FACULTAD DE MATEMÁTICAS

DEPARTAMENTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

Trabajo Fin de Grado

Caminos Pareto-eficientes en redes: aplicaciones

Paloma Martín Gómez

Dirigido por:
Eduardo Conde Sánchez

Sevilla, Junio 2015.

"Siempre que enseñes, enseña a la vez a dudar de lo que enseñas"

José Ortega y Gasset

Summary

The problem of finding the path in a network connecting two given nodes, source and sink, with minimum possible cost is known in the literature as the shortest path problem (SPP) and it is a core model that lies at the heart of network optimization. The reason is the wide range of its practical applications and the large amount of interesting generalizations that can be considered, among them, the analysis of the multiobjective shortest path problem. This last model allows us to find how to send a given product between two specified nodes of a network as quickly, as cheaply and as reliable as possible taking into account the so-called Pareto optimal paths.

The text presented here consists of two chapters. The first one is devoted to the introduction of some notation and properties related to networks where each arc is associated to just one cost value. The formulation of the SPP as a Mathematical Programming model is considered as well as its corresponding dual problem. The complementary slackness theorem provides us with a characterization of any basic feasible solution as a spanning tree in the original network. A sufficient optimality condition is also derived from this relation. Finally, the Dijkstra algorithm is studied as well as other specific algorithms that use special properties on acyclic networks.

The second chapter extends the hypotheses considered in the previous one to a multiobjective context. The continuous formulation of the problem of finding the Pareto optimal paths is compared with its discrete version and some properties are stated. A generalization of the Dijkstra algorithm is proposed in order to find the whole set of Pareto optimal solutions. This procedure allows us to determine that the optimal set of paths corresponds with a set of adjacent trees. This property is very important in order to generate the Pareto optimal set of solutions and is the basis of a new improved algorithm that, starting with a given optimal tree, explores its adjacent trees and finds those Pareto optimal ones by using the duality conditions of the complementary slackness theorem presented in the chapter one.

Índice general

Capítulo 1

El problema del camino más corto en un grafo

El problema del camino más corto en un grafo consiste en determinar la ruta o trayectoria de menor longitud posible desde el nodo s (origen) al resto de los nodos de la red. Primero definiremos algunas nociones básicas necesarias para la comprensión del trabajo y estableceremos una serie de hipótesis necesarias para poder resolver este problema.

Definición 1.1 *Se define* ***red dirigida*** *como un par $G = (V, A)$, que consta de un conjunto finito de nodos V y un conjunto de arcos A que unen pares de nodos en V. Se dice que el arco $(i, j) \in A$ es incidente a los nodos $i, j \in V$ y está dirigido del nodo i al nodo j. Además, cada arco $(i, j) \in A$ tendrá asociado un coste (longitud, tiempo, ...) que denotaremos $c_{i,j}$.*

Definiremos $A(i)$ como el conjunto formado por los arcos adyacentes al nodo i.

Definición 1.2 *Sea $G = (V, A)$ una red dirigida. Un* ***camino*** *en G es una sucesión de arcos $p_{st} = \{(s = i_0, i_1), (i_1, i_2), ..., (i_{p-1}, i_p = t)\}$ en la cual el nodo inicial de cada arco es el mismo que el nodo final del arco precedente en la sucesión y todos los nodos son distintos. Así, cada arco en el camino está dirigido hacia t y hacia fuera de s.*

Definición 1.3 *Se define la* ***longitud de un camino dirigido*** *como la suma de todas las longitudes de arcos que forman dicho camino.*

Definición 1.4 *Un* ***ciclo dirigido*** *en una red G es un camino cerrado en G donde todos los nodos y arcos que lo forman son distintos salvo en el hecho de que el primer y el último nodo coinciden.*

Si una red dirigida no contiene ciclos dirigidos diremos que es **acíclica**.

Definición 1.5 *Una red dirigida y conexa que no contiene ciclos, dirigidos o no dirigidos, se denomina* ***árbol****.*

Una vez vistas las definiciones básicas, podemos asumir las siguientes hipótesis:

Hipótesis 1.1 *Todos los costes son enteros.*

Esta hipótesis en realidad sólo es necesaria para algunos algoritmos y en la práctica no supone limitación alguna pues es fácil pasar de números racionales a enteros.

Hipótesis 1.2 *La red contiene un camino dirigido desde el nodo s al resto de nodos de la red.*

Hipótesis 1.3 *La red no contiene ciclos de longitud negativa.*

Suponemos que la red no contiene ciclos de longitud negativa debido a que estos complican bastante el problema que nos planteamos. Ya que si la red contiene ciclos de longitud negativa, el camino más corto podría recorrerlos una infinidad de veces, y cada vez que éste los recorre, la longitud disminuye. Sin embargo, si nos encontramos ante una red dirigida sin ciclos de longitud negativa, existe un procedimiento bastante simple y eficiente denominado algoritmo de Dijkstra, cuyo objetivo es determinar la menor longitud del camino desde el origen al resto de los nodos, sin repetir ninguno de ellos.

Hipótesis 1.4 *La red es dirigida.*

Ya tenemos todas las hipótesis necesarias para plantear el problema de encontrar el camino más corto en una red. Además como éste coincide con un problema de flujo en redes, lo podemos formular como un problema de programación matemática. La siguiente formulación corresponde al problema del camino más corto entre el nodo s y el nodo t.

$$min \sum_{(i,j)\in A} c_{i,j}\, x_{i,j}$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \; i = s \\ 0 & \text{si } \; i \neq s,t \\ -1 & \text{si } \; i = t \end{cases} \qquad (P)$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

Multiplicamos por -1 las restricciones del problema (P) para conseguir la condición de optimalidad y formulamos su dual:

$$\text{máx } \pi_t - \pi_s \qquad (D)$$
$$\text{s.a} \qquad -\pi_i + \pi_j \leq c_{i,j}$$

Si utilizamos el Teorema de Holgura Complementaria (THC) para relacionar la solución del problema (P) y la del problema dual (D) se obtienen las siguientes condiciones:

- Si una variable primal es estrictamente positiva, entonces su restricción dual correspondiente se da con igualdad.

$$x_{i,j} > 0 \rightarrow -\pi_i + \pi_j = c_{i,j}$$

- Si la restricción dual es estricta entonces la variable primal correspondiente es cero.

$$-\pi_i + \pi_j < c_{i,j} \rightarrow x_{i,j} = 0$$

1.1. Identificación de solución básica factible con árboles

En principio sabemos que nuestro problema (P) tendrá tantas variables básicas como número de restricciones (es decir, tendremos n soluciones). Pero si observamos las restricciones, podemos ver que una de ellas es redundante debido a que la suma de todas las restricciones nos da cero y esto corresponde con la definición de linealmente dependiente pues tenemos una combinación de filas no nulas que es cero.

Al quitar una de las restricciones, tenemos que toda solución básica factible de (P) tiene $(n-1)$ variables básicas, las cuales se corresponden con $(n-1)$ arcos. Como podemos observar, esto ya va pareciendose más a un árbol pues todo árbol tiene $(n-1)$ arcos. Cabe destacar que estos arcos no forman un ciclo, pues si lo formasen, podríamos establecer una orientación arbitraria en él y sumar las columnas de los arcos que van en el mismo sentido que la orientación establecida y restar los que vayan en contra. Esto hace que la suma de las columnas de la matriz de restricciones asociadas a los arcos que forman el ciclo sea nula y por tanto las variables no puedan ser básicas pues para que lo sean, las columnas tienen que ser linealmente independientes.

Como conclusión, tenemos que elegir n conexiones de forma que no halla ciclos y esto se corresponde con la definición de árbol dada, por tanto, se puede identificar toda solución básica factible con un árbol no dirigido de G.

1.2. Condición de optimalidad

Fijado el árbol T, podemos hacer $c_{i,j} + \pi_i - \pi_j = 0 \quad \forall\,(i,j) \in T$ por el THC y al resolver este sistema de ecuaciones obtenemos un candidato a solución dual en la que la condición de factibilidad dual equivale a

$$c_{i,j} + \pi_i - \pi_j \geq 0 \qquad \forall\,(i,j) \notin T$$

es decir, esta es una condición suficiente de optimalidad en el problema (P).

Si se incumple dicha condición, es decir, si se tiene que

$$\overline{c}_{i,j} = \pi_i - \pi_j + c_{i,j} < 0$$

para algún $(i,j) \notin T$ se puede intentar mejorar la solución. Para ello hacemos básico el arco (i,j) que no pertenece a T, es decir, le añadimos a T una nueva conexión y esto da lugar a un ciclo en T.

Al formarse el ciclo lo que nos cuestionamos es qué arco que forme parte del ciclo debe salir de la base pues lo que estamos haciendo es añadir una nueva conexión a la base. Para ello incrementamos el flujo a través del arco (i,j) y establecemos un sentido en el ciclo de forma que aumentamos las conexiones que van en dicho sentido y disminuimos las que van en contra hasta conseguir que un arco distinto de (i,j) alcance un flujo igual a cero. Este arco será el que saldrá de la base.

Al hacer este procedimiento de añadir y eliminar conexiones en un árbol, lo que estamos haciendo es pasar de un árbol a otro árbol que se corresponden con soluciones básicas factibles adyacentes. Por lo tanto, se dice que la solución que genera el nuevo árbol es adyacente a la anterior.

El procedimiento descrito anteriormente es la base de lo que se conoce como Algoritmo del Símplex sobre redes, pero en la literatura existen otros algoritmos que buscan los valores duales mediante otros procedimientos. Son los denominados algoritmos de etiquetado y podemos distinguir dos tipos: algoritmos de fijado secuencial de etiquetas y algoritmo de corrección de etiquetas.

Definición 1.6 *Se define como* ***algoritmo de fijado secuencial de etiquetas*** *a aquel en el que en cada iteración se le asigna a cada nodo una etiqueta fija con la distancia desde el nodo s.*

Este tipo de algoritmos se pueden aplicar sólo a problemas de camino más corto definidos en una red acíclica con longitudes de arco arbitrarias o a problemas sobre grafos generales con longitudes de arco no negativas.

