7	0.5	9.0	2456	33.3	38	0.4	38	162	38	0.3
C12-B	2500	484	2186	6.8	46.0	0.0	8.7	590	100.0	46	0.5	46	156	46	0.8
C13-A	2500	472	2113	9.8	237.0	0.2	17.9	5326	0.0	**236**	0.6	237	1050	236	0.5
C13-B	2500	471	2112	9.8	258.5	0.7	35.9	15455	60.0	258	18.5	258	733	258	52.5
C14-A	2500	466	2081	7.5	293.0	0.0	21.0	3163	100.0	293	1.7	293	829	293	0.4
C14-B	2500	459	2048	7.5	318.6	0.5	29.8	9211	43.3	318	1.0	318	766	318	0.4
C15-A	2500	406	1871	6.5	502.2	0.8	45.4	14727	20.0	501	4.7	501	957	501	0.5
C15-B	2500	370	1753	6.0	551.8	0.9	45.7	15607	46.7	551	0.8	551	837	551	0.4
C16-A	12500	500	4740	2.4	12.0	0.0	10.6	500	0.0	12	1.9	**11**	1920	11	0.9
C16-B	12500	500	4740	2.4	12.0	0.0	11.5	503	0.0	12	3.5	**11**	1758	11	13.8
C17-A	12500	498	4694	2.4	19.0	0.0	11.2	620	0.0	19	2.9	**18**	549	18	1.9
C17-B	12500	498	4694	2.3	18.2	0.4	12.7	1951	76.7	18	2.1	18	434	18	1.4
C18-A	12500	469	4569	2.6	112.4	0.7	24.1	7446	6.7	112	2.1	**111**	3990	111^{+}	—
C18-B	12500	465	4538	2.9	115.0	0.7	26.2	8361	6.7	116	219.5	**113**	3262	113^{+}	—
C19-A	12500	430	3982	2.9	146.2	0.4	17.9	5402	80.0	146	2.3	146	3928	146	0.6
C19-B	12500	416	3867	2.8	149.0	0.6	15.8	4035	0.0	147	3.0	**146**	3390	146	0.6
C20-A	12500	241	1222	6.1	266.0	0.0	7.3	598	100.0	266	0.2	266	4311	266	0.0
C20-B	12500	133	563	5.0	267.0	0.0	5.2	500	100.0	267	0.1	267	3800	267	0.1
D1-A	1250	231	440	4.9	18.0	0.0	3.1	500	100.0	18	0.0	18	6	18	0.0
D1-B	1250	233	443	4.9	106.0	0.0	3.8	1950	100.0	106	0.1	106	257	106	0.0
D2-A	1250	257	481	4.9	50.0	0.0	3.5	500	100.0	50	0.1	50	7	50	0.0
D2-B	1250	264	488	4.9	218.3	1.0	7.3	4157	93.3	**218**	0.1	228	486	218	0.0
D3-A	1250	301	529	5.5	807.0	0.0	7.4	500	100.0	807	0.1	807	734	807	0.1
D3-B	1250	372	606	6.3	1516.2	1.3	51.0	15976	0.0	**1509**	0.6	1510	2184	1509	0.3
D4-A	1250	311	541	5.6	1203.8	0.4	10.4	974	16.7	1203	0.3	1203	1263	1203	0.3
D4-B	1250	387	621	7.2	1885.2	2.0	49.6	9671	0.0	1881	11.0	1881	2233	1881	1.3
D5-A	1250	348	588	7.6	2157.0	0.0	29.1	1963	100.0	2157	3.1	2157	3352	2157	8.8
D5-B	1250	411	649	11.5	3137.7	0.9	65.1	7316	0.0	3135	2.2	3135	2555	3135	0.4
D6-A	2000	740	1707	14.4	18.0	0.0	7.7	500	100.0	18	0.3	18	20	18	0.1
D6-B	2000	741	1708	14.7	72.6	0.8	10.5	1192	0.0	71	0.5	**70**	702	67	0.9
