INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY
DIVISIÓN DE INGENIERÍA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN INGENIERÍA

TECNOLÓGICO
DE MONTERREY®

SOLUCIÓN EXACTA DEL PROBLEMA DISEÑO DE REDES MULTIPRODUCTO CON ARCOS NO DIRIGIDOS Y CAPACIDAD FINITA

TESIS

PRESENTADA COMO REQUISITO PARCIAL PARA OBTENER EL GRADO ACADÉMICO DE:

MAESTRA EN CIENCIAS CON ESPECIALIDAD EN SISTEMAS DE CALIDAD Y PRODUCTIVIDAD

POR:

ANA ELSA HINOJOSA HERRERA

MONTERREY, N. L. **FEBRERO DE 2007**

INSTTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS MONTERREY

DIVISIÓN DE INGENIERÍA Y ARQUITECTURA
PROGRAMA DE GRADUADOS EN INGENIERÍA

Los miembros del comité de tesis recomendamos que el presente proyecto de tesis presentado por la Lic. Ana Elsa Hinojosa Herrera sea aceptado como requisito parcial para obtener el grado académico de:

MAESTRA EN CIENCIAS CON ESPECIALIDAD EN SISTEMAS DE CALIDAD Y PRODUCTIVIDAD

Comité de tesis:

José Luis González Velarde, Ph. D.
Asesor

Ada M. Álvarez Socarrás, Ph. D
Sinodal

Francisco R. Angel-Bello Acosta, Ph. D.
Sinodal

Aprobado:

Francisco R. Angel-Bello Acosta, Ph. D.
Director del Programa de Graduados en Ingeniería

Febrero 2007

DEDICATORIA

A Dios, a mis padres, hermanos y sobrino

"I am enough of an artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world."

A. Einstein

AGRADECIMIENTOS

Al ITESM Campus Monterrey y CONACYT por otorgarme la beca que me permitió estudiar esta maestría.

Al Dr. Juan Antonio Díaz por proporcionarme un código genérico de programación, para resolver problemas por medio de Branch and Bound.

A todos mis maestros por compartir su conocimiento, tiempo y esfuerzo. Principalmente a mi asesor Dr. José Luis González Velarde y sinodales por guiar mi tesis y apoyarme en el proceso.

A mis amigos por acompañarme, darme momentos inolvidables y lecciones de vida.

Muy en especial a mi familia, por su amor y apoyo incondicional, por creer en mí, permitirme ir tras mis sueños y ser un gran ejemplo.

RESUMEN

A los vértices y arcos de una gráfica frecuentemente se les asocia información cuantitativa, por ejemplo a los vértices lo correspondiente a ofertas y demandas; distancias, longitudes, capacidad y costos en el caso de los arcos. Relativo a estas redes, hay problemas discretos de optimización que aparecen en estadística, ingeniería eléctrica, investigación de operaciones, telecomunicaciones, combinatoria, computación, entre otras disciplinas.

El problema en el que se centrará este trabajo, es el diseño de redes multiproducto, con capacidad fija y arcos no dirigidos (Multicommodity Capacitated Network Design Problem, MCND), el cual se aplica principalmente en las áreas de transporte, ruteo, comunicación y computación, donde varios artículos – bienes, datos, paquetes, personas – se necesitan transportar de un origen a un destino particular, a través de los arcos de una red con capacidades limitadas.

En este problema, se puede considerar dos tipos de costos asociados a cada arco: uno variable, de transportación, que está relacionado con el volumen de cada artículo que fluye a través del arco; y uno fijo, de construcción o utilización, costo que es pagado si se utiliza el arco o se añade la capacidad. Siendo el objetivo principal la minimización del costo total por diseñar y operar la gráfica de flujo.

Una solución óptima del problema, puede ser encontrada por la numeración completa o implícita de todos los caminos posibles, sin embargo consume demasiado tiempo aún para problemas de tamaño moderado, ya que se ha demostrado que este problema pertenece a la clase Np-completo [42]. Además al construir una solución, la compensación entre los dos tipos de costos, así como la interacción entre la capacidad finita de los arcos y los costos fijos, añade dificultad a la resolución eficiente de los problemas [12].

Métodos eficientes de solución son los metaheurísticos y la computación paralela, en particular el procedimiento de Búsqueda Tabú ([25-27]), o GRASP y Búsqueda Dispersa [3] que se utilizan para identificar buenas soluciones factibles para problemas de gran tamaño.

El objetivo y contribución principal de la investigación es la generación de un algoritmo computacional que resuelva este tipo de problemas de manera exacta, basado en un esquema de ramificación y acotamiento (Branch-and-Bound en inglés), combinándolo con la técnica de Generación de Columnas, es decir ramificación y precio (Branch-and-Price en inglés); utilizando el algoritmo de camino más corto, de Floyd y Warshall, para la elección de la columna cuya entrada a la base genere el mayor beneficio a la solución.

Para ilustrar el procedimiento propuesto se presenta un ejemplo del diseño de una red pequeña, además se compara dos estrategias de ramificación ya que esto afecta la eficiencia de los algoritmos en un ambiente Branch-and-Price.

Debido a que se presentan los mayores retos computacionales en problemas de gran escala, al final del trabajo se compara el método propuesto contra uno de resolución basado en un heurístico.

ÍNDICE

CAPÍTULO 1

1 Introducción y Antecedentes

Descripción General del Problema

A los vértices y arcos de una gráfica frecuentemente se les asocia información cuantitativa, como oferta y demanda (para los vértices); distancia, longitud, capacidad y costo (para los arcos). Relativo a estas redes, hay problemas discretos de optimización que aparecen en estadística, ingeniería eléctrica, investigación de operaciones, telecomunicaciones, combinatoria, computación, entre otras disciplinas.

Ejemplos de estos son el diseño de sistemas de telecomunicación a costo mínimo [6], la maximización del rendimiento en un sistema de manufactura [29], la búsqueda de un conjunto de rutas de distribución a costo mínimo [9], o la distribución de electricidad desde un conjunto de puntos generadores hacia puntos de demanda [40]. Dichas aplicaciones han sido estudiadas y se han resuelto por métodos exactos o heurísticos.

El problema en el que se centrará este trabajo, es el diseño de redes multiproducto, con capacidad fija y arcos no dirigidos (Multicommodity Capacitated Network Design Problem, MCND). Las principales áreas de aplicación son: transporte [8], ruteo [32], comunicación [36] y computación [39]. Donde varios artículos – bienes, datos, paquetes, personas – se necesitan transportar de sus respectivos orígenes a sus destinos particulares, a través de los arcos de una red con capacidades limitadas. Se puede considerar dos tipos de costos asociados a cada arco: uno variable, de transportación, que está relacionado con el volumen de cada artículo que fluye a través del arco; y uno fijo, de construcción o utilización, costo que es pagado si se utiliza el arco o se añade la capacidad [12].

El principal objetivo de los problemas MCND es decidir el conjunto de arcos no dirigidos que se incluirá en el diseño final de la red, de manera que a costo mínimo, sea exitosamente transportada la demanda de cada punto, para todos los productos, sin sobrepasar la capacidad de los arcos.

Las formulaciones de diseño de redes presentan retos algorítmicos considerables, especialmente cuando se desea resolver problemas de gran tamaño, ya que usualmente se modelan por medio de optimización lineal entera-mixta con restricciones de flujo y

una gran cantidad de restricciones adicionales. La formulación matemática es combinatoria y usualmente NP-duros (NP-hard en inglés). Además, al construir una solución la compensación entre los dos tipos de costos, así como la interacción entre la capacidad finita de los arcos y los costos fijos, añade dificultad a la resolución eficiente de los problemas [12].

Métodos eficientes de solución son los metaheurísticos y la computación paralela, en particular el procedimiento de Búsqueda Tabú ([25-27]), GRASP y Búsqueda Dispersa [15] se utiliza para identificar buenas soluciones factibles para problemas de gran tamaño.

Antecedentes

Se han obtenido resultados interesantes para problemas de Diseño de Redes con capacidad infinita. Entre otros, Balakrishnan et al., propusieron una familia de algoritmos de ascenso-dual que generaliza los procedimientos conocidos de ascenso para resolver problemas de ruta más corta, localización de plantas, redes de Steiner y árboles de expansión dirigida. Los resultados computacionales muestran que el procedimiento de ascenso-dual y un heurístico "drop-add" asociado, genera soluciones que en la mayoría de los casos garantiza estar dentro del 1 al 4% del óptimo, además de requerir poco tiempo computacional. [5]

El método propuesto por Holmberg et al, en 1998 [30], utiliza la relajación Lagrangiana como subproblema y resuelve el dual del Lagrangeano con optimización de subgradiente, combinándolo con el heurístico primal (el subproblema Benders) para obtener una solución primal factible. Dicho método se probó computacionalmente y los resultados mostraron que es poderoso y supera el estado del arte de los códigos de programación entera mixta, con respecto al tamaño de los problemas y tiempos de solución.

Sin embargo hasta el 2000 se había hecho menor esfuerzo en los problemas capacitados, que aunque son más generales y más convenientes debido a que las restricciones de capacidad se presentan frecuentemente en aplicaciones, el solucionarlos presenta mayor dificultad y retos algorítmicos. En ese año, Holmberg et al. propusieron un método eficiente para resolver este problema, basado en el heurístico Lagrangiano en un marco de Branch-and-Bound [31]. El heurístico utiliza la relajación para obtener fácilmente

soluciones de subproblemas y resolver el dual Lagrangiano por optimización subgradiente. Esto incluye la técnica de encontrar soluciones factibles del primal. El esquema Branch-and-Bound puede utilizarse como un método exacto para garantizar soluciones óptimas del problema o modificarlo para ser un heurístico veloz. El método se probó en redes con diferentes tamaños y es capaz de generar buenas soluciones factibles de problemas de gran tamaño en un tiempo razonable. Probado en 65 problemas, resultó ser mejor o más rápido en 52 casos que el estado del arte de los códigos de CPLEX, hasta ese tiempo.

Crainic et al. en el 2000 [11], presentaron un procedimiento eficiente para determinar cotas superiores estrechas en la solución óptima de problemas capacitados de gran tamaño y con muchos artículos. Obtuvieron soluciones factibles utilizando búsqueda Tabú combinándola con generación de columnas. Los experimentos computacionales realizados mostraron que el método superaba los métodos existentes hasta ese momento.

Posteriormente Crainic y Gendreau (2002) presentaron un modelo cooperativo de búsqueda tabú en paralelo, para costos fijos y multiproducto. La implementación del trabajo en paralelo genera excelentes mejoras en términos de la calidad de las soluciones y tiempos de solución. [12]

Agarwal en 2002 resolvió el problema para distintos costos, capacidades y varios artículos, lo enfocó al contexto de diseño de redes de telecomunicación. El algoritmo utilizado inicia con una solución y gradualmente la mejoraba resolviendo una serie de subproblemas, hasta llegar a un mínimo local. En la mayoría de los problemas, el algoritmo en promedio produjo soluciones dentro del 5% de la cota inferior, sin embargo no es capaz de calcular cotas inferiores para problemas grandes. [1]

En 2003 Ghamlouche et al. presentaron un procedimiento de ciclos nuevos basados en una estructura de vecindades, para metaheurísticas aplicadas a la formulación del problema. En este procedimiento las vecindades son evaluadas y probadas por medio del algoritmo de búsqueda Tabú. Los resultados experimentales muestran que el procedimiento propuesto es de gran poder y supera los métodos existentes divulgados en la literatura hasta ese momento. [21] [22]

También se ha utilizado Relajación Lagrangeana y mecanismos basados en memoria a largo plazo (Crainic et al, 2004). Los experimentos computacionales indican que en problemas de gran tamaño, el método propuesto es competitivo con los mejores heurísticos aplicados a este problema. [13]

De los resultados más actuales, es un procedimiento heurístico que combina GRASP con búsqueda dispersa. Se utilizó GRASP para generar una población de soluciones, punto inicial del procedimiento de la búsqueda dispersa (Álvarez y González-Velarde, 2005). Los resultados computacionales de la aplicación a problemas de gran escala muestran la conveniencia del procedimiento. [3], [15]

Hipótesis

Un algoritmo basado en un esquema Branch-and-Price es efectivo y eficiente para resolver de manera exacta el problema de diseñar una red multiproducto con capacidades y arcos no dirigidos.

Objetivo

El objetivo y contribución principal de la investigación es la generación de un algoritmo computacional que resuelva este tipo de problemas de manera exacta, basado en un esquema de ramificación y acotamiento (Branch-and-Bound en inglés), combinándolo con la técnica de Generación de Columnas, es decir ramificación y precio (Branch-and-Price en inglés); utilizando el algoritmo de camino más corto, de Floyd y Warshall, para la elección de la columna cuya entrada a la base genere el mayor beneficio a la solución.

Justificación

Durante la investigación bibliográfica se observaron distintos métodos de solución para el problema MCND, la gran mayoría basados en heurísticos, que aunque por lo general resuelven el problema en poco tiempo y proveen soluciones muy cercanas a la óptima, no aseguran la obtención de una solución exacta. Esto se debe a que dicho problema pertenece a la clase NP-completo. Sin embargo Díaz, J. y Fernández, E. [17], propusieron un algoritmo eficiente de resolución exacta para otro problema de la misma clase.

CAPÍTULO 2

2 Marco Teórico

Problemas Np-Hard

Los problemas de optimización se pueden dividir en dos categorías: los que tienen variables continuas (continuos), y los que tienen variables discretas (combinatorios). En los problemas continuos, generalmente se busca un conjunto de números reales o incluso una función; en los problemas combinatorios se busca un objeto de un conjunto finito o posiblemente infinito numerable – típicamente un entero, conjunto, permutación o gráfica [37]; se presentan frecuentemente en el diseño de rutas, en la localización de puntos de distribución, balanceo de líneas, entre otros. A pesar de que en principio pueden resolverse por enumeración, en la práctica es difícil o incluso imposible cuando el número de soluciones factibles es extremadamente alto.

La clase P describe a problemas que pueden ser solucionados por algoritmos que requieren en tiempo de resolución a lo más una función polinomial del tamaño de la entrada, es decir que para entradas de tamaño *n,* en el peor de los casos el tiempo de resolución es $O(n^k)$ para alguna constante *k*. Aún casos grandes del problema pueden ser resueltos de manera eficiente con rutinas exactas, ya que incrementos en el tamaño del problema normalmente tienen un impacto pequeño en el tiempo computacional. [10]

La definición de la clase NP está basada en la solución de un problema en lugar de estar basada en el algoritmo. Si la solución correcta (por ejemplo una respuesta afirmativa) puede comprobarse o verificarse en tiempo polinomial, entonces el problema es conocido como polinomialmente no determinístico (NP). A pesar de que los problemas en esta clase son fáciles de verificar, no son fáciles de resolver. Cabe resaltar que la clase NP contiene a la clase P, es decir que cualquier problema que pueda ser resuelto eficientemente, también es verificable de manera concisa. [37]

Por reducción se entiende que un problema X se reduce a X', si un algoritmo de tiempo polinomial para X' implica la existencia de un algoritmo de tiempo polinomial para X. Los problemas para los cuales los problemas NP se reducen polinomialmente se conocen como NP-hard, por lo tanto un algoritmo de tiempo polinomial para un problema NP-hard, producirá un algoritmo para cada miembro de NP al mismo tiempo.

A los problemas que se encuentran en la clase NP-hard y a los de la clase NP, se les llama NP-completos. Existen varios problemas que se han demostrado ser NP-completos, ejemplos son, el problema del agente viajero, el problema de ruteo con ventanas de tiempo (VRPTW) cuando el número de vehículos es fijo, el problema de diseño de red multiproducto con capacidades finitas (MCDN), o problemas de dispersión donde se seleccionan instalaciones para maximizar una cierta función de distancias entre las instalaciones, maximizando la distancia mínima entre cualquier par de instalaciones y la maximización de la distancia media. [33]

Los problemas NP-completos tienen las siguientes propiedades [37]:

1. Ningún problema NP-completo puede resolverse por un algoritmo polinomial conocido.

2. Si hay un algoritmo polinomial para algún problema NP-completo, entonces hay un algoritmo polinomial para todos los problemas NP-completos.

Metaheurísticos

En los últimos años se han propuesto gran número de metodologías que se utilizan para resolver problemas del tipo NP-completo. Una manera es mediante el uso de algoritmos polinomiales que no garantizan dar la mejor solución, pero sí una buena aproximación, a estos algoritmos iterativos se les conoce como heurísticos. Los cuales son argumentos derivados de la experiencia, se utilizan para obtener conocimiento o algunos resultados por medio de proposiciones inteligentes en lugar de alguna fórmula preestablecida [38].

Estas técnicas se caracterizan por buscar el mejor de cada opción, con el fin de obtener una buena solución en un tiempo eficiente y hacer alguna mejora cambiando esta solución por una cercana o por medio de una búsqueda local, aunque queda la posibilidad de quedar atrapados en una solución óptima local y no encontrar el óptimo global. Sin embargo muchos heurísticos tienden a converger a una buena solución en tiempo polinomial.

Los heurísticos también pueden utilizarse dentro del contexto de un algoritmo de optimización exacto, con el fin de aumentar la velocidad del proceso para alcanzar el nivel óptimo. La necesidad de “fortalecer” el algoritmo de optimización se hace más evidente con modelos de gran escala.

Los metaheurísticos son estrategias generales de diseño de procedimientos heurísticos para resolver con alto rendimiento problemas complejos. Estas técnicas han sido exitosamente utilizadas en ciencia e ingeniería. Están basadas en técnicas de búsqueda inteligente, la cual es un nivel más específico de búsqueda, especialmente diseñada para llegar a la información en forma exacta. Surgieron de la observación e imitación del comportamiento de sistemas naturales de adaptación, como selección natural, sistemas físicos y aprendizaje humano (redes neuronales).

Glover y Kochenberger (2003) [28] presentaron un compendio de los metaheurísticos más efectivos y conocidos, entre ellos están: Búsqueda Dispersa, Búsqueda Tabú, Algoritmos Genéticos, Programación Genética, Búsqueda Local Dirigida, GRASP, Optimización de Colonias de Hormigas, Búsqueda Local Iterativa, Métodos de Redes Neuronales para Optimización, Recocido y Simulado, Hyper-Heurísticos y Estrategias Paralelas para Metaheurísticos.

La Búsqueda Dispersa es un método evolutivo que combina restricciones y reglas de decisión. Captura información de puntos originales, utiliza otros métodos heurísticos para evaluar las combinaciones producidas de dichos puntos y para generar nuevos puntos; además utiliza estrategias en vez de la aleatorización.

La Búsqueda Tabú utiliza memoria a largo y corto plazo para buscar soluciones cercanas de calidad y evitar errores recientes.

Algoritmos Genéticos utiliza ideas de biología, como población de cromosomas, selección natural, cruza para producción de descendencia y mutación para diversidad.

El metaheurístico GRASP (Greedy Randomized Adaptative Search Procedure) es un procedimiento de búsqueda miope, aleatorizada y adaptativo. Consiste en dos fases, una de construcción donde se obtiene una solución inicial, y otra de mejora, en la que se realiza una búsqueda local, de la que se guarda la mejor solución examinada.

Los algoritmos de colonia de hormigas son los más populares de los algoritmos bioinspirados, parten del estudio del comportamiento de estos animales, observando la sinergia por la comunicación que se realiza entre los miembros de la colonia a través del ambiente en el que se mueven. De esta forma se puede explicar la realización de obras colectivas sin necesidad de la intervención de una autoridad central.

El enfoque Recocido Simulado, imita el proceso de calentamiento de sólidos hasta que la temperatura esté a punto de cambiarlo de estado y después bajarla lentamente para encontrar un estado de energía baja.

Programación Lineal Entera

La programación lineal entera (IP, Integer Programming) trata programas lineales en los que algunas o todas las variables suponen valores enteros o discretos. Se dice que la IP es mixta o pura si respectivamente alguna o todas las variables están restringidas a valores enteros.

Aunque se han creado varios algoritmos para la IP, ninguno de ellos es totalmente confiable desde el punto de vista del cálculo, sobre todo, cuando el número de variables enteras se incrementa. A diferencia de la programación lineal (LP, Linear Programming) donde problemas con miles de variables y miles de restricciones se pueden resolver en un tiempo razonable, la experiencia de cálculo con la IP, después de más de 30 años de haberse creado, permanece imprecisa.

En LP, el método simplex se basa en aceptar que la solución óptima ocurre en un punto extremo del espacio de soluciones. Este poderoso resultado reduce la búsqueda de la solución óptima de un número infinito a un número finito de soluciones posibles. Por otra parte, la IP comienza con un número finito de puntos solución. Sin embargo, la naturaleza entera de las variables hace difícil diseñar un algoritmo eficaz que localice los puntos enteros factibles del espacio de soluciones. En vista de esta dificultad, los investigadores han creado procedimientos de solución que se basan en el gran éxito obtenido al resolver problemas de LP. Esta estrategia puede resumirse en tres pasos:

1. Relajar el espacio de soluciones del problema entero, ignorando las restricciones enteras por completo. Este paso convierte el IP en un LP regular

2. Resolver el modelo LP "relajado" e identificar su punto óptimo continuo

3. Comenzando con el punto óptimo continuo, agregar restricciones especiales que forcen iterativamente el punto extremo óptimo del modelo LP resultante, hacia las restricciones enteras deseadas

La razón para comenzar la búsqueda del IP óptimo en el LP óptimo continuo, es que existe la posibilidad de que ambas soluciones resulten cercanas entre si, y que de esta manera aumente la posibilidad de localizar rápidamente la solución entera. La característica principal del procedimiento propuesto es que resuelve problemas sucesivos de LP, que son más accesibles desde el punto de vista del cálculo que los problemas de IP.

Existen dos métodos para generar las restricciones especiales que forzan la solución óptima del problema LP relajado, hacia la solución entera deseada:

Método de ramificar y acotar (Branch-and-Bound)

Método del plano de corte

En ambos casos las restricciones agregadas eliminan partes del espacio de soluciones relajado, pero nunca alguno de los puntos enteros factibles. Desafortunadamente no se asegura que alguno de los métodos sea efectivo en la solución de problemas de IP. No obstante, los métodos de ramificar y acotar son mucho mejores en cuanto al cálculo.

Branch-and-Bound

La técnica de resolución para problemas complejos de programación entera mixta (MIP, Mixed Integer Programming) que más frecuentemente se utiliza es Branch-and-Bound. La cual no es realmente un algoritmo, más bien una filosofía para resolver problemas, pertenece a la clase de métodos de enumeración implícita. Fue propuesta por primera vez por A. H. Land y A. G. Doing en 1960 [18]. Una buena referencia es Geoffrion y Marsten [20].

La idea básica es dividir el problema MIP en un conjunto de problemas candidatos, tal que los conjuntos de soluciones de los problemas candidatos sean mutuamente excluyentes y exhaustivos, por lo que tiene la ventaja de encontrar muy probablemente una solución óptima del problema original. A continuación se presenta un algoritmo basado en esta filosofía.

Suponiendo un problema de maximización, donde z es la cota inferior de la solución entera óptima del problema de IP. Inicialmente se fija $z = -\infty$ e $i = 0$.

Paso 1. Agotamiento y ramificación. Seleccionar LP*i* como el próximo subproblema para investigarse. Resolver el LP*i* y tratar de agotarlo utilizando las condiciones apropiadas.

a) Si el LP*i* se declara agotado (solución inferior al incumbente, infactible o entera), actualizar la cota inferior *z* si se encuentra una mejor solución del IP; en otro caso, seleccionar un nuevo subproblema *i* y repetir el paso 1. Si todos los subproblemas se han investigado, detenerse; la solución óptima del IP está asociada con la última cota inferior *z*, en caso de que ésta exista. Si no es así:

b) Si el LP*i* no está agotado, seguir con el paso 2 para efectuar la ramificación del LP*i.*

Paso 2. Ramificación. Seleccionar una de la variables x_j cuyo valor óptimo x_j^* en la solución del LP*i* no satisfaga la restricción de integralidad. Eliminar la región $\lfloor x_j^* \rfloor < x_j < \lfloor x_j^* \rfloor + 1$. (donde [A] define el mayor entero ≤ A), creando dos subproblemas LP que corresponden a las dos siguientes restricciones mutuamente excluyentes:

$$x_j \leq \lfloor x_j^* \rfloor \quad \text{y} \quad x_j \geq \lfloor x_j^* \rfloor + 1$$

Repetir el paso 1.

Tres factores que afectan la eficiencia del método son:

1. *Selección de la relajación del problema*: En vez de resolver cada problema candidato como MIP, se resuelve una relajación de éste. Dos consideraciones importantes en la selección del problema relajado es que debe ser relativamente fácil de resolver y una aproximación muy cercana del original, además los valores óptimos de ambos problemas deben ser cercanos o iguales

2. *Ramificación/Separación del problema:* De los candidatos actuales se crean nuevos problemas candidatos, que son restricciones de los anteriores. Esta ramificación se realiza aprovechando la estructura del problema

3. *Selección del problema:* Es importante realizar la selección de manera que la distancia entre la cota superior e inferior disminuya lo más rápido posible y que se encuentren soluciones factibles. Las cotas permiten la poda de ramas, con lo que no es necesario enumerar todos los subproblemas posibles

Este sistema de búsqueda también se conoce como árbol de búsqueda y existen varias estrategias para podar las ramificaciones y hacer más eficientes las búsquedas. La más sencilla de realizar es analizar todos los nodos de uno por uno hasta encontrar el óptimo, sin embargo es poco eficiente cuando el área de búsqueda es grande.