Definición 1.7 *Se define como* ***algoritmo de corrección de etiquetas*** *a aquel en el que en cada iteración asigna a cada nodo una etiqueta temporal con la distancia desde s, hasta el último paso, donde pasan a ser fijas o permanentes.*

Este algoritmo es más general que el anterior, lo que hace que se pueda aplicar a todo tipo de problemas incluyendo los que tienen longitudes de arco negativa.

En ambos algoritmos de etiquetado se utilizan dos propiedades fundamentales que enunciaremos a continuación.

Propiedad 1.1 *Si el camino* $s = i_1 - i_2 - \ldots - i_h = k$ *es el camino más corto desde el nodo s al nodo k , entonces para todo* $q = 2, 3, \ldots, h-1$*, el subcamino* $s = i_1 - i_2 - \ldots - i_q$ *es el camino más corto desde el origen al nodo* i_q*.*

Propiedad 1.2 *Sea d el vector que representa las distancias del camino más corto. Entonces un camino dirigido* p_{sk} *desde el nodo s hasta el nodo k es el camino más corto si y sólo si* $d(j) = d(i) + c_{i,j}$ *para todo arco* $(i, j) \in p_{sk}$*.*

La propiedad anterior es una consecuencia directa de una de las condiciones de holgura complementaria analizada anteriormente en la que simplemete hay que reemplazar las etiquetas $d(i)$ por las variables duales $\pi(i)$.

A continuación vamos a ver una especialización del algoritmo de fijado secuencial de etiquetas sobre grafos acíclicos. Estos grafos son especialmente importante puesto que sirven para modelar las denominadas redes de actividades que componen un proyecto.

1.3. El camino más corto en grafos acíclicos

Recordemos que un grafo se dice *acíclico* si no contiene ciclos.

Definición 1.8 *Se define* ***ordenación topológica*** *de un grafo dirigido acíclico $G = (N, A)$ como una lista de los nodos del grafo, tal que si el arco $(i, j) \in A$ entonces el nodo i aparece antes que j en la lista resultado, es decir, $i < j$.*

Decribamos la idea general del algoritmo en grafos acíclicos. Lo primero que hacemos es considerar $d(s) = 0$ y hacer que el resto de nodos tengan una distancia infinita. A continuación, examinamos todos los nodos en orden topológico y para cada nodo $i \in A$ examinamos el conjunto de sus nodos adyacentes. Si para algún arco $(i, j) \in A(i)$, se tiene que $d(j) > d(i) + c_{i,j}$, entonces podemos hacer $d(j) = d(i)+c_{i,j}$. Cuando el algoritmo halla recorrido todos los nodos en este orden, la distancia será mínima.

Teorema 1.1 *El algoritmo resuelve el problema del camino más corto en grafos acíclicos en tiempo $O(m)$, siendo m el número de arcos del grafo.*

Demostración.

A continuación, veremos por inducción que la distancia será mínima. Para ello supongamos que esto es correcto para un cierto nodo k (Hipótesis de inducción), es decir, suponemos que el algoritmo ha examinado los nodos $1, 2, \ldots, k$ y que sus etiquetas o distancias son óptimas.

Consideremos la iteración en la que el algoritmo examina el nodo $k + 1$. Si el camino más corto desde el origen a $k + 1$ es $s = i_1 - i_2 - \cdots - i_h - (k + 1)$, entonces el camino más corto desde el origen al nodo i_h debe de ser $i_1 - \cdots - i_h$ por la *Propiedad 1.1*. Además, todos los nodos estan ordenados topológicamente y si el arco $(i_h, k + 1) \in A$ se tiene que $i_h \in \{1, 2, \ldots, k\}$.

Entonces por la hipótesis de inducción, la distancia del nodo i_h es igual a la longitud del camino $i_1 - i_2 - \cdots - i_h$ y como consecuencia, la distancia del nodo $(k+1)$ es igual a la longitud del camino $i_1 - i_2 - \cdots - i_h - (k+1)$ pues cuando se examina el nodo i_h, el algoritmo tiene que recorrer el arco $(i_h, k+1)$. Por tanto, cuando el algoritmo examina el nodo $(k+1)$, la distancia es mínima.

□

El siguiente es el ejemplo prototipo de algoritmo de fijado secuencial de etiquetas y será generalizado en el siguiente capítulo para ser aplicado al problema de determinación de caminos eficientes.

1.4. Algoritmo de Dijkstra

El objetivo del algoritmo de Dijkstra es encontrar un camino de longitud mínima entre los nodos de la red, partiendo de un origen s y calculando el camino más corto al resto de nodos.

Antes de poner la formulación del algoritmo de Dijkstra, vamos a explicar con exactitud los pasos que lleva a cabo aunque es bastante similar al de los grafos acíclicos.

Paso 1 En cada iteración el algoritmo va dividiendo los nodos en dos conjuntos, los nodos con etiquetas definitivas y los nodos con etiquetas temporales, denotaremos dichos conjuntos por S y $\overline{S}$ respectivamente y $|S|$ el número de elementos en el conjunto S. Primero el algoritmo le asigna al nodo s (origen) la distancia cero y después considera la etiqueta del nodo definitiva.

Paso 2 A continuación, considera que la distancia al resto de los nodos en la red es ∞ pues son desconocidas. En cada iteración el algoritmo analiza los arcos adyacentes que tienen a s como nodo origen y elige el de menor longitud, supongamos que es el nodo i, es decir, $d(i)$ es la menor longitud; entonces este pasa a estar en el conjunto S, pues marcamos su etiqueta como definitiva.

Paso 3 Ahora nuestro camino más corto P estaría formado por el nodo s y el nodo i. Volvemos a actualizar las etiquetas, examinando de nuevo todos los nodos que son adyacentes en este caso al nodo i, teniendo en cuenta que si las distancias de dichos nodos sobrepasan la distancia a i más la longitud del arco que los une a i, entonces se sustituye dicha distancia por la obtenida a través de i. Posteriormente se elige el nodo con menor etiqueta no definitiva, se hace definitiva y lo pasamos al conjunto S.

Paso 4 Repetimos este procedimiento hasta que recorramos todos los nodos en la red. Al finalizar, el conjunto S coincidirá con el conjunto de nodos en la red, es decir, todas las etiquetas de los nodos serán consideradas definitivas y por tanto el algoritmo habrá acabado.

La formulación del algoritmo de Dijkstra es la siguiente:

algorithm *Dijkstra*
begin
$S := \emptyset$; $\overline{S} := N$
$d(i) := \infty$ para cada nodo $i \in N$;
$d(s) := 0$ *y* $pred(s) := 0$;
while $\mid S \mid < n$ **do**
begin
Si $i \in \overline{S}$ es un nodo para el cual $d(i) = min\{d(j) : j \in \overline{S}\}$
$S := S \bigcup i$;
$\overline{S} := \overline{S} - i$;
for cada $(i, j) \in A(i)$ **do**
if $d(j) > d(i) + c_{i,j}$ **then** $d(j) := d(i) + c_{i,j}$ y $pred(j) := i$;
end;
end;

Notese, que la función *pred* permite reconstruir el camino más corto desde s a cualquier otro nodo al finalizar el algoritmo, es decir, en realidad lo que nos indica es cual es el predecedor del nodo que le pedimos.

Ejemplo

Si aplicamos el algoritmo de Dijkstra a la Figura 1.1, considerando el nodo 1 como nuestro nodo origen, hacemos que éste tenga distancia 0, con-

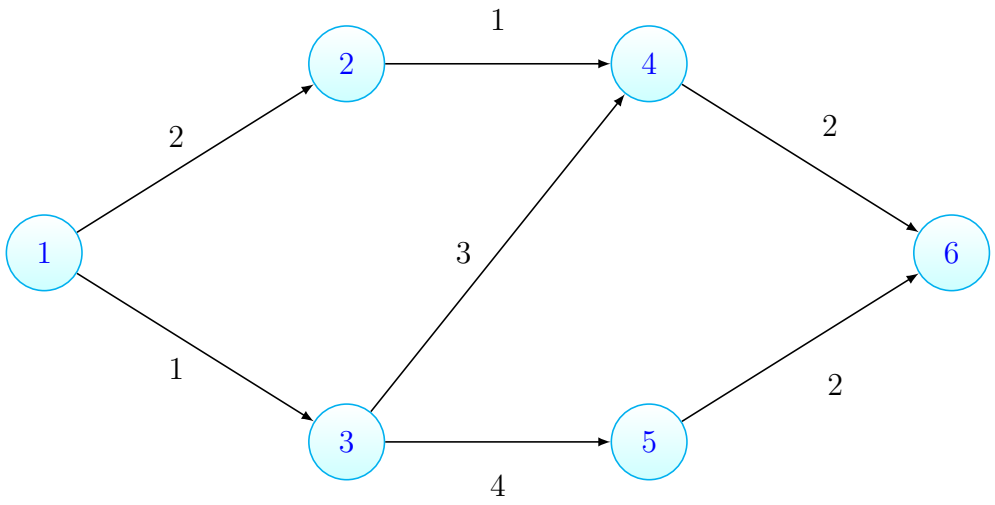

Figura 1.1: Ejemplo del Algoritmo de Dijkstra

siderando su etiqueta definitiva y después hacemos que la distancia al resto de los nodos sea infinita. Nuestro objetivo es buscar el camino más corto del origen al resto de los nodos. Partiendo del origen sólo podemos llegar a los nodos 2 y 3, pasando estos a tener etiquetas temporales 2 y 1 respectivamente, como buscamos la más pequeña nos quedamos con el nodo 3 y hacemos su etiqueta definitiva. A continuación realizamos el mismo procedimiento para los nodos 2, 4 y 5 y consideramos como etiqueta definitiva la del nodo 2. En la siguiente iteración hacemos definitiva la etiqueta del nodo 4 y así con todos los nodos hasta conseguir que todas las etiquetas sean definitivas. Una vez finalizado el algoritmo nos queda que el camino más corto del nodo 1 al 6 es el camino que pasa por los nodos $1, 2, 4$, y 6 con distancia mínima 5.

Tabla correspondiente a las etiquetas del Ejemplo

Nodos	Etiqueta 1	Etiqueta 2
1	0	
2	2	
3	1	
4	4	3
5	5	
6	5	7

1.4.1. Validez del algoritmo de Dijkstra

A continuación demostraremos que el algoritmo de Dijkstra genera una solución óptima.

Demostración.