D7-A	2000	734	1705	11.3	50.0	0.0	8.2	500	100.0	50	0.3	50	195	50	0.1
D7-B	2000	736	1707	11.4	105.0	0.0	9.5	520	0.0	105	0.3	105	711	103	0.1
D8-A	2000	764	1738	11.7	755.5	0.5	19.1	2788	50.0	755	15.6	755	1727	755	41.8
D8-B	2000	778	1757	12.3	1045.7	3.9	123.8	36313	0.0	**1037**	1013.4	1038	3175	1036	2.8
D9-A	2000	752	1716	17.9	1074.7	1.0	52.1	13718	0.0	1075	354.5	**1072**	4109	1070^{+}	—
D9-B	2000	761	1724	20.9	1436.4	3.0	151.2	31361	0.0	1420	1769.6	1420	2754	1420	4539.6
D10-A	2000	694	1661	14.6	1674.4	1.4	122.2	21289	0.0	1671	9.0	1671	4193	1671	2.2
D10-B	2000	629	1586	18.5	2089.8	2.1	107.3	14598	0.0	2079	44.1	2079	2644	2079	4.1
D11-A	5000	986	4658	27.7	18.0	0.0	15.4	500	100.0	18	1.8	18	540	18	0.5
D11-B	5000	986	4658	23.6	29.0	0.0	17.4	814	100.0	**29**	2.0	30	1280	29	4.7
D12-A	5000	991	4639	23.1	42.0	0.0	13.9	500	100.0	42	2.3	42	844	42	13.2
D12-B	5000	991	4639	22.3	42.0	0.0	15.1	620	100.0	42	2.3	42	687	42	0.4
D13-A	5000	966	4572	27.7	446.7	0.5	58.7	14308	0.0	445	1126.4	445	5047	445	5643.4
D13-B	5000	961	4566	28.0	491.7	1.9	97.2	22843	0.0	486	15.9	486	4288	486	2.6
D14-A	5000	946	4500	35.5	605.6	1.2	102.3	21486	0.0	602	34.2	602	6388	602^{*}	—
D14-B	5000	931	4469	37.2	674.2	1.4	102.8	17746	0.0	665	3409.5	665	6178	664^{*}	—
D15-A	5000	832	4175	47.1	1048.7	1.3	145.7	18343	0.0	1042	185.8	1042	7840	1042	12.8
D15-B	5000	747	3896	49.2	1114.7	0.8	95.6	11026	0.0	1108	117.0	1108	5220	1108	4.8
D16-A	25000	1000	10595	10.8	14.0	0.0	23.1	500	0.0	14	8.9	**13**	1397	13	24.8
D16-B	25000	1000	10595	10.8	13.3	0.4	26.4	1313	73.3	13	9.3	13	1043	13	42.0
D17-A	25000	999	10534	10.8	23.0	0.0	24.8	1983	100.0	23	9.5	23	3506	23	167.1
D17-B	25000	999	10534	10.7	23.0	0.0	23.7	948	100.0	23	10.2	23	2089	23	60.1
D18-A	25000	944	9949	11.7	220.8	0.7	81.4	19864	0.0	218	197.0	218	30044	218^{+}	—
D18-B	25000	929	9816	12.0	230.2	1.3	98.7	25585	0.0	224	25.2	224	36643	223	34.9
D19-A	25000	897	9532	12.4	317.7	2.7	87.6	18480	0.0	308	151.9	308	40955	306	1446.5
D19-B	25000	862	9131	13.1	317.8	2.2	81.9	17912	0.0	311	13.6	311	38600	310	62.8
D20-A	25000	488	2511	37.3	537.0	0.0	18.4	1036	0.0	536	1.0	536	28139	536	0.5
D20-B	25000	307	1383	32.9	537.0	0.0	12.7	1587	100.0	537	0.5	537	22104	537	0.1

