Diseño robusto de red overlay mediante heurísticas y codificación de modelo exacto

(Proyecto en el Marco de Convenio ANTEL-FING)

INFORME DE PROYECTO DE GRADO PRESENTADO AL TRIBUNAL EVALUADOR COMO REQUISITO DE GRADUACIÓN DE LA CARRERA INGENIERÍA EN COMPUTACIÓN

ESTUDIANTES

MARCELO LABANCA

MARCELO PÍRIZ

NATALIA RIVERA

MARTIN SELLANES

BAJO LA SUPERVISIÓN DE

DR. ING. FRANCO ROBLEDO

MSC. CARLOS TESTURI

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
URUGUAY, MONTEVIDEO, NOVIEMBRE DE 2010

Resumen General

El presente trabajo tiene como objetivo definir el diseño óptimo de una red overlay que se modela en dos capas: una red de datos y una red de transporte que permiten separar respectivamente, que nodos se conectan y con que volumen de tráfico, de las rutas físicas y reales que estos tráficos recorren.
Se definirá que enlaces de la red de datos deben participar de la solución y para cada uno se hallará la asignación óptima de tecnología (cada tecnología se corresponde con una capacidad diferente). Esta asignación de tecnologías deberá contemplar las demandas de tráfico entre los nodos, estás demandas son dato del problema. Los nodos de la red de datos son conocidos y no variarán.
Para cada enlace incluido en la red de datos, se deberá fijar el camino que recorrerá el tráfico correspondiente, en la red de transporte.

Como principal condición la red a hallar deberá ser robusta respecto a una falla simple, esto quiere decir que si en la red de transporte falla uno (solo uno a la vez) de los enlaces, las demandas se deben seguir cumpliendo. Se deberá garantizar para cada demanda, un camino alternativo en la red de datos cuando uno de los enlaces físicos del camino típico falla.

Para lograr esto se implementaron dos heurísticas que calculan una solución aproximada y también se implemento un algoritmo Branch and Bound para hallar una solución óptima.

La primera heurística implementada buscaba una solución inicial que sirviera de punto de partida para el algoritmo Branch and Bound. En tal sentido y por la temprana familiarización con el problema, se logró rápidamente una solución factible pero que descuida aspectos importantes para lograr una solución aproximada a la óptima.

Paralelamente, a partir de un modelo lineal binario del problema, que nos fue proporcionado, se diseño un algoritmo Branch and Bound para encontrar una solución exacta. Se encontró que el software Cplex ya contaba con la implementación de algoritmos Branch and Cut (que es una variante de Branch and Bound), por lo que se utilizaron sus bibliotecas para implementarlo.

Por último al notar que el algoritmo Branch and Bound no lograba hallar la solución para nuestro caso real (por las dimensiones del problema), se implementó una segunda heurística. Esta vez al pretender la solución final y por tener una gran vinculación con el problema, se obtuvo una heurística que contempla todos los aspectos a optimizar y logra una buena solución.

Índice

1. Introducción

1.1. Contexto Marco

El trabajo de nuestro proyecto surge en el marco de la temática de la actividad “Optimización de costos bajo diseño robusto en redes multiovelay” de fecha 18/08/08 del convenio entre ANTEL y la Facultad de Ingeniería, de tal forma que el problema y el juego de datos reales pertenecen a la empresa mencionada [1].

Como mencionamos en el resumen general, la solución buscada es una red overlay, a continuación profundizaremos en la comprensión de estas redes para contextualizar y comprender nuestro trabajo. Una red overlay, es aquella que virtualiza sus conexiones sobre una o más redes existentes. Los nodos de datos en un overlay pueden pensarse como conectados por enlaces virtuales o lógicos, cada uno de los cuales corresponde a un camino a través de enlaces físicos de la red de transporte que se encuentra por debajo. Por ejemplo las redes peer to peer son redes overlay porque sus nodos corren encima de Internet. Por otro lado Internet se construyó como un overlay encima de la red telefónica [7].

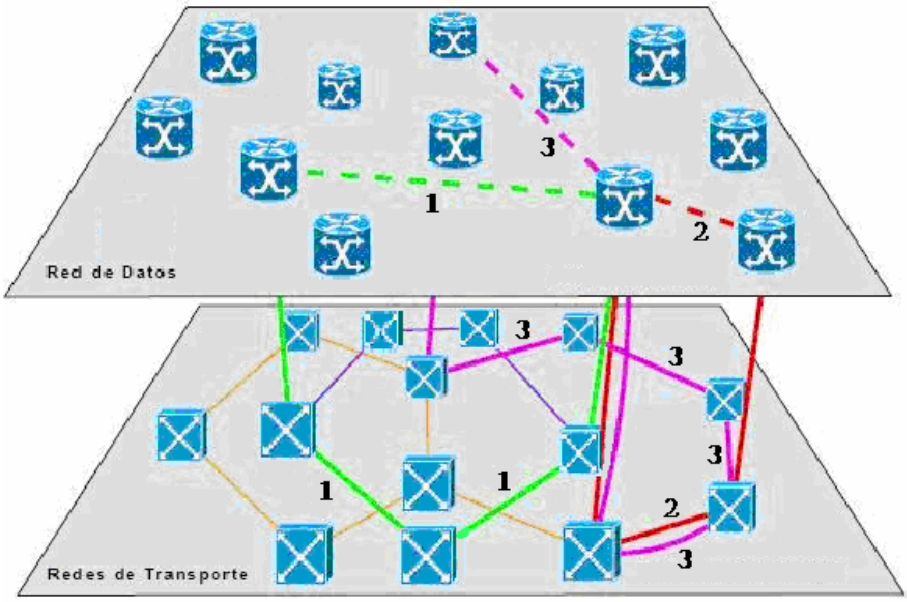

Figura 1.1.1: Red en Overlay.

En la figura 1.1.1 se visualiza un diagrama de una red Overlay. Por encima aparece la red de datos que tiene doce nodos y cuatro de ellos conectados. Las líneas punteadas simbolizan los enlaces virtuales o lógicos, es decir que entre estos nodos va a circular información. Sin embargo los enlaces físicos reales, por los que circulará la información, son los de la red de transporte que se visualizan debajo. Se observa por ejemplo que el enlace virtual 3 de la red de datos, se rutea en los cuatro enlaces de la red de transporte que están numerados con el número 3.

En el diagrama hay líneas que salen de los nodos de la red de trasporte hacia arriba, estas simbolizan la correspondencia que hay entre los nodos de la red de datos y los de la red de transporte. Se define para cada nodo de la red de datos cual es su correspondiente nodo en la red de transporte. En nuestro proyecto este mapeo se llama función *tns* (transport network station).

Recordemos que la red overlay a encontrar, tiene como condición soportar una falla simple. Una falla simple corresponde a que uno (solo uno) de los enlaces de la red de transporte deja de estar disponible. Esto puede pasar por ejemplo al cortarse un cable de fibra óptica que forme parte de esta red.
Al fallar un enlace de la red de transporte, automáticamente dejan de estar habilitados todos los enlaces de la red de datos que utilizan el enlace caído para rutear información. Si en la figura 1.1.1 cae el enlace 2 en la red de transporte, el enlace punteado 2 en la red de datos también caerá.
Para que se soporten estos tipos de falla, los nodos de la red de datos deben tener enlaces alternativos para rutear la información al dejar de estar disponibles los enlaces que típicamente utilizan. El ruteo nuevo se hace en la red de datos porque para cada enlace de esta red, el ruteo en la red de transporte es fijo.

De lo anterior se desprende que la red a encontrar tendrá distintas maneras de rutear los datos, una más utilizada que corresponde a un escenario sin fallas y otras correspondientes a los distintos escenarios de falla simple en la red de transporte.

1.2. Introducción al problema

El problema a resolver es diseñar una red overlay óptima, es decir, la que tenga costo mínimo. A continuación veremos cuales son los costos a minimizar.
Se señalan dos frentes a tomar en cuenta en la optimización, por un lado definir una red de datos óptima y por otro elegir en la red de transporte, los ruteos de información más cortos. Estos dos aspectos hay que resolverlos a la vez ya que no son independientes. Sin embargo al mirar las redes de datos y transporte por separado y las condiciones que estas deben cumplir, se visualizan los puntos relevantes a optimizar.

Veamos en primer lugar las características que debe tener la red de datos, y la información del problema que está vinculada.

Los nodos de la red de datos están fijados de ante mano, parte de nuestro trabajo es definir entre cuales nodos se van a establecer enlaces para la circulación de información. Además a cada enlace incluido se le deberá asignar una tecnología, es decir cuanto volumen de información permitirá.
Las distintas tecnologías permitidas son dato del problema, así como el costo que tiene cada una por unidad de distancia. Cuanto más grande es la capacidad de información de una tecnología, más alto es su costo, esto deberá ser tenido en cuenta para hallar una red de costo mínimo. Se deberá intentar asignar las tecnologías más pequeñas que logren cumplir con las restricciones del problema. La asignación de tecnologías está restringida inferiormente, por las demandas de tráfico de información entre los nodos, que deberá ser cumplida.

Se consideraban en principio dos tipos de tráfico de datos: comprometido y eventual. Aún en cualquier escenario de falla simple el tráfico comprometido debía poder ser cursado. El tráfico eventual solo estaría disponible cierto porcentaje del tiempo. Se define el factor de calidad z_Q, como la fracción de tiempo que el tráfico eventual estaría 100% utilizable.
Como simplificación en el problema a resolver en este Proyecto de Grado se considera que el factor de calidad z_Q vale uno para todos los casos. Por lo cual el tráfico eventual debe ser considerado disponible el 100% del tiempo. Consideraremos entonces en nuestro problema un solo tráfico, suma del comprometido y el eventual.

Las demandas son dato del problema. Se conoce la matriz M^0 y las matrices M^e, indican cuanta información debe poder circular entre los nodos. La M^0 representa las demandas, entre todo par de nodos, en el escenario sin fallas y las M^e las demandas para cada escenario de falla donde cae un enlace físico *e* de la red de transporte.
Las matrices de demandas son cuadradas simétricas, la cantidad de filas y columnas es igual a la cantidad de nodos de la red de datos. La manera de leerlas es la siguiente: si $M^0[1,2] = d$ quiere decir que en el escenario sin fallas se debe satisfacer una demanda *d* entre los nodos uno y dos. Análogamente se lee la información de las matrices M^e para cada escenario de falla.

Por un lado entonces, aparece como factor relevante a resolver en nuestro proyecto, el de fijar la menor cantidad de enlaces en la red de datos y para los mismos las tecnologías más pequeñas que permitan satisfacer todas las demandas de tráfico de información entre los nodos. Además de considerar el tráfico a garantizar en el escenario sin fallas y en cada escenario de falla simple, la elección de la red de datos está condicionada por la topología de la red transporte. A continuación explicaremos estos condicionantes.

Antes de analizar la red de transporte y los aspectos a definir sobre ésta, debemos comprender la función *tns* que es la que vincula físicamente la red de datos con la de transporte. Ésta función es un mapeo entre los nodos de la red de datos y los de la red de transporte. Para cada nodo *nd* de la red de datos existe *tns*(*nd*) = *nt*, donde *nt* es el nodo de transporte correspondiente a *nd*. Si un nodo *nd* de la red de datos envía información, estos datos circularan por la red de transporte partiendo de *tns*(*nd*).

Veamos ahora la red de transporte y la influencia de está en la solución a encontrar.

La topología de la red de transporte es conocida, se trata de una estructura anillada donde se conocen que nodos están conectados y las distancias que los separan. Se deberá decidir en esta red que camino seguirá la información, es decir, que para cada enlace incluido en la red de datos habrá un camino correspondiente en la red de transporte. La tecnología de los dispositivos de esta red, no permiten redireccionar información dinámicamente como si lo permiten los de la red de datos. Por tal motivo, el ruteo elegido para cada enlace de la red de datos es fijo.
El aspecto importante a tener en cuenta al elegir los caminos, es minimizar las distancias de los mismos.

Revisamos los puntos claves a optimizar, ahora veremos la condición de que la red buscada sea robusta respecto a una falla simple y como esto hace más complejo el diseño.

Se habla de una falla simple cuando un enlace en la red de transporte deja de estar disponible, el adjetivo simple corresponde a que solo uno de los enlaces dejó de funcionar. Si uno de los enlaces de la red de transporte es un cable de fibra óptica y por alguna razón este se corta, estaríamos en presencia de una falla simple.
Debemos distinguir entonces dos tipos de escenarios para el funcionamiento de la red, uno en el que no ocurre ninguna falla, que llamaremos “escenario sin fallas” y otro en el que ha ocurrido una falla simple, que llamaremos “escenario de falla *e*”, donde *e* es el enlace que dejó de estar disponible en la red de transporte.

Supongamos que tenemos el diseño de la red de datos para el escenario sin fallas cumpliendo con las demandas que figuran en M^0, y para cada enlace de dicha red, el ruteo correspondiente en la red de transporte. Si ocurre una falla simple en un enlace *e*, se verán afectados todos los enlaces de la red de datos que utilizan a *e* en sus ruteos. Para que la red sea robusta, se deberá encontrar caminos alternativos, para que circule la información de los enlaces perjudicados. Como mencionamos

antes, la red de transporte no tiene la capacidad de rerutear información dinámicamente, por ejemplo ante una falla simple, pero sí la red de datos. Por tal razón también se deberá tomar en cuenta en el diseño de la red de datos, que cuando ocurra una falla simple en un enlace *e*, todas las demandas de información expresadas en la matriz M^e correspondiente, deben poder circular sin utilizar los enlaces caídos (los que utilizan a *e* en sus ruteos).

Observamos todas las condiciones que debe cumplir la red a hallar y los aspectos importantes a minimizar para encontrar una red de costo mínimo. Llamaremos “costo de la solución” a la función que dada una red solución para nuestro problema, nos de el costo de implementarla. El resultado de esta función es lo que se busca minimizar y se calcula: sumando para cada enlace *ed* de la red de datos, el resultado de multiplicar el costo de la tecnología asignada a *ed*, por la distancia del camino elegido para *ed* en la red de transporte. Cabe aclarar que la moneda en la que se calcula esta función, es la misma en la que se expresan los costos de las tecnologías. Por ejemplo, si el costo de las tecnologías se expresa en dólares, el costo de una solución estará expresado en dólares.

1.3. Motivación

El desafío que se plantea en este proyecto de grado nos resulta interesante, ya que se trata de un problema real de una empresa de telecomunicaciones estatal. Buscamos la solución de costo mínimo, que satisfaga las demandas de los clientes (aún en escenarios de falla), por lo que su resolución afecta positivamente al país, ya que implica un mejor aprovechamiento de los recursos disponibles y una reducción de los gastos.