Aplicamos inducción en el cardinal del conjunto S. Las hipótesis de inducción serán las siguientes:

1. La distancia o longitud en S es óptima.
2. La distancia o longitud de cada nodo en $\overline{S}$ es la ruta (desde el origen a dichos nodos) de menor longitud que sólo contiene nodos interiores en S.

Veamos primero la primera hipótesis de inducción:
Recordemos que en cada iteración el algoritmo pasa del conjunto $\overline{S}$ al conjunto S el nodo i con menor longitud. Por tanto, lo que nosotros queremos demostrar es que $d(i)$ es la distancia óptima del origen al nodo i. Por hipótesis de inducción, $d(i)$ es la menor longitud del camino más corto al nodo i de entre los caminos que no contienen nodos que se encuentran en $\overline{S}$.

El caso en el que $i = s$ es evidentemente cierto pues la distancia sería cero. Consideremos P cualquier ruta desde el origen s al nodo i, la cual contiene al menos un nodo en el conjunto $\overline{S}$. Dicho camino lo podemos dividir en dos caminos P_1 y P_2, el primer camino va desde el origen s a un cierto nodo $k \in \overline{S}$ pero este no contiene ningún otro nodo de $\overline{S}$ y P_2 va desde k al nodo i.

Por hipótesis de inducción, la longitud de P_1 es al menos $d(k)$ y por la elección que hace el algoritmo del nodo i se tiene que $d(k) \geq d(i)$. Ya tenemos que P_1 tiene al menos longitud $d(i)$. Ahora bien, como todos los arcos son no negativos, la longitud de P_2 será no negativa y esto hace que la longitud de P sea al menos $d(i)$, es decir, $d(i)$ sería la longitud del camino más corto del origen al nodo i.

Una vez visto que se verifica la primera hipótesis, veamos que se cumple la segunda hipótesis de inducción.

Si se incumpliese dicha hipótesis debería existir algún nodo j del conjunto $\overline{S} \setminus \{i\}$ verificando $d(j) > d(i) + c_{i,j}$, pero esto no sucede, ya que si

recordamos como funciona el algoritmo, cuando este consideraba permanente un nodo i, examinaba cada arco $(i, j) \in A(i)$ y si $d(j) > d(i) + c_{i,j}$, entonces fijaba $d(j) = d(i) + c_{i,j}$ y consideraba que $pred(j) = i$. Por lo tanto, tras ser recalculada la longitud, por la hipótesis de inducción el camino desde el origen al nodo j definido por la función predecesor satisface la *Propiedad 1.2* y así la longitud de cada nodo en $\overline{S} - \{i\}$ es el camino de menor longitud sujeto a que cada nodo interior en un camino debe de pertenecer a $S \bigcup \{i\}$. □

1.4.2. Complejidad del algoritmo de Dijkstra

A continuación estudiamos la complejidad del peor caso del algoritmo de Dijkstra. Los calculos del algoritmo de Dijkstra se emplean para realizar dos tipos de operaciones básicas:

- *Selección de nodos.* El algoritmo desarrolla esta operación n veces y en cada operación necesita examinar los nodos con etiqueta temporal. Por lo tanto, el número de operaciones realizadas es:

$$n + (n-1) + (n-2) + \cdots + 1 = \frac{(n-1)n}{2} = O(n^2)$$

- *Actualización de distancias.* El algoritmo realiza esta operación $|A(i)|$ veces para cada nodo i. En total se realiza esta operación $\sum_{i \in N} |A(i)| = m$, donde m es el número de arcos en el grafo. Como cada ajuste requiere un número constante de operaciones elementales en total se tienen $O(m)$ operaciones en este proceso.

  Esto da lugar al siguiente resultado:

Teorema 1.2 *El algoritmo de Dijkstra realiza $O(n^2)$ operaciones para encontrar el camino más corto, siendo n el número de nodos del grafo.*

Capítulo 2

Caminos Pareto-eficientes en redes

A diferencia del capítulo anterior, en este capítulo estudiaremos en profundidad el problema de encontrar el camino más corto con multicriterios, es decir, con más de una función objetivo, ya que a veces una única función objetivo no es suficiente para completar el problema. Por ejemplo, en el caso de las redes asociadas a carreteras, a cada arco se le asigna varios parámetros como son el tiempo, la seguridad, el coste, la distancia, la probabilidad de accidentes... En general, la solución que obtendremos será la más eficiente pues lógicamente no será la mejor para todos los criterios, ya que no puede optimizar a la vez todos los parámetros. Sin embargo, veremos que existe un conjunto de caminos digamos privilegiados (Pareto optimos o no dominados) que contiene los caminos óptimos bajo cualquier objetivo construido como una función creciente de los costes individuales bajo cada criterio.

Si consideramos s y t dos nodos distintos en $G = (V, A)$ y $\wp$ el conjunto formado por todos los caminos desde s hasta t en G. Recordemos que un camino p_{st} es la secuencia de los nodos que lo forman $p_{st} = \{s = i_0, i_1, \ldots, i_p = t\}$. Diremos que un camino es **elemental** si no se repiten los nodos en la secuencia y es **no elemental** cuando algún nodo se repite.

Generalizamos el concepto visto anteriormente de coste de un arco definiendo $c^k_{i,j}$ como el **coste** asociado a cada arco $(i, j) \in A$ donde k representa los diferentes tipos de coste asociados a cada arco (tiempo, distancia...). La definición de longitud de un camino p_{st} para este tipo de problemas no varía con respecto al capítulo anterior. Por tanto, para todo $k \in \{1, \ldots, r\}$ y para todo camino p_{st} entre dos nodos distintos cualesquiera, la **longitud del camino** p_{st}, $c^k(p_{st})$, es la suma de todos los costes de los arcos que lo forman,

es decir, $c^k(p_{st}) = \sum_{(i,j)\in p_{st}} c^k_{i,j}$.

Partiendo de la *Definición 1.5* de árbol añadiremos algunos conceptos que serán utilizados en los algoritmos considerados en este capítulo.

Definición 2.1 *Un **árbol de unión** de $G = (V, A)$ es un árbol cuyo conjunto de nodos coincide con el de G, es decir, V y su conjunto de arcos es un subconjunto de A.*

Definición 2.2 *Un **árbol de unión dirigido** de G con raiz en s es un árbol de unión de G donde cada arco es dirigido hacia fuera de s*

Sea T un árbol de unión dirigido, podemos decir que un nodo $i \in V$ en T se denomina **hoja** si no existe ningún arco $(i, j) \in T$ para todo $j \in V$. Además, consideraremos que el camino p_{si} es una **rama** si el nodo i es una hoja.

Tras introducir todos los conceptos necesarios para la formulación del problema, la *formulación continua* del problema multicriterio sería:

$$'min' \left(\sum_{(i,j)\in A} c^1_{i,j} x_{i,j}\, , \ldots , \sum_{(i,j)\in A} c^r_{i,j} x_{i,j} \right)$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 \text{ si } & i = s \\ 0 \text{ si } & i \neq s,t \quad (i = 1, ..., n) \\ -1 \text{ si } & i = t \end{cases} \qquad (P1)$$

$$x_{i,j} \geq 0 \quad \forall\, (i, j) \in A$$

A partir de la cual, denotaremos por X el *poliedro convexo* resultante de establecer las restricciones de (P1) y V el *conjunto de vértices* de X.

Definición 2.3 *Sean $x, y \in X$ dos soluciones factibles distintas de (P1). Se dice que x **domina a** y (x D y) si y sólo si $c^k(x) \leq c^k(y)$ para todo $k \in \{1, \ldots, r\}$ y al menos en un caso se da con desigualdad estricta.*

Definición 2.4 *El conjunto $X_d = \{x \in X | \exists\, y \in X : y\ D\ x\}$ se denomina **conjunto de soluciones dominadas** de (P1). Luego $X_p = X - X_d$ es el **conjunto de soluciones no dominadas o Pareto optimas** de (P1). Y el conjunto de vértices no dominados de X se define como $V_p = X_p \bigcap V$.*

A continuación, se extienden las hipótesis 1.3 y 1.1 vistas en el capítulo anterior para adecuarlas al contexto multicriterio del problema (P1).

Hipótesis 2.1 *Supondremos la existencia de algún* $k \in \{1, \ldots, r\}$ *con* $c_{i,j}^k \geq 0$ *para todo* $(i, j) \in A$. *Asumimos que* $k = 1$ *pues es una generalización de la hipótesis 1.1.*

Hipótesis 2.2 *Para cualquier ciclo* C *en* G, *suponemos que* $c^k(C) \geq 0$ *para todo* $k \in \{1, \ldots, r\}$ *y que la desigualdad estricta se cumple para algún valor de* k.

Es fácil ver que esta hipótesis generaliza la *Hipótesis 1.3*.

Si discretizamos la formulación continua del problema (P1), necesitamos añadir la siguiente restricción al problema

$$x \in V$$

dando lugar a lo que denominaremos a partir de ahora *formulación discreta del problema* y denotaremos por (P2).

Proposición 2.1 *Todo camino* p_{st} *es un vértice de* X *y viceversa.*

Esta proposición es evidente teniendo en cuenta que la formulación continua del problema es un problema de flujo con todos sus coeficientes enteros y también deja claro que la formulación discreta del problema es una formulación matemática para caminos más cortos con multicriterios.

Como una solución básica factible de la formulación continua del problema es un árbol de unión de G, un vértice de X puede estar asociado con más de un árbol de unión de $G = (V, A)$. En [4] se presenta un algoritmo eficiente para transformar un árbol de unión de G, digamos $T \in \mathfrak{T}(x)$, en un árbol de unión dirigido de G con raíz en s, T', de manera que $T' \in \mathfrak{T}(x)$. Por tanto, tenemos la siguiente proposición:

Proposición 2.2 *Todo punto factible de* X *está asociado con un árbol de unión dirigido de* G, *con raiz en* s.

De esta proposición podemos concluir que $\wp$ puede identificarse con el conjunto de árboles de unión dirigidos de G con raiz en s, es decir, para determinar $\wp$ no es necesario generar todo el conjunto de soluciones básicas factibles del problema continuo (P1).

A partir de este momento, entenderemos por árbol a un árbol de unión dirigido con raíz en s.