Tabla 4.1. Resultados computacionales
Fuente: [Klau, Ljubic, Moser, Mutzel, Neuner, Pferschy, Raidl, Weiskircher, 2004]

Observaciones

- ⇨ La combinación de los métodos de programación lineal o programación lineal entera con algoritmos evolutivos pueden producir soluciones de alta calidad en tiempos computacionales breves incluso para algunos problemas de optimización duros.

- ⇨ La clasificación del problema es NP-completo, entonces se entiende que no existe un algoritmo exacto para su solución. En este aspecto, se puede advertir la cantidad de técnicas y procesos que se utilizan para buscar una solución.

- ⇨ Conocer una aplicación del problema de la ruta más corta, de manera concreta al observar en qué parte del proceso se aplica y tomando en cuenta las ventajas de los métodos de solución, como es el tiempo de corrida.

- ⇨ Conocer el tipo de técnicas utilizadas en el proceso de preprocessing para la reducción del problema original, es decir, se pueden aplicar diferentes pruebas para acotar el problema y el resultado es muy significativo.

- ⇨ La combinación de técnicas heurísticas con métodos de solución exactos. Esto es, entender como se alternan las soluciones parciales de ambos métodos para lograr una mejor aproximación a la solución óptima.

- ⇨ La importancia de la complejidad computacional. Los resultados que obtuvieron el grupo de investigadores fue muy similar a resultados ya publicados con anterioridad, sin embargo, lo que marcó una gran diferencia fue la reducción significativa de los tiempos de corrida.

CONCLUSIONES

El presente trabajo ha desarrollado de manera exhaustiva el Problema de la Ruta más Corta, particularmente, proporciona explicaciones de los aspectos fundamentales de la teoría, los diversos algoritmos e ilustra con algunas aplicaciones relevantes. Estas características, permiten que estudiantes o personas interesadas en este tema, les pueda ser de utilidad para ampliar y profundizar sus conocimientos y usarlos para abordar este tipo de problemas. Por ejemplo, la tesis hace hincapié que en el área de Redes, se debe:

- ✓ Primeramente, la importancia de conocer las aplicaciones prácticas, de esta manera saber si le es útil para el problema específico que deba resolver.

- ✓ Con la explicación de la teoría de la Complejidad Computacional, es posible saber previamente la eficiencia de los algoritmos, así, con sólo conocer el peor caso del tiempo de corrida se puede entender la eficiencia del mismo.

- ✓ Con la descripción detallada de cada uno de los métodos de solución puede entenderse cómo se utilizan, incluso tanto si se conocen las características de la red o no, saber qué tipo de algoritmo es más conveniente de aplicar y cuándo existe o no solución.

- ✓ Finalmente, las aplicaciones:

    - ⇨ Método Símplex, con una simple modificación es posible resolver el problema de flujo a costo mínimo, además con la no existencia de cotas superiores de flujo se requiere de una pequeña modificación en

el algoritmo, en el paso de pivoteo, haciendo de esta manera un algoritmo más sencillo de aplicar

- ⇨ Problema PCST, la aplicación como subrutina se menciona frecuentemente en clases y también en libros, sin embargo, no es común conocer cómo se realiza, es decir, cómo se trabaja con un problema NP-completo en qué momento se utiliza el algoritmo de ruta más corta para acotar el problema original. Así mismo, conocer el proceso completo con la cantidad de pasos y subrutinas implicadas en la búsqueda de una buena solución

Por estas razones, el desarrollo de este trabajo ha integrado los aspectos básicos del Problema de la Ruta más Corta, permitiendo su consulta como material de referencia. Es importante mencionar, que en un tema tan extenso como es este tipo de problemas, no se pueden exponer detalladamente todos sus componentes en una sola tesis por la misma naturaleza del trabajo. Por lo cual las personas interesadas en el tema tienen una gran cantidad de referencias de consulta, tanto de libros de redes como material publicado en Internet.

ANEXO 1
EJEMPLO DEL ALGORITMO DE DIJKSTRA UTILIZANDO LINGO

La utilización del software existe es una herramienta muy útil en la solución de problemas, por esta razón en este anexo se muestra el ejemplo del algoritmo de Dijkstra del capítulo 3, con la utilización de LINGO y con las modificaciones específicas para obtener la ruta más corta desde el nodo 1 (nodo fuente) a los nodos 7 y 8 cuando son considerados como nodos destino.

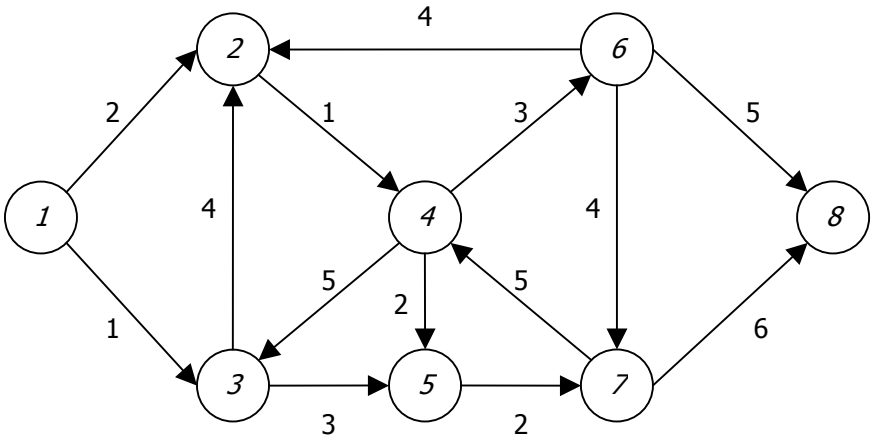


Figura A.1.1. Ejemplo del algoritmo de Dijkstra (capítulo 3)