Generación de Columnas

Los problemas de programación lineal con miles de renglones (restricciones) y/o columnas (variables), son resueltos por variantes del método simplex o por el método de puntos interiores. Sin embargo, para muchos problemas lineales que son de optimización combinatoria, el número de columnas (o renglones en el problema dual) son muchas para ser enumeradas explícitamente. Sin embargo frecuentemente las columnas tienen una estructura que puede ser explotada para generarlas como y cuando lo requiera el método simplex, con lo que se obtienen ventajas computacionales.

Ford y Fulkerson en 1958 fueron los pioneros en proponer un algoritmo en el que algunas variables se tratan de manera implícita. Trabajaron el problema de flujo máximo en una red multiproducto (Maximal Multicommodity Network Flows, MMNF), utilizando varias aplicaciones de algoritmos combinatorios para encontrar el camino más corto entre cada par de nodos en una red. [19]

Dantzig y Wolfe en 1960 desarrollaron una estrategia basada en el método simplex, para extender un programa lineal en el proceso de solución, llamado Generación de Columnas [14]. Éste fue utilizado por primera vez en el contexto de descomposición de problemas lineales de gran escala con alguna estructura especial. La idea básica consiste en iterativamente elegir de un gran número de alternativas, la columna más adecuada e incluirla en la base. Una variante del algoritmo básico que afecta su eficiencia, es reemplazar una columna de la base por la que se incluirá, con lo que se evita que crezca el subproblema. Lübbeke y Desrosiers en 2005 examinaron las más recientes contribuciones de esta estrategia. [35]

La aplicación más famosa de este método fue hecha por Gilmore y Gomory [23] y [24], como parte de un algoritmo heurístico y eficiente para resolver el problema de patrones de corte. Lübbecke, M. y Desrosiers en 2005 presentaron varias áreas de aplicación de la técnica de Generación de Columnas en programación entera. [35]

Branch-and-Price

La combinación de la técnica Generación de Columnas con la de programación lineal basada en un entorno de Ramas y Cotas, fue introducida por Desrosiers, Soumis, y Desrochers en 1984 [16], para resolver un problema de rutas de vehículo con ventanas de tiempo. Esta fue la base en el diseño de algoritmos exactos para problemas enteros de gran escala. Varios años después, Vanderbeck et al. (1996) [41] y Barnhart et al. (1998) [7] retomaron esta combinación (Branch-and-Price) y discutieron aspectos computacionales, reglas de ramificación y maneras eficientes de resolver las relajaciones LP.

En Branch-and-Price, un subconjunto de columnas $I \subset J$ se deja fuera de la relajación LP porque en el problema original son demasiadas columnas para ser manejadas eficientemente y además en la solución óptima muchas de ellas tendrán la variable asociada con valor igual a cero. A partir de este subproblema se calculan los costos reducidos (w_c), se evalúa el criterio de optimalidad y se añade a la base la columna $j \in J$ tal que $z_j - c_j > 0$ ($wa_j - c_j > 0$) para problemas de minimización, la cual producirá una mejora en la función objetivo. A continuación se reoptimiza el LP. La ramificación ocurre cuando no hay columnas que deban entrar a la base y la solución LP no satisfaga las condiciones de integridad. El procedimiento termina cuando se han agotado todas las ramas.

A primera vista, se puede observar que Branch-and-Price no es más que la combinación de dos ideas bien conocidas para resolver programas lineales: Generación de Columnas y Branch-and-Bound. Sin embargo, Appelgren en 1969 [4] observó que esto no es del todo cierto. Hay dificultades al aplicar las técnicas de generación de columnas para programación lineal en métodos de solución de programas enteros [34]. Entre otras, que la ramificación en la programación entera convencional, podría no ser efectiva porque al fijar variables se puede destruir la estructura del problema de precios; además el

resolver al óptimo estos LPs puede no ser eficiente, en este caso se pueden aplicar diferentes reglas para manejar el árbol de Ramificación y Precio.

Camino Más Corto

Existen problemas de optimización combinatoria que pueden formularse y resolverse como problemas de camino más corto, además hay problemas complejos que pueden resolverse con procedimientos que llaman como subrutinas a problemas de camino más corto, este es el caso del problema de MCND, en el cual se utilizará un algoritmo para obtener el camino más corto entre cada par de nodos en una red.

Cada arco y_{ij} de una red tiene asociada una distancia. El problema consiste en encontrar un camino entre dos nodos, tal que sea mínima la suma de las distancia de los arcos que lo conforman.

2.1.1 Algoritmo Floyd-Warshall

Este algoritmo obtiene la matriz de caminos más cortos con cómputos $\mathrm{O}(n^3)$, para una red con *n* nodos. Ya que dada una matriz de distancias $d[i,j]$, se necesita realizar operaciones triples para probar la optimalidad de la solución. Está basado en inducción y utiliza una técnica de programación dinámica. [2]

Dados dos nodos *i* y *j*, la longitud de la trayectoria más corta entre ellos es $d^k[i,j]$, bajo la condición de que la trayectoria sólo utiliza de los primeros *k*-1 nodos. Para esta trayectoria un vértice intermedio es cualquier vértice en la trayectoria, distinto a i, j .

El algoritmo calcula la distancia $d^1[i,j]$ para todos los pares de nodos, a partir de esta distancia calcula $d^2[i,j]$ para todos los pares de nodos y repite este procedimiento hasta obtener $d^{n+1}[i,j]$ para todos los pares de nodos.

Dado $d^k[i,j]$, se calcula $d^{k+1}[i,j]$ utilizando la siguiente propiedad:

$$d^{k+1}[i,j] = \min\left\{d^k[i,j],\ d^k[i,k] + d^k[k,j]\right\}$$

Simultáneamente se va construyendo la matriz Π de predecesores, donde $p_{ij}(k)$ es el predecesor del vértice *j* en un camino más corto desde *i,* con vértices intermedios en el

conjunto $\{1,\ldots, k\}$. Con esta matriz se construye el camino más corto, por medio de la recurrencia:

$$p_{ij}(0) = 0 \text{ si } i = j \text{ ó } a_{ij} = \infty$$

$$p_{ij}(k) = \begin{cases} p_{ij}(k-1) & si\ d^{k-1}[i,j] \le d^{k-1}[i,k] + d^{k-1}[k,j] \\ p_{kj}(k-1) & si\ d^{k-1}[i,j] > d^{k-1}[i,k] + d^{k-1}[k,j] \end{cases}$$

El camino más corto se construye usando la matriz $\Pi^{(n)}$.

CAPÍTULO 3

3 DESCRIPCIÓN DEL PROBLEMA

El objetivo principal del problema MCND es minimizar el costo total por diseñar y operar la gráfica de flujo. Una solución óptima del problema, puede ser encontrada por la numeración completa o implícita de todos los caminos posibles, sin embargo consume demasiado tiempo aún para problemas de tamaño moderado, ya que se ha demostrado que este problema pertenece a la clase Np-completo [42].

En el presente trabajo se propondrá la solución por medio de Branch-and-Price y el uso del algoritmo Floyd-Warshall para obtener la columna que más beneficie el valor objetivo de la función. Se utilizará relajación lineal en el proceso de ramificación.

Formulación del Modelo

Sea $G = (N, A)$ un grafo que representa una red no orientada con un conjunto N de nodos, un conjunto A de arcos no dirigidos (aristas) disponibles en la red y E el conjunto de arcos orientados asociados a las aristas de A. Sea K el conjunto de artículos con demanda d^k para el k-ésimo artículo. Los nodos $O(k)$ y $D(k)$ son los nodos origen y destino del producto k, respectivamente.

Sea $\{i, j\}$ el arco no dirigido entre los nodos i y j, con capacidad u_{ij}, donde F_{ij} es el costo fijo de utilizar el arco $\{i, j\}$ en el diseño de la red y c_{ij}^k el costo variable de transportar una unidad de flujo del artículo k a través del arco dirigido (i, j).

La formulación que se utilizará está basada en caminos, los cuales son una sucesión de nodos, donde cada par de nodos sucesivos pertenece al conjunto E de arcos disponibles. Para cada artículo k, sea L^k el conjunto de caminos que permiten transportar su demanda de origen a destino.

$$\delta_{ijl}^k = \begin{cases} 1 & si\ el\ arco\ (i,j)\ pertenece\ a\ l \\ 0 & en\ otro\ caso \end{cases},$$

donde l es un camino para el producto k.

El costo por transportar una unidad del producto *k* por el camino *l*, es $C_l^k = \sum_{(i,j)\in E} c_{ij}^k \delta_{ijl}^k$.

Las variables relacionadas son y_{ij}, variables de decisión que indican si la arista $\{i, j\}$ es incluida en el diseño final de la red, y h_l^k la cantidad de flujo del producto *k* en el camino $l \in L^k$; x_{ij}^k denota el flujo correspondiente del producto *k* en el arco (i, j), donde $x_{ij}^k = \sum_{l\in L^k} h_l^k \delta_{ijl}^k$.

La formulación del MCND basada en caminos es:

$$\min\left[\sum_{k\in K}\sum_{l\in L^k} C_l^k h_l^k + \sum_{\{i,j\}\in A} F_{ij}\, y_{ij}\right] \tag{3.1}$$

$$\sum_{l\in L^k} h_l^k = d^k \quad \forall k \in K \tag{3.2}$$

$$\sum_{k\in K}\sum_{l\in L^k} h_l^k \left(\delta_{ijl}^k + \delta_{jil}^k\right) \le u_{ij}\, y_{ij} \quad \forall \{i, j\} \in A \tag{3.3}$$

$$h_l^k \ge 0 \quad \forall k \in K; l \in L^k \tag{3.4}$$

$$y_{ij} \in \{0,1\} \quad \forall \{i, j\} \in A \tag{3.5}$$

El objetivo (3.1) captura la compensación básica entre los costos de transportar el producto *k* por el camino *l* y los costos fijos de utilizar los arcos. Cabe resaltar que una red grande puede ser más difícil de construirse, pero podría reducir costos de operación incluyendo más caminos activos origen-destino. De manera inversa, una red pequeña puede incrementar los costos de operación. Las ecuaciones de demanda (una para cada artículo *k*) están representadas por (3.2). En (3.3) se presentan las restricciones de capacidad, el flujo de todos los artículos que circulan en cualquier dirección del arco $\{i, j\}$ no puede exceder la capacidad del arco. Este conjunto de restricciones prohíbe el flujo a través de arcos inactivos, es decir arcos $\{i, j\}$ no incluidos en el diseño $(y_{ij} = 0)$. Y finalmente, las restricciones (3.4) aseguran la no negatividad de las variables h_{ij}^k y (3.5) forza a las variables y_{ij} a valores binarios.

Observaciones Sobre el Problema MCND

Aunque la red sea no dirigida, el flujo es dirigido, por lo que cuando se ha tomado la decisión de conectar los nodos i, j, el flujo es permitido en cualquier dirección, es decir se consideran los arcos (i, j) y (j, i) en el diseño de la red.

En el presente trabajo, los costos variables de transportación asociados a cada arista dependen únicamente del producto, no del sentido en el que está circulando. Además para cada artículo *k,* hay un único par de nodos origen-destino.

El algoritmo propuesto requiere de una solución inicial, la cual estará formada por caminos artificiales: $\{(O(k),\ D(k))\}$. Donde h_{k0} es la variable asignada al camino artificial $\{(O(k),\ D(k))\}$ y y_k la variable de decisión del arco $(O(k),\ D(k))$. Los costos respectivos de utilización y transportación se fijan altos para que no sean parte del diseño final de la red. Las restricciones de capacidad asociadas a las variables artificiales, son:

$$h_{k0} - d^k y_k \leq 0$$

CAPÍTULO 4

4 METODOLOGÍA DE SOLUCIÓN

Debido a que el problema en estudio presenta un reto algorítmico considerable especialmente cuando el problema es de gran tamaño, su resolución se ha abordado con heurísticos, sin embargo se tienen referencias de un problema NP-completo que se resolvió eficientemente con un algoritmo en un escenario de Branch-and-Price, tal es el caso del problema de localización de plantas con una fuente (SSCPLP, Single Source Capacitated Plant Location Problem) [17].

El procedimiento que se presenta para la resolución del problema MCND está formulado en un ambiente Branch-and-Price, donde cada columna que se agrega corresponde a un nuevo camino de $O(k)$ a $D(k)$ y las columnas se generan a partir del algoritmo Floyd-Warshall de camino más corto.

Pseudocódigo

El algoritmo de resolución propuesto, se codificó en C++ versión 6.0 y utiliza el optimizador CPLEX versión 9.1. A continuación se muestra el pseudocódigo del procedimiento.

```
Definir e Inicializar Variables
Lectura de Datos
Hacer{
        Optimizar el subproblema actual;
        Si la solución es factible y el valor objetivo es menor al incumbente:
        {       Si es entera:
                        Actualizar el incumbente y la mejor solución;
                En otro caso:
                        Ramificar sobre y_ij;
        }
        En otro caso:
                La rama está agotada, podarla y hacer backtrack;
} Mientras (La pila no está vacía)
Imprimir la solución, el costo de la red y el tiempo de resolución;
```

Problema y Solución Inicial

Debido a las características del problema, el encontrar una solución inicial factible es complicado, por lo que la solución con la que iniciará el algoritmo será obtenida utilizando variables artificiales.

Siguiendo la notación de la formulación matemática, se definen los parámetros:

E'	El conjunto de arcos $(O(k), D(k))$ para $k \in K$
$(O(k), D(k))$	Arco dirigido del nodo origen $O(k)$ al nodo destino $D(k)$, para $k \in K$
$\{O(k), D(k)\}$	Arco no dirigido entre los nodos $O(k)$ y $D(k)$, para $k \in K$
F_k	Costo fijo de utilizar el arco $\{O(k), D(k)\}$ en el diseño de la red
c_k	Costo variable de transportar una unidad de flujo del producto *k* a través del arco $(O(k), D(k))$
L^k,	Conjunto de caminos originales y artificial que permiten transportar la demanda del producto *k* de origen a destino

Los costos F_k y c_k asociados a los arcos artificiales se fijan de gran magnitud para evitar que formen parte del diseño final de la red. El costo por transportar una unidad del producto *k* por el camino artificial *l*, es $C_l^k = \sum_{(i,j)\in E'} c_k \delta_{ijl}^k$.

Las variables relacionadas son:

y_k	Variable de decisión que indica si la arista artificial $\{O(k), D(k)\}$ es incluida en el diseño final de la red
h_{k0}	La cantidad de flujo del producto *k* en el camino *l* = $\{(O(k), D(k))\}$
x_k	Flujo correspondiente del producto *k* en el arco artificial $(O(k), D(k))$

Con esta notación, el problema inicial en el algoritmo es:

$$\min\left[\sum_{k\in K}\sum_{l\in L^k,} C_l^k h_l^k + \sum_{\{i,j\}\in A} F_{ij}\, y_{ij} + \sum_{k\in K} F_k y_k\right] \tag{4.1}$$

$$-u_{ij} y_{ij} \le 0 \quad \forall\{i,j\} \in A \tag{4.2}$$

$$h_{k0} = d^k \quad \forall k \in K \tag{4.3}$$

$$h_{k0} - d^k\, y_k \le 0 \tag{4.4}$$

$$h_l^k \ge 0 \quad \forall k \in K; l \in L^k$$

$$h_{k0} \ge 0 \quad \forall k \in K \tag{4.5}$$

$$y_{ij} \in [0,1] \quad \forall\{i,j\} \in A$$

$$y_k \in [0,1] \quad \forall k \in K \tag{4.6}$$

Las ecuaciones en (3.3) se reducen a las de (4.2) y se satisfacen automáticamente por las restricciones (4.6), esto porque inicialmente no se cuenta con caminos no artificiales en la formulación, pero conforme se añadan columnas (caminos), esta restricción dejará de cumplirse de manera automática. De manera análoga las restricciones en (3.2) corresponden a las de (4.3).

Las restricciones (4.4) aseguran que el flujo que pasa por los caminos artificiales sea menor o igual a la demanda del producto, y que no haya flujo cuando la arista artificial no esté incluida en el diseño de la red.

Obsérvese la relajación lineal de las variables de decisión y lo sencillo que es obtener la solución optima de este problema, la cual es factible para el problema inicial, pero no para el original. Dicha solución es:

$$\begin{cases} y_{ij} = 0 \quad \forall\{i,j\} \in A \\ y_k = 1 \quad \forall k \in K \\ h_l^k = 0 \quad \forall k \in K;\, l \in L^k \\ h_{k0} = d^k \quad \forall k \in K \end{cases}$$

Es claro que el valor de la función en la solución inicial es muy grande, sin embargo a lo largo del algoritmo se irá reduciendo al eliminar caminos artificiales de la solución óptima de los subproblemas

Criterio de Optimalidad

Sea J el conjunto de columnas (caminos) del problema original. Sea w el vector de precios, donde w_i corresponde a la i-ésima restricción. El costo relativo a cada columna no básica a_j está dado por $\bar{c}_j = c_j - wa_j$.

Basándonos en el método simplex, para problemas de minimización, el criterio de optimalidad asegura que si $z_j - c_j \leq 0 \quad \forall j \in J$, entonces se ha llegado al óptimo. Se sabe que las columnas en J son demasiadas como para hacer la prueba explícita a cada una, sin embargo:

$$z_j - c_j \leq 0 \ \forall j \in J \quad \text{es equivalente a} \quad \max_{j \in J}(z_j - c_j) \leq 0$$

Lo cual sugiere resolver el problema de búsqueda de la columna *j,* resolviendo el problema de maximización. Utilizando las propiedades de máximos y mínimos, se tiene la siguiente equivalencia:

$$-\min_{j \in J}(-(z_j - c_j)) \leq 0 \equiv \max_{j \in J}(z_j - c_j) \leq 0$$

Finalmente, el criterio de optimalidad que se utilizará es:

$$\min_{j \in J}(-wa_j + c_j) \geq 0$$

recordando que $z_j = wa_j$.

Elección de la Nueva Columna

La columna que entre a la base debe ser tal que se obtenga el mayor beneficio posible en la función objetivo. Para encontrarla se resolverá un problema de minimización – el de camino más corto – aplicado a una red donde los costos de los arcos están dados por $-wa_j + c_j$. Se utilizará el algoritmo Floyd-Warshall, el cual requiere las siguientes matrices.

Para el producto *k*, la matriz d_k^0 es la matriz inicial de distancias, tal que:

$$d_k^0[i][j] = \begin{cases} 0 & \textit{Para } i = j, \\ -w_i + c_{ij}^k & \textit{Para los arcos disponibles}, \\ \infty & \textit{Para los arcos no disponibles} \end{cases}$$

donde c_{ij}^k es el costo variable de transportar una unidad de flujo del producto *k* por el arco (i, j).

La matriz Π_k^0 es la matriz inicial de vértices predecesores, donde:

$$\Pi_k^0[i][j] = i$$

Utilizando las matrices d_k^0, Π_k^0 y el siguiente pseudocódigo, se puede obtener el camino más corto entre cada par de nodos, en particular entre los nodos $O(k)$ y $D(k)$, para cada producto *k*.

```
Para (Todos los nodos, m)
    Para (Todos los nodos inicio de los arcos, i)
        Si(m ≠ i)
            Para(Todos los nodos final de los arcos, j)
                Si( j ≠ m && j ≠ i)
                    Si (d[i][j] > d[i][m] + d[m][j])
                    {   d[i][j] = d[i][m] + d[m][j];
                        π[i][j] = π[m][j];
                    }
```

Al tener los *k* caminos más cortos, se evalúa el criterio de optimalidad para cada uno y se elige el que genere mayor beneficio a la función objetivo. A continuación se muestra el pseudocódigo utilizado.

```
nomb[2] = 0;                          // Producto del camino que se incluirá
optimo[nomb[2]] = ∞;
cols = 0;                             // Número de columnas que cumplen el criterio
Hacer(Para todos los productos, k)
{   optimo[k] = 0;
    Para(Todos los renglones, i)      // Calcula –wa_j
        optimo[k] = -w[i]*a[i][k] + optimo[k];
    optimo[k] = optimo[k] + costo_camino[k];
```

```
        Si(optimo[k] ≥ 0)                    // Evalúa el criterio de optimalidad
                cols = cols +1;
        Si(optimo[k] < optimo[nomb[2]]) // Se elige la columna que genere mayor
                nomb[2] = k;                    beneficio a la función objetivo.
}
```

Dicha columna no básica corresponde a un camino por el que puede fluir el producto k, de $O(k)$ a $D(k)$. El introducirla a la base generará una disminución en el costo total de la operación y construcción de la red de distribución que se está diseñando, para añadirla se utiliza el siguiente pseudocódigo.

```
Si(Se satisface el criterio de optimalidad para los k caminos)
        nomb[2] = 0;        //Indica a la función main que no se agregó una columna
En otro caso:
        Añadir la columna del camino que va de O(k) a D(k).
```

Ramificación y Agotamiento

Dependiendo de la solución óptima que se va obteniendo para los subproblemas, se toma la decisión de ramificar o agotar cada rama.

Ramificación

Lo cual consiste en seleccionar una de las variables y_{ij}, cuyo valor óptimo en la solución del subproblema en estudio, no satisfaga la restricción de integralidad. Y a partir de ésta crear dos subproblemas LP que correspondan a dos restricciones mutuamente excluyentes, del tipo:

$$y_j \leq \left[y_j^*\right] \quad \text{y} \quad y_j \geq \left[y_j^*\right]+1$$

Las estrategias de ramificación pueden ser sencillas, como por ejemplo elegir ramificar sobre la primera variable fraccionaria que aparezca, o sobre la variable fraccionaria que más cercana esté de 1. Sin embargo para fines de eficiencia algorítmica, se recomienda ramificar tomando en cuenta las características particulares de la formulación lineal del problema.

Se probaron distintos métodos de ramificación (Apéndice A), siendo el mejor el que se expresa en el siguiente pseudocódigo:

```
Si(La solución es factible y menor al incumbente)
{   sta = 3;        // Estado de la solución. sta = 3: Solución factible y no entera
    Si (La solución es entera)
    {  incumbente = valor_obj_actual;     // Actualizar el incumbente
       Para(Todas las columnas, j)
          y*[j] = x*[j];                  // Actualiza la mejor solución
       sta = 2;                           // Estado: Solución factible y entera
       Imprimir el valor objetivo del incumbente, la iteración y el tiempo de cómputo
    }
    En otro caso:      // Encontrar el camino con el que se fijen más arcos
    {   Para(Todos los caminos que están en la solución, i)
       {  cumple[i] = 0;       // Inicializar contador
          Para(Todos los arcos, j)
          {    Si (En el camino i existe el arco j && Si x*[j] > 0 &&
               no se ha marcado antes el arco)
               {       cumple[i] = cumple[i] + 1;
                       costo = costo + costo fijo del arco j;
               }
               Si(cumple[i] > 0)   //Considera el costo de construir esos arcos:
                        cumple[i] = cumple[i]/costo;
               Si(cumple[i] > cumple[k])
                   k = i;
     }
     camino_rama = k;        // Es el camino al que pertenece el arco a ramificar
     j = -1;
     Para(Todos los arcos, xfrac)
        Si(En el camino i existe el arco xfrac && no se había fijado)
       {    Si (j = =-1)
              j = xfrac;
            Si(x*[xfrac] no es entero)
              j = xfrac;
       }
     xfrac = j;                          // Variable que alimentará la pila
```

```
}
En otro caso:
{   Si( la solución no es factible)
        sta = 1;                    // La solución no es factible
    En otro caso
        sta = 4;                    // La cota inferior excede el incumbente
    Y la rama queda agotada.
}
```

El fijar un arco $y_{ij} = 1$ significa que forma parte de la solución y que por él fluye un producto k. Esto requiere que y_{ij} pertenezca a un camino l_k de $O(k)$ a $D(k)$. Basándonos en esta observación, se decidió fijar en 1 un arco del camino que sea más económico construir. Lo que genera dos subproblemas excluyentes: uno que contiene al arco y_{ij} y podría contener al camino l_k, y el que no contiene a y_{ij}. Obsérvese que la restricción $y_{ij} = 0$ implica que l_k no pertenece al diseño de la red para ese subproblema.

Backtrack

Se realiza el backtrack cuando se poda una rama, por cualquiera de los siguientes criterios.

Optimalidad: La solución actual es factible y entera.

Infactibilidad: La solución actual no es factible.

Cota inferior > *Incumbente*: La cota inferior del problema actual es mayor que el incumbente, por lo tanto no encontraremos una mejor solución de esa rama.