Nota. Es obvio que las definiciones 2.3 y 2.4 dadas para el problema continuo (P1) pueden ser usadas para el problema discreto (P2). Así, T_d será el **conjunto de soluciones dominadas de (P2)** y T_p es el **conjunto de soluciones Pareto optimas del problema discreto (P2)**.

Definición 2.5 *Diremos que $p_{st} \in \wp$ es un* ***camino no dominado*** *si y sólo si no existe ningún camino $p'_{st} \in \wp$ tal que $c^k(p'_{st}) \leq c^k(p_{st})$ para todo $k \in \{1, \ldots, r\}$ y $c^i(p'_{st}) < c^i(p_{st})$ para algún $i \in \{1, \ldots, r\}$.*

Con el siguiente lema establecemos una importante diferencia entre la formulación continua del problema (P1) y la formulación discreta del problema (P2), pues a partir de la formulación continua generamos un conjunto de soluciones eficientes que no tiene que coincidir necesariamente con el de la formulación discreta.

Lema 2.1 *Se tiene que $V_p \subseteq T_p$ donde V_p es el conjunto de vértices no dominados de X y T_p es el conjunto de soluciones pareto óptimas de (P2).*

Demostración.

Para todo $T \in V_p$, no existe $x \in X$ tal que x D T. En concreto, no existe $T' \in V$ $(V \subset X)$ tal que T' D T. Por tanto, todo vértice no dominado de (P1) es una solución no dominada de (P2), es decir, $V_p \subseteq T_p$.

□

En muchos problemas la inclusión es estricta debido a que puede existir un $T \in T_p$ dominado por alguna solución no básica de (P1).

Nuestro objetivo es demostrar que los vértices Pareto óptimos obtenidos como solución en la formulación continua y discreta son vértices adyacentes, porque de esta forma, podremos generar todo el conjunto de soluciones eficientes a partir de una de ellas examinando las adyacentes.

Teorema 2.1 *Para cada par T, T' de vértices no dominados de (P1), existe una secuencia $\{T = T_1, T_2, \ldots, T_{l-1}, T_l = T'\}$ de vértices adyacentes no dominados de (P1). Es decir, T y T' estan conectados por vértices adyacentes no dominados de (P1).*

En general, este teorema no es cierto para el caso discreto.

La proposición que vamos a ver a continuación es directa.

Proposición 2.3 *Sea p^1_{st}, p^2_{st} y p^3_{st} tres caminos distintos desde s hasta t en G. Se verifica que si p^1_{st} D p^2_{st} y p^2_{st} D p^3_{st} entonces p^1_{st} D p^3_{st}.*

Lema 2.2 *Para todo camino dominado p_{st}, existe un camino no dominado p'_{st} tal que p'_{st} D p_{st}.*

Demostración.

Es directa como consecuencia de la *Proposición 2.3* y del hecho de que el número de caminos desde s hasta t en G es finito.

□

Lema 2.3 *Bajo la hipótesis 2.2, todos los caminos no elementales (se repiten los nodos en el camino) son dominados por los caminos no dominados.*

Demostración.

Consideremos un camino no elemental p_{st} desde s hasta t. Sin pérdida de generalidad, podemos suponer que en p_{st} sólo se repite un nodo i. Así, $p_{st} = p_{si} \bigcup p_{ii} \bigcup p_{it}$, donde p_{ii} es un ciclo. Como p_{ii} es un ciclo se tiene que $c^k(p_{ii}) \geq 0$ para todo $k \in \{1, \ldots, r\}$ y existe algún $j \in \{1, \ldots, r\}$ de manera que $c^j(p_{ii}) > 0$. Es claro que $c^k(p_{st}) = c^k(p_{si}) + c^k(p_{ii}) + c^k(p_{it})$ para todo $k \in \{1, \ldots, r\}$. Dado que $c^j(p_{st}) > c^j(p_{si}) + c^j(p_{it})$ y $c^k(p_{st}) \geq c^k(p_{si}) + c^k(p_{it})$ para todo $k \neq j$, podemos concluir que $(p_{si} \bigcup p_{it})$ D p_{st}. Luego, $(p_{si} \bigcup p_{it})$ es un camino elemental desde s hasta t. Si aplicamos el *Lema 2.2* ya hemos terminado pues ya tendríamos que los caminos no dominados dominan al camino elemental que hemos encontrado.

□

Lema 2.4 *Todo camino no dominado desde s hasta t en G contiene sólo subcaminos no dominados desde s hasta cualquier nodo intermedio i en el camino considerado.*

Demostración.

Sea p_{st} un camino no dominado desde s hasta t en $G = (V, A)$. Supongamos que existe algún nodo intermedio $i \in p_{st}$ tal que p_{si} es un subcamino dominado de p_{st} desde s hasta i. Luego, existe un camino elemental p'_{si} ($p'_{si} \neq p_{si}$), tal que $c^k(p'_{si}) \leq c^k(p_{si})$ para todo $k \in \{1, \ldots, r\}$ y se da al menos una desigualdad estricta. Sea $p'_{st} = p'_{si} \bigcup p_{it}$, donde p_{it} es el subcamino de p_{st} desde i hasta t. Claramente, $c^k(p'_{st}) = c^k(p'_{si}) + c^k(p_{it})$ para todo

$k \in \{1, \ldots, r\}$. Así, $c^k(p'_{st}) \leq c^k(p_{st})$ para todo $k \in \{1, \ldots, r\}$ y se verifica al menos una desigualdad estricta. Esto es, p'_{st} D p_{st} lo cual es imposible pues p_{st} es un camino no dominado. Por tanto, p'_{st} es un camino no elemental. Pero si fuese así, por el *lema 2.3* tendríamos que p'_{st} es dominado por cualquier camino elemental q_{st} y esto es imposible porque por la *Proposición 2.3* concluiríamos que q_{st} D p_{st}.

□

Como podemos ver, los lemas 2.3 y 2.4 dejan claro el porqué es necesario imponer la hipótesis 2.2.

Antes de ver el resultado más destacado en el capítulo, necesitamos saber cuando dos caminos son adyacentes. Sean p^1_{st} y p^2_{st} dos caminos distintos desde s hasta t en G, diremos que p^1_{st} y p^2_{st} son **adyacentes** cuando p^1_{st} y p^2_{st} están determinados por dos vértices adyacentes en X.

Teorema 2.2 *Para todo par de caminos no dominados p^1_{st} y p^2_{st}, existe una secuencia $\{p^1_{st}, p^i_{st}, \ldots, p^j_{st}, p^2_{st}\}$ de caminos adyacentes no dominados. Es decir, p^1_{st} y p^2_{st} estan conectados por caminos no dominados adyacentes desde s hasta t en G.*

La prueba del teorema se apoya en la generalización del algoritmo de etiquetado visto en el capítulo anterior, el algoritmo de Dijkstra. Por tanto, vamos a ver dicha generalización y después la demostración.

2.1. Aproximación de etiquetado múltiple

Al generalizar el algoritmo de Dijkstra, consideramos una **etiqueta** como una r-upla de enteros y dos indicadores. Como ya sabemos por dicho algoritmo, a lo largo de su ejecución a cada nodo se le asocian diferentes etiquetas.

Sea $i \in V$ cualquier nodo en G, entonces la **l-ésima etiqueta asociada a** i es $[(c^1(p_{si}), \ldots, c^r(p_{si}), (j, l_1)]_l$ donde j $(j \neq i)$ es un nodo cualquiera de G y l_1 indica alguna etiqueta de j para los cuales el coste es

$$c^k_l(p_{si}) = c^k_{l_1}(p_{sj}) + c^k_{ji}$$

para todo $k \in \{1, \ldots, r\}$. Es fácil ver que el coste de una etiqueta en este caso es exactamente igual que la definición que dimos en el capítulo anterior salvo que es para varios criterios, como ya hemos mencionado al pricipio del capítulo.

Observemos que $c_l^k(p_{si})$ es la k-ésima componente de la l-ésima etiqueta del nodo i.

Definición 2.6 *Sea i cualquier nodo en G. Se dice que $[(c^1(p_{si}), \ldots, c^r(p_{si}), (-, -)]_\xi$ es **más pequeña lexicográficamente** que $[(c^1(p'_{si}), \ldots, c^r(p'_{si}), (-, -)]_\delta$ si*

$$c_\xi^1(p_{si}) = c_\delta^1(p'_{si}), \ldots, c_\xi^{k-1}(p_{si}) = c_\delta^{k-1}(p'_{si})$$

y $c_\xi^k(p_{si}) < c_\delta^k(p'_{si})$ para algún $k \in \{1, \ldots, r\}$.

Al igual que en el Algoritmo de Dijkstra, dentro del conjunto de todas las etiquetas asociadas a cada nodo, podemos considerar las etiquetas permantentes o temporales como ya vimos en el *Capítulo 1*. Una vez fijada una etiqueta $j \in V$, se le asignan etiquetas temporales a los nodos $i \in V$ tales que el arco $(j, i) \in G$. Así pues, mientras que la primera etiqueta permanece fija, las etiquetas temporales pueden ser eliminadas a lo largo de la ejecución del algoritmo hasta conseguir que todas las etiquetas sean permanentes.

Veamos a continuación la formulación del algoritmo y después un ejemplo para que quede claro su ejecución.

Algoritmo 1

Paso 0 Se le asigna al nodo origen s la etiqueta temporal $[(0, \ldots, 0), (-, -)]_1$.

Paso 1

1. Si el conjunto de etiquetas temporales es vacío pasar al Paso 3. En otro caso, de entre todas las etiquetas temporales determinar cual es la más pequeña lexicográficamente. Si fuese la l-ésima etiqueta asociada al nodo i la más pequeña, hacer esta etiqueta permanente.

2. Mientras exista algún nodo $j \in V$ tal que $(i, j) \in A$ realizar:

   a) $c^k(p_{sj}) = c_l^k(p_{si}) + c_{ij}^k \ \forall k \in \{1, \ldots, r\}$ y considerar la siguiente etiqueta como etiqueta temporal del nodo j:

   $$[(c^1(p_{sj}), \ldots, c^r(p_{sj})), (i, l)]_\varepsilon$$

   b) De entre todas las etiquetas temporales asociadas al nodo j, eliminamos aquellas que representan un camino dominado desde s hasta j.

Paso 2 Volver al Paso 1.

Paso 3 Encontrar un camino no dominado desde s hasta t tal que los dos indicadores para cada etiqueta sean utilizados.