Durante el proceso inicial de investigación del estado del arte en cuanto a estrategias para resolver este problema, nos damos cuenta de que la solución requiere de mucha creatividad, ya que es necesario adaptar estrategias de resolución existentes o pensar nuevas (no existe un algoritmo solución ya elaborado). Dada la complejidad computacional del problema, las estrategias que existen para resolverlo (ver sección 2.2) se presentan de forma genérica y debemos ver si aplican para nuestro caso particular. Esto nos plantea un gran desafío, ya que para elaborar nuevas estrategias debemos aplicar adecuadamente los conocimientos adquiridos en la carrera, y para adaptar estrategias ya elaboradas debemos profundizar en su investigación. Debemos decidir la mejor forma de proceder.

Por último la finalización de este proyecto de grado nos permite culminar una etapa importante de nuestras vidas.

1.4. Objetivos

El objetivo general de este proyecto es hallar la solución al problema que tenga el menor costo posible. Es decir, encontrar la asignación de tecnologías en la red de datos que soporte las demandas requeridas entre los clientes (en el escenario sin fallas y ante cualquier falla simple), y los correspondientes ruteos en la red de transporte para cada enlace de la red de datos, que resulten en el menor costo posible. El costo se calcula sumando para cada enlace de la red de datos, el resultado de multiplicar el precio de la tecnología asociada al enlace por la distancia del camino en la red de transporte asociado a ese enlace.

Avanzando en el conocimiento de estrategias para resolver este problema, se plantean objetivos más específicos, como diseñar e implementar una heurística que permita hallar una solución aproximada a la óptima y un algoritmo arborescente del tipo Branch and Bound (ver sección 2.2), que permita hallar la solución optima al problema.

En primera instancia se busca una heurística sencilla que permita obtener una solución aproximada de forma rápida. Esta solución se utilizará como cota superior en el algoritmo arborescente, de forma de simplificar su recorrida.

Se quiere diseñar e implementar el algoritmo arborescente basándose en el esquema Branch and Bound.
Se investigará y utilizará el software Cplex que cuenta con librerías para la implementación eficiente de este tipo de algoritmos. [8]

Una vez implementado el algoritmo Branch and Bound y comprobados los grandes tiempos de cálculo que requiere para encontrar la solución óptima, se plantea el nuevo objetivo de realizar otra heurística más compleja que permita hallar una solución de menor costo que la encontrada por la primera heurística.

Ambas heurísticas serán realizadas en base a conocimientos adquiridos en la carrera, a la creatividad e innovación, no se basaran en una metaheurística en particular. Esta decisión fue tomada en conjunto con los tutores, dada la escasa experiencia del grupo en cuanto a adaptar alguna metaheurística existente, a este problema particular.

1.5. Método de solución

El trabajo realizado se compone de tres partes claramente diferenciadas:
1. El desarrollo de una heurística que logra una solución factible pero alejada de la óptima.
2. La realización de una segunda heurística más precisa que logra una mejor solución.
3. La implementación de un algoritmo Branch and Bound para la resolución exacta del problema.

Cada una de estas partes otorga una solución para nuestro problema pero también fueron combinadas para encontrar mejores soluciones. A continuación se describe el proceso de desarrollo de las mismas.

Además de la descripción general del problema, el insumo principal que tuvimos para realizar el proyecto fue un Modelo lineal binario, que representa el problema a resolver. Este modelo se adjunta en el Anexo I. El mismo nos fue proporcionado por los tutores del proyecto.

A partir del modelo se tuvo una comprensión mayor del problema estudiando la función objetivo y las restricciones del mismo. Por otra parte al avanzar en la investigación del software Cplex y notar que este cuenta con bibliotecas para la resolución exacta de este tipo de modelos, tuvimos el puntapié inicial para comenzar a desarrollar una solución.
Se transcribió el modelo a un formato que Cplex interpreta y se implementó un algoritmo Branch and Bound para la resolución exacta del mismo.

Paralelamente se diseño una heurística que tenía como cometido encontrar una solución factible que sirviera de cota superior para el algoritmo Branch and Bound, para limitar el espacio de búsqueda. En tal sentido se logró terminar antes que el algoritmo exacto. Al comenzar a desarrollar la heurística enfocamos el diseño con la idea de que la distancia de los caminos en la red de transporte era el aspecto más importante a optimizar. Además creyendo que el resultado más importante iba a hacer el que obtuviera el algoritmo exacto. Por tales motivos se obtuvo rápidamente un diseño pero que luego notamos, no encuentra buenas soluciones, es decir, cercanas a la óptima. Esto se debe fundamentalmente a no tomar en cuenta el otro aspecto importante a minimizar que es la asignación de tecnologías en la red de datos.

Luego continuamos con la etapa de pruebas, donde se logró ejecutar el algoritmo exacto teniendo como cota superior el costo de la solución encontrada por la heurística. En esta etapa se constató que para nuestro juego de datos reales, el algoritmo exacto no finalizaba su ejecución por las dimensiones de la entrada. Es decir que lo que teníamos para nuestro caso de estudio real era la solución encontrada por la heurística que como mencionamos no era buena.

Se decidió entonces encarar otro diseño para implementar una segunda heurística. Esta vez enfocados a encontrar una solución final y tratando de considerar todos los aspectos a optimizar de manera simultánea. Dado la mayor familiarización con el problema en esta etapa, se obtuvo un diseño que ataca todos los puntos relevantes y luego de implementado se constató que encuentra

soluciones más cercanas a la óptima para nuestro caso real. Destacamos entonces esta parte de nuestro trabajo como el mayor aporte a la resolución del problema.

1.6. Conclusiones

Para el tipo de problema que nos toca resolver existen métodos de resolución exacta y otros que encuentran buenas soluciones que para la realidad son aceptables.
Los métodos exactos generalmente dependen del tamaño de la entrada para funcionar. En nuestro caso las dimensiones son muy grandes para resolver con algoritmos exactos por lo cual la utilización de métodos aproximados como las heurísticas parece ser la mejor opción.

Resulta mucho más fácil resolver un problema de estas características a través de heurísticas, cuando se tiene experiencia en la resolución de problemas similares o cuando se tiene un dominio de técnicas aplicando metaheurísticas. Nuestro equipo no contaba con estas ventajas por lo que las heurísticas desarrolladas surgen a partir de la profunda compresión del problema, de las distintas técnicas de resolución adquiridas en la carrera, de la creatividad e innovación.

Una gran familiarización con la realidad que se plantea resultó fundamental para el desarrollo de las soluciones implementadas, en nuestro caso costo llegar a dominar todo el dominio del problema y se logró de forma gradual. Esto se refleja en las dos heurísticas desarrolladas, la primera, en los comienzos del proyecto, es muy sencilla y descuida aspectos importantes para la obtención de una buena solución. La segunda, ya al final del proyecto, es más compleja y ataca todos los problemas a resolver.

Como resultados se tienen dos heurísticas que permiten obtener soluciones factibles. Por otra parte se tiene un algoritmo exacto Branch and Cut implementado con el software Cplex, que busca encontrar una solución exacta al problema.

El algoritmo exacto no logró encontrar una solución para nuestro caso de estudio real dadas las dimensiones del mismo. Para ejemplos de dimensiones menores el algoritmo permite hallar el óptimo. Se valora sin embargo, la investigación de un software preparado para resolver problemas de optimización.

Se destaca como resultado de nuestro proyecto, la implementación de la segunda heurística, que además de encontrar una solución factible para nuestro caso de estudio real, el costo de esta solución es de la mitad que el de la solución encontrada por la primera heurística

1.7. Estructura del Informe

El presente documento está estructurado de la siguiente forma:

- Al inicio del documento se presenta un resumen general del problema a resolver.
- En el Capítulo 1 se realiza una introducción al problema.
- En el Capítulo 2 se presentan los conceptos básicos para entender el proyecto así como también el estado del arte en cuanto a estrategias para resolverlo.
- El Capítulo 3 corresponde a la parte central del trabajo, allí se describen las heurísticas implementadas, el diseño de un algoritmo Branch and Bound y la implementación de un algoritmo Branch and Cut utilizando Cplex. También se describen los ambientes que se utilizaron para el desarrollo de los algoritmos.
- En el Capítulo 4 se encuentran los casos de prueba y sus resultados.
- En el Capítulo 5 se encuentran las conclusiones y el trabajo a futuro.
- Finalmente en el capítulo 6 se encuentran las referencias.
- Se presentan nueve Anexos:
  - El Anexo I contiene el Modelo lineal binario que representa el problema a resolver.
  - El Anexo II contiene definiciones de conceptos básicos de optimización.
  - El Anexo III contiene los archivos de entrada y de salida de cada una de las soluciones implementadas, para el caso de prueba 1.
  - El Anexo IV contiene los archivos de entrada y de salida de cada una de las soluciones implementadas, para el caso de prueba 2.
  - El Anexo V contiene los archivos de entrada y de salida de cada una de las soluciones implementadas, para el caso de prueba 3.
  - El Anexo VI contiene los archivos de entrada y de salida de cada una de las soluciones implementadas, para el caso de prueba 4.
  - El Anexo VII describe las operaciones de los módulos de la *Primera solución aproximada*.
  - El Anexo VIII describe las operaciones de los módulos de la *Segunda solución aproximada*.
  - El Anexo IX describe el archivo de salida de la *Solución utilizando Cplex*.

Las secciones más importantes que se deben leer para entender el problema son:

- Resumen General.
- Sección 1.2, Introducción al problema.
- Sección 1.4, Objetivos.
- Sección 2.1, Conceptos Básicos.
- Dentro de la sección 2.2, Estado del arte en cuanto a estrategias para resolver el problema, las secciones: 7a) Branch and Bound y 7b) Branch and Cut.
- Capítulo 3, Parte central del trabajo.
- Capítulo 5, Conclusiones y trabajo a futuro.

2. Contexto

En este capítulo se definen los conceptos que son relevantes para comprender este proyecto y que se utilizarán en todo el documento. Posteriormente se incluye un resumen del estado del arte en cuanto a diferentes estrategias para resolver el problema.

2.1. Conceptos básicos de este proyecto

A continuación se explican conceptos que se consideran básicos para entender este proyecto. Los mismos se agrupan según refieran a la red de transporte, a la red de datos, o vinculen ambas redes.

2.1.1. Conceptos referentes a la red de transporte

- **Red de transporte**: corresponde a las estaciones (nodos) y conexiones físicas existentes (enlaces) que permiten la comunicación de los datos.
- **Falla simple**: corresponde a la falla de un solo enlace de la red de transporte.
- **Distancia**: es una medida de la dimensión que tiene un enlace de la red de transporte.

2.1.2. Conceptos referentes a la red de datos

- **Red de datos**: es una red que se encuentra en una capa superior a la red de transporte. Cada nodo de la red de datos tiene un correspondiente nodo en la red transporte. Los nodos de la red de datos se conectan por enlaces virtuales o lógicas, cada uno de los cuales corresponde a un camino a través de enlaces físicos de la red de transporte.
- **Tecnología**: corresponde a un par capacidad, costo que se asocia a los enlaces de la red de datos. La capacidad hace referencia a la cantidad de tráfico que soporta esa tecnología y costo al valor monetario de la misma. La capacidad de la tecnología debe ser soportada por los enlaces de transporte que se encuentran por debajo, que son los que efectivamente trasportan la información. En este proyecto de grado todas las tecnologías que se manejan son soportadas por los enlaces de transporte. Las tecnologías se identifican con un número entero creciente desde uno. Se reserva el valor cero para indicar que a un enlace no se le asignó ninguna tecnología, es decir que no existe el enlace.

- **Demanda**: corresponde a la cantidad de tráfico que se requiere que puedan intercambiar, todo par de nodos de la red de datos. Si un par de nodos no requieren tráfico entre si, la demanda es cero. La demanda deberá ser soportado por la capacidad de la tecnología que se asocié a los enlaces que conectan al par de nodos. Entre un mismo par de nodos se tiene un valor de demanda para el escenario sin fallas, y un valor de demanda para cada escenario de falla simple. Estos valores pueden ser distintos entre si.

- **Tráfico comprometido**: Es el tráfico entre un par de nodos de la red de datos que debe ser cursado el 100% del tiempo. Aún en cualquier escenario de falla simple el tráfico comprometido debe poder ser cursado.

- **Tráfico eventual**: Es el tráfico entre un par de nodos de la red de datos que solo está disponible cierto porcentaje del tiempo. Se define el factor de calidad z_Q, como la fracción de tiempo que el tráfico eventual está 100% utilizable. A título informativo, el tráfico eventual se asocia con el tráfico Best Effort (comúnmente asociado a Internet). Estos clientes aceptan degradaciones eventuales del nivel de servicio en un rango tolerable (subjetivo) [1].

2.1.3. Conceptos que vinculan la red de datos y la red de transporte

- ***tns***: abreviatura de transport network station, es una función que toma un nodo de la red de datos (capa superior) y devuelve el nodo de la red de transporte que le corresponde (capa inferior).

- **Escenario de falla**: representa el conjunto red de transporte, red de datos, con los cambios producidos por la falla simple de un enlace de transporte. En la red de transporte se representa la caída del enlace correspondiente y en la red de datos se deben considerar las nuevas demandas entre los clientes (estas demandas dependen del enlace que falló y son dato de entrada al problema). Hay tantos escenarios de falla como enlaces en el grafo de transporte.

- **Escenario sin fallas**: representa el conjunto red de transporte, red de datos, en el caso que no falla ningún enlace de la red de transporte.

- **Ruteo**: Es la elección de un camino en la red de transporte para comunicar dos nodos de la red de datos.

- **Camino más corto**: se utiliza para indicar una secuencia de enlaces en el grafo de transporte (ruteo), entre un nodo origen y un nodo destino dados, tal que la suma de la distancias de estos enlaces, es la menor posible entre las distintas secuencias de enlaces que

pueden existir que conecten ese origen y destino. Dados dos nodos *i*, *j* de la red de datos, se busca el camino más corto en transporte entre *tns*(*i*) y *tns*(*j*).

- **Solución Factible**: La solución consiste en el dimensionamiento de los enlaces de la red de datos y el ruteo de los mismos en la red de transporte. Para que la solución sea factible el dimensionamiento debe permitir que todas las demandas de datos puedan ser entregadas, no habiendo ninguna falla en la red de transporte y también ante la caída de cualquiera de los enlaces de transporte (una caída a la vez).

- **Costo de una solución**: Dada una solución factible a nuestro problema, el costo se calcula sumando los productos del "*costo de la tecnología asignada al enlace"* y la "*distancia del ruteo designado para dicho enlace*". Esto para cada enlace de la red de datos con tecnología distinta de cero, es decir, enlace que se incluyo en la solución.

2.2. Estado del arte en cuanto a estrategias para resolver el problema

A continuación se encuentra un resumen de diversas estrategias para resolver este tipo de problemas. De todas las estrategias presentadas, las utilizadas para resolver el problema son: ***Método heurístico*** y ***Métodos arborescentes (Branch and Bound)***.
Se pueden consultar definiciones de conceptos generales de optimización en el Anexo II.