El pseudocódigo para el backtrack es el siguiente:

```
Si(La pila no está vacía)
        Sea y_ij = 0 el nodo superior de la pila;
En otro caso
        Terminar = 1;                   // Ya se han analizado todas las ramas
Si(Terminar != 1)
{       Liberar el arco que se había fijado como y_ij = 1.
        Fijar y_ij  = 0;
        Para(Las ramas debajo del nodo y_ij = 1)
                Relajarlas, es decir acotar la variable entre 0 y 1;
```

```
        nomb[2] = 1;  // Permitirá la búsqueda de nuevas columnas;
}
```

Ejemplo Ilustrativo

Para ilustrar el procedimiento propuesto, considérese la siguiente red de distribución para dos productos:

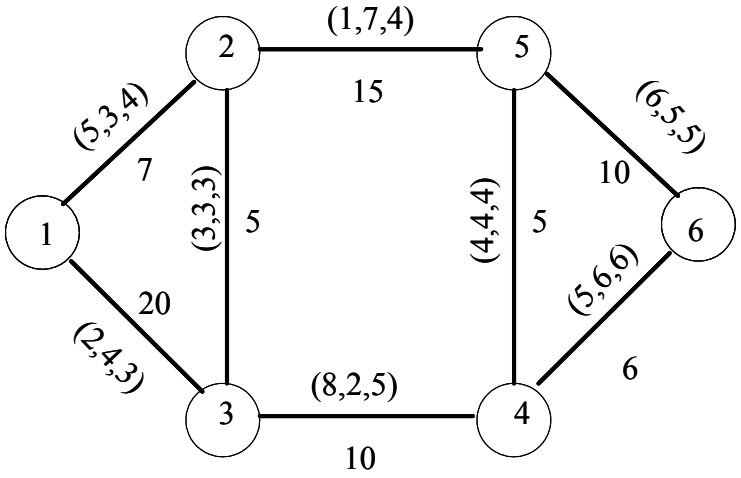

Figura 4-1. Gráfica del Ejemplo Ilustrativo

Donde $(c_{ij}^{(1)}, c_{ij}^{(2)} f_{ij})$ son los costos respectivos. $c_{ij}^{(k)}$ representa el costo unitario de transportación del producto k que se mueve a través del arco (*i, j*) y f_{ij} el costo fijo por utilizar el arco (*i, j*) en el diseño de la red. La capacidad del arco (*i, j*) es representada por u_{ij} .

El primer producto se requiere transportar del nodo 1 (O(1) = 1) al nodo 5 (D(1) = 5) y el segundo del nodo 2 (O(2) = 2) al nodo 6 (D(2) = 6). La demanda correspondiente es de 10 y 5.

Considérese los siguientes arcos artificiales (1, 5) y (2, 6), las respectivas variables de decisión y_1 y y_2, los caminos asociados: h_{10} = {(1, 5)} y h_{20} = {(2, 6)}. Se tomará 100 como costo fijo de utilización de cada arco artificial, por lo que el costo por transportar una unidad de producto por el camino respectivo es 100.

El problema inicial a resolver es:

Minimize

obj: $4\,y_{12} + 3\,y_{13} + 3\,y_{23} + 4\,y_{25} + 5\,y_{34} + 4\,y_{45} + 6\,y_{46} + 5\,y_{56} + 100\,y_1 + 100\,y_2 + 100\,h_{10} + 100\,h_{20}$

Subject To

c1: $-7\,y_{12} \leq 0$

c2: $-20\,y_{13} \leq 0$

c3: $-5\,y_{23} \leq 0$

c4: $-15\,y_{25} \leq 0$

c5: $-10\,y_{34} \leq 0$

c6: $-5\,y_{45} \leq 0$

c7: $-6\,y_{46} \leq 0$

c8: $-10\,y_{56} \leq 0$

c9: $h_{10} = 10$

c10: $h_{20} = 5$

c11: $-10\,y_1 + h_{10} \leq 0$

c12: $-5\,y_2 + h_{20} \leq 0$

Bounds

$0 \leq y_{12} \leq 1$

$0 \leq y_{13} \leq 1$

$0 \leq y_{23} \leq 1$

$0 \leq y_{25} \leq 1$

$0 \leq y_{34} \leq 1$

$0 \leq y_{45} \leq 1$

$0 \leq y_{46} \leq 1$

$0 \leq y_{56} \leq 1$

$0 \leq y_1 \leq 1$

$0 \leq y_2 \leq 1$

$0 \leq h_{10} \leq 10$

$0 \leq h_{20} \leq 5$

End

Iteración 1.1

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

$w_1 = 0.0$ $w_2 = 0.0$ $w_3 = 0.0$ $w_4 = 0.0$ $w_5 = 0.0$ $w_6 = 0.0$

$w_7 = 0.0$ $w_8 = 0.0$ $w_9 = 110.0$ $w_{10} = 120.0$ $w_{11} = -10.0$ $w_{12} = -20.0$

Para cada producto se buscará el camino más corto entre su origen y destino, utilizando el algoritmo Floyd-Warshall.

A los arcos que no están disponibles en la red, se les penaliza con costo de 400.0, por ejemplo para el arco (1, 4): $d_{14}^0 = 400.0$. Las distancias de los arcos disponibles se calcularon de la siguiente manera:

$$-wa_j + c_j,$$

Por ejemplo para el producto 1 y los arcos (2, 3) y (1, 5) se tiene: $d_{23}^0 = -0 + 3.0$ y $d_{15}^0 = -(-10) + 100$ respectivamente.

Camino para el producto 1:

A continuación se muestran la primera matriz de distancias mínimas y la de predecesores:

$$d^0 = \begin{bmatrix} 0.0 & 5.0 & 2.0 & 400.0 & 110.0 & 400.0 \\ 5.0 & 0.0 & 3.0 & 400.0 & 1.0 & 120.0 \\ 2.0 & 3.0 & 0.0 & 8.0 & 400.0 & 400.0 \\ 400.0 & 400.0 & 8.0 & 0.0 & 4.0 & 5.0 \\ 110.0 & 1.0 & 400.0 & 4.0 & 0.0 & 6.0 \\ 400.0 & 120.0 & 400.0 & 5.0 & 6.0 & 0.0 \end{bmatrix}$$

$$\Pi^0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 2 & 0 & 2 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 & 0 & 0 \\ 0 & 0 & 4 & 0 & 4 & 4 \\ 5 & 5 & 0 & 5 & 0 & 5 \\ 0 & 6 & 0 & 6 & 6 & 0 \end{bmatrix}$$

La matriz final de distancias mínimas y predecesores, obtenida por medio del algoritmo Floyd-Warshall es:

$$d^6 = \begin{bmatrix} 0.0 & 5.0 & 2.0 & 10.0 & 6.0 & 12.0 \\ 5.0 & 0.0 & 3.0 & 5.0 & 1.0 & 7.0 \\ 2.0 & 3.0 & 0.0 & 8.0 & 4.0 & 10.0 \\ 10.0 & 5.0 & 8.0 & 0.0 & 4.0 & 5.0 \\ 6.0 & 1.0 & 4.0 & 4.0 & 0.0 & 6.0 \\ 12.0 & 7.0 & 10.0 & 5.0 & 6.0 & 0.0 \end{bmatrix}$$

$$\Pi^6 = \begin{bmatrix} 0 & 1 & 1 & 3 & 2 & 5 \\ 2 & 0 & 2 & 5 & 2 & 5 \\ 3 & 3 & 0 & 3 & 2 & 5 \\ 3 & 5 & 4 & 0 & 4 & 4 \\ 2 & 5 & 2 & 5 & 0 & 5 \\ 2 & 5 & 2 & 6 & 6 & 0 \end{bmatrix}$$

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -110 + 6 = -104$.

Camino para el producto 2:

De manera análoga, la matriz final de distancias mínimas y predecesores es:

$$d^6 = \begin{bmatrix} 0.0 & 3.0 & 4.0 & 6.0 & 10.0 & 12.0 \\ 3.0 & 0.0 & 3.0 & 5.0 & 7.0 & 11.0 \\ 4.0 & 3.0 & 0.0 & 2.0 & 6.0 & 8.0 \\ 6.0 & 5.0 & 2.0 & 0.0 & 4.0 & 6.0 \\ 10.0 & 7.0 & 6.0 & 4.0 & 0.0 & 5.0 \\ 12.0 & 11.0 & 8.0 & 6.0 & 5.0 & 0.0 \end{bmatrix}$$

$$\Pi^6 = \begin{bmatrix} 0 & 1 & 1 & 3 & 2 & 4 \\ 2 & 0 & 2 & 3 & 2 & 4 \\ 3 & 3 & 0 & 3 & 4 & 4 \\ 3 & 3 & 4 & 0 & 4 & 4 \\ 2 & 5 & 4 & 5 & 0 & 5 \\ 3 & 3 & 4 & 6 & 6 & 0 \end{bmatrix}$$

El camino más corto del nodo 2 al 6 es: {6, 4, 3, 2}. La columna j asociada a éste es: (0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T. Evaluando el criterio de optimalidad: $-wa_j + c_j = -120 + 11 = -109$.

Se decide añadir la columna (0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T, que corresponde a un camino de distribución para el producto 2.

El problema resultante es:

Minimize

obj: 4 y_{12} + 3 y_{13} + 3 y_{23} + 4 y_{25} + 5 y_{34} + 4 y_{45} + 6 y_{46} + 5 y_{56} + 100 y_1 + 100 y_2 + 100 h_{10} + 100 h_{20} + 11 h_{21}

Subject To

c1: - 7 y_{12} <= 0

c2: - 20 y_{13} <= 0

c3: - 5 y_{23} + h_{21} <= 0

c4: - 15 y_{25} <= 0

c5: - 10 y_{34} + h_{21} <= 0

c6: - 5 y_{45} <= 0

c7: - 6 y_{46} + h_{21} <= 0

c8: - 10 y_{56} <= 0

c9: h_{10} = 10

c10: h_{20} + h_{21} = 5

c11: - 10 y_1 + h_{10} <= 0

c12: - 5 y_2 + h_{20} <= 0

Bounds

0 <= y_{12} <= 1

0 <= y_{13} <= 1

0 <= y_{23} <= 1

0 <= y_{25} <= 1

0 <= y_{34} <= 1

0 <= y_{45} <= 1

0 <= y_{46} <= 1

0 <= y_{56} <= 1

0 <= y_1 <= 1

0 <= y_2 <= 1

$0 <= h_{10} <= 10$

$0 <= h_{20} <= 5$

$0 <= h_{21}$

End

Iteración 1.2

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

$w = (0.0,\ 0.0,\ -0.6, 0.0, -0.5, 0.0, -1.0, 0.0, 110.0, 13.1, -10.0, -20.0)$

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -110 + 6 = -104$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -13.1 + 12 = -1.1$.

Se decide añadir la columna $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$, que corresponde a un camino de distribución para el producto 1.

Iteración 1.3

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

$w = (-103.733, 0.0, -0.6, -0.266, -0.5, 0.0, -1.0, 0.0, 110.0, 13.1, -10.0, -20.0)$

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 3, 1}. La columna j asociada a éste es: $(0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -103.133$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -0.833$.

Se decide añadir la columna $(0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$, que corresponde a un camino de distribución para el producto 1.

Iteración 1.4

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-103.733, 0.0, -103.733, -0.266, -0.5, 0.0, -1.0, 0.0, 110.0, 116.233, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 4, 3, 1}. La columna j asociada a éste es: $(0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -95.5$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -103.966$.

Se decide añadir la columna $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$, que corresponde a un camino de distribución para el producto 2.

Iteración 1.5

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-0.75, -0.15, -0.6, -0.266, -0.5, 0.0, -1.0, -0.5, 7.0166, 12.766, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Ya se satisface el criterio de optimalidad por lo que no se añadirán columnas.

En la Figura 4.2 se observa la red de caminos disponibles para este subproblema. Donde, por ejemplo ▬ ▬ representa al camino {(1, 3), (3, 2), (2, 5)}, y ..... a {(2, 3), (3,4), (4, 6)}.

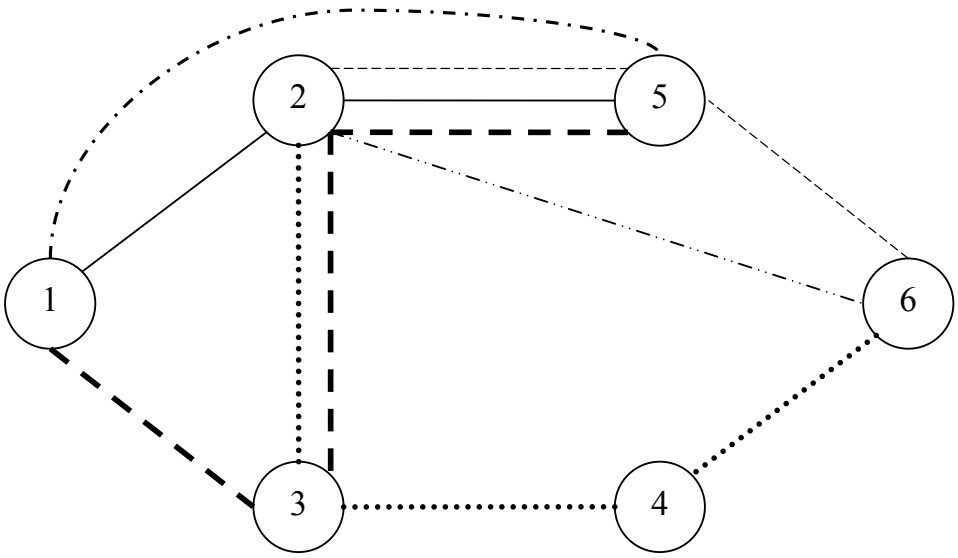

Figura 4-2. Caminos Disponibles en la Iteración 1

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1, 0.15, 0.6, 1, 0, 0, 0, 0.5, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 7, 3, 5)$$

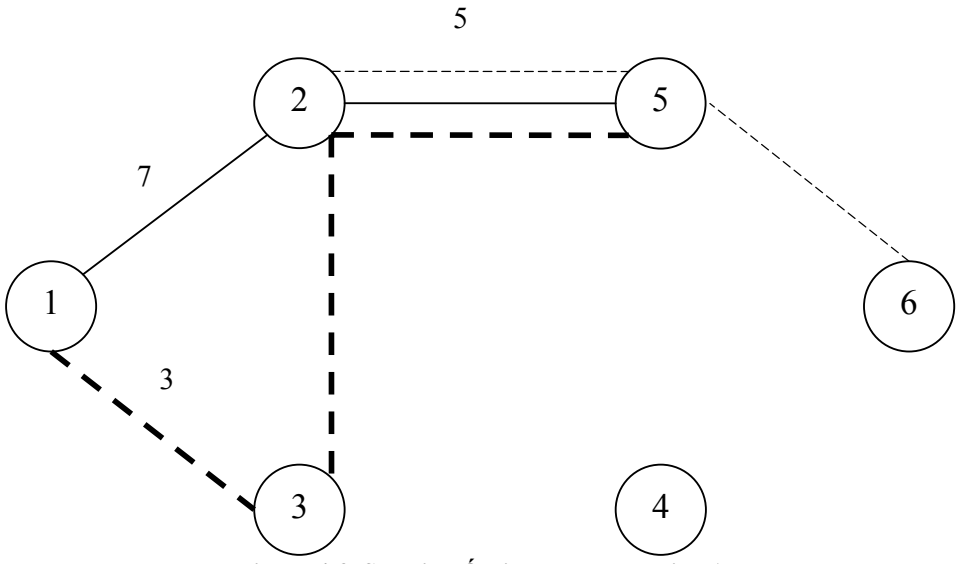

Figura 4-3. Solución Óptima en la Iteración 1

Obsérvese que la solución es factible ya que no se utilizan los arcos artificiales, sin embargo no es entera.

Paso [iii]

La solución óptima actual es

$$\boldsymbol{y} = (1, 0.15, 0.6, 1, 0, 0, 0, 0.5, 0, 0)$$

$$\boldsymbol{h} = (0, 0, 0, 7, 3, 5)$$

Las columnas de los caminos disponibles y no artificiales son:

$$C_1 = (0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$$

$$C_2 = (1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$$

$$C_3 = (0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$$

$$C_4 = (0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$$

Hasta ahora todos los arcos están libres, es decir no se ha ramificado sobre ellos, por lo que el siguiente cálculo se hará para los 10 arcos.

Sobre *i*, para los 4 caminos:

Si $h(C_i) > 0$

Sobre *j*, los 10 arcos:

Si $y(j) > 0$ y $C_i(i) = 1$

Cumple[*i*] ++;

Para esta iteración, el primer camino no artificial no se encuentra en la solución.

El segundo camino no artificial está en la solución, ya que h(C2) = 7. Además los primeros 4 arcos y el 8º están en la solución; los arcos 1, 4 y 7 en el camino C2. Coinciden siendo positivos para los arcos 1 y 4, por lo que Cumple[2] = 2 y el costo de construir estos dos arcos es 8.

Haciendo el mismo cálculo para los 4 arcos se obtiene el vector:

Cumple = (0, 2, 3, 2)

Dividiendo estos valores positivos por el costo total de construir los arcos correspondientes, es decir:

(0, 8, 10, 9)

se obtiene el vector:

Cumple = (0, 2/8, 3/10, 2/9)

= (0, 0.25, 0.3, 0.22)

Y se elige el camino con el valor mayor, es decir C_3. Este camino es el que contiene el mayor número de arcos nuevos para el árbol de búsqueda y el más económico si se construyeran todos estos. Con esto se busca construir el mayor número de arcos pero a menor costo.

La variable que se fijará a 1, es la primera del camino C_3 que esté en la solución. En este caso la segunda, y se alimenta la pila con la rama $y_{13} = 0$.

El árbol de búsqueda hasta ahora es:

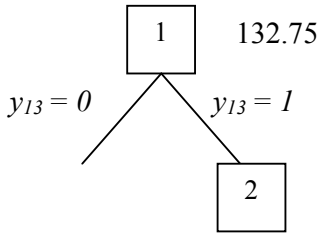

Figura 4-4. Árbol de Búsqueda en la Iteración 1

La cota inferior de la rama es el valor objetivo de la función: 132.75. Como es la primera solución factible, esta cota se convierte en la cota inferior del problema original.

Iteración 2

Paso [i]

No es necesario añadir columnas ya que no se han eliminado arcos en el subproblema.

Paso [ii]

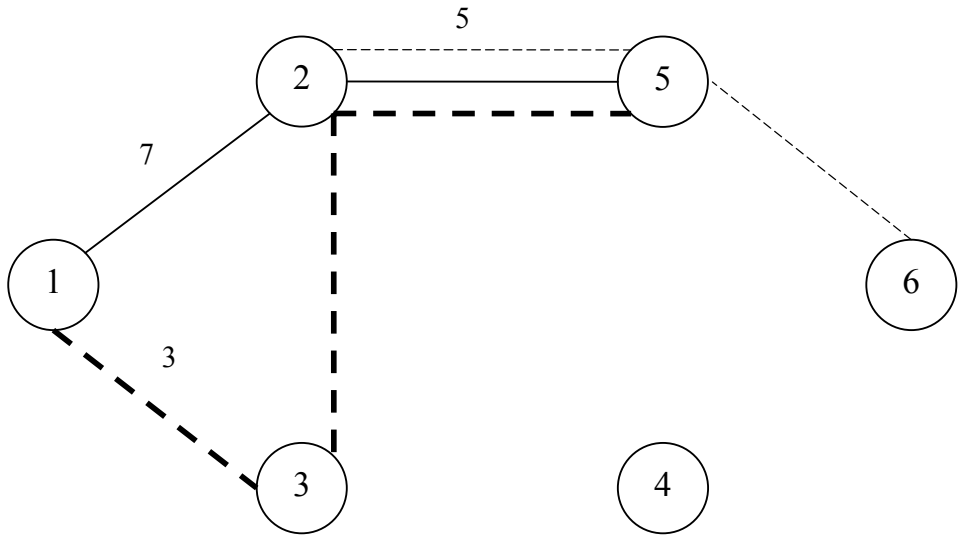

Figura 4-5. Solución Óptima en la Iteración 2

Al resolver con CPLEX el problema actual, la solución es:

$$y = (1, 1, 0.6, 1, 0, 0, 0, 0.5, 0, 0);$$

$$h = (0, 0, 0, 7, 3, 5)$$

La solución es factible ya que no se utilizan los arcos artificiales, sin embargo no es entera. La cota inferior de esta rama es el valor objetivo de la función: 135.3.

Paso [iii]

Se ramificará sobre y_{23} y y_{23} = 0 se guarda en la pila. El árbol de búsqueda correspondiente es:

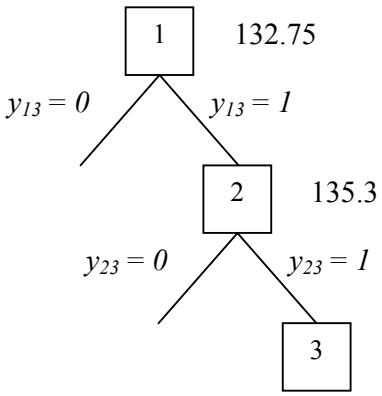

Figura 4-6. Árbol de Búsqueda en la Iteración 2

Iteración 3

Paso [i]

Debido a que no se ha restringido a 0 alguna variable, no es necesario añadir columnas.

Paso [ii]

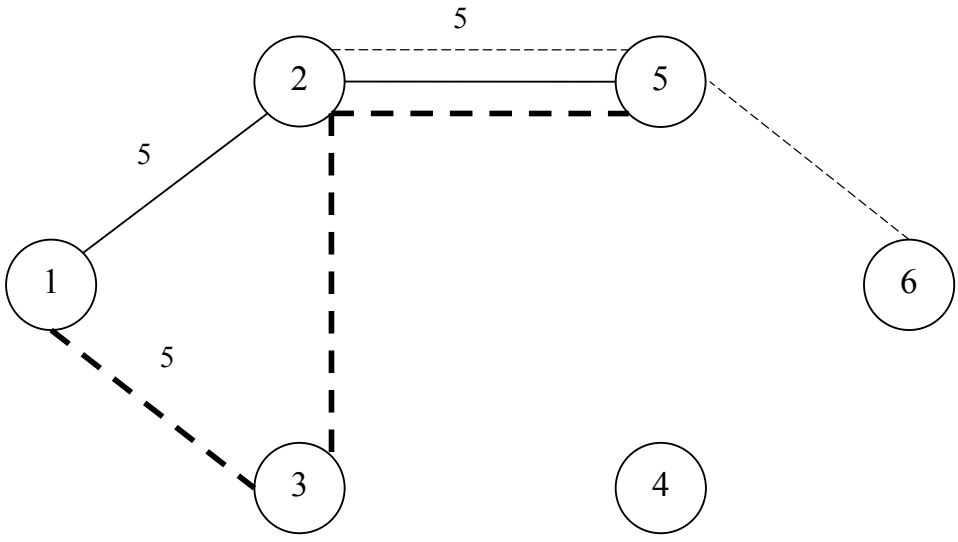

Figura 4-7. Solución Óptima en la Iteración 3

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (0.71, 1, 1, 1, 0, 0, 0, 0.5, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 5, 5, 5)$$

La solución es factible pero no entera. La cota inferior de esta rama es el valor objetivo de la función: 135.35.

Paso [iii]

Como la solución es factible pero no entera. Se ramificará a 1 sobre la variable y_{12} y se guardará $y_{12} = 0$ en la pila, como puede observarse en la siguiente figura.

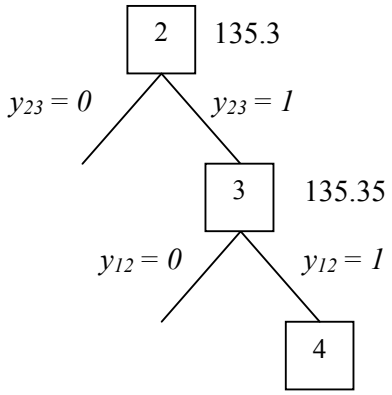

Figura 4-8. Árbol de Búsqueda en la Iteración 3

Iteración 4

Paso [i]

Debido a que no se ha restringido el problema, no es necesario añadir columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1, 1, 1, 0.86, 0.2, 0, 0.33, 0.3, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 2, 7, 3, 3)$$

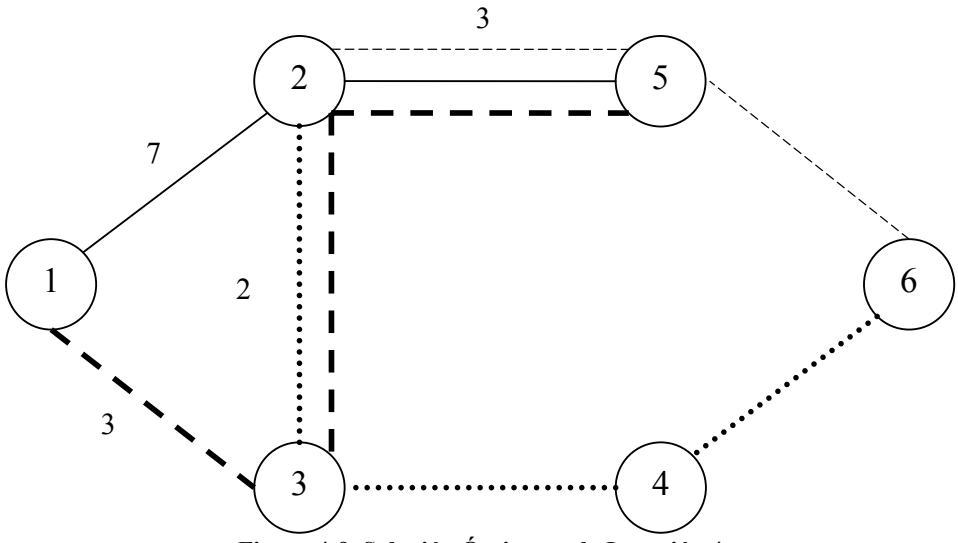

Figura 4-9. Solución Óptima en la Iteración 4

La solución es factible pero no entera. La cota inferior de esta rama es el valor objetivo de la función: 135.96.