Paso 4 Parar de ejecutar el algoritmo.

Como podemos observar el Paso 3 es fácil de ejecutar. De hecho, al final del algoritmo hay tantas etiquetas permanentes asociadas a cada nodo $t \in V$ como número de caminos no dominados desde s hasta t como vimos con el Algoritmo de Dijkstra, es más, cada etiqueta corresponde con un único camino. Para determinar cualquiera de esos caminos, elegimos una etiqueta del nodo $t \in V$ y suponemos que (j, l) son los indicadores de esta etiqueta. Así, $j \in V$ es un nodo que está justo antes del nodo t en el camino. Ahora bien, para determinar el nodo que va justo antes del nodo j, tenemos que saber los indicadores de la l-ésima etiqueta del nodo j. Es obvio que vamos a ir pasando por diferentes nodos hasta alcanzar el primer nodo del camino, es decir, s.

Señalemos que todos los caminos no dominados desde el nodo $s \in V$ a los nodos $i \in V - \{s\}$ pueden determinarse a partir del *Algoritmo 1.*

Es fácil de ver que la prueba de este algoritmo sigue el *Lema 2.4* y las *Hipótesis 2.1* y *2.2*.

A continuación, vamos a ver un ejemplo para que quede claro como se ejecuta el algoritmo.

Ejemplo

Consideremos el grafo que aparece en el Figura 2.1.
Lo primero que hace el algoritmo es asignarle al primer nodo (es decir, s=1) la etiqueta temporal $[(0, 0, 0, 0), (-, -)]_1$. Claramente, esta etiqueta será elegida en la primera ejecución del Paso 1 y como consecuencia, se considera permanente. Una vez fijado el primer nodo, sólo los nodos 2 y 3 tienen etiquetas temporales, las cuales serían: $[(10, 4, 2, 10), (1, 1)]_1$ y $[(6, 1, 18, 10), (1, 1)]_1$ respectivamente. En la siguiente ejecución del Paso 1, la primera etiqueta del nodo 3, $[(6, 1, 18, 10), (1, 1)]_1$ es elegida como la etiqueta temporal más pequeña lexicográficamente debido a que es más pequeña lexicográficamente que la etiqueta temporal del nodo 2. Ahora bien, desde la etiqueta del nodo 3, sólo los nodos 2 y 5 son etiquetados temporalmente con las siguientes etiquetas $[(12, 3, 28, 20), (3, 1)]_2$ y $[(7, 5, 26, 11), (3, 1)]_2$.

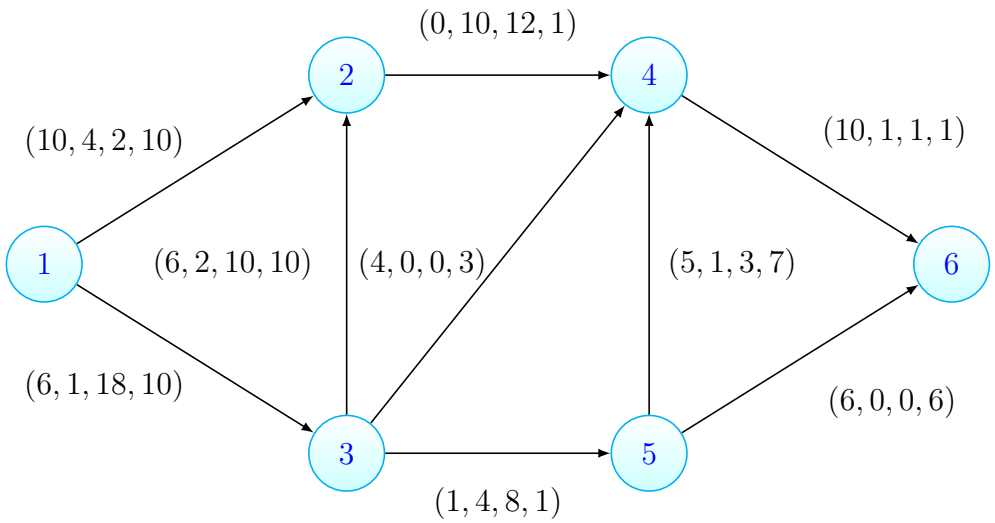

Figura 2.1: Ejemplo para el Algoritmo 1

Hasta este momento no hemos tenido que eliminar ninguna etiqueta temporal. Esto sólo sucederá cuando desde la segunda etiqueta del nodo 2, $[(12, 3, 28, 20)\,(3, 1)]_2$, el nodo 4 es etiquetado con la siguiente etiqueta $[(12, 13, 40, 21),\ (2, 2)]_3$, la cuál es dominada por la primera etiqueta del nodo 4.

El procedimiento continúa hasta que no existen más etiquetas temporales, es decir, hasta que todas las etiquetas son consideradas permanentes o fijas. Como podemos ver el resultado final lo recogeremos en la siguiente tabla. Al finalizar el algoritmo llegaremos a dos caminos no dominados desde $s = 1$ a $t = 6$, los cuales serían $p^1_{16} = \{1, 3, 5, 6\}$ y $p^2_{16} = \{1, 2, 4, 6\}$.
Por ejemplo, para determinar el segundo camino debemos comenzar con la siguiente etiqueta del nodo 6 : $[(20, 15, 15, 20),\ (4, 2)]_2$. Así, siguiendo la tabla, podemos ver que el nodo 6 está etiquetado a partir de la segunda etiqueta del nodo 4 y ésta se obtiene a partir de la primera del nodo 2, la cual se ha obtenido a partir de la primera etiqueta del nodo 1.

Tabla correspondiente a las etiquetas del Ejemplo

Nodos	Etiqueta 1	Etiqueta 2
1	$(0,0,0,0),(-,-)^p$	
2	$(10,4,2,10),(1,1)^p$	$(12,3,28,20),(3,1)^p$
3	$(6,1,18,10),(1,1)^p$	$(14,14,14,14),(4,2)^p$
4	$(12,6,29,18),(5,1)^p$	$(10,14,14,11),(2,1)^p$
5	$(7,5,26,11),(3,1)^p$	$(15,18,22,15),(3,2)^p$
6	$(13,5,26,17),(5,1)^p$	$(20,15,15,12),(4,2)^p$

Nodos	Etiqueta 3	Etiqueta 4
1		
2	$(20,16,24,24),(3,2)^d$	
3	$(16,6,29,21),(4,1)^d$	
4	$(12,13,40,21),(2,2)^d$	$(20,19,25,22),(5,2)^d$
5		
6	$(22,7,30,19),(4,1)^d$	$(21,18,22,21),(5,2)^d$

p corresponde con etiquetas permanentes
d a etiquetas dominadas

2.2. Conexión de caminos no dominados

En esta sección vamos a demostrar el *Teorema 2.2*, para ello necesitamos introducir algunas definiciones previas, una propiedad y un lema que nos serán de ayuda.

Consideramos que el **árbol más corto lexicográficamente** es un árbol en el que el camino más corto lexicográficamente desde el nodo s al nodo i es determinado para todo nodo $i \in N - \{s\}$.

Definición 2.7 *Se dice que un* ***árbol es no dominado*** *si todas sus ramas son caminos no dominados.*

Proposición 2.4 *El árbol más corto lexicograficamente es un árbol no dominado.*

Demostración.

Hagamos la prueba por reducción al absurdo. Supongamos que el árbol más corto lexicográficamente está dominado.

Sabemos por definición que un árbol está dominado si todas sus ramas son caminos dominados.

$\Rightarrow \exists\, i \in V$, una hoja de T, para el cual p_{si} está dominado (p_{st} está determinado en T).

Y ahora por definición de camino dominado, existe un camino p'_{si} tal que $c^k(p'_{si}) \leq c^k(p_{si})$ para todo $k \in \{1, \ldots, r\}$ y $c^k(p'_{si}) < c^k(p_{si})$ para algún $j \in \{1, \ldots, r\}$.

Esto lleva a contradicción pues implica que T no sea el árbol más corto lexicográficamente ya que hemos encontrado otro camino con costes menores. □

Tras ver esta demostración podemos observar que el árbol más corto lexicográficamente se puede determinar facilmente a partir de cualquiera de los algoritmos disponibles para encontrar el árbol más corto. Obviamente, tendremos que hacerle una breve modificación a dichos algoritmos. Aunque también podemos utilizar el *Algoritmo 1* para determinar el árbol más corto lexicográficamente. Claramente, tras ejecutar del *Algoritmo 1*, la etiqueta más pequeña lexicográficamente de cada nodo $i \in V - \{s\}$ corresponde con el valor de la menor distancia lexicográficamente desde el origen s hasta el nodo i. Una vez que tenemos esas etiquetas, es fácil construir el árbol más corto lexicográficamente pues todos los caminos más cortos lexicográficamente están formados por subcaminos pequeños lexicográficamente.

Consideramos ahora la segunda etiqueta más pequeña lexicográficamente del nodo t, cuyo valor corresponde con la distancia del segundo camino más corto lexicográficamente desde el origen s hasta el nodo t. Cuando llegamos a este punto, sólo podemos considerar dos casos, o esta etiqueta se obtuvo a partir de la menor etiqueta lexicográficamente o a partir de la segunda etiqueta más pequeña lexicográficamente. En este caso, es indiferente empezar a partir de una etiqueta u otra. Como el nodo s sólo tiene una etiqueta por ser el nodo origen, yendo hacia atrás, encontraremos un camino más corto lexicográficamente. Así, el segundo camino más corto lexicográficamente desde s hasta t se determina en un árbol no dominado, adyacente al árbol más corto lexicográficamente.

Con un razonamiento similar, el siguiente lema podría concluir la demostración del *Teorema 2.2*.

Lema 2.5 *Sea p_{st} cualquier camino no dominado pero que no es el camino más corto lexicográficamente, a no ser que el camino más corto lexicográfico no sea único. Existe un camino no dominado p'_{st} adyacente a p_{st} tal que uno de los siguientes casos se verifica:*

1. $c^k(p'_{st}) = c^k(p_{st})$ *para todo* $k \in \{1, \ldots, r\}$.