1. *Algoritmo aproximado [2]*

Un ***algoritmo aproximado*** es un algoritmo que construye una solución que no se puede garantizar que sea óptima. Para problemas de gran tamaño se utilizan métodos heurísticos. Los métodos heurísticos no trabajan eficazmente sobre todos los datos. Por esta razón muchas veces se trabaja con muchas soluciones introduciendo, para la construcción de soluciones, sorteos o por ejemplo por métodos de "vecinos". Los métodos heurísticos son generalmente no validados matemáticamente pero eficientes en la práctica.

2. *Método heurístico [2]*

Un ***método heurístico*** es un algoritmo aproximado o procedimiento inexacto que está bien definido paso a paso y que permite llegar a una solución de buena calidad rápidamente. Las heurísticas son utilizadas en problemas donde se prioriza el tiempo de cálculo frente a la calidad de las soluciones. El resultado de una heurística depende de su capacidad de adaptarse a casos particulares, de que evite quedar atrapada en mínimos locales y de su capacidad de exploración de la estructura del problema.

Algunas técnicas para mejorar una heurística son: preprocesamiento de datos, diseño y utilización de estructuras de datos eficientes, utilización de procedimientos aleatorios en forma controlada (para diversificar el espacio de soluciones obtenidas), búsqueda intensa en regiones "promisorias" según cierto criterio, etc.

3. *Metaheurísticas [2]*

Las ***Metaheurísticas*** son una clase de métodos aproximados que dan una metodología general, un marco de trabajo que permite la creación de nuevos algoritmos híbridos por combinación de diferentes conceptos derivados de las heurísticas clásicas, etc. Algunos ejemplos de metaheurísticas son: Búsqueda Local, Búsqueda Tabú, GRASP, ANT-Systems, etc.. Las metaheurísticas utilizan estrategias de aprendizaje de forma de poder estructurar la información para buscar eficientemente soluciones cercanas a la óptima.

Características de las metaheurísticas:

- Su objetivo es explorar eficientemente el espacio de búsqueda, de forma de encontrar soluciones (sub)óptimas.
- Son estrategias para guiar los procesos de búsqueda.
- Son algoritmos aproximados y no determinísticos.
- Incorporan mecanismos para evitar óptimos locales.
- No son específicas al problema que se intenta resolver, incorporan el conocimiento específico del problema o la experiencia (memoria) "desviando" la búsqueda.
- Amplio espectro: desde técnicas sencillas como la búsqueda local a técnicas complejas (procesos de aprendizaje, por ejemplo).

Diferentes formas de clasificar las metaheurísticas:

- Según se inspiren en procesos naturales o no.
- Basadas en poblaciones versus un único punto o métodos de trayectoria.
- Dinámicas versus funciones objetivo estáticas.
- Una versus varias estructuras de vecindades.
- Según use o no memoria.

4. Resumen de las metaheurísticas revisadas

Dentro de las que están basadas en un único punto o métodos de trayectoria se estudiaron metaheurísticas como Búsqueda Local, Búsqueda Tabú y GRASP (Greedy Randomized Adaptive Search Procedure). Antes de pasar a dar un pequeño resumen sobre estas metaheurísticas se define el concepto de vecindad ya que será utilizado.

Definición: Una *estructura de vecindad* es un mapeo $\mathcal{N}:\Omega\rightarrow\Omega$, el cual define para cada $S\in\Omega$, un conjunto $\mathcal{N}(S)\subseteq\Omega$ de soluciones llamadas "vecinas" de S. El conjunto $\mathcal{N}(S)$ se llama *"vecindad"* de S y cada elemento $S'\in\mathcal{N}(S)$ es una solución *"vecina"* de S [2].

Nota: Las soluciones no necesariamente son factibles, la factibilidad se maneja a nivel de la implementación (en el mecanismo de generación de vecinos, o en el criterio de aceptación de soluciones, etc.) [2].

a. Búsqueda Local [2]

La Búsqueda Local es también llamada mejora iterativa ya que los movimientos se realizan sólo si se mejora la solución.
Pasos del algoritmo:

s ← *Generar SolucionInicial*()
Repetir
    s ← *Mejorar*(*s*,$\mathcal{N}$(*s*))

Hasta no hay mejora posible

En la figura 2.3.1 se describe un ejemplo de búsqueda local con cuatro iteraciones.

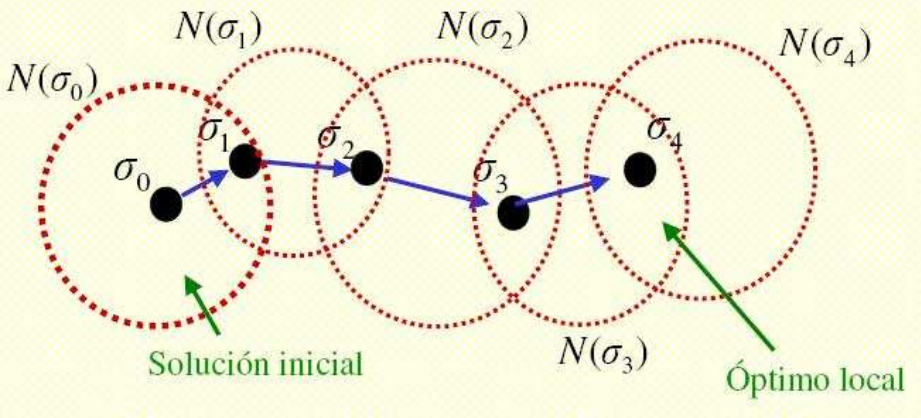

Figura 2.2.1: Ejemplo de búsqueda local con cuatro iteraciones. Se parte de la solución inicial σ_o, explorando en cada iteración la vecindad de la solución actual (denotada en la figura como N). En la iteración cuatro, no se encuentra una solución mejor en la vecindad de σ_4, por lo que la búsqueda local termina. Notar que σ_4 es un óptimo local no necesariamente el óptimo global.

En la figura 2.3.2 se ilustra mediante una curva, posibles óptimos locales que pueden ser hallados con la búsqueda local, pero que no necesariamente corresponden al óptimo global.

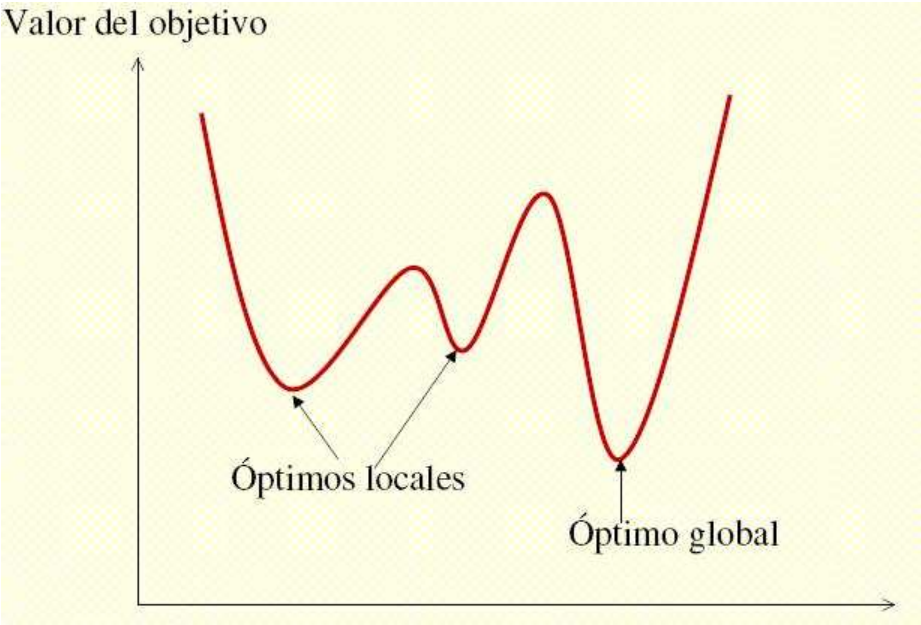

Figura 2.2.2: Óptimos locales y óptimo global.

b. Búsqueda Tabú [2]

La Búsqueda Tabú utiliza una búsqueda local con memoria a corto plazo que le permite "escapar" de mínimos locales y evita ciclos. La memoria de corto plazo esta representada por la lista Tabú la cual registra las últimas soluciones "visitadas" e impide volver a ellas en los próximos movimientos. La lista Tabu se actualiza normalmente en forma FIFO.

Pasos del algoritmo:

s ← *GenerarSolucionInicial*()
ListaTabu ← ∅
Mientras no condición fin hacer
    s ← *MejorSolucion*(*s*, $\mathcal{N}$(*s*) \ *ListaTabu*)
    Actualizar(*ListaTabu*)
Fin Mientras

Algunas de las posibles condiciones de fin son:

- Tiempo máximo de CPU.
- Cantidad de iteraciones.
- Alcanzar una solución *s*, tal que *f*(*s*) mejor que un cierto nivel prefijado de antemano, siendo *f* la función a optimizar.
- No se obtiene una solución mejor después de varias iteraciones
- Todas las soluciones en $\mathcal{N}$(*s*) están prohibidas por la lista tabú, o sea, no hay ningún movimientos permitido

El largo de la lista tabú controla la memoria del proceso de búsqueda. Una lista tabú corta, controla áreas reducidas del espacio de búsqueda y una larga, fuerza a una búsqueda en áreas mayores. El largo de la lista puede cambiar a lo largo del proceso de búsqueda.

c. GRASP [2]

GRASP (Greedy Randomized Adaptive Search Procedure) es una heurística que combina procedimientos constructivos y de búsqueda local.

Pasos del algoritmo:

Mientras no condición fin hacer
    s ←*ConstruirSolucionGRASP*()
    AplicarBusquedaLocal(*s*)
    MemorizarMejorSolución()
Fin Mientras

GRASP es un procedimiento iterativo en dos fases: una de construcción de la solución y otra de mejora.

Pasos del algoritmo para construcción de una solución:

s ←∅ (*s* es una solución parcial en este caso)
Mientras no condición fin hacer
RCL ← *GenerarListaCandidatosRestringidos*()
x ←*SeleccionRandom*(*RCL*)
s ← *s* U *x*
ActualizarFuncionGreedy(*s*)
Fin Mientras

Es un método de búsqueda local e iterativo (ya que una solución factible es construida en cada iteración)
En cada iteración se realiza lo siguiente:

- Se elige el próximo elemento: se ordenan, de acuerdo a una función "golosa", todos los elementos disponibles en una lista de candidatos.
- Se elige aleatoriamente un candidato entre los mejores de la lista de candidatos.

Es una heurística adaptativa ya que los beneficios asociados a cada elemento son actualizados, de forma de tomar en cuenta la elección hecha en el paso anterior
RCL es la lista restringida de mejores candidatos. La elección aleatoria en *RCL* permite generar soluciones diferentes en cada iteración.

d. ANT systems-Colonias de agentes cooperativos [2]

Dentro de las metaheurísticas basadas en poblaciones (en cada iteración se trabaja con un conjunto o población de soluciones) se revisó la metaheurística ANT systems o Colonias de agentes cooperativos.

- Se basa en imitar el comportamiento de las colonias de hormigas.
- Cada hormiga considerada individualmente tiene capacidades básicas, pero la colonia logra en conjunto un comportamiento inteligente, como resultado de la interacción entre las hormigas.
- Pese a ser insectos casi ciegos logran encontrar el camino mas corto entre el hormiguero y la fuente de comida.
- La comunicación se logra por medio de sustancia química dejada como rastro.
- Una hormiga se mueve esencialmente al azar pero si encuentra una huella dejada previamente es muy probable que la siga, reforzándola además con su propio rastro.
  Esta idea sirve como base para esta heurística, que tiene las siguientes características:
  - Retroalimentación positiva.
  - Cálculo distribuido que evita la convergencia prematura.
  - Uso de una heurística greedy constructiva que ayuda a encontrar soluciones aceptables en las primeras etapas del proceso de búsqueda.
- Los agentes cooperativos (hormigas) pueden tener memoria, tener cierta inteligencia, etc.

Como resumen comparativo de los métodos basados en trayectoria (aquí vimos Búsqueda Local, Búsqueda Tabú y GRASP) y los basados en poblaciones (aquí vimos ANT-systems), podemos decir lo siguiente:

- Los métodos basados en **poblaciones** son mejores en identificar posibles "buenas áreas" en el espacio de búsqueda (permiten dar "pasos largos"), que los métodos de **trayectoria** los cuales son mejores para explorar localmente las áreas.
- Algunos métodos de trayectoria tratan de dar "pasos más largos", pero normalmente los pasos no son guiados. En los métodos de población hay mecanismos para guiar las búsquedas y hay una memoria de más largo plazo.
- Las heurísticas "híbridas" tratan de aprovechar las bondades de ambos tipos de métodos y han tenido éxito en algunas aplicaciones, es un tipo de técnica a continuar mejorando en el futuro.

5. Métodos exactos para problemas NP difíciles

Dentro de los métodos exactos para problemas NP difíciles se encuentran los métodos arborescentes y programación dinámica. A continuación se describe cada uno de estos métodos [2].

a. Métodos arborescentes (Branch and Bound)

Métodos arborescentes: son métodos guiados de exploración del espacio de soluciones factibles, donde la estrategia de recorrido depende del problema particular a tratar. Dentro del grupo de métodos arborescentes se encuentran los procedimientos Branch and Bound [2].

A continuación se describen los principios básicos de operación de los procedimientos Branch and Bound, y de dos de sus variantes como son Branch and Cut y el Algoritmo Aditivo de Balas.

i. Branch and Bound [2]

Sea un problema entero (IP): Min $f(x)$ / $x \in S$
Los procedimientos Branch and Bound permiten determinar por enumeración implícita las soluciones realizables de S y encontrar la solución óptima en S. La exploración de soluciones realizables sigue dos principios: branching y bounding.

Branching: permite reducir el estudio de S al de subconjuntos de S cada vez más reducidos, de forma de terminar resolviendo problemas muy sencillos (por ejemplo de un sólo elemento). Normalmente esta subdivisión de S es una partición.

Bounding: Para cada subconjunto de *S* obtenido por branching se calcula una cota por defecto de *f*(*x*) en ese subconjunto o función de evaluación (para un problema de mínimo, de lo contrario es una cota superior).

El proceso de exploración es representado por una arborescencia. La raíz del árbol es todo el conjunto *S*, a partir de allí se construye el árbol. El modo de construcción de la arborescencia depende de la elección del nodo del árbol que se elige para ramificar (proceso de branching).

Los procedimientos de Branch and Bound se realizan en profundidad, la solución optima corresponde a la hoja de mejor evaluación.