Paso [iii]

Se ramificará fijando $y_{25} = 1$ y se guardará $y_{25} = 0$ en la pila.

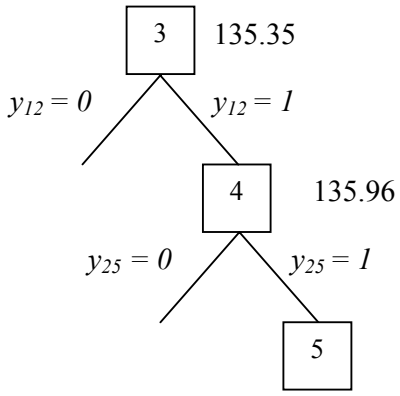

Figura 4-10. Árbol de Búsqueda en la Iteración 4

Iteración 5

Paso [i]

Debido a que no se ha restringido el problema, no es necesario añadir columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es la siguiente:

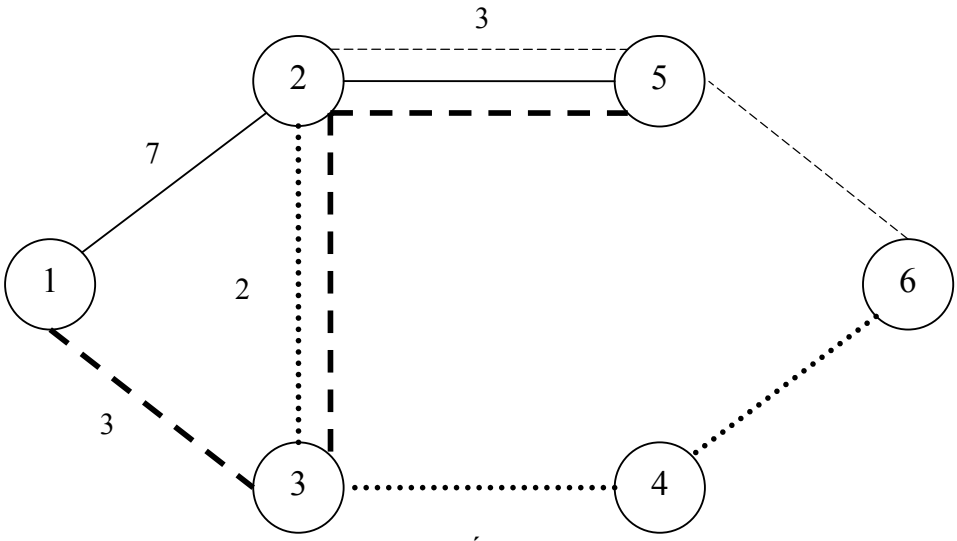

Figura 4-11. Solución Óptima en la Iteración 5

$\boldsymbol{y}$ = (1, 1, 1, 1, 0.2, 0, 0.33, 0.3, 0, 0);

$\boldsymbol{h}$ = (0, 0, 2, 7, 3, 3)

La solución es factible pero no entera. La cota inferior de esta rama es el valor objetivo de la función: 136.5.

Paso [iii]

Se ramificará fijando $y_{56} = 1$ y se guardará $y_{56} = 0$ en la pila.

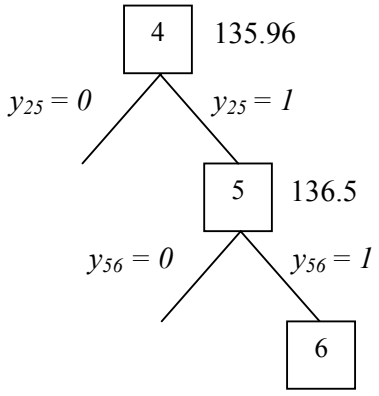

Figura 4-12. Árbol de Búsqueda en la Iteración 5

Iteración 6

Paso [i]

Debido a que no se ha restringido el problema, no es necesario añadir columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

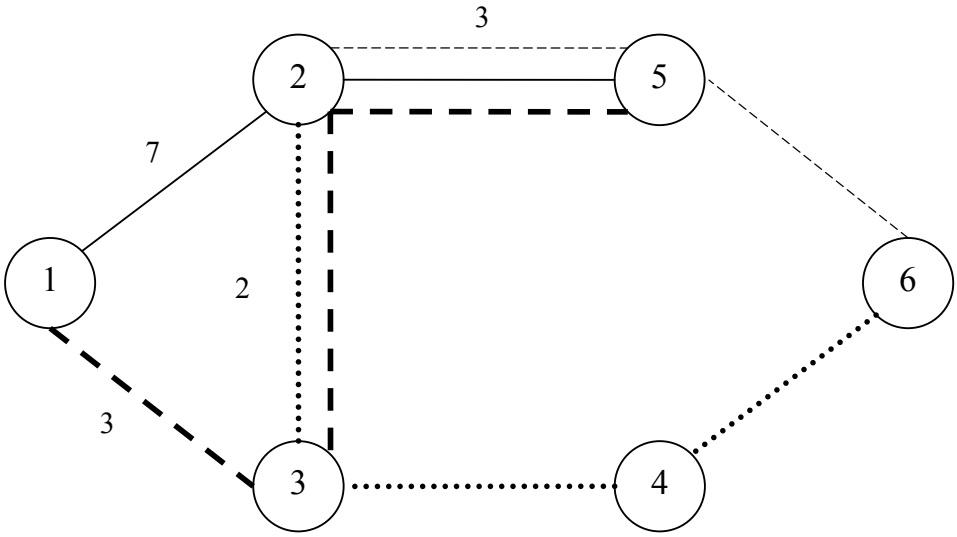

Figura 4-13. Solución Óptima en la Iteración 6

$$\boldsymbol{y} = (1, 1, 1, 1, 0, 0, 0, 1, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 5, 5, 5)$$

La solución es factible y entera. Por lo que la solución óptima está acotada por 132.75 y 139.0, es decir dentro del 0.045%.

Paso [iv]

Como la solución es entera se convierte en el primer incumbente y se agota esta rama. Se hace backtrack sobre $y_{56} = 0$.

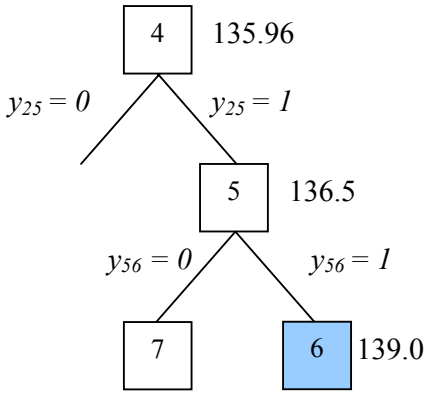

Figura 4-14. Árbol de Búsqueda en la Iteración 6

Iteración 7.1

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-104.0, 0.0, -104.0, 0.0, -0.5, 0.0, -1.0, -104.0, 110.0, 116.5, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 4, 3, 1}. La columna j asociada a éste es: $(0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -95.5$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 4, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -98.5$.

Se añadió la columna $(0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0)^T$, que permite transportar el producto 2 del nodo 2 al 6.

Iteración 7.2

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-6.3, 0.0, -6.3, 0.0, -0.5, -0.8, -1.0, -6.8, 12.3, 18.8, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 4, 3, 2}. La columna j asociada a éste es: $(0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Ya se satisface el criterio de optimalidad por lo que no se añadirán columnas.

La red de este subproblema es

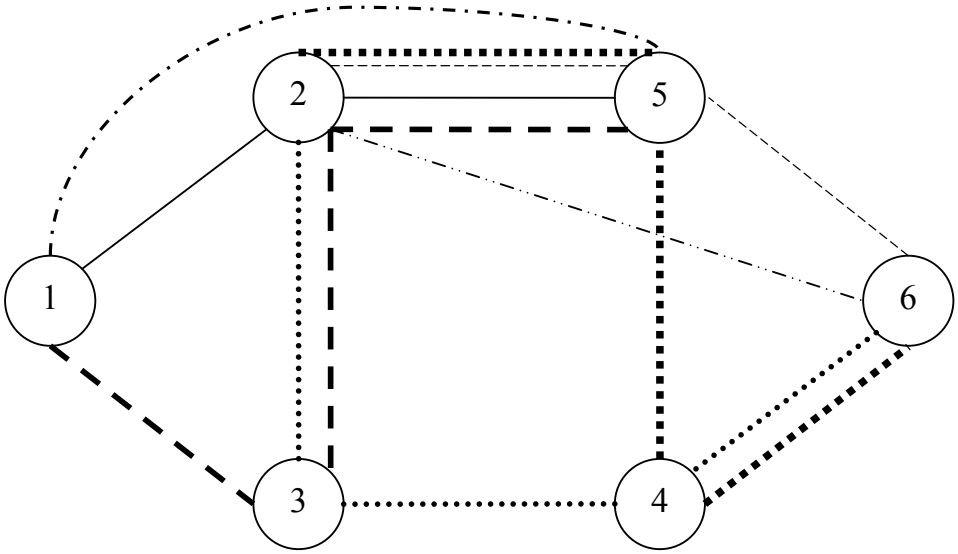

Figura 4-15. Caminos Disponibles en la Iteración 7

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1, 1, 1, 1, 0.2, 0.6, 0.833, 0, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 2, 7, 3, 0, 3)$$

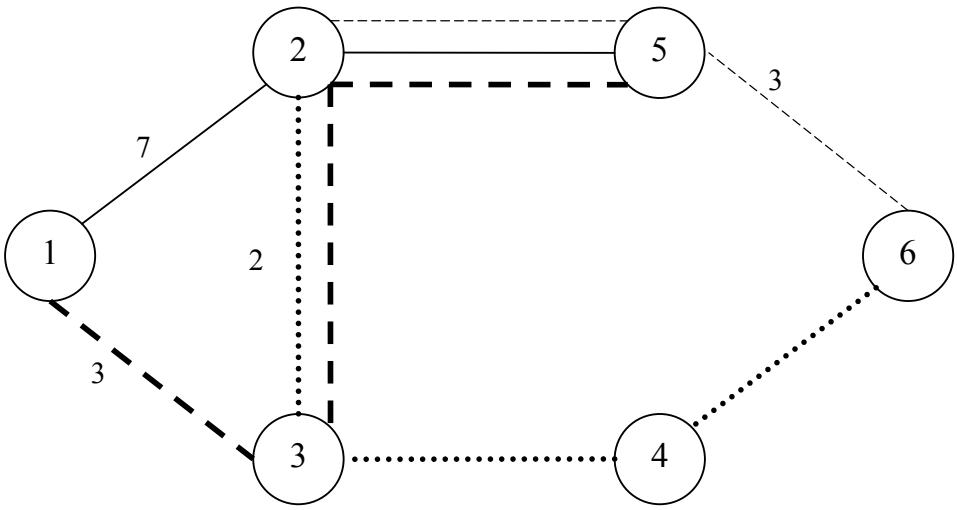

Figura 4-16. Solución Óptima en la Iteración 7

Obsérvese que la solución es factible ya que no se utilizan los arcos artificiales, sin embargo no es entera. La cota inferior de esta rama es: 155.4.

Paso [iv]

Se hace backtrack sobre $y_{25} = 0$.

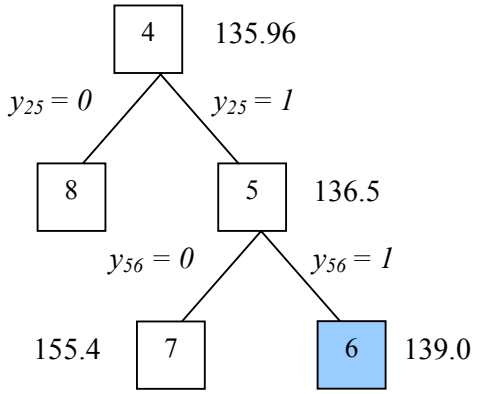

Figura 4-17. Árbol de Búsqueda en la Iteración 7

Iteración 8.1
Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (0.0, 0.0, 0.0, -104.0, -0.5, -0.8, -1.0, -0.5, 110.0, 12.5, -10.0, -20.0)

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 4, 3, 1}. La columna j asociada a éste es: $(0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -94.7$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 4, 3, 2}. La columna j asociada a éste es: $(0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Por lo que se añadirá la columna correspondiente al camino {5, 4, 3, 1}.

Iteración 8.2
Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (0.0, 0.0, 0.0, -104.0, -0.5, -95.5, -1.0, -0.5, 110.0, 12.5, -10.0, -20.0)

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 6, 4, 3, 1}. La columna j asociada a éste es: $(0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = -87.0$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 4, 3, 2}. La columna j asociada a éste es: $(0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Por lo que se añadirá la columna correspondiente al camino {5, 6, 4, 3, 1}.

Iteración 8.3
Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (0.0, 0.0, 0.0, -104.0, -87.5, -8.5, -1.0, -0.5, 110.0, 99.5, -10.0, -20.0)

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 1}. La columna j asociada a éste es: $(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0)^T$.

Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 4, 3, 2}. La columna j asociada a éste es: $(0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Se satisface el criterio de optimalidad por lo que no se añadirán columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1, 1, 1, 0, 1, 1, 0.83, 0, 0.5, 0);$$

$$\boldsymbol{h} = (5, 0, 5, 0, 0, 0, 0, 5, 0)$$

La solución no es factible.

Paso [iv]

La rama queda agotada por no factibilidad, por lo que se hace backtracking sobre y_{12}, como puede observarse en la figura 4. 18.

<u>Iteración 9</u>

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

$$w = (-9.03, 0.0, -9.03, -0.266, -0.5, -0.8, 0, -0.5, 15.3, 12.76, -10.0, -20.0)$$

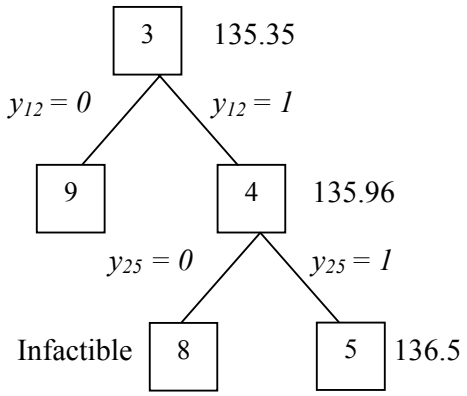

Figura 4-18. Árbol de Búsqueda en la Iteración 9

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$.

Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Se satisface el criterio de optimalidad por lo que no se añadirán columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (0, 1, 1, 0.66, 0.5, 1, 0, 0.5, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 0, 5, 5, 0, 5, 0)$$

La solución es factible. La cota inferior de esta rama es: 177.66.

Paso [iv]

Como la cota inferior es mayor que el incumbente, se agota esta rama y se realiza backtrack sobre $y_{23} = 0$.

<u>Iteración 10</u>

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-9.03, 0.0, -9.03, -0.26, -0.5, -0.8, 0.0, -0.5, 15.3, 12.76, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 2, 1}. La columna j asociada a éste es: $(1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Se satisface el criterio de optimalidad por lo que no se añadirán columnas.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1, 1, 0, 0.8, 0.3, 0.6, 0, 0.5, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 7, 0, 5, 0, 3, 0)$$

La solución es factible. La cota inferior de esta rama es 160.6.

Paso [iv]

La cota inferior de esta rama es el valor objetivo de la función: 160.6, que es mayor al incumbente, por lo que la rama queda agotada. Se realiza backtrack sobre $y_{13} = 0$.

Iteración 11

Paso [i]

Al resolver el problema actual, los valores duales obtenidos son:

w = (*-103.73, -103.73, -0.6, -0.26, -0.5, -0.8, -0.6,-0.5, 110.0, 12.76, -10.0, -20.0)*

Camino para el producto 1:

Por lo que el camino más corto del nodo 1 al 5 es: {5, 1}. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$.

Camino para el producto 2:

Por lo que el camino más corto del nodo 2 al 6 es: {6, 5, 2}. La columna j asociada a éste es: $(0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0)^T$. Evaluando el criterio de optimalidad: $-wa_j + c_j = 0$

Debido a que ambas columnas cumplen el criterio de optimalidad, no se agregan columnas al problema.

Paso [ii]

Al resolver con CPLEX el problema actual, la solución es:

$$\boldsymbol{y} = (1.0, 0, 0, 0.8, 0, 0, 0, 0.5, 0.3, 0);$$

$$\boldsymbol{h} = (3, 0, 0, 7, 0, 5, 0, 0, 0)$$

La solución no es factible.

Paso [iv]

La solución no es factible, pero la pila ya está agotada, por lo que se ha llegado al óptimo.

El costo mínimo de la red es 139.0, que corresponde a la solución óptima:

$$\boldsymbol{y} = (1, 1, 1, 1, 0.0, 0.0, 0.0, 1, 0, 0);$$

$$\boldsymbol{h} = (0, 0, 0, 5, 5, 5, 0, 0, 0)$$

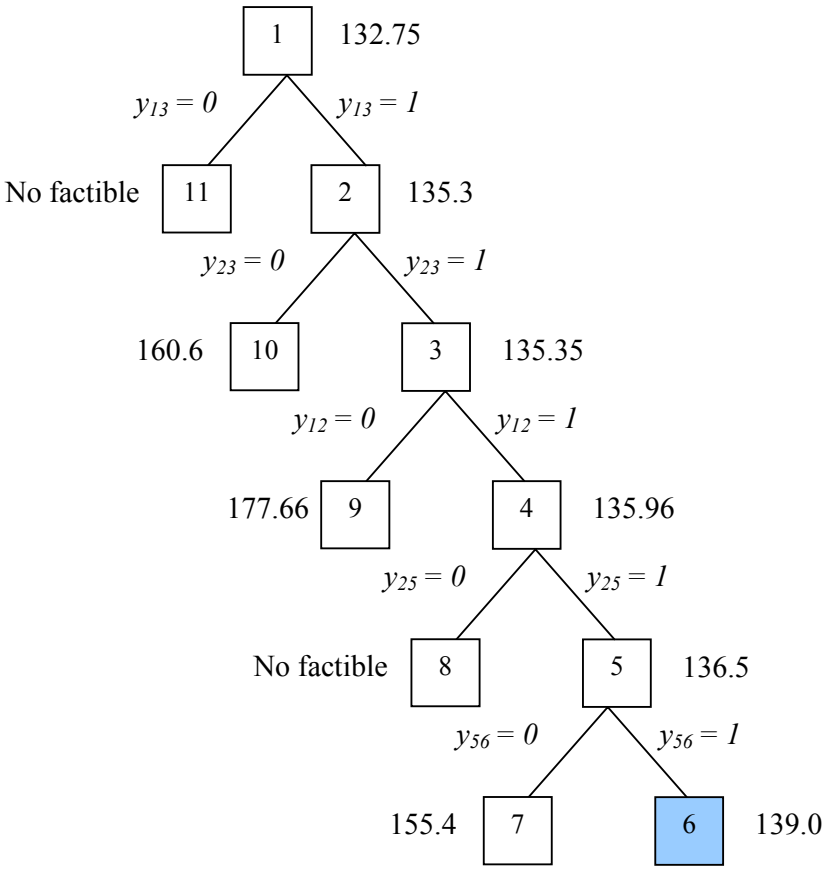

Figura 4-19. Árbol de Búsqueda Completo

El árbol completo de búsqueda puede observarse en la figura anterior, donde se muestra que sólo se requirieron 11 iteraciones para encontrar el óptimo. Cabe resaltar que este algoritmo es sencillo y fácil de seguir.

El tiempo computacional utilizado se muestra en la siguiente tabla:

Primer Incumbente	Último Incumbente	Algoritmo
0.016	0.016	0.063

Tabla 1. Tiempo Computacional. Método Propuesto

Con esto podemos ver que fue posible acotar la solución óptima en el primer tercio del tiempo total que requirió el algoritmo, lo cual es importante porque ayuda a descartar ramas del análisis.

Comparación de Estrategias de Ramificación

Como se sabe, la estrategia de ramificación afecta la eficiencia de los algoritmos en un ambiente de ramificación y acotamiento. Conforme más se aproveche la estructura del problema en particular, mejores resultados se tendrán. A continuación se ejemplificará esto comparando el método propuesto contra uno alternativo en el que no se toma en cuenta el hecho de que un arco forma parte de un camino y que el objetivo principal del problema es construir caminos para transportar los productos de origen a destino.

La estrategia alterna de ramificación consiste en elegir la variable y_{ij}, tal que su valor óptimo en la solución del subproblema, sea el más cercano a 1 sin ser entero. Se puede consultar el Apéndice A para observar a detalle las iteraciones del algoritmo con esta estrategia de ramificación.

Comparando las Figuras 4.19 y 4.20, el método propuesto originalmente, produce un árbol de búsqueda de menor tamaño, a saber 11 vs 21 subproblemas analizados.

En cuanto al tiempo computacional empleado, para cada estrategia de ramificación se resolvió el mismo problema 10 veces bajo las mismas condiciones, utilizando una computadora con procesador AMD Turion 64 ML-37, con 512 MB DDR. (Tabla 2)

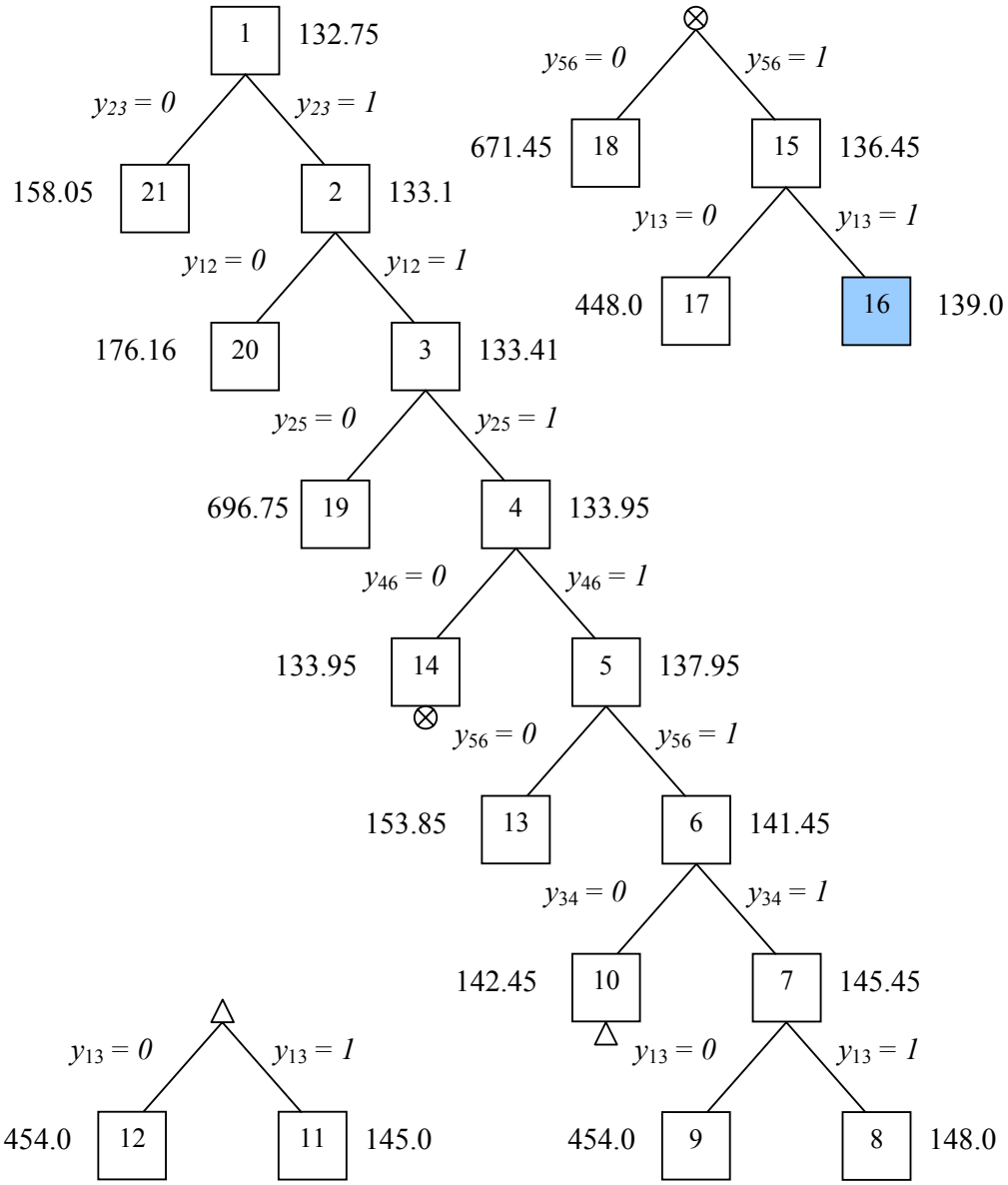

Figura 4-20. Árbol de Búsqueda. Método Alterno

	Primer Incumbente		Último Incumbente		Tiempo Total	
	Original (s)	Alterno (s)	Original (s)	Alterno (s)	Original (s)	Alterno (s)
	0.016	0.046	0.016	0.125	0.063	0.203
	0.032	0.047	0.032	0.125	0.078	0.188
	0.016	0.046	0.016	0.109	0.063	0.187
	0.016	0.062	0.016	0.14	0.063	0.218
	0.016	0.046	0.016	0.125	0.063	0.218
	0.016	0.062	0.016	0.156	0.063	0.234
	0.016	0.062	0.016	0.125	0.078	0.203
	0.031	0.046	0.031	0.125	0.078	0.203
	0.016	0.047	0.016	0.125	0.063	0.203
	0.016	0.063	0.016	0.141	0.063	0.219
Promedio (s)	0.019	0.053	0.019	0.130	0.068	0.208
DesvEst	0.007	0.008	0.007	0.013	0.007	0.015
Proporción	0.362		0.147		0.325	
Prueba t	0.000		0.000		0.000	
Prueba F	0.503		0.057		0.047	

Tabla 2. Comparación de Tiempos Computacionales

Basándose en un análisis estadístico de diferencias de varianzas y medias, se concluye que los métodos tienen distinta eficiencia computacional, teniendo mejor desempeño el método con la estrategia de ramificación propuesta originalmente, el cual para obtener la solución óptima requiere 32.5% del tiempo de lo que requiere el método alterno. Cabe resaltar que el haber encontrado rápidamente un incumbente ayudó a disminuir el espacio de soluciones.