2. $[c^1(p'_{st}), \ldots, c^r(p'_{st})] <^L [c^1(p_{st}), \ldots, c^r(p_{st})]$.

donde $<^L$ *significa menor lexicográficamente.*

2.3. Algoritmo de aproximación por bases

Una vez demostrado el *Teorema 2.2* vamos a generar con su ayuda un nuevo algoritmo que sólo usa las bases del problema (P1). El *Teorema 2.2* lo que nos quiere decir es que si tenemos un camino no dominado, sólo tenemos que generar todos los caminos adyacentes a él.

La idea general del algoritmo consiste en considerar cualquier árbol T lexicograficamente pequeño y asociarle la variable del dual, $\pi_i^k(T)$, respecto al nodo i para todo $i \in V$, siempre y cuando consideremos $c^k(x) = \Sigma c_{i,j}^k x_{i,j}$ como función objetivo. Primero hacemos $\pi_s^k(T) = 0$ para todo $k \in \{1, \ldots, r\}$ y para cualquier árbol T. Como podemos observar $\pi_i^k(T)$ coincide con el valor $c^k(p_{si})$ donde p_{si} es el camino desde s hasta i en T.

A continuación, se calculan los costes reducidos de T, $\overline{c}_{i,j}^k(T)$, o simplificando la notación $\overline{c}_{i,j}^k$, para cada arco $(i,j) \in A$ y para todo $k \in \{1, \ldots, r\}$ definidos como sigue

$$\overline{c}_{i,j}^k = \pi_i^k - \pi_j^k + c_{i,j}^k$$

tal y como se comentó en la *Sección 1.2.* del *Capítulo 1.*

Después establecemos para algún arco $(i,j) \notin T$ un nuevo árbol adyacente a $T : T' = T \bigcup \{(i,j)\} - \{(h,j)\}$ donde $(h,j) \in T$ si y sólo si el vector formado por los costes reducidos (digamos v) es positivo lexicográficamente, es decir, si la primera componente distinta de cero es positiva. Esto último lo denotaremos como $v >^L 0$. Para dar por finalizado el algoritmo sólo nos quedaría eliminar los árboles que sean dominados pues buscamos aquellos que sean no dominados.

Si formulamos y organizamos estas ideas, la formulación del algoritmo es la siguiente:

Algoritmo 2

Paso 0

1. Determinar T, el árbol más pequeño lexicográficamente.

2. Calcular $[\pi_i^1(T), \ldots, \pi_i^r(T)]$ para todo $i \in V$.

3. Colocar T en la lista LIST.

Paso 1 Si LIST está vacía, entonces el algoritmo ha finalizado. En otro caso, sacar de LIST el árbol más pequeño lexicográficamente T' respecto a $[\pi_t^1(T'), \ldots, \pi_t^r(T')]$ y hallar sus costes reducidos $[\overline{c}_{ij}^1(T'), \ldots, \overline{c}_{ij}^r(T')]$ para todo $(i,j) \notin T'$.

Paso 2 Si p'_{st}, el camino desde s hasta t en T' no pertenece a la lista LIST1, entonces p'_{st} se introduce en LIST1.

Paso 3 Mientras existan $(i,j) \notin T'$ tal que $[\overline{c}_{ij}^1(T'), \ldots, \overline{c}_{ij}^r(T')] >^L 0$ y $\overline{c}_{ij}^k < 0$ para algún $k \in \{1, \ldots, r\}$ o $\overline{c}_{ij}^k = 0$ para todo $k \in \{1, \ldots, r\}$; realizamos:

1. $T = T' \bigcup \{(i,j)\} - \{(h,j)\}$ donde $(h,j) \in T'$.

2. Calcular $[\pi_i^1(T), \ldots, \pi_i^r(T)] \quad \forall\, i \in V$.

3. Si T no pertenece a la lista LIST entonces añadimos T en LIST.

Paso 4 Eliminar de LIST todos los árboles dominados.

Paso 5 Volver al Paso 1.

Usemos este algoritmo para resolver el mismo ejemplo que hemos resuelto con el *Algoritmo 1* y así comprobar que con ambos se obtienen las mismas soluciones eficientes. Por tanto, los dos algoritmos se pueden utilizar indistintamente para obtener caminos Pareto eficientes en redes.

Ejemplo

El árbol más pequeño lexicográficamente en el grafo de la Figura 2.1 es el siguiente:

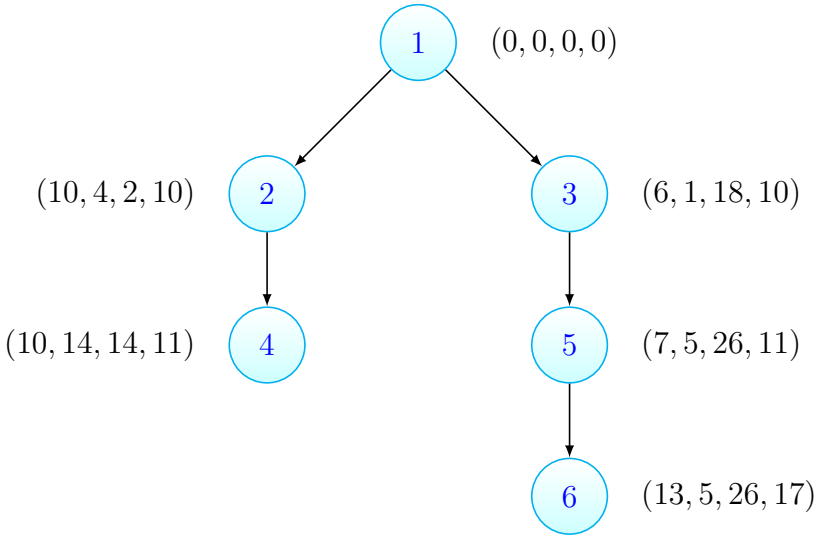

Figura 2.2: T_1

Una vez obtenido el árbol T_1, lo movemos a la lista LIST. Pero al pasar al Paso 1, éste es eliminado de la lista LIST pues es el más pequeño lexicográficamente dentro de ella. Calculemos ahora sus costes reducidos, $\overline{c}^k_{i,j}(T_1)$, para $k = 1, \ldots, 4$ y los recogemos en la siguiente tabla:

Tabla costes reducidos de T_1

$(i,j) \notin T_1$	$\overline{c}^1_{i,j}$	$\overline{c}^2_{i,j}$	$\overline{c}^3_{i,j}$	$\overline{c}^4_{i,j}$
$(3,2)$	2	-1	26	10
$(4,3)$	8	13	-4	4
$(4,6)$	7	10	-11	-5
$(5,4)$	2	-8	15	7

Calculemos los costes reducidos para el primer arco, para ver con claridad las operaciones necesarias. El resto se calculan de la misma forma.

- $\overline{c}^1_{3,2} = \pi^1_3 - \pi^1_2 + c^1_{3,2} = 6 - 10 + 6 = 12 - 10 = 2$
- $\overline{c}^2_{3,2} = \pi^2_3 - \pi^2_2 + c^2_{3,2} = 1 - 4 + 2 = -1$
- $\overline{c}^3_{3,2} = \pi^3_3 - \pi^3_2 + c^3_{3,2} = 18 - 2 + 10 = 26$
- $\overline{c}^4_{3,2} = \pi^4_3 - \pi^4_2 + c^4_{3,2} = 10 - 10 + 10 = 10$

donde $\pi_j^i = c^i(p_{ij})$.

Entonces se obtiene el primer camino $p_{16}^1 = \{1, 3, 5, 6\}$ y se añade a la lista LIST1 $= \{p_{16}^1\}$. En la primera ejecución del Paso 3, como todos los nodos tienen los costes reducidos positivos lexicográficamente se generan los árboles T_2, T_3, T_4 y T_5, y se colocan en la lista LIST $= \{T_2, T_3, T_4, T_5\}$.

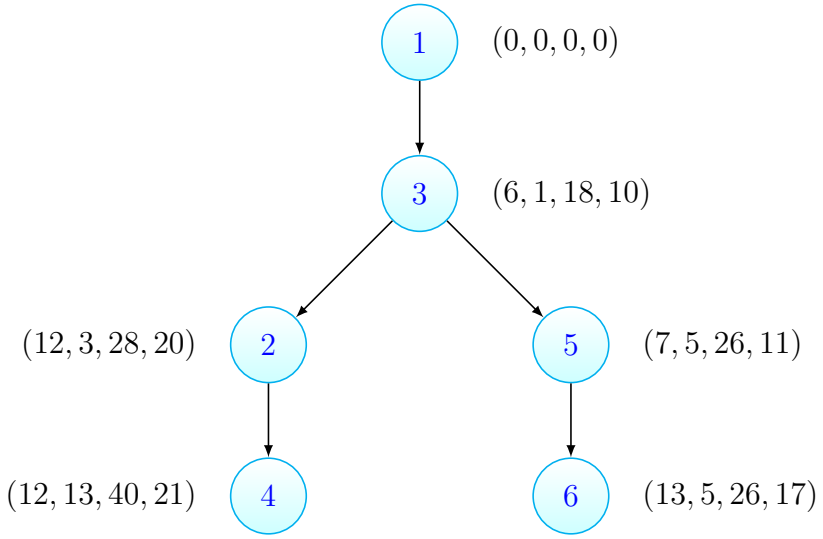

Figura 2.3: T_2

En el Paso 4, T_2 y T_3 son eliminados de la lista LIST porque T_5 D T_2 y T_4 D T_3. Por tanto, la lista quedaría LIST $= \{T_4, T_5\}$. Al volver de nuevo al Paso 1 T_5 es eliminado de la lista LIST debido a que es el más pequeño lexicográficamente y se calculan sus costes reducidos, es decir, $\bar{c}_{i,j}^k(T_5)$ para $k = 1, \ldots, 4$. Al igual que antes, recogemos estos costes en la siguiente tabla:

Tabla costes reducidos de T_5

$(i,j) \notin T_5$	$\bar{c}_{i,j}^1$	$\bar{c}_{i,j}^2$	$\bar{c}_{i,j}^3$	$\bar{c}_{i,j}^4$
$(3,2)$	2	-1	26	10
$(4,3)$	10	5	11	11
$(4,6)$	9	2	4	2
$(2,4)$	-2	8	-15	-7

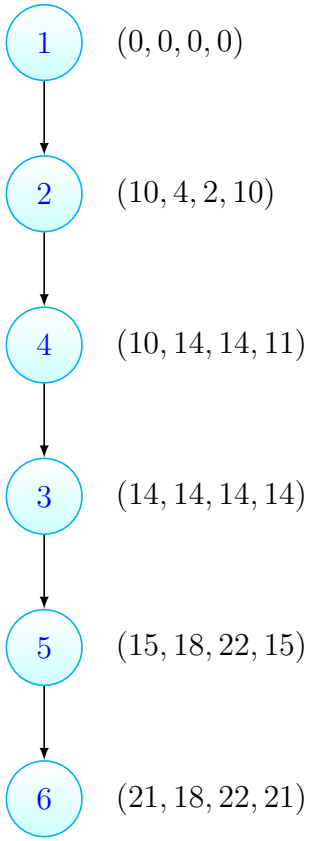

Figura 2.4: T_3

Obtenemos el mismo camino que antes, es decir, $p_{16}^5 = p_{16}^1$, por tanto, en el Paso 3 se genera un único árbol T_6 y tras el Paso 4, nos queda que LIST $= \{T_4, T_6\}$.