Del nodo 1 al nodo 7:

min =
2*x12+x13+x24+4*x32+3*x35+5*x43+2*x45+3*x46+2*x57+4*x62+4*x67+
5*x68+5*x74+6*x78;

x12+x13=1;
x12+x32+x62-x24=0;
x13+x43-x32-x35=0;
x24+x74-x43-x45-x46=0;

```
x35+x45-x57=0;
x46-x62-x67-x68=0;
x57+x67-x74-x78=1;
x68+x78=0;


@bin(x12);
@bin(x13);
@bin(x24);
@bin(x32);
@bin(x35);
@bin(x43);
@bin(x45);
@bin(x46);
@bin(x57);
@bin(x62);
@bin(x67);
@bin(x68);
@bin(x74);
@bin(x78);
```

Global optimal solution found at step: 13
Objective value: 6.000000

Variable	Value
x12	0.0000000
x13	1.000000
x24	0.0000000
x32	0.0000000
x35	1.000000
x43	0.0000000
x45	0.0000000
x46	0.0000000
x57	1.000000
x62	0.0000000
x67	0.0000000
x68	0.0000000
x74	0.0000000
x78	0.0000000

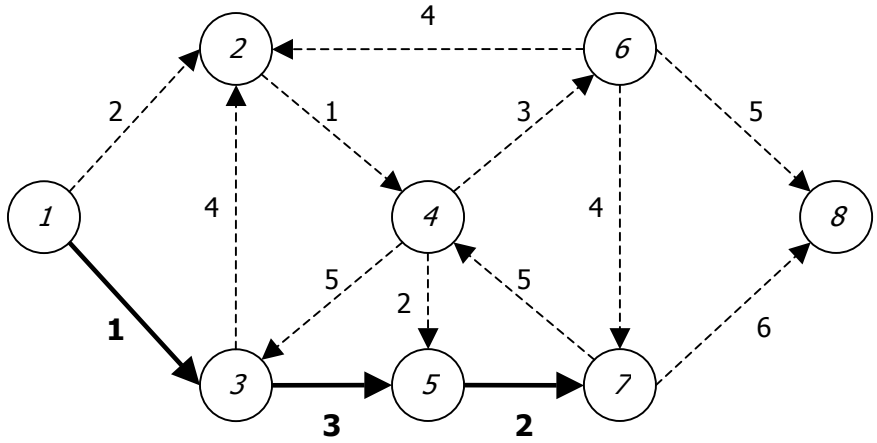


Figura A.1.2. Ejemplo del algoritmo de Dijkstra
(nodo fuente 1, nodo destino 7)

Del nodo 1 al nodo 8:

min =
2*x12+x13+x24+4*x32+3*x35+5*x43+2*x45+3*x46+2*x57+4*x62+4*x67+
5*x68+5*x74+6*x78;

x12+x13=1;
x12+x32+x62-x24=0;
x13+x43-x32-x35=0;
x24+x74-x43-x45-x46=0;
x35+x45-x57=0;
x46-x62-x67-x68=0;
x57+x67-x74-x78=0;
x68+x78=1;

@bin(x12);
@bin(x13);
@bin(x24);
@bin(x32);
@bin(x35);
@bin(x43);
@bin(x45);
@bin(x46);
@bin(x57);
@bin(x62);
@bin(x67);
@bin(x68);
@bin(x74);
@bin(x78);

Global optimal solution found at step: 10
Objective value: 11.00000
Branch count: 0

Variable	Value
x12	1.000000
x13	0.0000000
x24	1.000000
x32	0.0000000
x35	0.0000000
x43	0.0000000
x45	0.0000000
x46	1.000000
x57	0.0000000
x62	0.0000000
x67	0.0000000
x68	1.000000
x74	0.0000000
x78	0.0000000