• Un nodo S_i de la arborescencia no se divide más si:
  – S_i no factible, o sea $S_i = \phi$
  – Se encontró la solución óptima
  – Toda solución en S_i es un valor dominado: $f(x) \geq f(y) \forall x \in S_i$ para algún $y \in S_j \,/\, j \neq i$

Una forma de encontrar una cota inferior es relajar las restricciones enteras. En el caso de problemas de minimización un nodo S_i no se divide más si el valor objetivo del problema relajado en ese nodo es mayor que el mejor valor objetivo logrado hasta el momento para una solución entera. Esto se fundamenta en que al tratarse de un problema relajado se sabe que su valor objetivo es una cota inferior del la solución entera que podría lograrse a partir de ese nodo. Si ya se cuenta con una solución entera que es mejor que la solución del problema relajado en ese nodo, cualquier solución entera que se lograra a partir de él no sería mejor que la ya obtenida, por lo que no es necesario seguir ramificando en ese nodo.

ii. *Branch and Cut*

Es una variante de Branch and Bound donde el proceso de ramificación se realiza a través de planos de corte. Estos planos de corte se construyen a partir de elegir una variable para ramificar e imponer condiciones a los valores que puede tomar esta variable. Cada una de estas condiciones determina un plano de corte que se aplicará al subproblema del nodo generado a partir del mismo. Se denomina plano de corte ya que se trata de un plano que corta la región factible asociada al subproblema que se está resolviendo en el nodo.

Ejemplo para el caso de problemas binarios:
Dado un nodo n_1 se elige para ramificar alguna de las variables que tiene valor fraccionario en el problema relajado asociado a n_1 (sea esta variable v_1). A partir

de n_1 se ramifica en dos nodos más, a uno se le agrega el plano de corte v_1=0 (n_{11}) y al otro v_1=1 (n_{12}).
Se repite el procedimiento en n_{11} y n_{12} y así sucesivamente en el resto de los nodos.

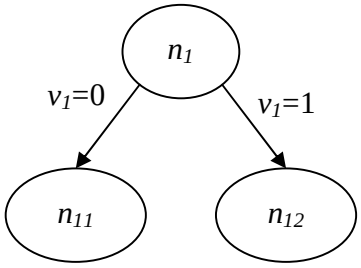

Figura 2.2.3: Ejemplo explicativo de cómo funciona el algoritmo Branch and Cut. n_1, n_{11} y n_{12} son nodos y v_1 es la variable utilizada para ramificar, generando dos cortes v_1=0 y v_1=1.

iii. Algoritmo Aditivo de Balas [6]

A continuación se da una introducción a al algoritmo Aditivo de Balas.

Es una variante de Branch and Bound optimizada para problemas binarios. En los problemas binarios, cada variable puede tomar los valores uno o cero, esto puede representar elegir o descartar una opción, prender o apagar una llave, una respuesta si o no o muchas otras situaciones. El Algoritmo Aditivo de Balas es un algoritmo Branch and Bound especializado en resolver problemas binarios.

Como primer paso se requiere que el problema sea llevado a la siguiente forma estándar:

1. La función objetivo tiene la forma minimizar $z = \sum_{j=1}^{n} c_j x_j$
2. Las m restricciones deben ser todas inecuaciones de la forma $\sum a_{ij} x_j \geq b_i \forall i = 1,2,...,m$
3. Todas las $x_j \forall j = 1..n$ son variables binarias
4. Todos los coeficientes de la función objetivo son positivos
5. Las variables se ordenan de acuerdo a su coeficiente en la función objetivo $0 \leq c_1 \leq c_2 \leq ... \leq c_n$

Si el problema no se encuentra en esta forma se puede convertir fácilmente. Por ejemplo las variables que tienen coeficiente negativo en la función objetivo se manejan con un cambio de variable en el cual x_j se reemplaza por $(1 - x_j)$ (punto 4). También es fácil reordenar las variables (punto 5). El lado derecho de las restricciones puede ser negativo, por lo que las restricciones de <= se

convierten fácilmente en >= multiplicando por (-1). La restricción más grande es que el algoritmo de Balas no maneja restricciones de igualdad.

La clave de la forma en como trabaja el Algoritmo de Balas radica en su estructura especial:

- El sentido de la función objetivo es minimización y todos los coeficientes son positivos, por lo que se comienza setenado todas las variables en cero para obtener el valor más pequeño de *Z*.
- Si no se pueden setear todas las variables en cero sin violar alguna restricción, luego se setea en uno la variable que tiene el índice mas pequeño. Esto es porque las variables están ordenadas crecientemente, por lo que la de menor índice tiene el coeficiente más pequeño por lo que *Z* se incrementará lo menos posible.

Estas características también afectan el resto del procedimiento Branch and Bound. El Branching es simple, cada variable puede tomar solo dos valores cero o uno. El Bounding es la parte más compleja, la función de bounding mira el costo de la próxima solución factible posible. Hay dos casos que se describen a continuación:

Si la variable actual x_N tiene valor uno, el algoritmo asume que a partir de allí se puede lograr una solución factible, la de menor costo sería la que asigna valor cero a todas las variables siguientes a x_N en la función objetivo. Por lo que el valor de la cota es $\sum_{j=1}^{N} c_j x_j$, las variables posteriores a x_N se consideran que valen cero.

Si la variable actual x_N tiene valor cero, las cosas son un poco diferentes. Necesitamos calcular una cota para los nodos que son actualmente infactibles. En este caso una de las restricciones >= no se satisface. Si seteamos la variable actual en cero, el lado izquierdo de la restricción que no se satisface no cambiará. Por lo que se deberá setear al menos una variable en uno y la que menos costo aporta es x_{N+1}, por lo que la función de bounding será $\sum_{j=1}^{N} c_j x_j + c_{N+1}$. Al igual que el caso anterior el algoritmo asume que este seteo de menor costo seguramente proveerá una solución factible, por lo que procede.

b. Programación dinámica [2]

Métodos de programación dinámica: para procesos de decisión secuenciales, fundamentada en una resolución recursiva del problema

- Es una técnica para descomponer el problema original en una serie de problemas embebidos, a partir de una aproximación recursiva para resolver el problema original.
- Fue desarrollada originalmente para resolver problemas de decisión secuenciales.
- Es un proceso de decisión secuencial en tiempo discreto: $t = 1,...T$ Al comienzo del período t el proceso está en estado s_{t-1} que depende del estado inicial s_0 y de las variables de decisión $x_1,\ldots,x_{t-1}$ para $t = 1,\ldots,T$.
- La contribución a la función objetivo en el período t depende solamente de s_{t-1} (estado anterior) y de x_t (variable de decisión en tiempo actual) y el estado en t solamente de s_{t-1} y de x_t.

Formulación de un problema por programación dinámica:

$z = \max \Sigma g_t(s_{t-1},x_t)$ /

$s_t = \Phi_t(s_{t-1},x_t)$ $t=1...T\text{-}1$ (s_t son las variables de estado y x_t las de decisión)

s_0 dato

El dominio de las variables de estado y de decisión depende de la aplicación.

Llevando esta formulación a una expresión recursiva resulta:

$z_k(s_{k-1})=\max\ g_k(s_{k-1},x_k) + z_{k+1}(s_k)$

/ $s_k=\Phi_t(s_{k-1},x_k)$

- La fórmula recursiva anterior permite transformar el problema original con T variables de decisión, T-1 variables de estado y T-1 restricciones de estado, en una secuencia de T subproblemas, dado por la fórmula anterior.
- Inicialmente $z_{T+1}(s_T) = 0$
- La fórmula recursiva expresa un principio intuitivo de optimalidad para un proceso de decisión secuencial: una vez alcanzado un estado particular, una condición necesaria de optimalidad es que las decisiones remanentes deben ser elegidas óptimamente con respecto al estado (principio de optimalidad de Bellman).

3. Parte central del trabajo

En este capítulo se describen las soluciones desarrolladas para resolver el problema. Por tratarse de un problema de optimización lineal entero, es decir un problema NP-hard (ver Anexo II), se enfocó su resolución con heurísticas que calculen una solución aproximada a la óptima. Las mismas se describen como *"Primera solución aproximada"* (en la sección 3.1) y *"Segunda solución aproximada"* (en la sección 3.2).
Por otra parte también se investigó la forma de encontrar una solución exacta al problema. En este contexto se diseño un algoritmo Branch and Bound que se describe en la sección 3.3. Se investigó el software Cplex encontrándose que tenía operaciones que implementan algoritmos Branch and Cut, por lo que estas operaciones se utilizaron para desarrollar un algoritmo que permitiera calcular una solución exacta al problema. El mismo se describen como *"Solución utilizando Cplex"* (en la sección 3.4). Se utilizó el costo de la solución aproximada encontrada por la primera heurística como cota superior para el Branch and Cut implementado con Cplex.

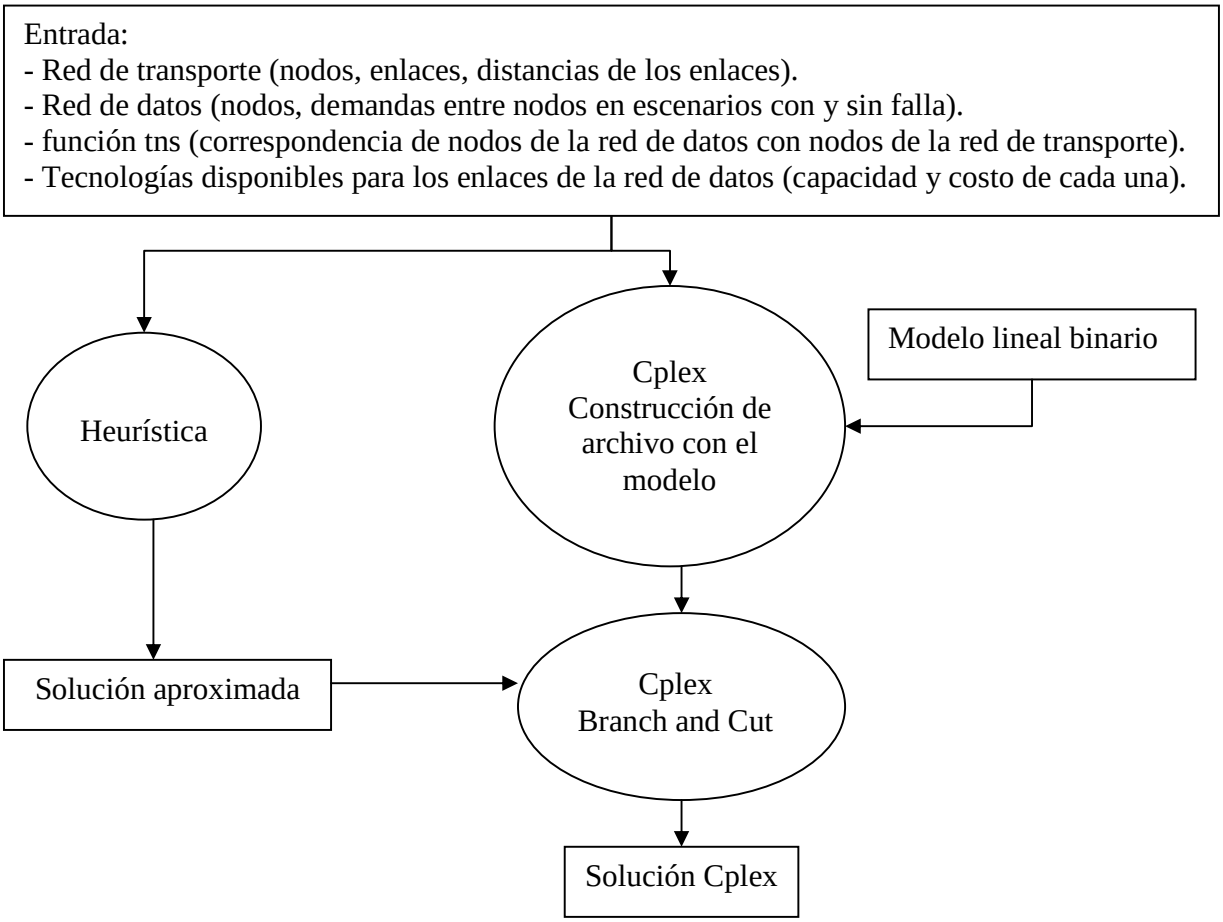

Figura 3.1: Diagrama ilustrando las distintas soluciones implementadas (con sus entradas y salidas) y como se relacionan entre si.

En la figura 3.1 se ilustra el trabajo realizado, donde cada componente simboliza lo siguiente:

Entrada:	En la figura se presentan los datos de entrada al problema que ya fueron especificados en la sección 1.2 (introducción al problema). Para estos datos se definió un formato que fue utilizado en todas las soluciones implementadas. Este formato se especifica en la sección 3.1.3 (organización en módulos de la solución aproximada uno) y se puede encontrar un ejemplo completo en el Anexo III.
Heurística:	Corresponde a las heurísticas implantadas “Primera solución aproximada” ó “Segunda solución aproximada”.
Solución Aproximada:	Es una solución a nuestro problema, hallada con los métodos heurísticos. Los datos presentes en la solución ya fueron especificados en la sección 1.2 (introducción al problema), se puede resumir como los enlaces de la red datos con la correspondiente tecnología asignada, los ruteos en la red de transporte para cada enlace de la red de datos y los reruteos de información para cada escenario de falla simple. Un ejemplo de archivo de salida de la primera heurística implementada se puede ver en el Anexo III.
Modelo lineal binario:	Es la realidad a resolver modelada como un problema de optimización matemática lineal binario.
Cplex, Construcción de Archivo con el modelo:	Programa que toma la entrada y el modelo de programación matemática lineal, y genera un archivo en formato *lp* con la especificación del modelo.
Cplex Branch and Cut:	Es un programa Cplex que toma el modelo *lp* y una solución aproximada como cota superior, y encuentra una solución óptima.
Solución Cplex:	Es una solución a nuestro problema, hallada con un algoritmo Branch and Bound implementado con el software Cplex. Los datos presentes en la solución ya fueron especificados en la sección 1.2 (introducción al problema), se puede resumir como los enlaces de la red datos con la correspondiente tecnología asignada, los ruteos en la red de transporte para cada enlace de la red de datos y los reruteos de información para cada escenario de falla simple. Un ejemplo de archivo de salida se puede ver en el Anexo III.

3.1. Primera solución aproximada

Fue la primera solución que diseñamos tomando en cuenta la necesidad de tener rápidamente una solución factible y sobre todo una cota superior para el costo de la solución.
La idea era obtener la cota para inicializar la solución Cplex que se encargaría de hallar la solución óptima.
En tal sentido se realizó una heurística que logra una solución factible rápidamente.

Como virtud de esta heurística podemos mencionar que es sencilla a la hora de entender la construcción de la solución, lo que permite ver fácilmente que la solución encontrada es factible.
Otro punto que también consideramos favorable, es el tamaño del código (mil líneas de código aproximadamente) que consideramos manejable.
Otra ventaja es el hecho de que minimiza la distancia de los caminos elegidos en la red de transporte para el tráfico de datos, es decir que uno de los aspectos importantes a minimizar es tenido en cuenta.
Por último es importante resaltar que es general, es decir, que funciona para distintas configuraciones en los datos de entrada. Se puede variar la red de transporte, las demandas requeridas o las tecnologías disponibles y la heurística sigue encontrando una solución factible (si es que existe).