Al volver al Paso 1, T_6 es eliminado de LIST pues en este caso es el más pequeño lexicográficamente de la lista y se calculan sus costes reducidos, $\overline{c}_{i,j}^k(T_6)$.

Tabla costes reducidos de T_6

$(i,j) \notin T_6$	$\overline{c}_{i,j}^1$	$\overline{c}_{i,j}^2$	$\overline{c}_{i,j}^3$	$\overline{c}_{i,j}^4$
$(1,2)$	-2	1	-26	-10
$(4,3)$	10	5	11	11
$(4,6)$	9	2	4	2
$(2,4)$	0	7	11	3

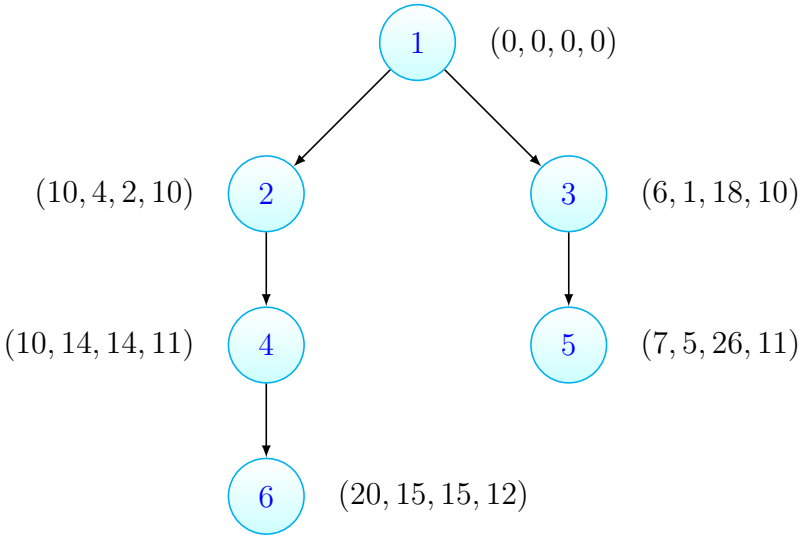

Figura 2.5: T_4

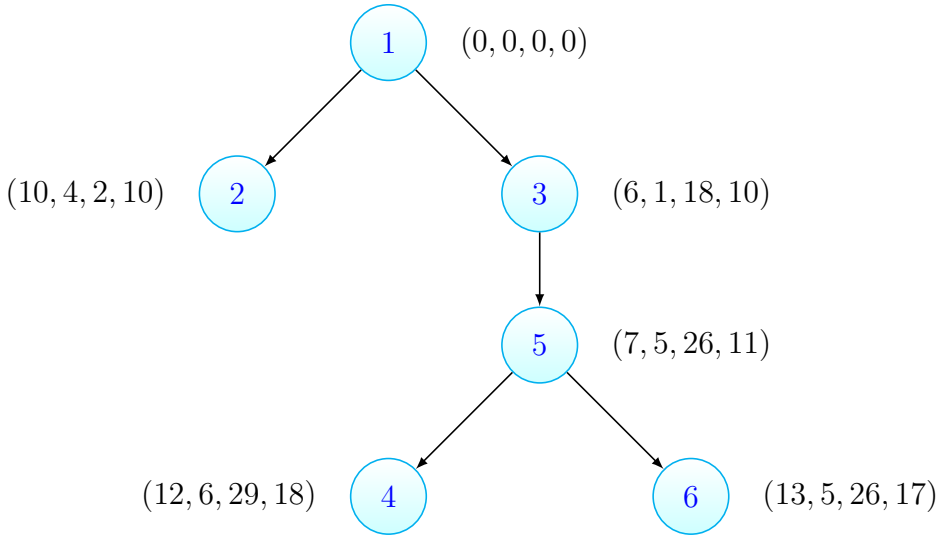

Figura 2.6: T_5

Al igual que antes el camino que obtenemos $p_{16}^6 = p_{16}^1$ y al calcular los costes reducidos de T_6, o todos los costes de los nodos son positivos o tienen la primera componente negativa, por tanto, cuando se ejecuta el Paso 3 no se genera ningún árbol y por ello tras el Paso 4 lo que nos queda es LIST = $\{T_4\}$. Sin embargo al ejecutar el Paso 1 se elimina T_4 de LIST y se calculan

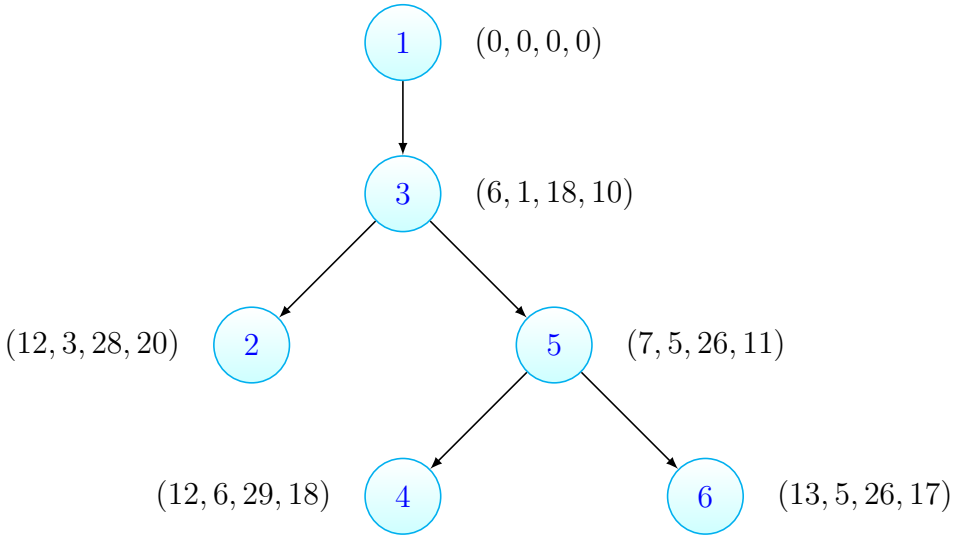

Figura 2.7: T_6

sus costes reducidos, $\overline{c}^k_{i,j}(T_4)$ para $k = 1, \ldots, 4$.

Tabla costes reducidos de T_4

$(i,j) \notin T_4$	$\overline{c}^1_{i,j}$	$\overline{c}^2_{i,j}$	$\overline{c}^3_{i,j}$	$\overline{c}^4_{i,j}$
$(3,2)$	2	-1	26	10
$(4,3)$	8	13	-4	4
$(5,6)$	-7	-10	11	5
$(5,4)$	2	-8	15	7

Tras ejecutar el Paso 2 se tiene un nuevo camino $p^4_{16} = \{1, 2, 4, 6\}$ y por tanto se amplía la lista LIST1 $= \{p^1_{16}, p^4_{16}\}$. En el Paso 3 se generan los árboles T_7, T_8 y T_9 y se colocan en la lista LIST. Pero T_6 D T_7 y T_5 D T_9 entonces al ejecutar el Paso 4, nos queda que LIST $= \{T_8\}$. Sin embargo, en el Paso 1 éste es eliminado de la lista LIST y se calculan sus costes reducidos $\overline{c}^k_{i,j}(T_8)$ para $k = 1, \ldots, 4$.

Tabla costes reducidos de T_8

$(i,j) \notin T_8$	$\overline{c}^1_{i,j}$	$\overline{c}^2_{i,j}$	$\overline{c}^3_{i,j}$	$\overline{c}^4_{i,j}$
$(3,2)$	10	12	22	14
$(1,3)$	-8	-13	4	-4
$(5,6)$	1	3	7	9
$(5,4)$	10	5	11	11

Al ejecutar de nuevo el Paso 3 no se genera ningún árbol pues ya obtenemos todos los costes positivos o la primera componente negativa y la lista LIST está vacía, por tanto, el *Algoritmo 2* ha terminado con p^1_{16} y p^4_{16} como caminos no dominados desde $s = 1$ hasta $t = 6$.

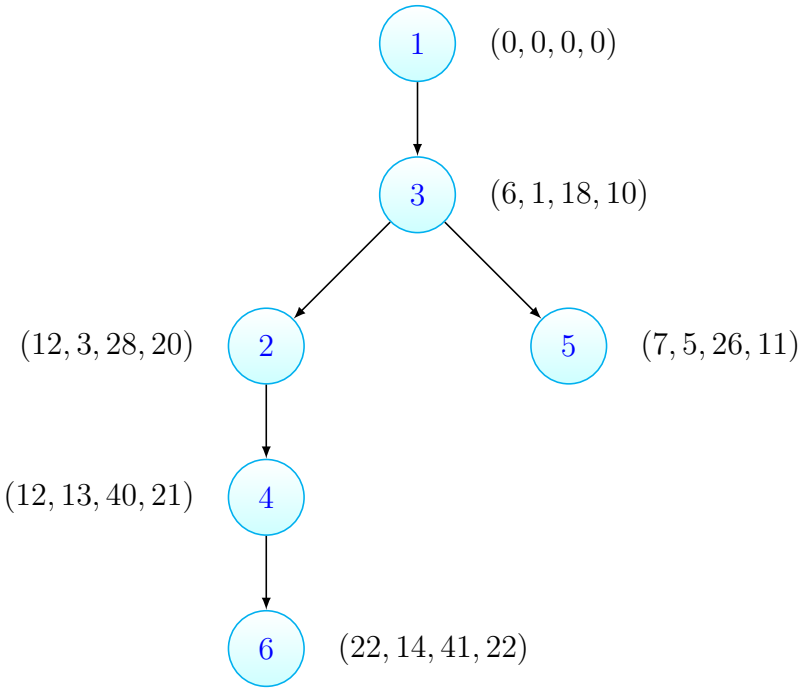

Figura 2.8: T_7

A continuación demostramos la validez del algoritmo.