Problemas de Gran Escala

Se probó el algoritmo en problemas con gran cantidad de variables ya que debido a las características del mismo, es en este tipo de instancias en las que se observan las mayores dificultades durante la resolución, debido a que los árboles de búsqueda crecen rápidamente conforme aumenta el tamaño del problema, lo que dificulta o hace incluso imposible de obtener una solución óptima. En particular, se probó el algoritmo en problemas de 30 nodos y 10 ó 50 productos. Los resultados de los primeros 70 minutos de un ejemplo del primer caso se observan en la Tabla 3.

Este mismo problema se resolvió con un algoritmo que utiliza un heurístico que combina GRASP con búsqueda dispersa (De Alba; 2004), el cual obtuvo un mejor entero en 254.3 segundos, a saber 85302, con un gap de 10.46%. Para más detalles consúltese el Apéndice D.

Iteración	Incumbente	Gap	Tiempo (s)
25	90091	28.76%	5.531
27	90070	28.75%	6.578
35	89510	28.3%	8.562
200	87039	26.27%	36.093
202	87018	26.25%	36.453
210	86458	25.77%	37.35
14460	86183	25.53%	4225.719
14462	86162	25.52%	4226.89
14470	85602	25.03%	4230.42

Tabla 3. Ejemplo p3010fl4

Los resultados de la prueba en un problema con 30 nodos y 50 productos se resumen en la Tabla 4 (mayores detalles en Apéndice D).

Iteración	Incumbente	Gap	Tiempo (s)
84	313620	43.41%	14.06
444	305712	41.95%	58.5
2860	301151	41.07%	248.3
8870	297766	40.40%	803.01
42371	292745	39.38%	4174.328
46320	292036	39.23%	4604.563

Tabla 4. Ejemplo p3050vt5

Al resolverse el mismo problema con un heurístico, el primer entero obtenido fue 279708, con un gap de 36%; el cual es menor al mejor obtenido en los primeros 77 minutos con el método exacto, además logró acotar de manera más estrecha a la solución óptima.

La siguiente tabla resume los resultados obtenidos al comparar ambos métodos.

Instancia	Método Exacto	Método heurístico
P3010ft4	Solución: 166585 Gap: 20.92 % Tiempo: 1110.72 s Tiempo Máx: 2957.9 s	Solución: 158322.5 Gap: 14.25% Tiempo: 31.66 s
P3010fl3	Solución: 118013 Gap: 30.58 % Tiempo: 5.44 s Tiempo Máx: 2306.1 s	Solución: 112999 Gap: 10.24% Tiempo: 10.34 s
P3010fl4	Solución: 86458 Gap: 25.77 % Tiempo: 34.75 s Tiempo Máx: 2319.38 s	Solución: 85302 Gap: 10.26% Tiempo: 254.95 s
P3010fl1	Solución: 94400.5 Gap: 27.46 % Tiempo: 43.52 s Tiempo Máx: 6637.5 s	Solución: 81400 Gap: 14.84% Tiempo: 13.77 s

P3010ft3	Solución: 219916 Gap: 18.9 % Tiempo: 957.34 s Tiempo Máx: 4595.86 s	Solución: 206582 Gap: 9.69% Tiempo: 33.58 s
P3010ft2	Solución: 158006 Gap: 19.37 % Tiempo: 8111.78 s Tiempo Máx: 8111.78 s	Solución: 147314 Gap: 9.42% Tiempo: 90.48 s
P3010vl1	Solución: 33082 Gap: 27.9 % Tiempo: 736.67 s Tiempo Máx: 1817.2 s	Solución: 27355 Gap: 4.56% Tiempo: 15.2 s

Tabla 5. Comparación de Métodos

Obsérvese que el método exacto no aporta buenas cotas inferiores, ya que la primera cota inferior es la solución obtenida de la relajación lineal del problema en cuestión. Esta cota inferior puede mejorar hasta que se hayan analizado por completo una de las dos primeras ramas generadas, lo cual no se logró en ninguno de los ejemplos anteriores. Esta limitante dificulta la reducción del espacio de soluciones a analizar.

Además al ir creciendo en tamaño las ramas, se incrementa rápidamente el número de subproblemas por resolver, lo cual es otra limitante en cuanto a la eficiencia del método.

En general el método heurístico requiere poco tiempo para obtener un mejor incumbente que el obtenido por el método exacto y provee un mejor acotamiento de la solución óptima.

CAPÍTULO 5

5 CONCLUSIONES

Generales

El algoritmo propuesto resuelve de manera exacta y en poco tiempo problemas MCND de tamaño limitado, ya que conforme va aumentando el número de columnas, se vuelve ineficiente, y no logra generar una buena cota para la solución del problema.

Se observa que al ramificar utilizando la estructura del problema, es decir tomando en cuenta que un arco forma parte de un camino, se redujo significativamente el tiempo de resolución, teniendo mayor impacto en el tiempo empleado para encontrar el primer incumbente, lo cual permite agotar ramas de manera más rápida, es decir provee soluciones más cercanas al óptimo en menor tiempo.

Otra característica positiva del método propuesto es que al utilizar la formulación basada en caminos, se maneja de manera más eficiente la compensación entre costos fijos y variables, que es una de las dificultades principales de este problema.

Por otra parte, el uso de generación de columnas basado en el método simplex permitió añadir las columnas que más benefician a la solución, esto resultó ser una buena ayuda para encontrar mejores soluciones en cada iteración.

Además el añadir las columnas que mayor beneficio tengan sobre el valor objetivo de la función, se resolvió acertadamente por medio del problema de camino más corto utilizando el algoritmo Floyd-Warshall, el cual tiene un tiempo de resolución polinomial. Con esto se asegura que conforme crezca el problema, el tiempo computacional requerido en esta fase no se verá gravemente afectado.

Propuestas de Mejora

Para mejorar la eficiencia del algoritmo, se propone reemplazar las columnas de los caminos que no se utilizan en la solución óptima del subproblema, por nuevas columnas obtenidas por medio del algoritmo Floyd-Warshall, tantas conforme indique el criterio de optimalidad.

En cuanto a la resolución del subproblema de camino más corto, se recomienda truncar los cálculos en cuanto se encuentren los caminos que permiten transportar los productos de su origen a destino.

Otra propuesta, es alternar la entrada y salida de columnas, no sólo de las columnas que no aparecen en la solución óptima del subproblema, sino también alguna que al eliminarla se perjudique menos el valor de la función objetivo, y a continuación ingresar una columna, la que mayor beneficio provea a dicho valor objetivo. Esta búsqueda puede realizarse por medio de métodos exactos ya que el número de columnas en la base no es tan grande, o por metaheurísticos como algoritmos genéticos o búsqueda dispersa, lo que podría servir para obtener mejores cotas en menor tiempo.

El trabajo en paralelo, para cualquiera de las recomendaciones anteriores, o inclusive para el algoritmo propuesto, podría reducir los tiempos computacionales y generar un algoritmo más eficiente. Una propuesta en particular es resolver de manera paralela las dos primeras ramas del árbol de búsqueda y cuando se agote cualquiera de las dos, replicar la idea sobre el siguiente nodo al que aplique. Con esto además de acelerar la búsqueda, se logra obtener más rápidamente una mejor cota inferior, y podría obtenerse más rápidamente un mejor incumbente, lo que reduce aún más el espacio de soluciones.

Tomando en cuenta que al incrementarse el tamaño de la rama aumenta rápidamente el número de subproblemas por resolver, se propone que si la rama excede cierto tamaño, parar la búsqueda en ésta y enviar el nodo a una cola y hacer backtrack hacia el siguiente nodo en la pila. Al vaciarse la pila se continuará la búsqueda sobre los nodos de la cola. Esto con la finalidad de encontrar un mejor incumbente que permita agotar algunas de estas ramas de gran tamaño y reducir el número de subproblemas por resolver. (Apéndice E)

Otra manera de reducir más rápidamente el espacio de soluciones, es iniciar con una solución entera factible para el problema, la cual puede ser obtenida por medio de un método heurístico. Ya que al tener un mejor incumbente se podría evitar resolver algunos subproblemas debido al criterio de acotamiento. (Apéndice E)

Una limitante del algoritmo es que no resuelve ejemplos donde esté disponible el arco que une un origen con su destino correspondiente, sin embargo esto es sencillo de resolver añadiendo un nodo y otro arco artificial. (Apéndice E)

BIBLIOGRAFÍA

[1] Agarwal, Y. (2002). Design of Capacitated Multicommodity Networks with Multiple Facilities. *Operations Research*, Vol. 50, No. 2, pp. 333.

[2] Ahuja, R.; Magnanti, T. y Orlin, J. (1993) Network Flows: Theory, Algorithms, and Applications. *Englewood Cliffs, N. J. Prentice Hall.* pp. 147-150.

[3] Álvarez, A.; González-Velarde, J. L. y De-Alba, K. (2005). Scatter Search for Network Design Problem. *Annals of Operations Research*. Vol. 138, pp. 159-178.

[4] Appelgren, L. (1969). A Column Generation Algorithm for a Ship Scheduling Problem. *Transport Science*. Vol. 3, pp. 53-68.

[5] Balakrishnan, A.; Magnanti, L.; Wong, R. T. (1989). A Dual-Ascent Procedure for Large-Scale Uncapacitated Network Design. *Operational Researchers*. Vol. 37, pp. 716-740.

[6] Balakrishnan, A.; Magnanti, T. L. y Wong, R. T. (1995). A Decomposition Algorithm for Local Access Telecommunication. *Operation Research*. Vol. 43, No. 1, pp. 58.

[7] Barnhart, C.; Johnson, E.; Nemhauser, G.; Savelsbergh, M. y Vance, P. (1998). Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operation Research*. Vol. 46, No. 3, pp. 316.

[8] Barnhart, C.; Krishnan, N.; Kim, D.; y Ware, K. (2002). Network Design for Express Shipment Delivery. *Computational Optimization and Applications*. Vol. 21, No. 3, pp. 239.

[9] Chung-Piaw, T. y Jia, S. (2004). Warehouse-Retaler Network Design Problem. *Operations Research*. Vol. 52, No. 3, pp. 396.

[10] Cormen, T., Leiserson, Ch., Rivest, R. (c2001). Introduction to Algorithms. Mc. Graw Hill.

[11] Crainic, T.; Gendron, B. y Farvolden, J. (2000). A Simplex-Based Tabu Search Method for Capacitated Network Design. *INFORMS Journal on Computing*. Vol. 12, No. 3, pp. 223.

[12] Crainic, T. y Gendreau, M. (2002). Cooperative Parallel Tabu Search for Capacitated Network Design. *Jornal of Heuristics*. Vol. 8, No. 6, pp. 601.

[13] Crainic, T.; Gendron, B. y Hernu, G. (2004). A Slope Scaling/Lagrangean Perturbation Heuristic with Long-Term Memory for Multicommodity Capacitated Fixed-Charge Network Design. *Journal of Heuristics, Kluwer Academia Publishers*. Vol. 10, pp. 525-545.

[14] Dantzig, G. y Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operational Research*. Vol. 8, pp. 101.

[15] De Alba, K. (2004). Un procedimiento Heurístico para un Problema de Diseño de Redes Multiproducto con Capacidad Finita y Cargos Fijos, Tesis Doctoral, FIME, UANL, San Nicolás de los Garza, NL.

[16] Desrosiers, J.; Soumis, F. y Desrochers, M. (1984). Routing with Time Windows by Column Generation. *Networks*. Vol. 14, pp. 545-565.

[17] Díaz, J. A. y Fernández, E. (2002). A Branch-and-Price Algorithm for the Single Source Capacitated Plant Location Problem. Journal of the Operational Research Society. Vol. 53, pp. 728-740.

[18] Doing, A. y Land, A. (1960). An Automatic Method for Solving Discrete Programming Problems. *Econometrica*. Vol. 28, pp. 497-520.

[19] Ford, L. y Fulkerson, D. (pre-1986; 1958). A Suggested Computation for Maximal Multicommodity Network Flows. *Management Science.* Vol. 5, No. 1, pp. 97.

[20] Geoffrion, A. y Marsten, R. (1972). Integer Programming: A Framework and State-of-the-Art Survey. *Management Science*. Vol. 18, pp. 465-491.

[21] Ghamlouche, I.; Crainic, T. y Gendreau, M. (2003). Cycle-based Neighborhoods for Fixed-Charge Capacitated Multicommodity Network Design. *Operations Research*. Vol. 51, No. 4, pp. 655.

[22] Ghamlouche, I.; Crainic, T. y Gendreau, M. (2004). Path Relinking, Cycle-Based Neighborhoods and Capacitated Multicommodity Network Design. *Annals of Operation Research*. Vol. 131, pp. 109-133.

[23] Gilmore, P. y Gomory, R. (1961). A Linear Programming Approach to the Cutting Stock Problem. *Operations Research*. Vol. 9, pp. 849-859.

[24] Gilmore, P. y Gomory, R. (1963). A Linear Programming Approach to the Cutting Stock Problem: Part II. *Operations Research*. Vol. 11, pp. 863-888.

[25] Glover, F. (1989). Tabu Search-Part I. *ORSA Journal on Computing*. Vol. 1, No. 3, pp. 190-206.

[26] Glover, F. (1990). Tabu Search-Part II. *ORSA Journal on Computing*. Vol. 2, No. 1, pp. 4-32.

[27] Glover, F. y Laguna, M. (1997). Tabu Search. *Norwell, MA; Kluwer*.

[28] Glover, F. y Kochenberger, A. (2003). *Handbook of Metaheuristics*. Editorial Klumer. Academic Publisher, Norwell, MA.

[29] Herrmann, J. W.; Ioannou, G.; Minis, I.; Nagi, R. y Proth, J. M. (1995). Design of Material Flow Networks in Manufacturing Facilities. *Journal of Manufacturing Systems; Dearborn*. Vol. 14, No. 4, pp. 277.

[30] Holmberg, K. y Hellstrand, J. (1998). Solving the Uncapacitated Network Design Problem by a Lagrangean Heuristic and Branch-and-Bound. *Operations Research*. Vol. 46, No. 2, pp. 247.

[31] Holmberg, K. y Yuan, D. (2000). A Lagrangian Heuristic Based Branch-and-Bound Approach for the Capacitated Network Design Problem. *Operations Research*. Vol. 48, No. 3, pp. 461.

[32] Holmberg, K. y Yuan, D. (2003). A Multicommodity Network-Flow Problem with Side Constraints on Paths Solved by Column Generation. *INFORMS Journal on Computing*. Vol. 15, No. 1, pp. 42.

[33] Johnson, D. y Garey, M. (1979). Computers and Intractability. A guide to the theory of up-completeness. W.H. Freeman and Company, New York.

[34] Johnson, E. (1989). Modeling and Strong Linear Programs for Mixed Integer Programming. *Algorithms and Model Formulations in Mathematical Programming. S. W.Wallace*. NATO ASI Series 51, pp. 1-41.

[35] Lübbecke, M. y Desrosiers, J. (2005). Selected Topics in Column Generation. *Operations Research*. Vol. 53, pp. 1007-1023.

[36] Mateus, G.; Luna, H. y Sirihal, A. (2000). Heuristics for Distribution Network Design in Telecommunication. *Journal of Heuristics*. Vol. 6, No. 1, pg.131.

[37] Papadimitriou, C.; Steiglitz, K. (1982). Combinatorial optimization: Algorithms and Complexity. Prentice Hall.

[38] Pinedo, M. (2002). Scheduling: Theory, Algorithms, and Systems, Englewood Cliffs, Prentice Hall.

[39] Riis, M.; Skriver, A. y Moller, S. (2005). Internet Protocol Network Design with Uncertain Demand. *Journal of the Operational Research Society*. Vol. 56, pp. 1184-1195.

[40] Stigler, H. y Todem, C. (2005). Optimization of the Austrian Electricity Sector (Control Zone of VERBUND APG). Central European Journal of Operations Research. Vol. 13, No. 2, pp. 105.

[41] Vanderbeck, F. y Wolsey, L. (1996). An Exact Algorithm for IP Column Generation. Operational Research. Vol. 19. pg. 151-159.

[42] Wolsey, L. (1984). Integer Programming. John Wiley and Sons, Chichester, UK; 1998.

LISTA DE FIGURAS

APÉNDICE A

Otro Ejemplo Ilustrativo del Método

Ramificando sobre la variable y_{ij}, tal que su valor óptimo sea el más cercano a 1, sin ser entero; se requieren las siguientes iteraciones para resolver el problema de la Figura 4.1.

Problema	**Columnas añadidas**	**Valor Óptimo**	**Solución**	**Acción**
1	C_1, C_2, C_3, C_4	132.75	(1, 0.15, 0.6, 1, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 7, 3, 5)	Ramificar sobre y_{23}. Fijar $y_{23} = 1$.
2		133.11	(0.71, 0.25, 1, 1, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 5, 5, 5)	Ramificar sobre y_{12}. Fijar $y_{12} = 1$.
3		133.41	(1, 0.15, 1, 0.86, 0.2, 0, 0.3, 0.3, 0, 0, 0, 0, 2, 7, 3, 3)	Ramificar sobre y_{25}. Fijar $y_{25} = 1$.
4		133.95	(1, 0.15, 1, 1, 0.2, 0, 0.3, 0.3, 0, 0, 0, 0, 2, 7, 3, 3)	Ramificar sobre y_{46}. Fijar $y_{46} = 1$.
5		137.95	(1, 0.15, 1, 1, 0.2, 0, 1, 0.3, 0, 0, 0, 0, 2, 7, 3, 3)	Ramificar sobre y_{56}. Fijar $y_{56} = 1$.
6		141.45	(1, 0.15, 1, 1, 0.2, 0, 1, 1, 0, 0, 0, 0, 2, 7, 3, 3)	Ramificar sobre y_{34}. Fijar $y_{34} = 1$.
7		145.45	(1, 0.15, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 2, 7, 3, 3)	Ramificar sobre y_{13}. Fijar $y_{13} = 1$.
8		148.0	(1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 2, 7, 3, 3)	Incumbente. Backtrack: Fijar $y_{13} = 0$.

9		454.0	(1, 0, 1, 1, 1, 0, 1, 1, 0.3, 0, 3, 0, 5, 7, 0, 0)	No Factible. Backtrack: Fijar $y_{34} = 0$.
10		142.45	(1, 0.15, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 7, 3, 5)	Ramificar sobre $y_{13.}$ Fijar $y_{13} = 1$.
11		145.0	(1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 7, 3, 5)	Incumbente. Backtrack: Fijar $y_{13} = 0$.
12		454.0	(1, 0, 1, 1, 0, 0, 1, 1, 0.3, 0, 3, 0, 0, 7, 0, 5)	No Factible. Backtrack: Fijar $y_{56} = 0$.
13	C_5	153.85	(1, 0.15, 1, 1, 0.2, 0.6, 1, 0, 0, 0, 0, 0, 2, 7, 3, 0, 3)	Cota inferior > Incumbente. Backtrack: Fijar $y_{46} = 0$.
14		133.95	(1, 0.15, 1, 1, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 7, 3, 5)	Ramificar sobre $y_{56.}$ Fijar $y_{56} = 1$
15		136.45	(1, 0.15, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 7, 3, 5, 0)	Ramificar sobre $y_{13.}$ Fijar $y_{13} = 1$
16		139.0	(1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 7, 3, 5, 0)	Incumbente. Backtrack: Fijar $y_{13} = 0$.
17		448.0	(1, 0, 1, 1, 0, 0, 0, 1, 0.3, 0, 3, 0, 0, 7, 0, 5, 0)	No Factible. Backtrack: Fijar $y_{56} = 0$.

18		671.45	(1, 0.15, 1, 1, 0, 0, 0, 0, 0, 1, 0, 5, 0, 7, 3, 0, 0)	No Factible. Backtrack: Fijar $y_{25} = 0$.
19	C_6, C_7, C_8	696.75	(1, 0.25, 1, 0, 1, 1, 0.83, 0, 0.5, 0, 5, 0, 5, 0, 0, 0, 0, 0, 5, 0)	No Factible. Backtrack: Fijar $y_{12} = 0$.
20		176.16	(0, 0.5, 1, 0.6, 0.5, 1, 0, 0.5, 0, 0, 0, 0, 0, 0, 5, 5, 0, 0, 5, 0)	Cota inferior > Incumbente. Backtrack: Fijar $y_{23} = 0$.
21		158.05	(1, 0.15, 0, 0.8, 0.3, 0.6, 0, 0.5, 0, 0, 0, 0, 0, 7, 0, 5, 0, 0, 3, 0)	Cota inferior > Incumbente. La Pila está vacía.