Como desventaja podemos mencionar que no encuentra una buena solución, es decir que a pesar de encontrar una solución factible, la misma no es cercana a la óptima.
Por otro lado al no contar con los datos reales del problema a la hora de ser implementada, la heurística, es muy general y pierde precisión para nuestro problema.
Por último es importante mencionar que por la poca familiarización que teníamos con el problema a la hora de diseñarla, la heurística no toma en cuenta todos los aspectos a minimizar para obtener una buena solución. Por ejemplo no aprovecha temas de holguras y reutilización de canales.

3.1.1. Aclaraciones, conceptos y definiciones para entender la solución

- Grafo de Datos: La red de datos se modela con un grafo, es más, para cada escenario sin fallas y con fallas se manejará un grafo diferente y luego se construirá el final. En esta solución se le llamara “clientes” a los vértices del grafo de datos y “aristas” a las aristas.

- Grafo de Transporte: La red de transporte se modela con un grafo. En esta solución se le llamara “nodos” a los vértices del grafo de transporte y “link” a las aristas.

3.1.2. Descripción de los pasos del algoritmo

A continuación se incluye una descripción de los pasos del algoritmo:

1) Sea V_D el conjunto de nodos de la red de datos. Para cada par de clientes $(i, j) \in V_D$, hallar el camino más corto en distancia entre *tns*(*i*) y *tns*(*j*), utilizando el algoritmo de Dijkstra en el grafo de transporte, aplicado a *tns*(*i*). Luego de esto queda fijado el ruteo *Ruteo*(*i, j*) en transporte para cada par de clientes $(i, j) \in V_D$.

2) Construir grafo de datos G_i para el escenario sin fallas:

   - Tomar M^0, matriz de demandas en el escenario sin falla.
   - Para cada demanda *d* entre dos clientes *i, j*, agregar a G_i la arista (*i, j*) con valor *d*. Para los clientes que no tienen demandas se tomará que *d* = 0.

   De esta forma queda construido el grafo completo G_i con tantos vértices como clientes haya en el problema, con valor de las aristas (*i, j*) = $M^0[i, j]$ y con distancia de aristas igual a la del ruteo *Ruteo*(*i, j*) hallado en el paso uno.

3) Para cada link *e* del grafo de transporte construir el grafo de datos G_e correspondiente:

   - $G_e = G_i$
   - Sacar de G_e, todas las aristas afectadas por la caída del link *e*, es decir, las que utilizan en su ruteo de transporte, al link *e*.

   - Para cada arista que no fue afectada por la caída de *e*, ver la matriz de demanda M^e, matriz de demandas en el escenario de falla del link *e*. Cambiar en G_e las capacidades para soportar la demanda M^e.

   - Para cada arista afectada (*i, j*) por la caída de *e*:

     - Encontrar en G_e el ruteo más corto de *i* hasta *j*.
     - Ver en M^e la demanda *d* entre (*i, j*).
     - Sumar el valor *d* al valor de todas las aristas que integran el ruteo hallado.

   - Guardar en grafo G_e.

4) Construir un grafo G_u, unión de los resultados obtenidos en los pasos dos y tres. Para esto se asignará a cada arista (*i, j*), el valor máximo *d* obtenido de entre G_i y todos los G_e, para dicha arista.

5) Construir grafo resultado G_r asignando a cada arista (i, j) de G_u, la mínima tecnología que satisfaga la capacidad asignada en el paso cuatro.

En este punto logramos obtener una solución factible inicial.

3.1.3. Organización en módulos

La solución se encuentra organizada en tres módulos: ***Grafo***, ***LeerArchivo*** y ***Main***.

El módulo ***Grafo*** contiene estructuras para manipular los grafos de datos y transporte y las demandas entre los nodos del grafo de datos. También contiene operaciones para trabajar con estas estructuras. No tiene dependencias con los otros módulos.

El módulo ***LeerArchivo*** contiene operaciones para leer un archivo con los datos de entrada al problema y utilizarlos para cargar las estructuras. Se relaciona con el módulo Grafo para el acceso a estas estructuras.
Recordar que los datos de entrada son: red de transporte (nodos, enlaces, distancias de los enlaces), red de datos (nodos, demandas entre nodos en escenarios con y sin falla), función tns (correspondencia de nodos de la red de datos con nodos de la red de transporte) y tecnologías disponibles para los enlaces de la red de datos (capacidad y costo de cada una).
A continuación se describe el formato del archivo de entrada, utilizando un archivo de entrada simplificado, es decir que se le borraron algunas líneas (se indica con una línea de puntos suspensivos). Se puede encontrar un ejemplo completo de archivo de entrada en el Anexo III.

```
CANT_NODOS_TRANSPORTE: 30  //Cantidad de nodos del grafo de transporte.
CANT_LINKS_TRANSPORTE: 35  //Cantidad de links del grafo de transporte.

-------------------------------------------
- LINKS_CON_DISTANCIAS_ASOCIADAS           -
//Se definen los links del grafo de transporte y sus distancias.
-------------------------------------------

LINK: 0 1 255.9
LINK: 0 5 139.01
LINK: 1 3 174.52
LINK: 2 3 119.35
LINK: 3 6 125.82
LINK: 4 7 148.24
……..

-------------------------------------------
- TECNOLOGÍAS                             -
//Se definen las tecnologías, para cada una se incluye primero la capacidad y
//luego el costo.
-------------------------------------------

CANT_TECNOLOGIAS: 3
```

```
TECNOLOGIA: 0 0
TECNOLOGIA: 1000 50
TECNOLOGIA: 9000 108

-----------------------------------------
- CLIENTES
//Se definen los clientes del grafo de datos y para cada uno se indica su
//correspondiente nodo en el grafo de transporte.
-----------------------------------------

CANT_CLIENTES: 45
TNS_0: 4
TNS_1: 1
TNS_2: 25

……..……

-----------------------------------------
- Matrices
//Para cada link del grafo de transporte se incluye la matriz de demandas en
//el escenario de falla de ese link.
//La última matriz es la correspondiente al escenario sin fallas.
//Para cada matriz se guarda solo la triangular superior ya que las matrices
//de demanda son simétricas.
//Para simplificar la lectura de la matriz se omitieron los espacios al
//comienzo de cada fila (correspondientes a las entradas que no se incluyen
//por tomarse en cuenta solo la matriz triangular superior).
-
-----------------------------------------

LINK1: 0 1

5 0 0 3 0 1 0
0 4 0 0 9 0
0 0 10 0 0
0 80 0 0
0 0 0
23 0
0

LINK2: 0 5

14 0 0 3 0 1 0
0 4 0 0 6 0
0 0 10 0 0
0 90 0 0
0 0 0
13 0
0
……..……
```

El módulo ***Main*** tiene como cometido calcular la solución aproximada ejecutando los pasos uno a cinco descritos anteriormente. Para esto necesita acceder a las estructuras de datos y cargarlas

a partir del archivo de entrada, por lo que se relaciona tanto con el módulo Grafo como con el módulo LeerArchivo. Los resultados obtenidos se guardan en un archivo de salida con el siguiente formato, se puede encontrar un ejemplo completo de archivo de salida en el Anexo III.

```
//Para cada arista del grafo de datos se incluyen los nodos del
//grafo de transporte que componen el camino por el que se rutea
//el tráfico correspondiente a la arista en el escenario sin falla.
****** Ruteos en transporte en el escenario sin fallas ******
Arista (0,5):     5     6     3     1
Arista (1,2):     2     5     3
Arista (4,7):     5     4     10    14    15

.........

//Para cada link del grafo de transporte, se incluyen las aristas
//de datos que se ven afectadas ante su caída y los clientes de
//datos por los que se rerutea el trafico de la arista afectada.
****** Ruteos en datos en escenarios de falla ******
Falla link (0,1):
      arista (15,24) es:      15    19    24
      arista (2,35) es: 2     22    35
      arista (7,27) es: 7     18    0     27
      .........
Falla link (0,8):
      arista (22,40) es:      22    23    35    40
      arista (29,39) es:      29    31    33    7     2     39
      .........

.........

//Para cada arista de datos se incluye, la tecnología que se le
//asignó, el costo de la misma y la distancia del camino en
//transporte que la rutea.
****** Tecnologías asignadas a las aristas y distancias en transporte ******
Arista (0,1):
      tecnologia 1000.000000
      Costo Tec. 50.000000
      distancia 535.785521
Arista (0,2):
      tecnologia 0.000000
      Costo Tec. 0.000000
      distancia 327.810706
Arista (2,3):
      tecnologia 9000.000000
      Costo Tec. 108.000000
      distancia 774.539399

.........

//Se calcula el costo de la solución encontrada sumando para cada arista de
//datos, el resultado de multiplicar el costo de la tecnología asignada a la
//arista, por la distancia del camino que la rutea en el grafo de transporte.

RESULTADO DE LA FUNCION OBJETIVO = 4705332.500000
```

```
//La holgura de la red se calcula a partir de sumar para cada arista de
//datos, las diferencia entre la capacidad requerida para la
//arista y la capacidad de la tecnología asignada. Este valor se
//tiene en el paso cinco del algoritmo antes de asignar la mínima
//tecnología que satisfaga la capacidad requerida.

Holgura total de la capacidad de la red = 578405.000000
```

En la figura 3.1.3.1 se encuentra un diagrama que muestra las dependencias entre los módulos descritos anteriormente.

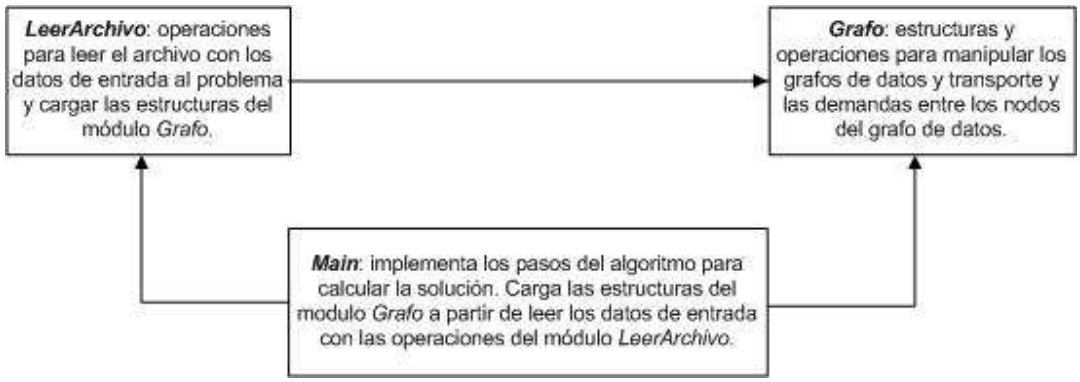

Figura 3.1.3.1: Diagrama ilustrando los módulos de la solución aproximada 1 y como se relacionan entre si.

En el Anexo VII se describen las operaciones incluidas en cada módulo.

3.1.4. Estructuras de datos más importantes

Las siguientes son las estructuras de datos más importantes que se utilizaron:

- Estructura ***Grafo***.
- Estructura ***Grafos***.
- Estructura ***Matriz***.
- Estructura ***Matrices***.
- Estructura ***Tabla_link_matrices***
- Estructura ***AristasAfectadas***.
- Estructura ***Caminos***.
- Estructuras ***Tecnologías***.
- Estructura ***TablaTNS***.

A continuación se describe cada una de ellas:

- Estructura ***Grafo***: es un arreglo de listas que se utiliza para almacenar un grafo mediante listas de adyacencia. En esta implementación se utiliza tanto para el grafo de datos como para el de transporte. El índice de cada posición del arreglo se utiliza para identificar al cliente/nodo origen. El cliente/nodo destino es el indicado en el campo *id nodo ady* que se encuentra en la lista de adyacentes del cliente/nodo origen. Para el caso del grafo de datos, *Distancia* es la suma de las distancias de los links que componen el camino en transporte desde el *tns* del cliente origen al *tns* del cliente destino. Para el caso del grafo de transporte, *Distancia* es el largo del link desde el nodo origen al nodo destino. Como ambos grafos son no dirigidos, la estructura se carga de forma simétrica. Es decir que por ejemplo para representar la arista/link entre los clientes/nodos 0 y 2, se agrega en la lista de la posición cero el adyacente 2 y en la lista de la posición dos el adyacente 0, ambos con la misma distancia.

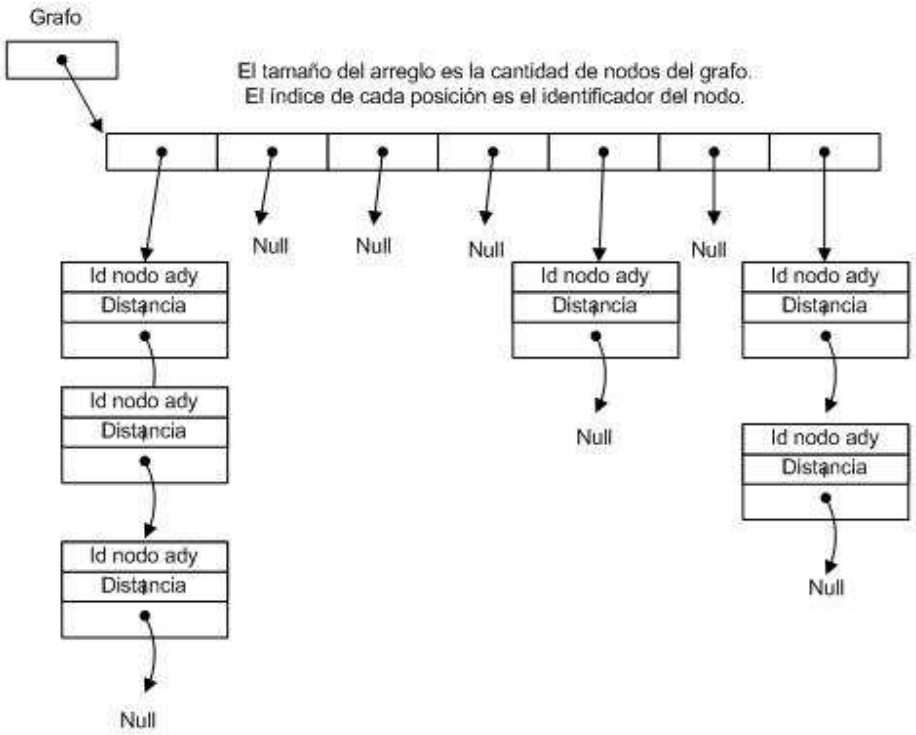

Figura 3.1.4.1: Descripción de la estructura *Grafo* de la solución aproximada 1.

- Estructura ***Grafos***: es un arreglo de estructuras Grafo. Se utiliza para almacenar los grafos de datos que se generan en el paso tres del algoritmo. Es decir que en cada posición *i* se guarda el grafo de datos resultado del reruteo debido a la falla del link *i*. En la última posición se guarda el grafo de datos en el escenario sin falla (calculado en el paso dos).