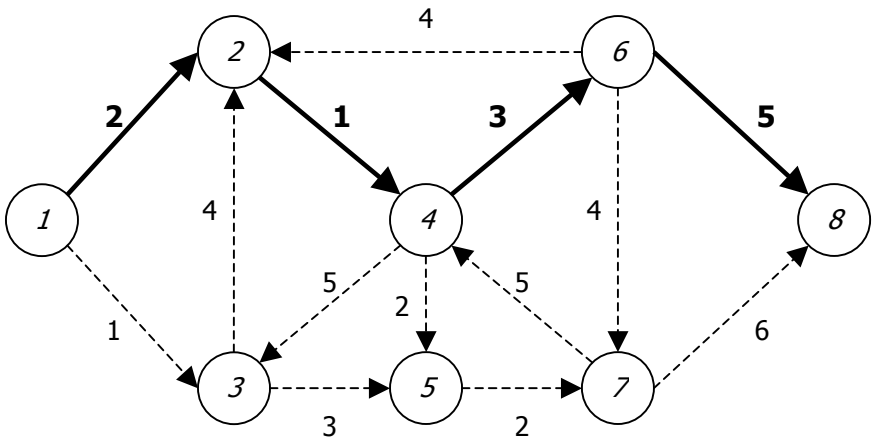


Figura A.1.3. Ejemplo del algoritmo de Dijkstra (capítulo 3)

Como puede observarse la función objetivo no cambia, es en las restricciones donde se debe revisar cuáles son los nodos fuente y destino para el envío y colocación de una unidad de flujo, de igual forma en las demás restricciones la conservación de flujo.

REFERENCIAS

1. Ahuja, Ravindra K.; Magnanti, Thomas L.; Orlin, James B., 1993. *Network Flows: Theory, Algorithms and Applications.* Prentice Hall. New Jersey, EE. UU.

2. Ball, M.O.; Magnanti, T.L.; Monma, C.L.; Nemhauser, G.L.,1995. *Handbooks in Operations Research and Management Science: Networks Models.* Elsevier Science.

3. Bazaraa, Mokhtar S.; Jarvis, John J.; Sherali, Hanif D., 1990. *Linear Programming and Network Flows.* John Wiley & Sons, 2ª edición.

4. Christofides, Nicos, 1975. *Graph Theory, an Algorithmic Approach*. Academic Press. London, UK.

5. Dreyfus, Stuart E.; Law, Averill M., 1977. *The Art and Theory of Dynamic Programming.* Academic Press. London, UK.

6. Flores de la Mota, Idalia, 1999. *Apuntes de Teoría de Redes*. Facultad de Ingeniería, U.N.A.M. México.

7. Flores de la Mota, Idalia, 2002. *Apuntes de Programación Entera*. Facultad de Ingeniería, U.N.A.M. México.

8. Hillier, Frederick S.; Lieberman, Gerald J, 1991. *Introducción a la Investigación de Operaciones*. Mc Graw Hill, 5ª edición.

9. Jensen, Paul A; Barnes, J. Wesley, 1980. *Network Flow Programming*. John Wiley & Sons.

10. Klau, Gunnar; Ljubic, Ivana; Moser, Andreas; Mutzel, Petra; Neuner, Philipp; Pferschy, Ulrich; Raidl, Günter; Weiskircher, René, 2004. *Combining a Memetic Algorithm with Integer Programming to Solve the Prize-Collecting Steiner Tree Problem.* Institute of Computer Graphics and Algorithms, Vienna University of Technology, Viena, Austría.
11. http://www.ads.tuwien.ac.at/publications/bib/pdf/klau-04.pdf

12. Misra, Preeti, 2005. *Fundamental Computer Concepts of Informatics.* http://www.informatics.indiana.edu/predraq/2005falli500/lecture_notes_5_misra.pdf

13. Rockafellar, R.T., 1984. *Network Flows and Monotropic Optimization.* John Wiley & Sons.

14. Sedgewick, Robert, 1988. *Algorithms*. Addison-Wesley Publishing, 2ª edición.

15. Taha, Hamdy A., 1995. *Investigación de Operaciones*. Alfaomega, 5ª edición.

16. Taha, Hamdy A., 1975. *Integer Programming: Theory, Applications and Computations*. Academic Press.

17. http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml

18. http://www.mat.uc.pt/~eqvm/OPP/OPP1.html

19. http://en.wikipedia.org/wiki/Shortest_path_problem

20. http://research.compaq.com/SRC/3D-animate/shortestpath.html

21. http://www.ics.uci.edu/~eppstein/161/960208.html

22. http://www.cs.sunysb.edu/~algorith/lectures-good/node18.html

23. http://www.cs.utexas.edu/users/psp/WalkShortestPath.pdf

24. http://www.husdal.com/gisruk2002/Sherlock_Mooney_Winstanley_Husdal_NUI_Maynooth.pdf

25. http://math.furman.edu/~mwoodard/fuejum/content/2004/paper2_2004.pdf

26. http://www.idsia.ch/~roberto/papers/science-2.pdf

27. http://web.mit.edu/sloan-msa/Papers/2.13.pdf