Tabla 6. Iteraciones con Ramificación Alterna

Las columnas que se introdujeron a lo largo del algoritmo son:

C_1	$(0, 0, 1, 0, 1, 0, 1, 0, 0, 1)^T$
C_2	$(1, 0, 0, 1, 0, 0, 0, 0, 1, 0)^T$
C_3	$(0, 1, 1, 1, 0, 0, 0, 0, 1, 0)^T$
C_4	$(0, 0, 0, 1, 0, 0, 0, 1, 0, 1)^T$
C_5	$(0, 0, 0, 1, 0, 1, 1, 0, 0, 1)^T$
C_6	$(1, 1, 0, 0, 1, 0, 1, 0, 0, 1)^T$
C_7	$(0, 1, 0, 0, 1, 1, 0, 0, 1, 0)^T$
C_8	$(0, 1, 0, 0, 1, 0, 1, 1, 1, 0)^T$

Tabla 7. Caminos Agregados

APÉNDICE B

Definición de Estructuras

```
        //Definición de la estructura de ingreso de datos del problema
typedef struct _inputdata
{       int    nodos;              /* Numero de nodos                    */
        int    art;                /* Numero de artículos por transportar */
        int    arc;                /* Arcos                              */
        double *cost;              /* Costo fijo por utilizar el arco    */
        int    *cap;               /* Capacidades de arcos               */
        int    *dem;               /* Demandas                           */
        int    **i;                /* Inicio y fin del arcos             */
        double **uncost;           /* Costo unitario de transportar el producto k por
                                   el arco (i, j)*/
        double **unorig;           /* Costo unitario de transportar producto k
                                   por el arco (i, j), este quedará fijo */
} inputdata;

        //Definición de la estructura de problema cplex
typedef struct _datacplex
{       char     *probname;
        int      numcols;
        int      numrows;
        int      objsen;
        double   *obj;
        double   *rhs;
        char     *sense;
        int      *matbeg;
        int      *matcnt;
        int      *matind;
        double   *matval;
        double   *lb;
        double   *ub;
        char     *ctype;
        int       *index;
        double *values;
```

```
        int         solstat;
        double   objval;
        double   *z;
        double   *pi;
        double   *slack;
        double   *dj;
        int       **a;
        int       **an;
        int         *nomb;
} datacplex;
```

//Definición de la estructura de ingreso de datos del subproblema para el algoritmo de camino más corto

```
typedef struct _fwinputdata
{       double **a;          /* Matriz de longitudes de los caminos |a|=nodos*nodos  */
        int    **p;          /* Matriz de los vértices predecesores                  */
        int    **b;          /* Matriz de nodos de los caminos más cortos            */
        double *sumab;       /* Vector de costo de los caminos más cortos            */
} fwinputdata;
```

//Definición de la estructura que contiene los datos de la solución

```
typedef struct _soldata
{       double cotainf;          /* Cota inferior                                   */
        double incumbente;       /* Matriz de los vértices predecesores             */
        int    tiposol;          /* tiposol != 1 si la solución no es factible      */
        int    xfrac;            /* Variable sobre la que se ramificará             */
        int    sta;              /* sta==3: Solución factible y no entera           */
        double *x;               /* Sólución óptima del subproblema actual          */
        double *y;               /* Sólución óptima incumbente                      */
        int    paro;             /* Marcador del ciclo de búsqueda del tipo de la solución */
        int    tam;              /* Tamaño de la solución                           */
        int    *cumple;          /* Valores que coinciden con la solución           */
        int    **c;              /* Caminos                                         */
        int    **ctemp;          /* Caminos temporal                                */
        int    caminos;          /* Número de caminos no artificiales               */
```

```
        int    crama;          /* Camino sobre el que se ramificará          */
        int    *arco;          /* Arco que ya se fijó                        */
        clock_t cerotime;      /* Tiempo al inicio del proceso               */
        clock_t inctime;       /* Tiempo al encontrar el primer incumbente   */
        clock_t soltime;       /* Tiempo al encontrar la solución óptima     */
        clock_t algortime;     /* Tiempo del algoritmo                       */
        double  *time;         /* Vector de tiempos                          */
        float   *primcot;      /* Primera cota                               */
        int     iter;          /* Cuenta las iteraciones                     */
} soldata;

        //Definición de estructura de las variables utilizadas en la ramificación
typedef struct _ramadata
{       unsigned  nodo;        /* Nodo donde se ramificará                   */
        int       fin;         /* Marca para ramificar                       */
        int       cnt;         /* Contador de columnas que se cambiarán      */
        int       *indices;    /* Índice de la variable que se cambiará      */
        char      *lu;         /* Cotas que se modificarán                   */
        double    *bd;         /* Nuevos valores de las cotas                */
        int       tm;          /* Marcador para ramificar                    */
        int       itt;         /* Marcador para ramificar                    */
} ramadata;

        // Definición de estructuras de datos para el manejo de la pila
typedef struct TNODE
{       int    numero;              /* Número de nodo                        */
        int    indice;              /* Índice de la variable fijada a cero o a uno */
        int    valor;               /* Valor al que se fija la variable: 0 ó 1 */
        struct TNODE *Padre;        /* Apuntador al nodo padre               */
} TNODE;
```

Prototipo de las Funciones Utilizadas

Función para ingresar los datos del problema general:

```
void   input_data (char *filename, inputdata *pdata);
```

Función que crea el problema en lenguaje CPLEX:

```
void   setproblemdata (datacplex *pcplex, inputdata pdata);
```

Función para liberar memoria de los datos de pcplex

```
void   libera(datacplex *pcplex);
```

Función para obtener los caminos más cortos:

```
void   caminos(inputdata pdata, datacplex pcplex, fwinputdata fwpdata);
```

Función para ingresar los datos del subproblema del camino más corto:

```
void   fwinput_data (datacplex *pcplex, inputdata pdata, fwinputdata *fwpdata, int k);
```

Función para el algoritmo Floyd-Warshall:

```
void   fw (fwinputdata *fwpdata, inputdata pdata, int nodo);
```

Función que analiza criterio de optimalidad y añade una columna si es necesario:

```
void   calyadd(datacplex *pcplex, inputdata pdata, fwinputdata *fwpdata, CPXENVptr
env, CPXLPptr  lp, soldata *sol);
```

Función para actualizar el tamaño del vector de solución

```
void   tamano(datacplex *pcplex, soldata *sol);
```

Función que analiza si la solución es factible:

```
void   factible(soldata *sol, inputdata pdata, datacplex pcplex);
```

Función que analiza si la solución es entera:

```
void   entero(soldata *sol, inputdata pdata);
```

Función para encontrar la cota inferior del problema general

```
void    cotainf(soldata *sol, datacplex *pcplex, ramadata rama);
```

Función que analiza sobre qué variable se ramificará:

```
void   branch(soldata *sol, inputdata pdata, datacplex pcplex);
```

Funciones para el manejo de la pila:

```
int    push (StackNodePtr *, TNODE *, StackNodePtr *, int);
TNODE *pop(StackNodePtr *topPtr )
int isEmpty( StackNodePtr topPtr )
```

Código de la Función Principal

```
void main(int argc, char **argv)
{       void main(int argc, char **argv)
{       int     status;                 // Definición de variables
        int     j, n=1;
        int     Terminar=0;
        float  masgrande = 1.0E40;  //Debe costar más que los costos fijos de los arcos disponibles

        inputdata     pdata;            // Estructura con los datos del problema
        datacplex     pcplex;           // Estructura con los arreglos CPLEX
        soldata       sol;              // Estructura con los datos de la solución
        TNODE         *q, *r, *s;       // Estructura de nodos para la pila
        fwinputdata fwpdata;            // Estructura con los datos del problema FW
        ramadata      rama;             // Estructura con los datos de la rama

        CPXENVptr env = NULL;
        CPXLPptr  lp  = NULL;
        StackNodePtr stackPtr = NULL;
        StackNodePtr stackprim = NULL;

        if (argc != 2)
        {       printf("usage: genred <input file>\n");
                exit(1);
        }
        input_data(argv[1], &pdata);
                        // Inicializa valores de ramificación
        rama.nodo=0;
        rama.fin=0;
        rama.cnt=1;
        rama.tm=0;
        rama.itt=-1;
                // Inicializar nodo que se analiza
        q=(TNODE *) malloc (sizeof (TNODE));
        q->numero=rama.nodo;
        q->indice=-1;     // No aplica para el nodo 0: no hay ninguna variable fijada
        q->valor=-1;      // No aplica para el nodo 0
        q->Padre=NULL;
        rama.nodo++;
```

```
        // Inicializa valores de la solución
    sol.incumbente= 1.0E20;     // Inicializa valor del incumbente
    sol.tam = 0;                // Inicializa tamaño de la solución
    sol.caminos = 0;            // Número de caminos no artificiales
    sol.primcot = dvector(0,0);
    sol.primcot[1] = -1;        // Inicializa para encontrar la primera cota inferior
    sol.iter=0;
        // Inicializa el ambiente CPLEX
    env = CPXopenCPLEX (&status);
    if ( env == NULL )
    {       char  errmsg[1024];
        fprintf (stderr, "Could not open CPLEX environment.\n");
        CPXgeterrorstring (env, status, errmsg);
        fprintf (stderr, "%s", errmsg);
    }
        // Crea problema en Cplex
    setproblemdata (&pcplex,pdata);
    lp = CPXcreateprob(env, &status, pcplex.probname);
    status = CPXcopylp (env, lp, pcplex.numcols, pcplex.numrows, pcplex.objsen, pcplex.obj,
pcplex.rhs, pcplex.sense, pcplex.matbeg, pcplex.matcnt, pcplex.matind, pcplex.matval, pcplex.lb,
pcplex.ub, NULL);
        // Libera memoria de los parámetros del problema
    libera(&pcplex);
        /* Tamaño del problema */
    pcplex.numrows = CPXgetnumrows(env,lp);
    pcplex.numcols = CPXgetnumcols(env,lp);
        /* Asignación de memoria e inicialización*/
    sol.arco = ivector(0, pdata.arc-1);   // Vector en donde se marcarán los arcos que ya se fijaron
    for(j=0; j<pdata.arc; j++)
        sol.arco[j]=-1;
        //Problema inicial
    status = CPXwriteprob(env, lp, "inicial.lp", NULL);
        /* Asignación de memoria e inicialización */
    sol.time = dvector(0, 1);           // Vector de tiempos de primer incumbente y solución
    sol.time[0]=0.0;
    sol.cerotime = clock();

        // Optimiza, genera columnas y ramifica hasta obtener el óptimo general
    do              // Optimiza el problema actual añadiendo columnas
    {       if(pcplex.nomb[2]!=0)           // Si existe una columna para agregar
        {       do
```

```
		{	status = CPXlpopt(env, lp);
			if(status)
			{	fprintf(stderr, "Failed to optimize LP.\n");
				goto TERMINATE;
			}
				//Status de la solución
			pcplex.solstat = CPXgetstat (env, lp); // Solución factible y óptima = 1
				// Obtiene los duales
			status=CPXgetpi(env, lp, pcplex.pi, 0, CPXgetnumrows(env,lp)-1);
				// Asignación de Memoria. Nodos del camino
			fwpdata.b = imatrix(1, pdata.nodos+1, 1, pdata.art);
				//Para cada artículo obtiene el camino más corto
			caminos(pdata, pcplex, fwpdata);
				// Calcula -min(-waj-v1+cj) y añade columna
			calyadd(&pcplex, pdata, &fwpdata, env, lp, &sol);
		} while (pcplex.nomb[2]!=0);
		pcplex.numcols = CPXgetnumcols(env,lp);
		CPXgetobjval(env, lp, &pcplex.objval);
			//Actualiza el tamaño del vector de solución
		tamano(&pcplex, &sol);
}
else
{	status = CPXlpopt(env, lp);
		if(status)
		{	fprintf(stderr, "Failed to optimize LP.\n");
			goto TERMINATE;
		}
			//Status de la solución
		pcplex.solstat = CPXgetstat (env, lp);    // Para solución factible y óptima = 1
		CPXgetobjval(env, lp, &pcplex.objval);
}
		// Guarda la solución, local y general
status = CPXgetx(env, lp, sol.x, 0, pcplex.numcols-1);
if(sol.iter==0)
		for(j=0; j<pcplex.numcols-1; j++)
			sol.y[j]=sol.x[j];
		//Analizar si la solución es factible
factible(&sol, pdata, pcplex);
		//Analizar si la solución es entera
entero(&sol, pdata);
```

```
        // Cotas inferiores (minimización)
cotainf(&sol, &pcplex, rama);
        // Aplicar criterios para la poda de la rama del árbol de enumeración a la que pertenece el nodo q i.e. infactibilidad, optimalidad o acotamiento
        //Elige la variable sobre la que se ramificará, alimenta la pila y analiza el status de la solución
branch(&sol, pdata, pcplex);
        //Asignación de memoria
rama.lu = cvector(0,0);
rama.bd = dvector(0,0);
rama.indices = ivector(0,0);
        // Termina la exploración de la rama actual (el nodo q está agotado)
if (sol.sta != 3)               //Si la solución no es factible o entera:
{       s=q;
                // Backtracking:
        // 1.- Verificar que la pila no esté vacía y obtener el siguiente nodo a procesar.
        if(q->numero!=0 && q->numero!=1)
        {       q=pop( &stackPtr );     //Saca al nodo de hasta arriba de la pila
                rama.tm++;
        }
        else
                Terminar=1;
        if(q->numero==0 && rama.fin==1)
                Terminar=1;
        // 2.- Actualizar las variables que están fijadas en el nodo q
        if(Terminar != 1)
        {       r = q;
                if(q->numero==1)
                {       rama.fin=1;
                        sol.primcot[1]=-1;
                }
                if (r != NULL)
                {               // Liberar los arcos que se habían fijado a 1
                        for(j=0; j<pdata.arc; j++)
                                if(sol.arco[j]==r->indice)
                                {       rama.indices[0]=j;
                                        rama.lu[0]='L';
                                        rama.bd[0]=0;
                                        status = CPXchgbds(env, lp, rama.cnt, rama.indices, rama.lu, rama.bd);
```

```
                                        sol.arco[j]=-1;  //Libera la marca de los arco
                                        que se habían fijado en 1
                                }
                        // Se fija a cero la variable que antes se había fijado en 1
                        rama.indices[0]=r->indice;
                        rama.lu[0]='B';
                        rama.bd[0]=0;
                        status = CPXchgbds(env, lp, rama.cnt, rama.indices, rama.lu,
                        rama.bd);
                        // Devuelve la relajación de los nodos de ramas abajo
                        if(rama.tm>1)
                        {       do
                                {       rama.indices[0]=s->indice;
                                        rama.lu[0]='U';
                                        rama.bd[0]=1;
                                        status = CPXchgbds(env, lp, rama.cnt,
                                        rama.indices, rama.lu, rama.bd);
                                        s=s->Padre;
                                } while (s->indice != q->indice);
                        rama.tm=1;
                        }
                        pcplex.nomb[2]=1;   //Permite buscar columnas para agregar
                }
        }
}
else    //La solución es factible pero no entera
{       // El nodo q será el siguiente a explorar y el nodo s se envía a la pila
        s=(TNODE *) malloc (sizeof (TNODE));
        s->numero=rama.nodo++;
        s->indice=sol.xfrac;        //Índice del arco en el que se desea ramificar
        s->valor=0;
        s->Padre=q;
        if(rama.itt==-1)
        {       rama.itt=push(&stackprim, q, &stackprim, &rama);
                rama.itt=0;
        }
        rama.itt=push(&stackPtr, s, &stackprim, &rama);
        r=(TNODE *) malloc (sizeof (TNODE));
        r->numero=rama.nodo++;
        r->indice=sol.xfrac;        //Índice del arco en el que se desea ramificar
        r->valor=1;
        r->Padre=q;                 // r->Padre es el nodo que se estaba analizando
```

```
                    q=r;                        // Se actualiza q : el nodo que se analizará
                        // Se fijara la variable ramificada en 1
                    rama.lu[0]='B';
                    rama.bd[0]=1;
                        //Fija a 1 el arco
                    rama.indices[0]=sol.xfrac;
                    status = CPXchgbds(env, lp, rama.cnt, rama.indices, rama.lu, rama.bd);
                    sol.arco[sol.xfrac]=sol.xfrac;      //Marca que se fijó este arco en 1,
                    rama.tm=0;
                }
                free_cvector (rama.lu,0);
                free_dvector (rama.bd,0);
                free_ivector (rama.indices, 0);
                sol.iter++;
                if(sol.iter==2000*n)   //Cada 2000 iteraciones imprime el tiempo
                {       printf("%d \n", sol.iter);
                    sol.algortime = clock();
                    sol.time[2] = (double) (sol.algortime - sol.cerotime)/CLOCKS_PER_SEC;
                    printf("Tiempos: %lf\n", sol.time[2]);
                    n++;
                }
        } while (!Terminar);
TERMINATE:
        sol.algortime = clock();            // Tiempo total de solución
        sol.time[2] = (double) (sol.algortime - sol.cerotime)/CLOCKS_PER_SEC;
        sol.time[1] = (double) (sol.soltime - sol.cerotime)/CLOCKS_PER_SEC;
        printf("Tiempos: Primer incumbente: %lf, Solución: %lf, Algoritmo: %lf\n", sol.time[0],
        sol.time[1], sol.time[2]);
            // Escribe la solución
        printf ("\n\nLa solución óptima es:\n");
        for(j = 0; j < pcplex.numcols ; j++)
        {       if(sol.y[j] > 0)
                {       printf ("x%d: Value = %f\n", j+1, sol.y[j]);
                }
        }
        if(sol.iter>1)
                printf ("El valor objetivo es: %f\n", sol.incumbente);
        else
                printf ("No es solución entera. El valor objetivo es: %f\n", pcplex.objval);
}
```

Código de la Función input_data

```
void input_data(char *filename, inputdata *pdata)
{       int  i, j;                      // Variables auxiliares
        char a;                         // Utilizado para leer y saltar headers
        float grande=1.0e15;    //Costo fijo del arco artificial > costo fijo de arcos normales
        FILE *fp;                       // puntero del archivo
        fp = fopen(filename,"r");
        if (fp == NULL) nrerror("Unable to open input file.");
                /* lee el tamaño del problema antes de los skipping headears */
        a = getc(fp);
        while(!isdigit(a)) a = getc(fp);
        ungetc(a, fp);
        fscanf(fp,"%d %d %d", &pdata->nodos, &pdata->art, &pdata->arc);
                /* asignación de memoria */
        pdata->cost = dvector(1, pdata->arc);
        pdata->cap  = ivector(1, pdata->arc);
        pdata->dem  = ivector(1, pdata->art);
        pdata->i    = imatrix(1, 2, 1, pdata->arc+pdata->art);
        pdata->uncost = dmatrix(1, pdata->art, 1, pdata->arc+pdata->art);
        pdata->unorig = dmatrix(1, pdata->art, 1, pdata->arc+pdata->art);
                /* lee costos fijos de utilizar los arcos */
        a = getc(fp);
        while(!isdigit(a)) a = getc(fp);
        ungetc(a, fp);
        for (i = 1; i <= pdata->arc; ++i)
                fscanf(fp, "%lf", &pdata->cost[i]);
                /* lee la capacidad de los arcos */
        a = getc(fp);
        while(!isdigit(a)) a = getc(fp);
        ungetc(a, fp);
        for (i = 1; i <= pdata->arc; ++i)
                fscanf(fp, "%d", &pdata->cap[i]);
                /* lee la demanda total de cada producto */
        a = getc(fp);
         while(!isdigit(a)) a = getc(fp);
        ungetc(a, fp);
        for (i = 1; i <= pdata->art; ++i)
                        fscanf(fp, "%d", &pdata->dem[i]);
```

```
            /* lee inicio y fin del arco */
        a = getc(fp);
        while(!isdigit(a)) a = getc(fp);
         ungetc(a, fp);
        for (i = 1; i <= 2; ++i)
                for (j = 1; j <= pdata->arc+pdata->art; ++j)
                        fscanf(fp, "%d", &pdata->i[i][j]);
                /* lee costos unitarios de transportación del producto k que se mueve por el arco (i, j) */
        a = getc(fp);
        while(!isdigit(a)) a = getc(fp);
        ungetc(a, fp);
        for (i = 1; i <= pdata->art; ++i)
        {       for (j = 1; j <= pdata->arc; ++j)
                {       fscanf(fp, "%lf", &pdata->uncost[i][j]);
                        pdata->unorig[i][j]=pdata->uncost[i][j];
                }
                for (j = pdata->arc+1; j <= pdata->arc+pdata->art; ++j)
                {       pdata->uncost[i][j]=grande;
                        pdata->unorig[i][j]=grande;
                }
        }
}
```

Código de la Función setproblemdata

```
void setproblemdata (datacplex *pcplex, inputdata pdata)
{       int i, j, n;                    //Variables auxiliares
        int numcont, numint;
        float grande=1.0e15; //Costo de los arcos y caminos artificiales, igual al de input data

        pcplex->probname=cvector(1, 13);
        strcpy (pcplex->probname, "lpgencolbbfw");
        pcplex->numcols = pdata.arc + pdata.art*2;
        pcplex->numrows = pdata.arc + pdata.art*2;
        pcplex->objsen = 1;   // El problema es de minimización

        numcont = pdata.arc + 2*pdata.art;  // Número de variables continuas
        numint = 0;                         // Número de variables enteras
                /* Asignación de memoria */
        pcplex->obj=dvector(0,pcplex->numcols-1);
        pcplex->rhs=dvector(0,pcplex->numrows-1);
        pcplex->sense=cvector(0,pcplex->numrows-1);
        pcplex->matbeg=ivector(0,pcplex->numcols-1);
        pcplex->matcnt=ivector(0,pcplex->numcols-1);
        pcplex->matind=ivector(0,pcplex->numcols+pdata.art-1);
        pcplex->matval=dvector(0,pcplex->numcols+pdata.art-1);
        pcplex->lb=dvector(0,pcplex->numcols-1);
        pcplex->ub=dvector(0,pcplex->numcols-1);
        pcplex->ctype=cvector(0,pcplex->numcols-1);
        pcplex->index = ivector(0,pcplex->numrows-1);
        pcplex->values=dvector(0,pcplex->numrows-1);
        pcplex->pi=dvector(0,pcplex->numrows-1);
        pcplex->slack=dvector(0,pcplex->numrows-1);
        pcplex->a=imatrix(0,pcplex->numrows-1,0,pcplex->numcols-1);
        pcplex->nomb=ivector(1,2);

        pcplex->nomb[1] = 0;
        pcplex->nomb[2] = 1;

        for(i=0; i<pcplex->numrows; i++)
                for(j=0; j<pcplex->numcols; j++)
                        pcplex->a[i][j]=0;
```

```
            /* Define matbeg */
for(i=0; i < pdata.arc+pdata.art; i++)
      pcplex->matbeg[i] = i;
n=pdata.art+pdata.arc;
for(i=pdata.arc+pdata.art; i<pdata.arc+2*pdata.art; i++)
{           pcplex->matbeg[i] = n;
            n=n+2;
}
            /* Define matind y matval */
for(i=0; i < pdata.arc; i++)
{           pcplex->matind[i] = i;
            pcplex->matval[i] = -pdata.cap[i+1];  //Capacidad del arco. Restricciones de Capacidad
}
n=1;
for(i=pdata.arc; i<pdata.arc+pdata.art; i++)
{           pcplex->matind[i] = i+pdata.art;
            pcplex->matval[i] = -pdata.dem[n]; //Capacidad de los arcos artificiales
            n++;
}
n=0;
for(i=pdata.arc+pdata.art; i<pdata.arc+3*pdata.art; i++)
{    pcplex->matind[i] = n+pdata.arc;
            pcplex->matval[i] = 1;
            i++;
            pcplex->matind[i] = n+pdata.arc+pdata.art;
            pcplex->matval[i] = 1;
            n++;
}
            // Define las cotas de las variables
for(j=0; j<pdata.arc + pdata.art; ++j)
{           pcplex->lb[j] = 0.0;
            pcplex->ub[j] = 1.0;
            pcplex->ctype[j] = 'C';
}
i=1;
for(j=pdata.arc + pdata.art; j<pdata.arc + 2*pdata.art; ++j)
{           pcplex->lb[j] = 0.0;
            pcplex->ub[j] = pdata.dem[i];
            pcplex->ctype[j] = 'C';
```

```
            i++;
        }
            /* Define el lado derecho y la dirección de las inecuaciones */
        for (i=0;i<pdata.arc;i++)
        {       pcplex->sense[i] = 'L';
            pcplex->rhs[i] = 0.0;
        }
        j=1;
        for (i=pdata.arc; i<pdata.arc+pdata.art; i++)
        {       pcplex->sense[i] = 'E';
            pcplex->rhs[i] = pdata.dem[j];
            j++;
        }
        for (i=pdata.arc+pdata.art; i<pdata.arc+2*pdata.art; i++)
        {       pcplex->sense[i] = 'L';
            pcplex->rhs[i] = 0.0;
        }
            /* Define los coeficientes de la función objetivo */
        for(i = 1;i <= pdata.arc;i++)
        {       pcplex->obj[i-1] = pdata.cost[i];
            pcplex->matcnt[i-1] = 1;
        }
        for(i = pdata.arc+1;i <= pdata.arc+pdata.art;i++)
        {       pcplex->obj[i-1]=grande;
            pcplex->matcnt[i-1] = 1;
        }
        for(i = pdata.arc+pdata.art+1;i <= pdata.arc+2*pdata.art;i++)
        {       pcplex->obj[i-1]=grande;
            pcplex->matcnt[i-1] = 2;
        }
            // Libera memoria de los datos que ya no utiliza
        free_ivector (pdata.cap, 1);
        free_ivector (pdata.dem, 1);
}
```

Código de la Función libera

```
void libera(datacplex *pcplex)
{       free_dvector (pcplex->obj, 0);
        free_dvector (pcplex->rhs, 0);

        free_cvector (pcplex->sense, 0);

        free_ivector (pcplex->matbeg, 0);
        free_ivector (pcplex->matcnt, 0);
        free_ivector (pcplex->matind, 0);
        free_dvector (pcplex->matval, 0);
        free_dvector (pcplex->lb, 0);
        free_dvector (pcplex->ub, 0);
        free_cvector (pcplex->ctype, 0);
        free_ivector (pcplex->index, 0);
        free_dvector (pcplex->values, 0);
        free_dvector (pcplex->slack, 0);
}
```

Código de la Función caminos

```
void   caminos(inputdata pdata, datacplex pcplex, fwinputdata fwpdata)
{   int k;
        for(k=1; k<=pdata.art; k++)
        {                   // Guarda las distancias entre los arcos, para utilizar FW
                fwinput_data(&pcplex, pdata, &fwpdata, k);
                            // Da los caminos más cortos
                fw(&fwpdata, pdata, k);
        }
}
```