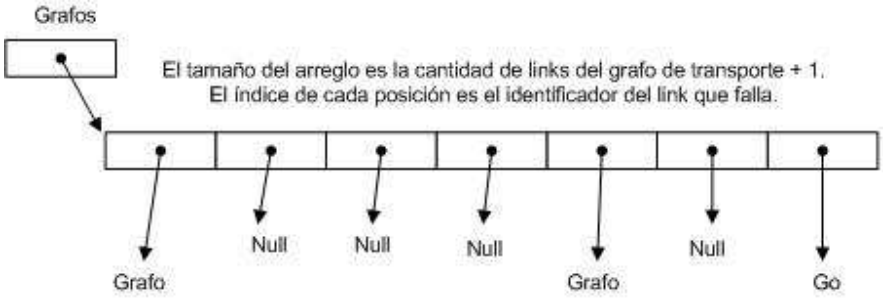

Figura 3.1.4.2: Descripción de la estructura *Grafos* de la solución aproximada 1.

- Estructura ***Matriz***: es un arreglo de arreglos de números flotantes. Se utiliza para guardar las demandas entre clientes. El índice del arreglo horizontal representan el cliente origen y el del arreglo vertical representa el cliente destino. Por tanto el numero en la posición *Matriz*[*i*][*j*] representa la demanda entre el cliente *i* el cliente *j*. Como el grafo de datos es no dirigido se guarda el mismo valor en *Matriz*[*i*][*j*] y en *Matriz*[*j*][*i*]. La demanda que se

almacena corresponde a la suma del tráfico comprometido más el tráfico eventual entre los clientes.

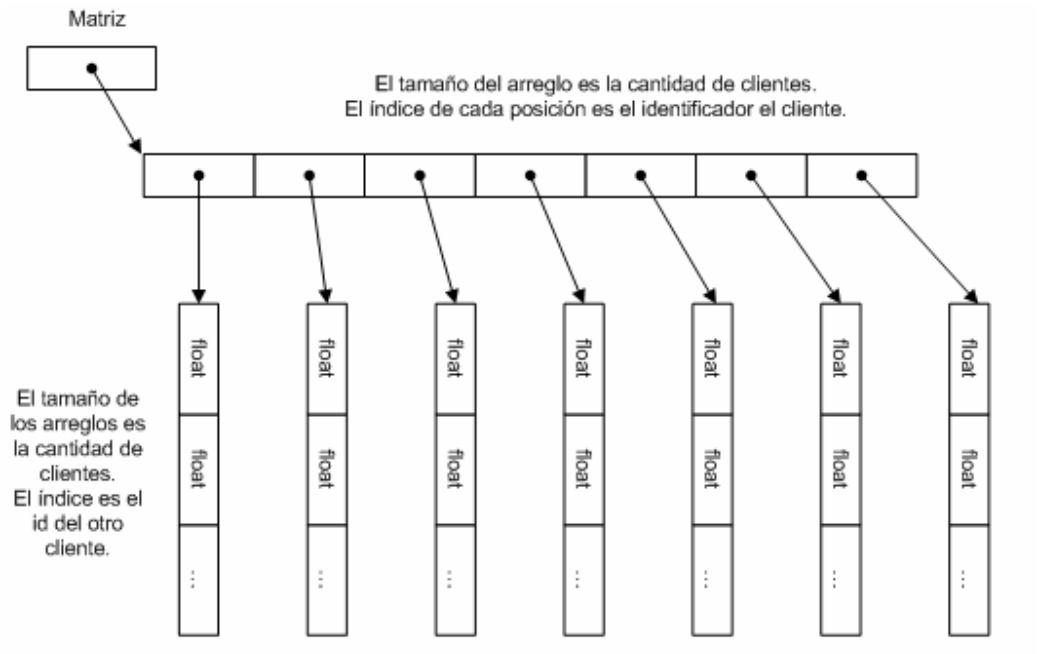

Figura 3.1.4.3: Descripción de la estructura *Matriz* de la solución aproximada 1.

- Estructura ***Matrices***: es un arreglo de estructuras *Matriz*. Se utiliza para guardar las matrices de demandas entre clientes para cada escenario de falla. En la última posición se guarda la matriz de demandas en el escenario sin falla M_0.

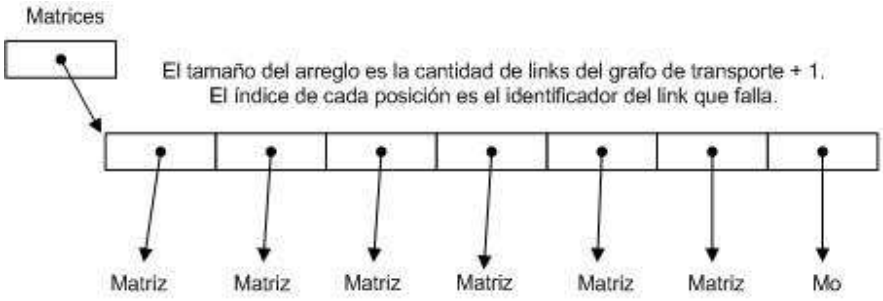

Figura 3.1.4.4: Descripción de la estructura *Matrices* de la solución aproximada 1.

- Estructura ***Tabla_link_matrices***: es un arreglo de links, donde para cada link se guarda el origen y el destino. La posición del arreglo es el identificador del link. La estructura se utiliza para obtener el *id* de un link dado el origen y el destino del mismo. Este *id* se usa como índice en el arreglo Matrices, para obtener la matriz de demandas en el escenario de falla de ese link.

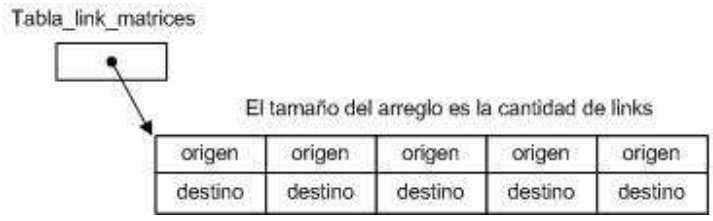

Figura 3.1.4.5: Descripción de la estructura *Tabla_link_matrices* de la solución aproximada 1.

- Estructura ***AristasAfectadas***: es un arreglo donde cada posición representa un link de transporte. Para cada link de transporte se tiene una lista de las aristas de datos que se ven afectadas ante su caída. Las aristas de datos se identifican a través del identificador del origen y del destino.

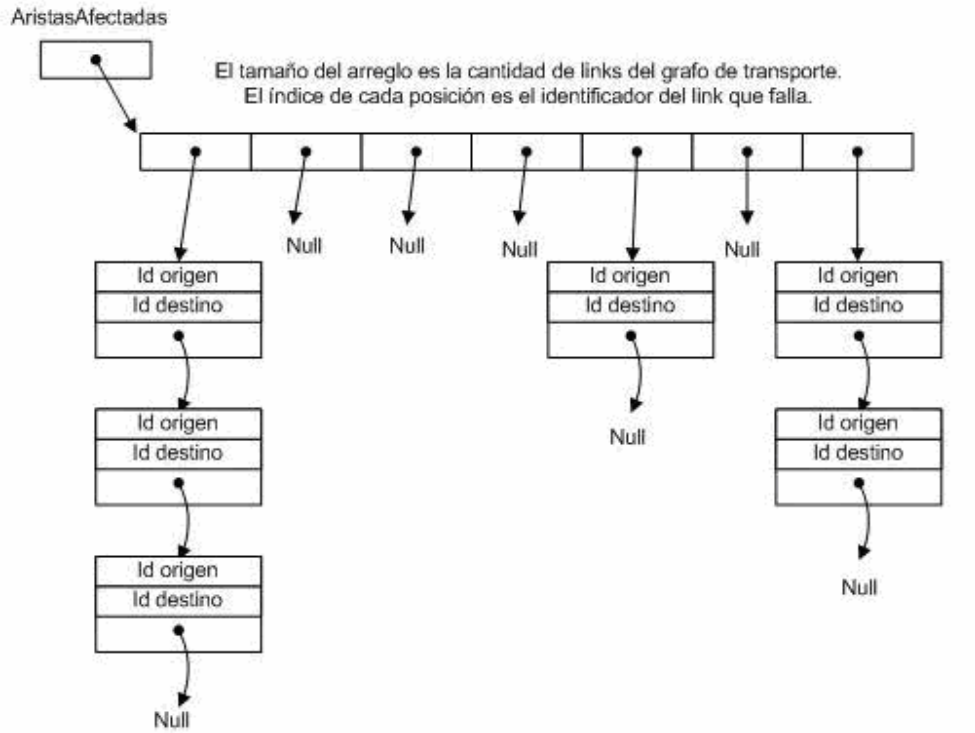

Figura 3.1.4.6: Descripción de la estructura *AristasAfectadas* de la solución aproximada 1.

- Estructura ***Caminos**:* es un arreglo que tiene para cada par de clientes *i*, *j* del grafo de datos, el camino más corto en el grafo de transporte entre *tns*(*i*) y *tns*(*j*). También se guarda la distancia entre cada para de nodos del camino.
  Se carga con los resultados de la ejecución del algoritmo de Dijkstra en el grafo de transporte.

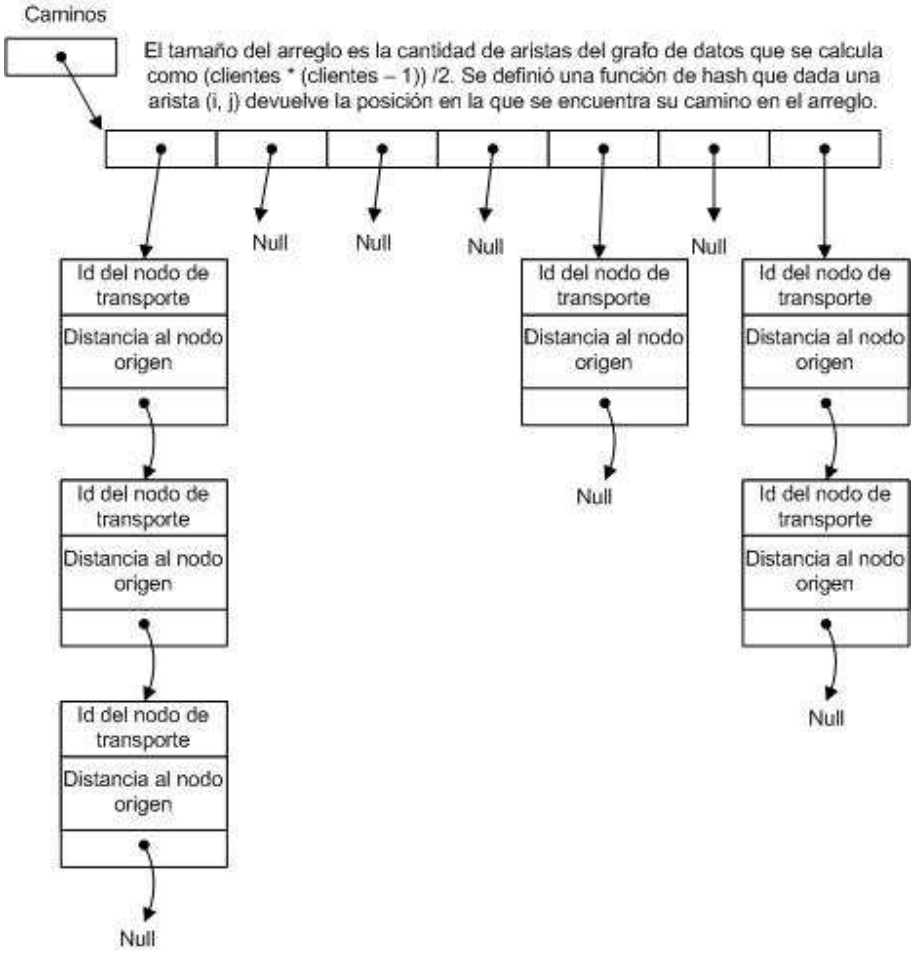

Figura 3.1.4.7: Descripción de la estructura *Caminos* de la solución aproximada 1.

- Estructuras ***Tecnologías***: es un arreglo donde se guardan los datos de las tecnologías. Para cada tecnología se guarda su capacidad y su costo.

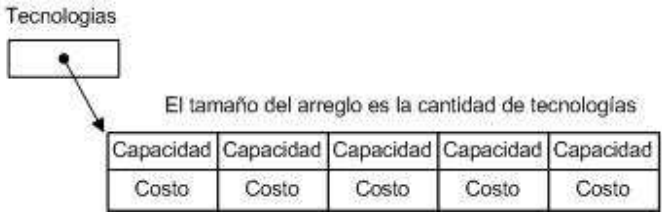

Figura 3.1.4.8: Descripción de la estructura *Tecnologías* de la solución aproximada 1.

- Estructura ***TablaTNS***: es un arreglo donde cada posición representa un cliente del grafo de datos. Para cada cliente se guarda el correspondiente nodo en transporte.

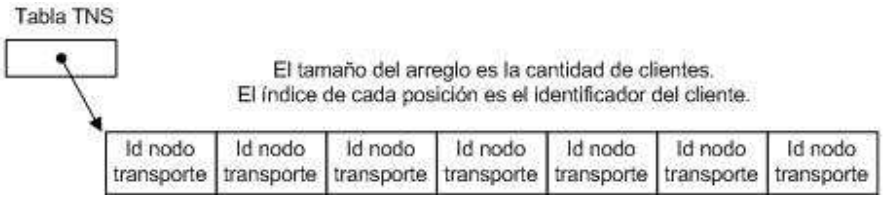

Figura 3.1.4.9: Descripción de la estructura *TablaTNS* de la solución aproximada 1.

3.2. Segunda solución aproximada

En esta heurística se buscó obtener una solución más cercana al óptimo que la lograda con la primera solución aproximada. Básicamente consiste en dividir los nodos del grafo de datos en grupos, de forma tal que los nodos contenidos en un grupo no tengan demanda entre si. Posteriormente se calcula la demanda total entre los grupos y se crean aristas que conecten los grupos (denominadas aristas caño) que soporten estas demandas y los posibles escenarios de falla. Finalmente se crean ciclos dentro de cada grupo, de forma tal que todos los nodos que requieran comunicarse con nodos de otros grupos puedan hacerlo, utilizando las aristas del ciclo correspondiente, y luego a través de las aristas caño para salir del grupo.

Lo que se pretende con la separación de los grupos es intentar reducir el desperdicio de conexión entre los nodos que se conectan entre si por las demandas, o sea la separación entre grupos es para generar aristas caños que soporten las restricciones estructurales de soportar la caída de un link en transporte y a la vez intentar concentrar todo el flujo de datos en dichas aristas con lo que se reduce el desperdicio potencial entre nodos. Con esta técnica logramos abstraernos del problema de conexión entre nodos y lo abstraemos a la conexión entre grupos entonces un nodo interno a un grupo ya no tiene que preocuparse por conectarse a otro nodo sino a otro grupo, con lo que lo importante es estar conectado a la salida del grupo actual del nodo, cumpliendo las restricciones de demanda y estructurales (soportar caída de un link) de forma menos compleja. La idea es intentar buscar conexiones con distancias mínimas locales y con demandas saturadas en cada ciclo interno del grupo. Con esto minimizamos las conexiones entre grupos, entre elementos del mismo grupo. Además buscamos minimizar las distancias en forma local y en forma global entre grupos. En la heurística se tomo en cuenta la idea de minimizar primero los ciclos, por lo que se intenta que cada ciclo dentro de cada grupo cumpla las condiciones estructurales y posea la máxima cantidad de nodos, finalmente se intenta minimizar el perímetro total del ciclo.