Demostración.

Supongamos que cuando el algoritmo finaliza existe algún camino no dominado p'_{st} que no ha sido determinado. Como el camino más corto lexicográfico ha sido determinado, aplicando el *Lema 2.5* debe de existir un

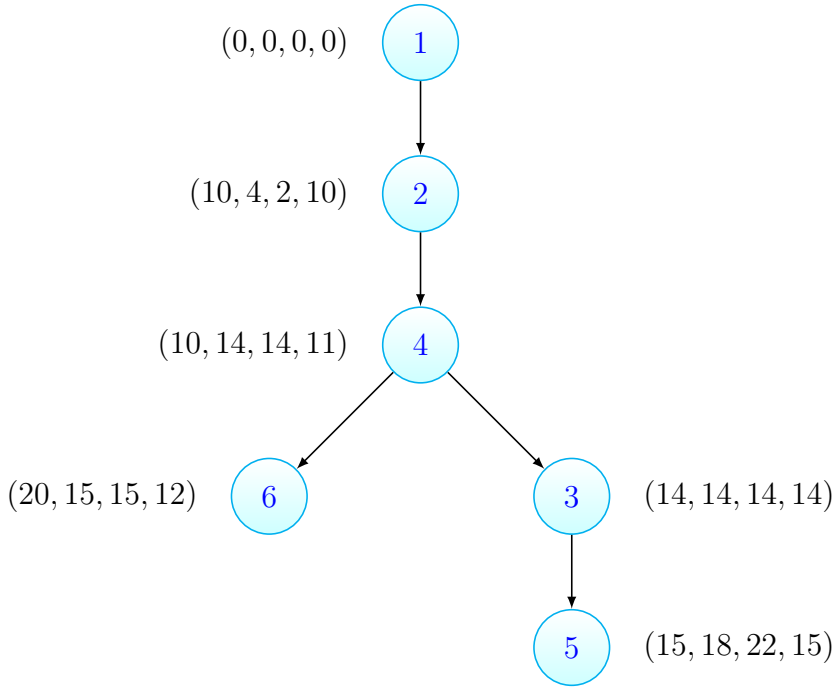

Figura 2.9: T_8

camino no dominado p_{st} adyacente a p'_{st} para el que $[\,c^1(p_{st}), \ldots,\, c^r(p_{st})] <^L [\,c^1(p'_{st}), \ldots,\, c^r(p'_{st})]$ o $c^k(p_{st}) = c^k(p'_{st}) \quad \forall k \in \{1, \ldots, r\}$.

Como el número de caminos no dominados es finito, podemos suponer sin pérdida de generalidad que p_{st} ha sido determinado por el algoritmo. Esto conlleva a que todo árbol no dominado donde p_{st} es determinado, ha sido determinado también por el algoritmo como hemos visto en el paso 3.

Además, ya que $[\,c^1(p_{st}), \ldots,\, c^r(p_{st})] <^L [\,c^1(p'_{st}), \ldots,\, c^r(p'_{st})]$ o $c^k(p_{st}) = c^k(p'_{st}) \quad \forall k \in \{1, \ldots, r\}$, podremos decir que para algún árbol no dominado T debe existir un arco $(i, j) \notin T$ para el que una de las condiciones del Paso 3 se verifica y $j \in p_{st}$. Esto implica que p'_{st} está en un árbol adyacente $T' = T \bigcup \{(i, j)\} - \{(h, j)\}$ donde $(h, j) \in T$. Ahora bien, como el camino p'_{st} no ha pertenecido a la lista LIST1, podemos concluir que T' ha sido eliminado en el Paso 4; esto es posible porque T' es no dominado. De hecho, T y T' sólo se diferencian en dos ramas, una de ellas correspondiente a p'_{st} en T' y la otra a p_{sh} que es un subcamino de p_{st} en T y además es no dominado. □

Nota. Es importante destacar que la condición impuesta en el Paso 3 no es suficiente pero si necesaria para que un árbol adyacente a un árbol no

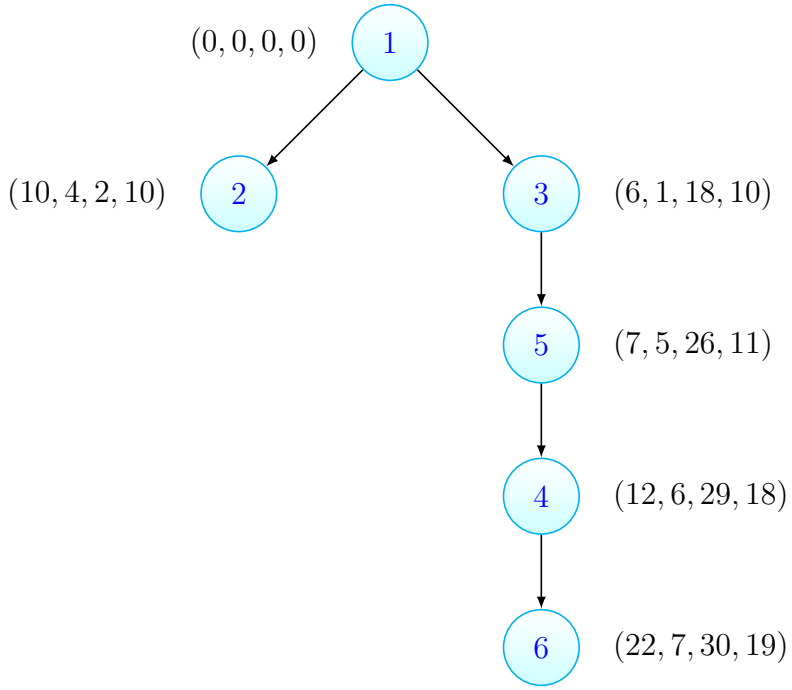

Figura 2.10: T_9

dominado sea también no dominado. Esto hace que los árboles que se generan en el Paso 3 puedan ser dominados y por ello se incluye el Paso 4.

2.4. Aplicaciones

En esta sección vamos a considerar una aplicación directa del problema de determinar caminos eficientes. Se trata de encontrar caminos en una red de comunicaciones que minimicen la distancia a recorrer y la seguridad que ofrecen (probabilidad de no tener accidentes). Aparte de este tipo de aplicaciones existen otras menos evidentes pero muy interesantes como la de determinar caminos más cortos sujetos a una restricción sobre los recursos necesarios para recorrerlos.

Nuestro primer objetivo es encontrar un camino donde la distancia sea mínima, es decir,

$$min \sum_{(i,j)\in A} c_{i,j}\, x_{i,j}$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \; i = s \\ 0 & \text{si } \; i \neq s,t \\ -1 & \text{si } \; i = t \end{cases}$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

como ya vimos en el *Capítulo 1*. Y el segundo, es encontrar un camino lo más seguro posible, es decir,

$$\text{máx} \prod\; p_{i,j}\, x_{i,j}$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \; i = s \\ 0 & \text{si } \; i \neq s,t \\ -1 & \text{si } \; i = t \end{cases}$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

donde $p_{i,j}$ es la probabilidad de no tener accidente en el arco (i,j). Se corresponde con productos porque estamos bajo independencia, ya que buscamos la probabilidad de no tener accidentes en cada uno de los arcos que forman la red.

Como tenemos un producto de probabilidades, podemos tomar logaritmo en él y como es una función creciente nos queda:

$$\text{máx} \sum\; \ln\; p_{i,j}\, x_{i,j}$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \; i = s \\ 0 & \text{si } \; i \neq s,t \\ -1 & \text{si } \; i = t \end{cases}$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

Nos interesa tener un problema de minimizar como hemos visto en el *Capítulo 2* pues queremos encontrar el camino más corto con multicriterios, los cuales se corresponden con distancia y seguridad. El objetivo en el primer problema es minimizar la distancia, por tanto no tenemos que realizar ningún cambio; pero el objetivo de nuestro segundo criterio es maximizar, por tanto multiplicamos por (-1) la función objetivo quedando entonces el siguiente problema equivalente

$$min \ \left(-\sum \ \ln \ p_{i,j}\, x_{i,j}\right)$$

.

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \ i = s \\ 0 & \text{si } \ i \neq s,t \\ -1 & \text{si } \ i = t \end{cases}$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

Como conclusión, el problema que tendríamos que resolver es el siguiente:

$$'min' \left(\sum_{(i,j)\in A} c_{i,j}\, x_{i,j}, - \sum \ \ln \ p_{i,j}\, x_{i,j} \right)$$

$$s.a. \sum_{j:\,(i,j)\in A} x_{i,j} - \sum_{j:\,(j,i)\in A} x_{j,i} = \begin{cases} 1 & \text{si } \ i = s \\ 0 & \text{si } \ i \neq s,t \\ -1 & \text{si } \ i = t \end{cases}$$

$$x_{i,j} \geq 0 \quad \forall\, (i,j) \in A$$

Bibliografía

[1] Ravindra K.Ahuja, Thomas L.Magnanti, James B.Orlin, *Network Flows:theory, algorithms, and applications .*

Ed. Pearson(2014)

[2] Mokhtar S. Bazaraa, John J. Jarvis, Hanif D. Sherali, *Programación Lineal y Flujo en Redes.*

Ed Limusa(1998)

[3] E. Q. V. Martins, *On a multicriteria shortest path problem.*

European Journal of Operational Research 16 (1984) 236-245

[4] Cunningham, W. H., *A network simplex method. Mathematical Programming, 11(1), 105-116.*

[5] A. Ravindran, Don T. Phillips, James J. Solberg, *Operations Research: Principles and practice.*

John Wiley & Sons (1976)

[6] Tao Peng and Xiaowen Wang, *A Mobile-based Navigation Web, Application: Finding the Shortest-time,Path based on Factor Analysis.*

June 2012