Código de la Función fwinput_data

```
void fwinput_data(datacplex *pcplex, inputdata pdata, fwinputdata *fwpdata, int k)
{       int  i, j, nodo, l, m, o=0;
        int cero=0;
        int cont;
        float masgrande = 1.0e40; //Peso si el arco no existe, igual al de main
        int interruptor=0;
                /* Asignación de memoria */
        fwpdata->a = dmatrix(1, pdata.nodos, 1, pdata.nodos);
        fwpdata->p = imatrix(1, pdata.nodos, 1, pdata.nodos);
                * Lee la distancia entre nodos */
        cont=0;
        for(nodo=1; nodo<=pdata.nodos; ++nodo)
        {   for(l=nodo; l<=pdata.nodos; ++l)
            {   if(nodo==l)
                {   fwpdata->a[nodo][l]=cero;
                    fwpdata->p[nodo][l]=cero;
                }
                else
                {   j=0;
                    m=0;
                    for(i=1; i<=pdata.arc+pdata.art; i++)
                    {   if(pdata.i[1][i] > nodo)
                        {   m=1+m;
                            i=pdata.arc+m;
                            if(i<=pdata.arc+pdata.art)
                            {   if(pdata.i[2][i]>l)
                                    i=pdata.arc+pdata.art;
                            }
                            else
                                    i=pdata.arc+pdata.art;
                        }
                        if(pdata.i[1][i] == nodo)
                        {   if(pdata.i[2][i] == l)
                            {   j=1;
                                if(m==0)
                                {   fwpdata->a[nodo][l]= -pcplex->pi[cont]+pdata.uncost[k][cont+1];
                                    // Distancia del arco : -precio sombra +  costo variable del arco
```

```
                                        fwpdata->a[l][nodo] = fwpdata->a[nodo][l];
                                        //Por ser bidireccional el arco y con el mismo costo
                                        cont = cont +1;
                                        //Contabiliza los valores sombra
                                    }
                                    else        //Si es arco artificial
                                    {   fwpdata->a[nodo][l]=-pcplex->pi[pdata.arc+pdata.art+o]
                                                                +pdata.uncost[k][pdata.arc+k];
                                        //Distancia del arco: -valor precio restricciones de capacidad
                                        fwpdata->a[l][nodo]=fwpdata->a[nodo][l];
                                        //Por ser bidireccional y con las mismas distancias
                                        o++;
                                    }
                                    fwpdata->p[nodo][l]=nodo;           // Inicializa predecesores
                                    fwpdata->p[l][nodo]=l;     //Por ser bidireccional
                                    i=pdata.arc+pdata.art;
                                }
                            }
                        }
                        if(j==0)
                        {           fwpdata->a[nodo][l]= grande;
                                    fwpdata->a[l][nodo]= grande;
                                    fwpdata->p[nodo][l]= 0;
                                    fwpdata->p[l][nodo]= 0;
                        }
                    }
                }
            }
        }
```

Código del Algoritmo Floyd-Warshall

```
void fw(fwinputdata *fwpdata, inputdata pdata, int nodo)
{       int k, i, j, m;
        int nzcnt=1;
        for(k=1; k<=pdata.nodos; k++)   //Sobre todos los posibles nuevos nodos en el camino (i,k),(k,j)
        {       for(i=1; i<=pdata.nodos; i++)                   //Sobre todos los nodos inicio del arco (i,j)
                {       if(k!=i)
                        {       for(j=1; j<=pdata.nodos; j++) //Sobre todos los nodos final de (i,j)
                                {       if(j!=k && j!=i)
                                        {   if(fwpdata->a[i][j]>fwpdata->a[i][k]+fwpdata->a[k][j])
                                            {   fwpdata->a[i][j]=fwpdata->a[i][k]+fwpdata->a[k][j];
                                                        fwpdata->p[i][j]=fwpdata->p[k][j];
                                            }
                                        }
                                }
                        }
                }
        }
        i=pdata.i[1][pdata.arc+nodo];     //El camino mínimo inicia donde inicia el k-ésimo arco artificial
        fwpdata->b[1][nodo]=pdata.i[2][pdata.arc+nodo];
        j=fwpdata->p[i][fwpdata->b[1][nodo]]; //En j está el predecesor del arco final
        m=2;
        if(i!=j)
        {       do
                        //Hasta llegar al nodo inicio del camino mínimo para el k-ésimo producto
                {       fwpdata->b[m][nodo]=j;
                        j=fwpdata->p[i][j];
                        m++;
                } while (i!=j);
        }
        fwpdata->b[m][nodo]=i;              //El último nodo del camino mínimo es el inicio de éste
        m++;
        fwpdata->b[m][nodo]=0;   //Coloca un cero para indicar que ya se terminó el camino mínimo
        free_imatrix (fwpdata->p,1, pdata.nodos, 1);
        free_dmatrix (fwpdata->a,1, pdata.nodos, 1);
}
```

Código de la Función calyadd

```
void calyadd(datacplex *pcplex, inputdata pdata, fwinputdata *fwpdata, CPXENVptr env, CPXLPptr  lp,
soldata *sol)
{       int    k, i, j;
        int    cont, cols = 0;
        double *optimo;

        int    status = 0;
        int    numcols = 1;
        int    numnz = 0;
        double *obj;
        int    *matbeg;
        int    *matind;
        double *matval;
        double *lb;
        double *ub;

                / Asignación de memoria
        fwpdata->sumab = dvector(1, pdata.art);                 //Vector de costos del nuevo camino
        pcplex->an = imatrix(0,pcplex->numrows-1,1,pdata.art);      //Matriz de matriz de valores aij,
        optimo = dvector(1, pdata.art);                          //Vector criterio de optimalidad
        obj = dvector(0,0);                                 //Costo del nuevo camino, columna a añadir
        matbeg = ivector(0,0);                             //Posición en la columna primer distinto de cero
        lb = dvector(0,0);                                //Cota inferior de la nueva variable (camino nuevo)
        ub = dvector(0,0);                               //Cota superior de la nueva variable (camino nuevo)
        pcplex->nomb[2]=1;              //Inicializa el número del artículo de la columna que se añadirá
        k=0;
        do      //Para obtener costo total y coeficientes de la nueva columna
        {       k++;                                        //Para cada artículo
                for(i=0; i<=pcplex->numrows-1; i++)         //Inicializa coeficientes de nueva columna
                        pcplex->an[i][k]=0;
                fwpdata->sumab[k]=0;                        //Inicializa costo de la nueva columna
                pcplex->an[pdata.arc-1+k][k]=1;
                i=1;
                do                                          //Sobre todos los nodos
                {       for(cont=1; cont<=pdata.arc+pdata.art; cont++)
                        {       if(pdata.i[1][cont]==fwpdata->b[i][k])
                                        if(pdata.i[2][cont]==fwpdata->b[i+1][k
                                        {       fwpdata->sumab[k]=fwpdata->sumab[k] +
```

```
                                                    pdata.unorig[k][cont];
                                    if(cont<=pdata.arc)
                                            pcplex->an[cont-1][k] =1;
                                    else
                                            pcplex->an[cont-1+pdata.art][k] = 1;
                                    cont=pdata.arc+pdata.art;
                            }
                                    //Por ser bidireccional
                            if(pdata.i[2][cont]==fwpdata->b[i][k])
                                    if(pdata.i[1][cont]==fwpdata->b[i+1][k])
                                    {       fwpdata->sumab[k]=fwpdata-> sumab[k] +
                                            pdata.unorig[k][cont];
                                            if(cont<=pdata.arc)
                                                    pcplex->an[cont-1][k] =1;
                                            else
                                                    pcplex->an[cont-1+pdata.art][k] = 1;
                                            cont=pdata.arc+pdata.art;
                                    }
                    }
                    i++;
                    if(fwpdata->b[i+1][k]==0)
                            i=pdata.nodos+2;
            } while (i<=pdata.nodos+1);
            numnz = 0;      // Verifica si el nuevo camino es un arco
            for(i=0; i<=pcplex->numrows-1; i++)
            {       numnz = numnz + pcplex->an[i][k];   //Va contando los elementos no cero
                    if(numnz==3)
                            i=pcplex->numrows+2; //Rompe el for cuando no es un sólo arco
            }
            if(numnz>1)
            {       if(sol->iter>0)
                            for(i=0; i<sol->caminos; i++)      //Verifica que la columna sea nueva
                            {       numnz=0;
                                    for(j=0; j<pdata.arc; j++)
                                            if(sol->c[i][j]==pcplex->an[j][k])
                                                    numnz=numnz+1;
                                    if(numnz>=pdata.arc)
                                            i=sol->caminos+1;
                            }
                    if(numnz<pdata.arc)
                    {       optimo[k]=0;
```

```
                        //Evaluar el criterio de optimalidad para camino del artículo k
                                //Calcula -waj
                        for(i=0; i<pdata.arc; i++)
                                optimo[k]=-pcplex->pi[i]*pcplex->an[i][k]+optimo[k];
                        for(i=pdata.arc+pdata.art; i<pdata.arc+pdata.art*2; i++)
                                optimo[k]=-pcplex->pi[i]*pcplex->an[i][k]+optimo[k];
                                //A waj le suma -vk
                        optimo[k]=-pcplex->pi[pdata.arc+k-1]+optimo[k];
                                //Guarda valor del criterio de optimalidad para la columna k
                        optimo[k]=optimo[k]+fwpdata->sumab[k];
                                //Número de columnas que cumplen con el criterio
                        if(optimo[k]>=-0.009)
                                cols = cols +1;
                                //Guarda artículo del camino que dé más beneficio
                if(optimo[k] < optimo[pcplex->nomb[2]])
                                pcplex->nomb[2]= k;
                }
                else
                {       cols = cols +1;
                        optimo[k]=0;
                }
        }
        else
                cols = cols+1;
}while(k<pdata.art);
numnz=0;
if(cols == pdata.art)         //Si se cumple el criterio de optimalidad para todos los artículos
{       pcplex->nomb[2]=0;                              //Producto del que se añadió columna = 0,
}
else //Añadir columna k
{       cont=0;
        j=0;
        k = pcplex->nomb[2];
        pcplex->nomb[1] = pcplex->nomb[1]+1;     //Número de columnas que se han agregado
        for(i=0; i<=pcplex->numrows-1; i++)
        {       numnz = numnz + pcplex->an[i][k];  //Va contando los elementos no cero
        }
        //Asignación de memoria
        matind = ivector(0, numnz-1);
        matval = dvector(0, numnz-1);
```

```
for(i=0; i<=pcplex->numrows-1; i++)
{       if(pcplex->an[i][k]==1)
        {       matind[j]=i;                    //Ubicación del elemento no cero
                matval[j]=1;                    //Valor del elemento no cero (=1)
                j++;
        }
}
matbeg[0]=0;                                    //Donde inicia la columna
obj[0]=fwpdata->sumab[k];                       //Valor objetivo
lb[0]= 0.0;                                     //Cotas de la nueva variable
ub[0] = CPX_INFBOUND;
                //Añade la columna
status=CPXaddcols(env,lp,numcols,numnz,obj, matbeg, matind, matval, lb, ub, NULL);
        //Actualiza matriz de caminos
sol->caminos = sol->caminos+1;
if(sol->caminos==1)
{       // Asignación de memoria
        sol->c = imatrix(0, 0, 0, pdata.arc);           //Matriz de caminos
        sol->ctemp = imatrix(0, 0, 0, pdata.arc);       //Matriz temporal de caminos
        for(i=0; i<pdata.arc; i++)
        {       sol->c[0][i]=pcplex->an[i][k];
                sol->ctemp[0][i]=pcplex->an[i][k];
        }
        sol->c[0][i]=0;
        sol->ctemp[0][i]=0;
}
else
{       // Asignación de memoria
        sol->c = imatrix(0, sol->caminos-1, 0, pdata.arc);      //Matriz de caminos
        for(j=0; j<sol->caminos-1; j++)
                for(i=0; i<=pdata.arc; i++)
                        sol->c[j][i]=sol->ctemp[j][i];
        for(i=0; i<pdata.arc; i++)
                sol->c[j][i]= pcplex->an[i][k];
        sol->c[j][i]= 0;
        sol->ctemp = imatrix(0, sol->caminos-1, 0, pdata.arc); //Matriz temporal
        for(j=0; j<sol->caminos; j++)
                for(i=0; i<=pdata.arc; i++)
                        sol->ctemp[j][i]=sol->c[j][i];
}
```

```
        }
        free_imatrix (fwpdata->b,1, pdata.nodos+1, 1);
        free_imatrix (pcplex->an ,0,pcplex->numrows-1,1);
        free_dvector (obj,0);                   //Costo del nuevo camino, columna a añadir
        free_ivector (matbeg,0)                 //Posición en la columna del primer distinto de cero
        free_dvector (lb,0);                    //Cota inferior de la nueva variable (camino nuevo)
        free_dvector (ub,0);                    //Cota superior de la nueva variable (camino nuevo)
}
```

Código de la Función tamano

```
void   tamano(datacplex *pcplex, soldata *sol)
{       int j;
        //Actualiza el tamaño del vector de solución
        if(sol->iter==0)
        {       sol->x = dvector(0,pcplex->numcols+1);
                sol->y = dvector(0,pcplex->numcols+1);
                sol->tam= pcplex->numcols;
        }
        else
        {       //Liberación y asignación de memoria
                free_dvector (sol->x,0);
                sol->x = dvector(0,pcplex->numcols+1);
                if(sol->tam<pcplex->numcols)
                {       for(j=0; j<sol->tam; j++)
                                sol->x[j]=sol->y[j];
                        free_dvector (sol->y, 0);
                        sol->y = dvector(0,pcplex->numcols+1);
                        for(j=0; j<sol->tam; j++)
                                sol->y[j]=sol->x[j];
                        sol->tam = pcplex->numcols;
                }
        }
}
```

Código de la Función factible

```
void   factible(soldata *sol, inputdata pdata, datacplex pcplex)
{       double cero = 0.000001;
        sol->tiposol=0;
        sol->xfrac = pdata.arc-1;           //Inicializa el contador de variables artificiales
        sol->paro=0;                        //Marcador del ciclo de búsqueda del tipo de la solución
        do
        {       sol->xfrac++;
                  //Si se llega a pdata.arc+pdata.art sig. que no hubo variables artificiales en la solución
                if(sol->xfrac==pdata.arc+pdata.art)
                {       sol->tiposol=1;   //tiposol != 1 si la solución no es factible
                        sol->paro=1;      //Termina la búsqueda al revisar las variables artificiales
                        if(sol->primcot[1]==-1)                 //Para conocer la primera cota
                                sol->primcot[1]=-2;
                        else
                                sol->primcot[1]=-3;
                }
                if(sol->x[sol->xfrac]>cero)   //Termina la búsqueda al encontrar la 1ª variable artificial
                        sol->paro=1;
        }while(sol->paro==0);
        if(pcplex.solstat!=1)
                sol->tiposol=0;             //Solución no factible
}
```

Código de la Función entero

```
void   entero(soldata *sol, inputdata pdata)
{       int j=-1;           //j=-1 la solución es entera; j=índice del primer fraccionario
        int valorint;
        double cero = 0.000001;
                //Analiza si la solución es entera
        for(sol->xfrac=0; sol->xfrac<pdata.arc + pdata.art; sol->xfrac++)
        {       valorint=sol->x[sol->xfrac];
                if(sol->x[sol->xfrac]-valorint>cero)
                {       j=sol->xfrac;
                        sol->xfrac = pdata.arc+pdata.art+1;          //Rompe el for
                }
        }
        if(j==-1)
                sol->xfrac=-1;
}
```

Código de la Función cotainf

```
void	cotainf(soldata *sol, datacplex *pcplex, ramadata rama)
{	if(sol->primcot[1]==-2)
	{	if(rama.fin ==0)
			sol->primcot[0]=pcplex->objval;
		else
			if(sol->primcot[0]<sol->cotainf)
				sol->primcot[0]=sol->cotainf;
		printf("La cota general: %lf\n", sol->primcot[0]);
	}
		//Cota inferior más grande, solución factible y menor al incumbente
	if(sol->primcot[1]==-3 && pcplex->objval > sol->cotainf && sol->tiposol==1 && sol->xfrac!=-1 && pcplex->objval < sol->incumbente)
	{	sol->cotainf=pcplex->objval;
		sol->primcot[1]=-4;
	}
}
```

Código de la Función branch

```
void   branch(soldata *sol, inputdata pdata, datacplex pcplex)
{       int i, j;
        int k=0;
        int valorint;
        double cero = 0.000001;
        double costo;
                //Solución factible y no sobrepasa la cota
        if(pcplex.objval < sol->incumbente && sol->tiposol==1)
        {       sol->sta = 3;       //Estado de la solución. sta==3: Solución factible y no entera
                        //Actualiza el Incumbente
                if(sol->xfrac==-1)
                {       if(sol->time[0]==0.0)
                        {       sol->inctime = clock();
                                sol->time[0] = (double) (sol->inctime –
                                        sol->cerotime)/CLOCKS_PER_SEC;
                                printf("Tiempo del Primer incumbente: %lf\n", sol->time[0]);
                        }
                        sol->soltime = clock();
                        sol->time[1] = (double) (sol->soltime - sol->cerotime)/CLOCKS_PER_SEC;
                        printf("Tiempo de Solución: %lf\n", sol->time[1]);
                        sol->incumbente=pcplex.objval;
                                //Actualiza el incumbente
                        for(j = 0; j < pcplex.numcols; j++)
                                sol->y[j]=sol->x[j];
                        sol->sta=2;
                                //Estado: Solución factible y entera
                        printf("En la iteración %d, se encontró el incumbente: %lf\n", sol->iter,
                                sol->incumbente);
                        printf("Se tiene %d columnas.\n", pcplex.numcols);
                        sol->primcot[1]=-4;
                }
                else    //Busca el camino que permitiría fijar más arcos a 1, según la solución actual
                {       //Asignación de memoria
                        sol->cumple = dvector(0, sol->caminos-1);
                        for(i=0; i<sol->caminos; i++)
                        {       if(sol->c[i][pdata.arc]==-1)
                                {       sol->cumple[i] = 0.0;
                                        costo=0.0;
```

```
                        //Si el camino i está en la solución óptima del subproblema
                        if(sol->x[i+pdata.arc+pdata.art*2]>0.000001)
                        {       for(j=0; j<pdata.arc; j++)
                                {       //Si en el camino i existe el arco j && Si la solución
                                        > 0 && no se ha marcado antes el arco
                                        if(sol->c[i][j]==1 && sol->x[j] > cero && sol-
                                        >arco[j]==-1)
                                        {       sol->cumple[i] = sol->cumple[i] + 1;
                                                costo=costo+pdata.cost[j+1];
                                        }
                                }
                                //También toma en cuenta el costo de construir estos arcos
                                if(sol->cumple[i]>0)
                                        sol->cumple[i]=sol->cumple[i]/costo;
                                }
                                if(sol->cumple[i]>sol->cumple[k])
                                        k=i;
                        }
                }
                sol->crama = k;         //Camino al que pertenece la variable de la próxima rama
                        // Encuentra el primer fraccionario del camino sobre el que se ramificará a 1,
                        para enviarlo a la pila
                j=-1;
                for(sol->xfrac=0; sol->xfrac<pdata.arc; sol->xfrac++)
                {       if(sol->c[sol->crama][sol->xfrac]==1 && sol->arco[sol->xfrac]==-1)
                        // Si en el camino i existe el arco j y no se fijó antes el arco
                        {       if(j==-1)
                                        j=sol->xfrac;
                                valorint=sol->x[sol->xfrac];
                                if(sol->x[sol->xfrac]-valorint>cero &&
                                pdata.cost[sol->xfrac+1]<pdata.cost[j+1])
                                        j=sol->xfrac;
                                }
                        }
                        sol->xfrac=j;                                   // Variable que se enviará a la pila
                        free_dvector (sol->cumple,0);
                }
        }
        else
        {       if(sol->tiposol!=1)  //Solución no factible
```

```
                    sol->sta = 1;
                else    //Agotar la rama por el criterio de acotamiento
                    sol->sta = 4;
            }
        }
```

Código de la Función push

```
int push(StackNodePtr *topPtr, TNODE *info, StackNodePtr *toptop, int itt)
{       StackNodePtr newPtr;
        newPtr = (StackNode *)malloc(sizeof(StackNode));
        if (newPtr != NULL)
        {       newPtr->info = info;                    //El nodo que quedará hasta arriba de la pila
                if(itt==0)
                {       newPtr->nextPtr = *toptop;
                        itt=1;
                }
                else
                        newPtr->nextPtr = *topPtr; //De dónde viene
                *topPtr = newPtr;                       //Pone en hasta arriba de la pila el nodo ramificado
        }
        else
        {       printf( "Not inserted. No memory available.\n");
        }
        return itt;
}
```

Código de la Función pop

```
TNODE *pop(StackNodePtr *topPtr )
{       TNODE *popValue;
        popValue = (*topPtr)->info;
        *topPtr = ( *topPtr )->nextPtr;
        return popValue;
}
int isEmpty( StackNodePtr topPtr )
{
        return topPtr == NULL;
}
```

APÉNDICE C

Formato del Archivo de Datos del Problema

En la siguiente tabla se muestra el formato que debe tener el conjunto de datos de una red para diseñarla con el código propuesto.

Número de nodos Número de artículos Número de arcos
Vector de costos Fijos de los arcos
Vector de capacidades de los arcos
Demandas de los productos
Vector de nodos inicio de los arcos disponibles Vector de nodos final de los arcos disponibles
Matriz de costos unitarios por transportar una unidad de flujo del producto *k* por los arcos disponibles

Figura C-1. Formato de Instancias

Para ejemplificar el formato, considérese la siguiente red

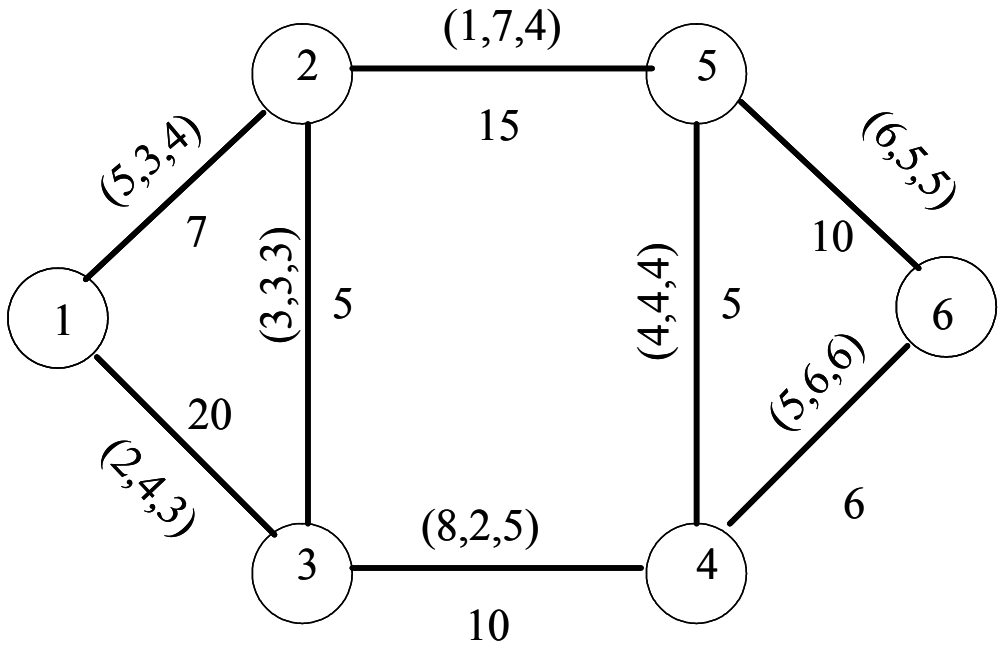

Figura C-2. Ejemplo de Red

A continuación se muestra el archivo correspondiente con el que se alimentará el programa:

```
6   2   8

4   3   3   4   5   4   6   5

7   20  5   15  10  5   6   10

10  5

1   1   2   2   3   4   4   5   1   2
2   3   3   5   4   5   6   6   5   6

5   2   3   1   8   4   5   6
3   4   3   7   2   4   6   5
```

Figura C-3. Ejemplo de Formato de Instancias

Obsérvese los arcos (1, 5) y (2, 6); los cuales corresponden a los arcos artificiales que van del nodo origen del producto *k* a su nodo destino.

APÉNDICE D

Instancia p3010f14. Resuelto con un Heurístico

Nod=30 Arc=343 Com=10 Sce=1

Termino la lectura de p3010fl4.txt...

Restricciones de capacidad

Temino las restricciones de capacidad

Liberando la memoria

Ya poblo el problema

Default variable names x1, x2 ... being created.

Default row names c1, c2 ... being created.

Ya escribio el problema

Tried aggregator 1 time.

Reduced MIP has 643 rows, 7203 columns, and 20923 nonzeros.

Presolve time = 0.05 sec.

MIP emphasis: balance optimality and feasibility

Root relaxation solution time = 0.36 sec.