Las virtudes de esta heurística son las siguientes:

- Encuentra una solución mejor que la solución aproximada 1, es decir que encuentra una solución factible y más cercana a la óptima (la principal virtud)
- Si bien es menos general que la solución aproximada 1, esta solución también fue implementada para que pueda ejecutarse con diferentes grafos de entrada, respetando el formato de entrada.
- Útil para grafos de entrada grandes
- El tiempo que demora para obtener un resultado es de alrededor de doce segundos para un grafo de la envergadura del grafo dado por ANTEL.

Las desventajas son las siguientes:

- Se necesita una cantidad mínima de nodos de entrada (por lo menos nueve nodos) ya que necesita armar ciclos y aristas que comuniquen los diferentes grupos.
- Es compleja a la hora de entender la construcción de la solución implementada.
- Es complejo analizar la factibilidad de la solución.
- La implementación es mucho mayor (en el orden de las seis mil líneas de código)
- El consumo de memoria RAM es muy elevado (el proceso consume 1,1 GB de RAM en el grafo de ANTEL)
- No muestra cual es el camino alternativo que debe rutear cuando ocurre una falla.

3.2.1. Aclaraciones, conceptos y definiciones para entender la solución

- Grafo de Datos: En esta solución cada grupo va a tener un grafo que lo represente, por lo que el grafo de datos resultante va a ser la unión de dichos grafos junto con las aristas_caño. En esta solución se le llamara “clientes” a los vértices del grafo de datos y “aristas” a las aristas del grafo.
- Grafo de Transporte: La red de transporte se modela con un grafo. En esta solución se le llamara “nodos” a los vértices del grafo de transporte y “link” a las aristas del grafo.
- Aristas_caño: Se le llaman aristas_caño a las aristas que comunican los grupos entre sí.
- Nodos de salida: Son los nodos extremos de las aristas_caño.
- *Grupo*: Es un conjunto de nodos que no tienen demandas entre sí.
- Dos aristas son link disjuntas cuando tienen ruteos en transporte que no comparten links.
- Puntas: Le llamamos Puntas a un conjunto de nodos unidos encadenados entre si mediante aristas. Entre otras cosas posee los extremos izquierdo, derecho, el camino que forman los nodos unidos para llegar de un extremo al otro, y la demanda de toda la cadena hacia otros grupos.

3.2.2. Descripción de los pasos del algoritmo

A continuación se brinda una descripción en alto nivel de los pasos del algoritmo, cada paso tiene un objetivo definido, más adelante en la sección 3.2.4 se detalla mas precisamente como se implementa cada paso.

1. Para cada par de clientes ejecuto el algoritmo de Dijkstra entre los tns de los mismos.
   El objetivo de este paso es simplemente tener previamente calculado el Dijkstra entre todos los nodos del grafo para ser utilizado en los pasos posteriores del algoritmo.

2. Este paso se encarga de separar los nodos en grupos

   Separo las demandas en *grupos*, para cada demanda entre los nodos (*Ci*, *Cj*)
   a. Agrego *Ci* a un *grupo* y *Cj* a otro
      i. Si no hay ningún *grupo* creo los dos iniciales.
      ii. Si no existe un *grupo* en donde agregar *Ci* o *Cj* sin romper la condición de grupo, creo un nuevo grupo para aquel nodo (*Ci* o *Cj*) que no fue posible agregarlo en ninguno de los anteriores.

3. Este paso calcula la demanda total de cada grupo hacia los demás.

   Para cada par de *grupos Gi*, *Gj*: Calculo la demanda total entre ellos, como la suma de las demandas que tienen los nodos de Gi con los de Gj.
   Denominaremos a este valor *Dtotal*(*Gi*, *Gj*).

4. Este paso calcula la cantidad de *aristas_caño* necesarias para comunicar a los grupos entre sí junto con su tecnología correspondiente

   Calculo la cantidad de *aristas_caño* y las tecnologías a asignar a las mismas de la siguiente manera:

   a. Tomo la menor *tecnologíaA* que cubre *Dtotal*(*Gi*, *Gj*) ó la más grande si no hay ninguna que cubra
   b. Cantidad _ caños = 0;
   c. While demanda – *tecnologíaA* > 0
      i. sumo una *arista_caño Caño_i*(*Gi*, *Gj*) y le asigno *tecnologíaA*
      ii. demanda = demanda – *tecnologíaA*
      iii. cantidad _ caños++;
   d. si demanda > 0
      i. Tomo la menor *tecnologíaB* que cubra la demanda_resultado calculada de la siguiente forma:

- Obtengo aquel nodo (nodoi) del grupo Gi que tiene una mayor demanda hacia el grupo Gj
- Obtengo aquel nodo (nodoj) del grupo Gj que tiene una mayor demanda hacia el grupo Gi
- Demanda_resultado se calcula como el Max{nodoi, nodoj, demanda}

ii. sumo una *arista_caño Caño_i*(*Gi, Gj*) y le asigno *tecnologíaB*
iii. cantidad _ caños++;

e. sumo una *arista_caño Caño_i*(*Gi, Gj*) y le asigno la tecnología mayor de todas las anteriores (cubro caída del algún caño)
f. cantidad _ caños++;

5. Este paso asigna cuales nodos de cada grupo son los que van a ser los extremos de cada *arista_caño* junto con sus ruteos en transporte

   Para cada *Caño_i*(*Gi, Gj*) agregado en el paso anterior:

   a. Tomo primero la o una de las *aristas_caño* de mayor tecnología que todavía no tiene un ruteo asignado.
   b. Defino el ruteo en transporte para la arista *Caño_i*(*Gi, Gj*) tomada en esta iteración. El ruteo debe ser link disjuntos con las otras *Caño_j*(*Gi, Gj*) ya que nunca se puede caer mas de una arista caño a la vez, en tal caso no se estaría soportando la demanda de los *grupos*.
      i. Saco del grafo de transporte, los links que ya fueron utilizados por otro *Caño_j*(*Gi, Gj*) para buscar un camino link disjunto.
      ii. Para cada par de nodos (*c1, c2*) donde *c1* pertenece a un *Gi* y *c2* a *Gj*, y no existe aún una arista (*c1, c2*), calculo el camino más corto de uno a otro usando Dijkstra.
      iii. Elijo el par (*c1, c2*) que tenga el camino de distancia menor.
   c. Asigno a *Caño_i*(*Gi, Gj*) el ruteo encontrado en el paso anterior y la defino como la arista (*c1, c2*).

Hasta el momento tenemos los grupos de clientes separados, y unidos por aristas_caños con ruteos links disjuntos (disjuntos entre las aristas _ caño de cada par de grupos, no de todos los grupos).

6. Este paso busca generar los ciclos internos de cada grupo así como la asignación de la tecnología de cada arista que pertenece a cada ciclo, junto con su ruteo correspondiente.

   a. Para cada par de nodos de salida que tienen aristas_caño hacia el mismo grupo, se le asigna una arista con la menor distancia y disjunta con dichas aristas_caño, y la

tecnología que se le asigna a la arista es la mayor de las aristas_caño que llegan a los nodos participantes de la arista que van al mismo grupo.

Nota: Es importante destacar que es posible que el conjunto de aristas_caño que van hacia el mismo grupo no sea único, por lo que puede existir otro conjunto de aristas_caño que van hacia otro grupo diferente cuyos extremos son los nodos seleccionados. En dicho caso, la arista asignada entre los nodos seleccionados también deberá ser link disjunta con las otras aristas_caño de los demás conjuntos y la tecnología asignada se calcula como el máximo entre las tecnologías mayores de cada conjunto.

En el caso de que la arista que une un nodo con su respaldo sea única y que además, al caerse dicha arista deje a algún nodo sin conexión (aislado) es necesario utilizar algún nodo que no tenga demanda (y que por lo tanto fue descartado desde el paso 2) como nodo pivot para formar un ciclo con 3 nodos y así, por la propiedad que tienen los ciclos que construimos de ser arista disjunto, evitar que quede el nodo aislado.

Para cada grupo *Gi*:

b. Separo los nodos del grupo en
   i. Nodos de salida.
   ii. Nodos sin procesar: Son los nodos que aún no forman parte de ningún ciclo.

c. Mientras (exista un nodo que no pueda comunicarse con los demás grupos)

d. Mientras (haya nodos sin procesar):

   i. Unir nodos de salida elegidos como representantes hacia cada grupo de la siguiente forma:

   i. Separo los nodos de salida en subgrupos GrupoSalida(Gi, Gj), de acuerdo al grupo Gj, destino de los mismos
   ii. Ordenamos los nodos de salida de cada GupoSalida por su tecnología de mayor a menor.
   iii. Generamos una punta por cada nodo de salida de mayor tecnología de cada GrupoSalida(Gi, Gj).
   iv. Se genera una punta con todos los nodos conectados con minima distancia, links disjuntos entre sí, link disjunto con respecto a las aristas caño elegidas para cada nodo de la punta.
   v. Se asigna provisoriamente la menor tecnología de las aristas_caño que llegan a cada arista que forma parte de la punta creada en la parte anterior. En el caso de que le lleguen varias aristas_caño a un mismo nodo de salida, se elige la de menor tecnología.

vi. Verificamos si todos los nodos no procesados del grupo se pueden unir individualmente generando ciclo con los nodos de la punta creada. En caso de que no sea posible unir algún nodo a la punta generada en el paso i 3), se vuelve a recalcular un ruteo diferente para la punta, esta vez tomando como base el camino calculado en la primera instancia y tirando uno a uno los links, calculando para cada link tirado un nuevo camino de la punta y volviendo al paso ii. En el caso de que se hayan tirado todos los links y que no se haya encontrado un ruteo para la punta que pueda satisfacer la unión de los nodos no procesados individualmente se retorna como resultado que el problema no es factible.
vii. Le asigno la tecnología a la punta de forma definitiva.
viii. Obtenemos los nodos internos que no han sido marcados aún. (no procesados)
ix. Creo una lista de puntas con dichos nodos.
x. Trato de armar un ciclo con la punta generada con los nodos de salida y la lista de puntas generada con los nodos no procesados con las siguientes reglas:
    1) Tratar de colocar la mayor cantidad de nodos en cada ciclo.
    2) Tratar de que cada ciclo tenga el menor perímetro posible
    3) Cada ciclo debe ser arista disjunto.
    4) La demanda total de los nodos del ciclo no puede superar la holgura disponible hacia cada grupo de cada arista caño seleccionada.
    5) La demanda total de los nodos del ciclo no puede superar la tecnología máxima disponible.
    6) Las aristas entre nodos de salida pueden ser compartidas entre varios ciclos, en dicho caso, se debe considerar la holgura de las aristas disponible para considerarlas en un nuevo ciclo.
xi. Le asignamos la tecnología a las aristas del ciclo que no unen 2 nodos de salida entre sí, que se calcula como la mínima tecnología que soporte las demandas que tienen sólo los nodos que no son de salida hacia los demás grupos, y en caso de que los nodos de salida seleccionados no pertenecieran a un ciclo anteriormente, se le debe sumar la demanda de dichos nodos hacia los demás grupos.
xii. Se marcan los nodos utilizados en el ciclo construido
xiii. Se agrega la demanda que genera el ciclo construido en las aristas_caño que pertenecen a los nodos de salidas utilizados en el ciclo.
xiv. Se arma una nueva lista de nodos de salida candidatos con los nodos de salida cuya arista_caño tiene holgura suficiente para contener al nodo no procesado con mayor demanda hacia otro grupo (con esto nos aseguramos el poder armar un ciclo con al menos un nodo no procesado).

Fin Mientras (haya nodos sin procesar)

Si la cantidad de grupos es mayor a 2, se debe verificar que todos los nodos del sistema pertenezcan a un ciclo. En el caso de que existan nodos de salida que no

pertenezcan a ningún ciclo se genera una nueva lista de nodos de salida con la misma lógica del paso xi pero esta vez tomando como nodos internos a los nodos de salida que no pertenecen a ningún ciclo.

Fin Mientras (exista un nodo que no pueda comunicarse con los demás grupos)

Inicializo la demanda cargada en cada arista caño para que no se considere las demandas ocupada en los demás grupos que las utilicen.

Fin Para cada grupo *Gi*

7. Este paso tiene como objetivo calcular el resultado final

    a. Resultado = 0
    b. Para cada arista ***e*** del grafo solución:
        i. *Resultado* = *Resultado* + (*costoTecnología de* ***e*** * *Distancia del ruteo de* ***e***)

3.2.3. Organización en módulos

La solución se encuentra organizada en los siguientes tres módulos:

- Módulo ***Grafo***.
- Módulo ***LeerArchivo***.
- Módulo ***Main***.

A continuación se describe cada uno de ellos y se presenta un diagrama ilustrando como se relacionan.

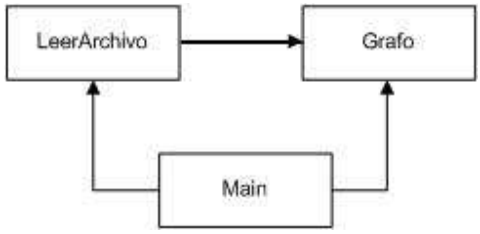

Figura 3.2.3.1: Diagrama ilustrando los módulos de la solución aproximada 2 y como se relacionan entre si.

- Módulo ***Grafo***: contiene estructuras para manipular el grafo de transporte y operaciones para trabajar con éstas.

- Módulo ***LeerArchivo***: operaciones para leer un archivo de entrada (de extensión txt) y cargar las estructuras con los datos leídos. Se relaciona con el módulo Grafo para el acceso a estas estructuras. El formato del archivo de entrada es igual al descrito para la primera solución aproximada en la sección 3.1.2. Se puede encontrar un ejemplo completo de archivo de entrada en el Anexo III.