	Nodes				Cuts/			
Node	Left	Objective	IInf	Best Integer		Best Node	ItCnt	Gap
	0 0	63981.0000	19			63981.0000	1257	
*	0+ 0		0	91850.5000		63981.0000	1257	30.34%
		70222.3472	46	91850.5000	Cuts:	42	1754	23.55%
		72885.7588	51	91850.5000	Cuts:	27	2196	20.65%
		73518.3899	63	91850.5000	Cuts:	17	2463	19.96%
		74282.8810	71	91850.5000	Cuts:	20	2845	19.13%
		74613.1474	76	91850.5000	Cuts:	17	3082	18.77%
		74799.5387	76	91850.5000	Flowcuts:	6	3219	18.56%
		74999.0813	83	91850.5000	Flowcuts:	3	3405	18.35%
		75135.5050	80	91850.5000	Flowcuts:	5	3576	18.20%
		75154.8076	84	91850.5000	Flowcuts:	2	3625	18.18%
		75241.4732	81	91850.5000	Flowcuts:	8	3768	18.08%
		75291.4216	80	91850.5000	Flowcuts:	3	3858	18.03%
		75322.9268	86	91850.5000	Flowcuts:	3	3953	17.99%

```
                75365.9426    87    91850.5000  Flowcuts:  2     4022   17.95%
                75402.3911    87    91850.5000  Flowcuts:  4     4077   17.91%
                75415.7553    86    91850.5000  Flowcuts:  3     4152   17.89%
                75428.8893    85    91850.5000  Flowcuts:  2     4195   17.88%
                75429.3847    85    91850.5000       Cuts:  0     4204   17.88%
     100   100    81447.2365    33    91850.5000    75435.3495    13487   17.87%
     200   200    83494.7988    16    91850.5000    75435.3495    19103   17.87%
     300   300    85647.5892    15    91850.5000    75435.3495    22022   17.87%
     400   392    82548.8991    40    91850.5000    75442.8787    30735   17.86%
     500   492    88803.6955    41    91850.5000    75442.8787    38361   17.86%
     600   580    84444.9290    43    91850.5000    75556.0488    44041   17.74%
     700   680    89070.9289    12    91850.5000    75556.0488    45896   17.74%
     800   777    85529.1368    57    91850.5000    75588.3852    52202   17.70%
     900   871    85096.6223    28    91850.5000    75651.5505    61618   17.64%
    1000   971    89579.9527    23    91850.5000    75651.5505    65001   17.64%
Elapsed time =  49.91 sec. (tree size = 10.10 MB)
*  1000+  683               0    88258.0000    75651.5505    65001   14.28%
*  1000+  671               0    88137.0000    75651.5505    65001   14.17%
*  1000+  646               0    87241.0000    75651.5505    65001   13.28%
*  1000+  639               0    87123.0000    75651.5505    65001   13.17%
*  1000+  621               0    86571.5000    75651.5505    65001   12.61%
    1100   716    84974.1914    26    86571.5000    75656.4365    72207   12.61%
    1200   810    83395.3021    58    86571.5000    75659.0847    77379   12.61%
    1300   904    82513.1786    58    86571.5000    75661.2918    83923   12.60%
    1400  1001    83507.6471    38    86571.5000    75676.4160    93311   12.59%
    1500  1087    85583.4411    35    86571.5000    75680.3693   101650   12.58%
    1600  1183    84628.8493    25    86571.5000    75682.4193   108978   12.58%
    1700  1267    82697.2223    34    86571.5000    75741.9759   115450   12.51%
    1800  1363    83834.8961    12    86571.5000    75750.1793   122947   12.50%
    1900  1449    79215.0635    72    86571.5000    75792.2093   126517   12.45%
    2000  1545    83786.9911    52    86571.5000    75844.4234   138422   12.39%
Elapsed time =  95.97 sec. (tree size = 16.34 MB)
    2100  1641    84486.1057    31    86571.5000    75866.7784   146459   12.37%
    2200  1733    84672.8258    27    86571.5000    75871.9704   153054   12.36%
```

```
   2300  1825    84828.5349    38    86571.5000    75878.8561   160549   12.35%
   2400  1914    78091.3829    77    86571.5000    75930.9211   169796   12.29%
   2500  2004    77623.5866    80    86571.5000    75940.0167   178159   12.28%
   2600  2100    79246.9138    69    86571.5000    76040.3809   187399   12.16%
   2700  2196    83555.4804    20    86571.5000    76065.2165   195606   12.14%
   2800  2292    84395.7332    53    86571.5000    76210.1351   201497   11.97%
   2900  2388    83948.3391    29    86571.5000    76216.4531   209720   11.96%
   3000  2478    80555.5187    42    86571.5000    76217.3193   214643   11.96%
Elapsed time = 142.94 sec. (tree size = 26.44 MB)
   3100  2576    85431.4358    17    86571.5000    76217.3193   218634   11.96%
   3200  2665    79206.5075    55    86571.5000    76232.7225   226670   11.94%
   3300  2755    81682.6390    60    86571.5000    76239.5427   238347   11.93%
   3400  2851    85547.8232    33    86571.5000    76246.1371   247181   11.93%
   3500  2941    84374.1331    29    86571.5000    76271.9066   259209   11.90%
   3600  3033    76504.2578    75    86571.5000    76304.0825   267323   11.86%
   3700  3129    84628.0196    20    86571.5000    76333.9193   279060   11.83%
   3800  3225    85736.7826    19    86571.5000    76338.7984   287762   11.82%
   3900  3309    78630.6442    78    86571.5000    76368.4524   298372   11.79%
   4000  3405    78573.5772    39    86571.5000    76380.1047   307050   11.77%
Elapsed time = 200.19 sec. (tree size = 36.46 MB)
*  4000+ 2703                    0    85302.0000    76380.1047   307050   10.46%
   4100  2799    84077.4173    20    85302.0000    76380.1047   311840   10.46%
   4200  2885    81597.5429    61    85302.0000    76418.3554   321126   10.41%
   4300  2981    85022.1678    51    85302.0000    76425.2224   331099   10.41%
   4400  3067    80216.0962    68    85302.0000    76441.7242   344252   10.39%
   4500  3157    81255.6321    32    85302.0000    76480.9863   357226   10.34%
   4600  3251    81453.9477    56    85302.0000    76484.4875   362781   10.34%
   4700  3337    82256.1546    42    85302.0000    76530.8506   373395   10.28%
   4800  3431    83844.0915    17    85302.0000    76535.3761   381282   10.28%
   4900  3521    78810.3213    69    85302.0000    76551.9270   389194   10.26%
   5000  3617    83576.1706    41    85302.0000    76553.5808   399847   10.26%
Elapsed time = 254.95 sec. (tree size = 38.87 MB)
```

Instancia p3050vt5 Resuelto con Método Exacto

Cota inferior: 177461.234375
Tiempo del Primer incumbente: 14.063000
Tiempo de Solución: 14.063000
En la iteración 84, se encontró el incumbente: 313620.000000
Son 610 columnas.
Tiempo de Solución: 17.688000
En la iteración 93, se encontró el incumbente: 311986.000000
Son 643 columnas.
Tiempo de Solución: 18.406000
En la iteración 100, se encontró el incumbente: 310484.000000
Son 645 columnas.
Tiempo de Solución: 19.172000
En la iteración 105, se encontró el incumbente: 309494.000000
Son 649 columnas.
Tiempo de Solución: 22.250000
En la iteración 131, se encontró el incumbente: 308105.000000
Son 663 columnas.
Tiempo de Solución: 23.797000
En la iteración 148, se encontró el incumbente: 307254.000000
Son 666 columnas.
Tiempo de Solución: 41.516000
En la iteración 281, se encontró el incumbente: 307216.000000
Son 753 columnas.
Tiempo de Solución: 47.625000
En la iteración 335, se encontró el incumbente: 307214.000000
Son 776 columnas.
Tiempo de Solución: 48.641000
En la iteración 349, se encontró el incumbente: 307131.000000
Son 776 columnas.
Tiempo de Solución: 58.500000
En la iteración 444, se encontró el incumbente: 305712.000000
Son 807 columnas.
Tiempo de Solución: 73.031000
En la iteración 627, se encontró el incumbente: 304861.000000
Son 821 columnas.
Tiempo de Solución: 105.438000
En la iteración 1020, se encontró el incumbente: 304816.000000
Son 862 columnas.
Tiempo de Solución: 106.000000
En la iteración 1023, se encontró el incumbente: 303822.000000
Son 865 columnas.
Tiempo de Solución: 158.828000
En la iteración 1698, se encontró el incumbente: 302971.000000
Son 905 columnas.
2000
Tiempos: 182.750000
Tiempo de Solución: 186.688000

En la iteración 2053, se encontró el incumbente: 302063.000000
Son 923 columnas.
Tiempo de Solución: 188.156000
En la iteración 2073, se encontró el incumbente: 301980.000000
Son 923 columnas.
Tiempo de Solución: 212.281000
En la iteración 2386, se encontró el incumbente: 301387.000000
Son 931 columnas.
Tiempo de Solución: 213.938000
En la iteración 2408, se encontró el incumbente: 301271.000000
Son 931 columnas.
Tiempo de Solución: 247.047000
En la iteración 2842, se encontró el incumbente: 301212.000000
Son 935 columnas.
Tiempo de Solución: 248.344000
En la iteración 2860, se encontró el incumbente: 301151.000000
Son 935 columnas.
Tiempo de Solución: 274.844000
En la iteración 3215, se encontró el incumbente: 300536.000000
Son 938 columnas.
Tiempo de Solución: 276.313000
En la iteración 3235, se encontró el incumbente: 300442.000000
Son 938 columnas.
4000
Tiempos: 355.047000
Tiempo de Solución: 376.469000
En la iteración 4200, se encontró el incumbente: 300080.000000
Son 1166 columnas.
Tiempo de Solución: 377.594000
En la iteración 4214, se encontró el incumbente: 299997.000000
Son 1166 columnas.
Tiempo de Solución: 508.578000
En la iteración 5601, se encontró el incumbente: 299404.000000
Son 1313 columnas.
Tiempo de Solución: 509.891000
En la iteración 5617, se encontró el incumbente: 299288.000000
Son 1313 columnas.
6000
Tiempos: 543.641000
Tiempo de Solución: 621.500000
En la iteración 6869, se encontró el incumbente: 299229.000000
Son 1372 columnas.
Tiempo de Solución: 622.469000
En la iteración 6881, se encontró el incumbente: 299168.000000
Son 1372 columnas.
8000
Tiempos: 724.750000
Tiempo de Solución: 765.531000
En la iteración 8448, se encontró el incumbente: 298553.000000
Son 1448 columnas.

Tiempo de Solución: 766.735000
En la iteración 8462, se encontró el incumbente: 298459.000000
Son 1448 columnas.
Tiempo de Solución: 788.406000
En la iteración 8696, se encontró el incumbente: 298136.000000
Son 1462 columnas.
Tiempo de Solución: 803.016000
En la iteración 8870, se encontró el incumbente: 297766.000000
Son 1462 columnas.
Tiempo de Solución: 811.203000
En la iteración 8964, se encontró el incumbente: 297609.000000
Son 1464 columnas.
Tiempo de Solución: 828.047000
En la iteración 9147, se encontró el incumbente: 296581.000000
Son 1471 columnas.
10000
Tiempos: 904.203000
Tiempo de Solución: 917.328000
En la iteración 10150, se encontró el incumbente: 295872.000000
Son 1487 columnas.
Tiempo de Solución: 1070.406000
En la iteración 11868, se encontró el incumbente: 295752.000000
Son 1517 columnas.
12000
Tiempos: 1082.266000
Tiempo de Solución: 1201.453000
En la iteración 13289, se encontró el incumbente: 295043.000000
Son 1565 columnas.
14000
Tiempos: 1265.688000
Tiempo de Solución: 1455.078000
En la iteración 15902, se encontró el incumbente: 294531.000000
Son 1655 columnas.
16000
Tiempos: 1466.547000
18000
Tiempos: 1660.547000
Tiempo de Solución: 1841.391000
En la iteración 19903, se encontró el incumbente: 293822.000000
Son 1710 columnas.
20000
Tiempos: 1852.141000
22000
Tiempos: 2055.547000
24000
Tiempos: 2257.938000
26000
Tiempos: 2456.438000
Tiempo de Solución: 2520.000000
En la iteración 26623, se encontró el incumbente: 293702.000000

Son 1831 columnas.
28000
Tiempos: 2665.828000
30000
Tiempos: 2871.672000
32000
Tiempos: 3072.688000
Tiempo de Solución: 3128.610000
En la iteración 32550, se encontró el incumbente: 292993.000000
Son 1938 columnas.
34000
Tiempos: 3285.594000
36000
Tiempos: 3490.656000
Tiempo de Solución: 3641.891000
En la iteración 37367, se encontró el incumbente: 292865.000000
Son 2050 columnas.
38000
Tiempos: 3708.750000
40000
Tiempos: 3922.235000
42000
Tiempos: 4136.172000
Tiempo de Solución: 4174.328000
En la iteración 42371, se encontró el incumbente: 292745.000000
Son 2119 columnas.
44000
Tiempos: 4349.297000
46000
Tiempo de Solución: 4604.563000
En la iteración 46320, se encontró el incumbente: 292036.000000
Son 2165 columnas.
48000
Tiempos: 4801.516000

APÉNDICE E

Utilizar Pila y Cola

Considérese el siguiente árbol de solución:

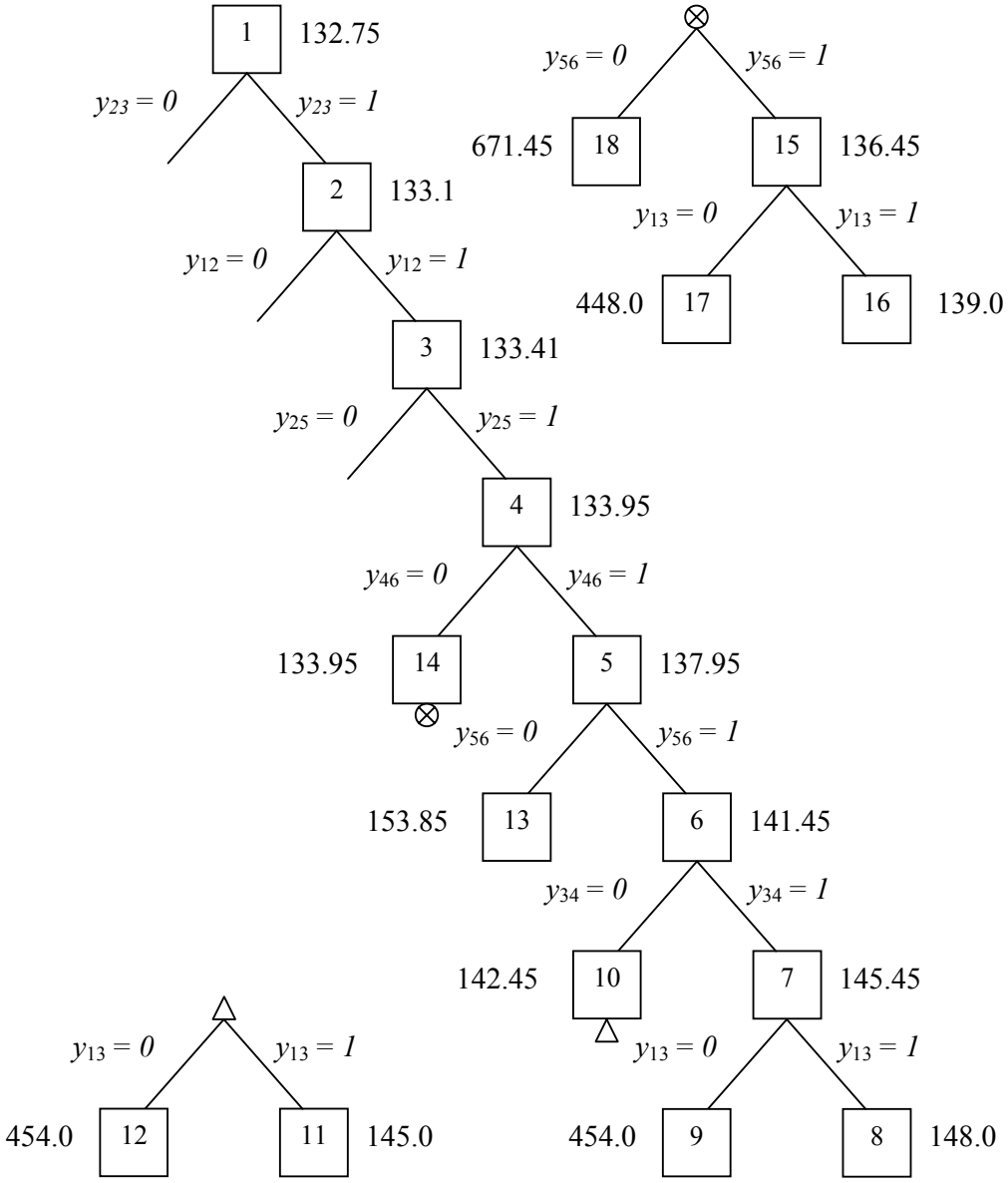

Figura E-1. Ejemplo de Red con Pila

En la iteración 16, se tiene la pila de la Figura E-2.

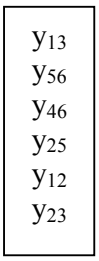

Figura E-2. Pila sin Cola

Utilizando una estructura de Cola, se podría enviar la variable y_{46} a la cola y eliminar algunas variables de la pila (Figura E-3), a continuación se realizaría backtrack sobre la variable y_{25}.

Al quedar vacía la pila, se procederá a analizar los subproblemas generados por las variables de la cola.

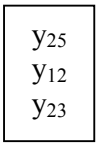

Figura E-3. Pila con Cola

Con esto se esperaría reducir el número de subproblemas por resolver ya que podría encontrarse de manera más rápida un mejor incumbente, con lo que algunos de los elementos de la pila no será necesario analizar o las ramas de éstos serán agotadas de manera más rápida.

Solución Inicial Obtenida por un Heurístico

Para iniciar con una solución entera factible sin variables artificiales, el procedimiento propuesto en este trabajo se modificará con el siguiente pseudocódigo:

1. Leer la solución
2. Añadir la columna correspondiente
3. Llenar pila
4. Modificar valores
    - Incumbente
    - sol.tam
    - sol.arco
    - sol.y

Supóngase que algún heurístico eficiente como el propuesto por De Alva [15] se utiliza para encontrar una solución factible, por ejemplo:

$(y_{12}, y_{15}, y_{24}, y_{28}, y_{34}, y_{39}, y_{45}, y_{56}, y_{59}, y_{68}, y_{78}, y_{79}) = (1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1)$

Dicha columna se agregará al modelo inicial por resolver, con lo que se obtendrá una disminución en el primer incumbente encontrado.

El árbol correspondiente a dicha solución es el que se observa en la Figura E-4, donde se han fijado a 1 las variables que aparecen en la solución inicial factible.

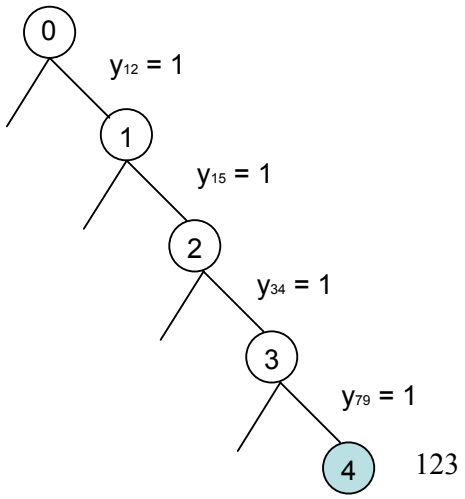

Figura E-4. Árbol de Solución con Heurístico

En este paso del procedimiento, la pila es la que se observa en la Figura E-5.

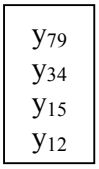

Figura E-5. Pila con Solución de Heurístico

Se inicializan los valores de la siguiente manera:

- Incumbente = 123;
- sol.tam = pcplex->numcols + 1;
- sol.arco = (0, 1, -1, -1, 4, -1, -1, -1, -1, -1, -1, 11)

- sol.y = (1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1)

Al tener rápidamente un mejor incumbente, se espera agotar más fácilmente ramas debido al criterio de acotamiento.

Añadir un Nodo Artificial

Considérese la siguiente red

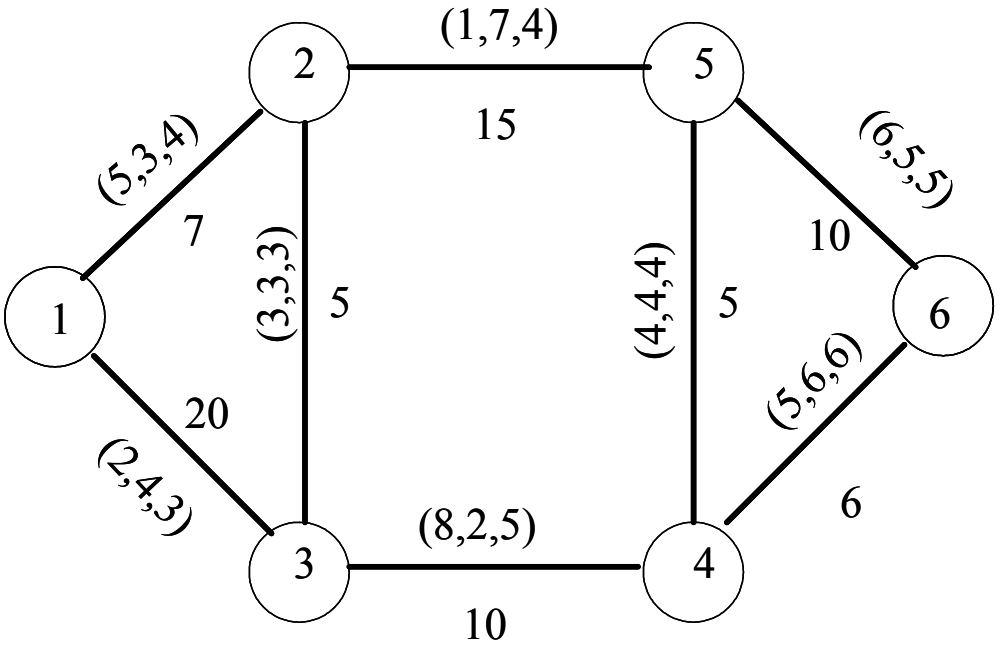

Figura E-6. Ejemplo de Red

Supongamos que se desea enviar un producto del nodo 1 al nodo 2. El algoritmo tendrá dos arcos que unen un mismo par de nodos, a saber el artificial y el disponible, sin embargo esto puede corregirse agregando el nodo k y el arco artificial (1, k), como en la Figura E-7.

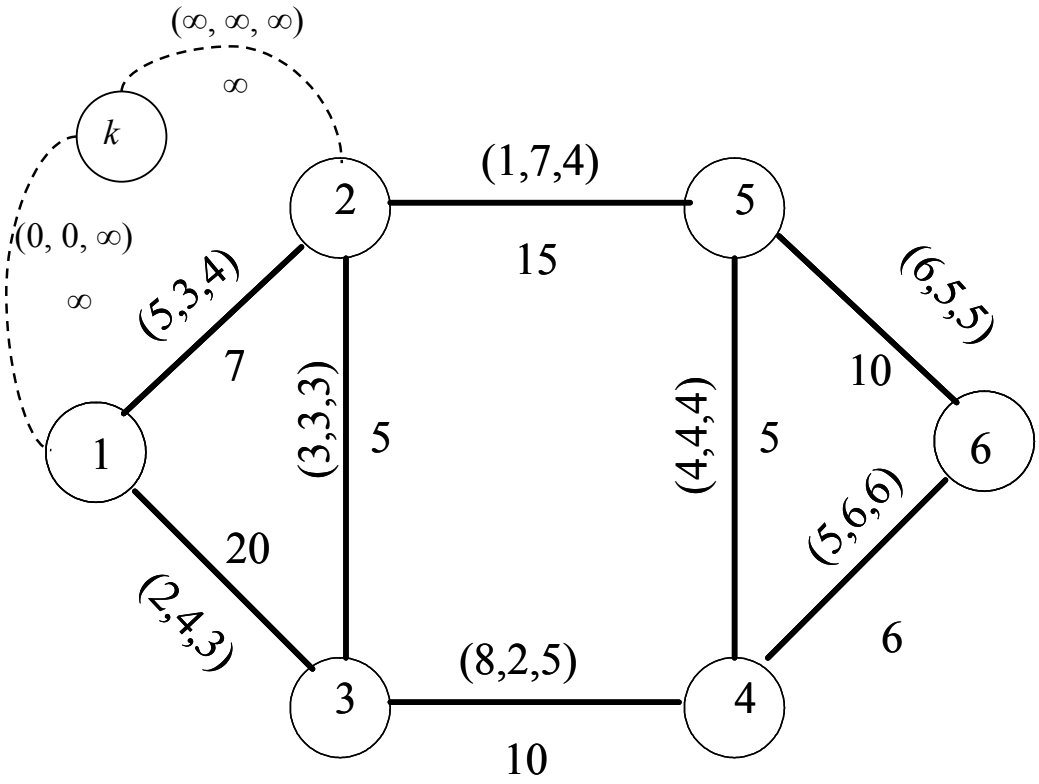

Figura E-7. Ejemplo de Red con Nodo Artificial