- Módulo ***Main***: su cometido es calcular la solución aproximada ejecutando los pasos uno a siete descritos anteriormente. Para esto necesita acceder a las estructuras de datos y cargarlas a partir del archivo de entrada, por lo que se relaciona tanto con el módulo Grafo como con el módulo LeerArchivo.
  Los resultados obtenidos se guardan en un archivo de salida (cuya extensión también es txt) con el formato que se indica a continuación. Se puede encontrar un ejemplo completo de archivo de salida en el Anexo V.

```
//En primer lugar aparecen las demandas que hay entre cada par de grupos:

############# Demanda entre Grupos#################
grupo: 1, grupo: 2, demanda: 66.000000
############# Fin de demanda entre Grupos##########
```

```
//Luego se listan los grupos creados, que clientes pertenecen a cada uno y
//que demandas tienen estos hacia el resto de los grupos:

========================= Lista de grupos ==============================
Grupo 1
idNodo 8 demanda con grupo1 = 0.000000 demanda con grupo2 = 12.000000
idNodo 7 demanda con grupo1 = 0.000000 demanda con grupo2 = 16.000000
idNodo 6 demanda con grupo1 = 0.000000 demanda con grupo2 = 20.000000
idNodo 4 demanda con grupo1 = 0.000000 demanda con grupo2 = 18.000000

Grupo 2
idNodo 3 demanda con grupo1 = 18.000000     demanda con grupo2 = 0.000000
idNodo 2 demanda con grupo1 = 12.000000     demanda con grupo2 = 0.000000
idNodo 1 demanda con grupo1 = 20.000000     demanda con grupo2 = 0.000000
idNodo 0 demanda con grupo1 = 16.000000     demanda con grupo2 = 0.000000

======================== Fin Lista de grupos ======================

//Se listan las aristas caño que comunican cada par de grupos:

======================== Lista de aristas caño ====================

Grupo 1

Aristas Caño con grupo 2

(6,3)    con tecnologia 7
(4,1)    con tecnologia 7
(4,2)    con tecnologia 5

Grupo 2

Aristas Caño con grupo 1

(1,4)    con tecnologia 7
(3,6)    con tecnologia 7
(2,4)    con tecnologia 5

================= Fin Lista de aristas caño ====================

//Luego viene la solución, es decir que enlaces aparecerán en la red de
//datos, que tecnología tendrán asignados y cuales son los ruteos en
//transporte para los mismos:

############## Red Resultado ###################################

-------------------------------------------------
i: 0, j: 0           //(i, j) corresponde a una arista de la red de datos.
distancia :0.000000  //En el caso de estar (i,j) en la sol. Es la
                     //distancia del ruteo en transporte.
Camino:              //Es el ruteo en la red de transporte.
idTecnologia :0      //Es la tecnología asignada a (i,j).

-------------------------------------------------
i: 0, j: 1
```

```
distancia :10.000000
Camino: 0     1
idTecnologia :5

-------------------------------------------------
.
.
.
-------------------------------------------------
i: 8, j: 7
distancia :9.000000
Camino: 7     8
idTecnologia :6

-------------------------------------------------
i: 8, j: 8
distancia :0.000000
Camino:
idTecnologia :0

########### Red Resultado FIN ##################

//Por último se muestra el costo de la solución:

El costo de la solución es: 652500.000000
```

En el Anexo VIII se describen las operaciones incluidas en cada módulo.

3.2.4. Estructuras de datos más importantes

Las siguientes son las estructuras de datos más importantes que se utilizaron:

- Estructura ***MatrizDemandas***.
- Estructura ***arrIdGrupos***.
- Estructura ***listaGrupo***.
  - Estructura ***GrafoGrupo***.
  - Estructura ***listaNodosSalida***.
  - Estructura ***listaNodos***.
- Estructura ***listaCombinacionGrupo***:
  - Estructura ***listaTecnologías***.
- Estructura ***MatrizCaminos***.
  - ***Estructura ListaDeNodos.***
- Estructura ***listaMatrizCamino***.
- Estructura ***listaPuntas***.
  - Estructura ***listaCamino***.
- Estructura ***listaAristaYDestinoUsados***.
- Estructura ***arrNodosSalidaUsados***
- Estructura ***MatrizHolgurasPuntaSalida***

A continuación se describe cada una de ellas:

- Estructura ***MatrizDemandas***: Esta estructura guarda la demanda desde cada nodo del grafo de datos hacia el resto. Se crea usando un arreglo de tamaño igual a la cantidad de nodos del grafo de datos. Cada índice del arreglo identifica un nodo, y contiene una lista de nodos a los cuales tiene demanda (es importante resaltar que esta relación es simétrica, o sea que si un nodo *n1* tiene demanda hacia un nodo *n2*, éste también tendrá demanda hacia el nodo *n1*).
  Cada nodo *j* de la lista correspondiente a la posición i del arreglo, tiene dos parámetros: el identificador del nodo *j* y el valor de la demanda que se encuentra entre el nodo *i* y el nodo *j* (el identificador del nodo es un campo entero y el de la demanda es real).
  Esta estructura fue creada para evitar cálculos innecesarios en la programación del software usando la matriz cuadrada con las demandas, que es ingresada desde la carga original del archivo de entrada. La matriz cargada a partir de los datos de entrada tiene generalmente muchos ceros, en esta estructura solo se representan las demandas distintas de cero, por lo que resulta más eficiente.

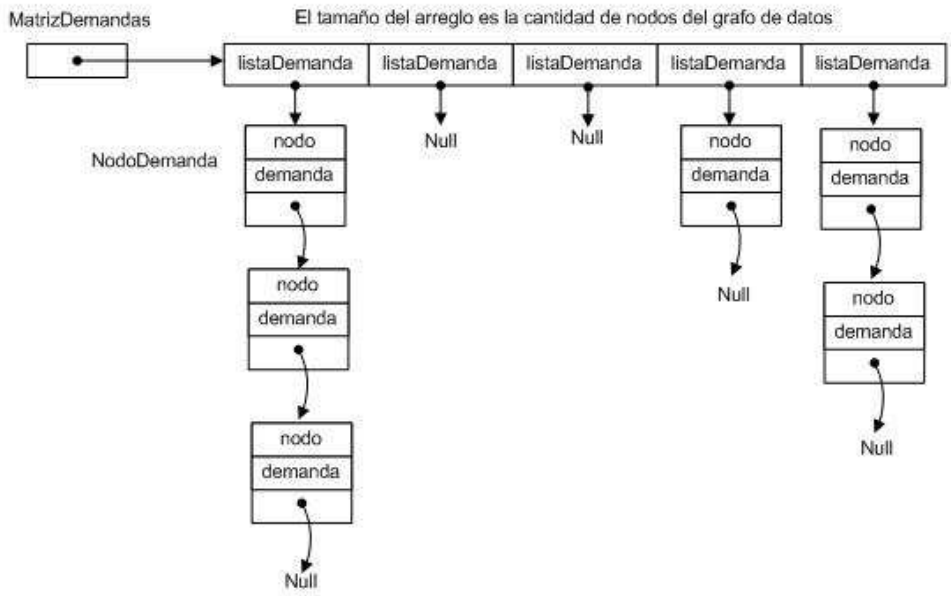

Figura 3.2.4.1: Descripción de la estructura *MatrizDemandas* de la solución aproximada 2.

- Estructura ***arrIdGrupos***: Esta estructura es un arreglo de enteros de tamaño la cantidad de nodos del grafo de datos. Básicamente es una estructura auxiliar que nos permite saber tomando el Id del Nodo como el índice del arreglo, en que grupo se encuentra dicho nodo.
  Se define como centinela o caso base el 0 en el valor del grupo, para expresar que el nodo del índice del arreglo aun no ha sido procesado y no pertenece a ningún grupo (o sea que en el momento de la consulta dicho nodo no pertenece aun a ningún conjunto). También se define el -2 como centinela para expresar la idea de que dicho nodo no participa en ningún conjunto. Esta estructura se inicializa en 0.

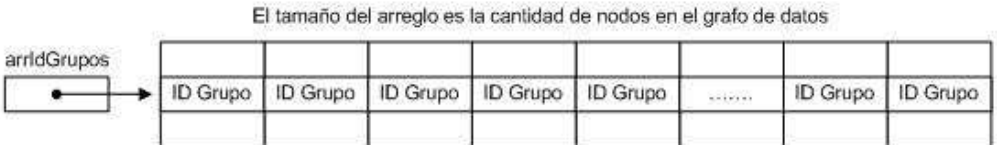

Figura 3.2.4.2: Descripción de la estructura *arrIdGrupos* de la solución aproximada 2.

- Estructura ***listaGrupo***: Esta estructura es la más compleja de las que se crearon para solucionar el problema de separar en grupos los nodos dependiendo de sus demandas. La estructura representa la lista de grupos diferentes que se generan por la separación de los nodos. Dicha estructura esta formada por una lista de nodos que llamaremos nodos tipo A para simplificar su explicación. Dichos nodos tipo A representan a un determinado grupo y tienen la siguiente información:
  - *idGrupo*: Es el número que identifica al grupo, el cual es un entero mayor a cero.
  - *lista_nodos*: es una variable de tipo ***listaNodos***. La estructura ***listaNodos*** consiste de una lista que contiene los nodos que pertenecen al grupo. Para cada uno de ellos se guarda la siguiente información:
    - *nodo*: Es el identificador del nodo.
    - *marcado*: Es un booleano que indica si el nodo ya fue usado en algún ciclo o no.
    - *demandas*: Es un arreglo de tamaño la cantidad de grupos total mas uno (ya que la posición cero no se utiliza). Cada posición representa el identificador de un grupo. En cada celda del arreglo se guarda la demanda que hay desde el nodo al grupo dado por el índice de la celda.
  - *grafo*: es una variable de tipo ***GrafoGrupo***. La estructura ***GrafoGrupo*** es un puntero a un grafo, que es preservado usando la forma de arreglo de listas. Este grafo corresponde al subgrafo que se construye para ese grupo, con los distintos pasos del algoritmo. El tamaño del arreglo para armar el grafo es la cantidad de nodos totales del grafo principal (misma cantidad que también tienen las estructuras *MatrizDemandas* (demanda entre nodos) y *arrIdGrupos* (grupo de cada nodo)). En las listas del arreglo de nodos se colocara hacia que nodo destino existe el enlace adyacente.
    Como lo hicimos con *arrIdGrupos* (grupo de cada nodo) que definimos una lista de valores predeterminados con una semántica determinada, en este caso el valor Null significa que el nodo esta en el grupo pero aun no se le asignó un nodo adyacente (es un estado intermedio), el menos uno significa que el nodo esta en otro grupo, el menos dos significa que el nodo no esta en ningún grupo porque no tiene demanda o sea que no es un nodo valido. La estructura de grafo para cada grupo se inicializa en menos uno.
  - *arrNodosSalida*: es un arreglo de ***listaNodosSalida*** cuyo tamaño es la cantidad de grupos total mas uno y cuyo índice representa el identificador de un grupo destino. La estructura ***listaNodosSalida*** es una lista de nodos en el que en cada posición se guardan los datos de las aristas caño que van desde nodos de *idGrupo* a nodos del grupo dado por esa posición. Cada nodo contiene la siguiente información:
    - *nodo*: es el identificador del nodo de salida en el grupo *idGrupo*, que es uno de los extremos de la arista caño.
    - *nodoDestino*: es el identificador del nodo de salida en el grupo dado por la posición del arreglo, que es el otro extremo de la arista caño.
    - *indiceMenorTecn*: representa el índice del arreglo de la menor tecnología que le llega al nodo *nodo*.

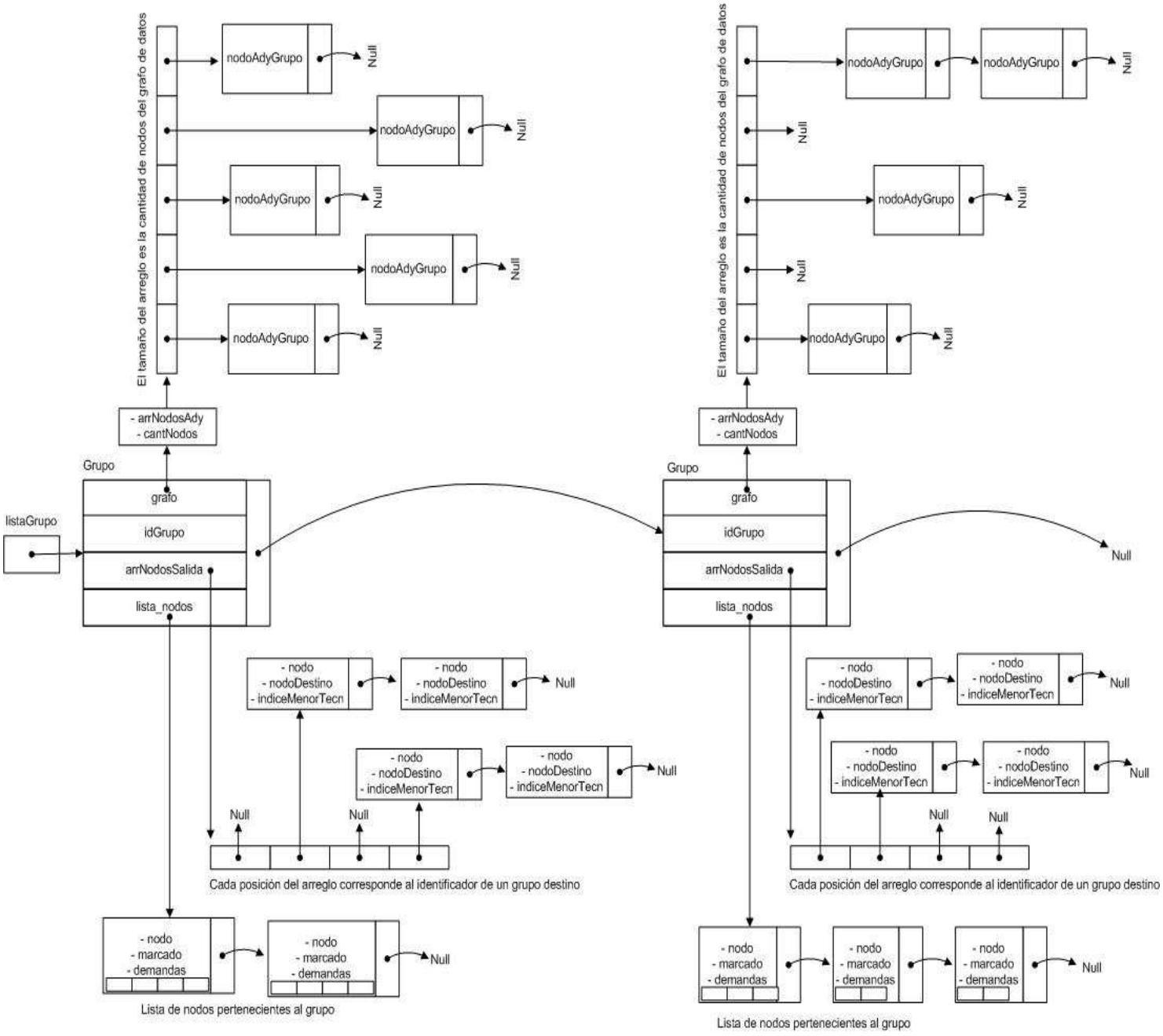

Figura 3.2.4.3: Descripción de la estructura *listaGrupo* de la solución aproximada 2.

- Estructura ***listaCombinacionGrupo***: es una lista que guarda información sobre los grupos que se encuentran conectados. Cada nodo de la lista tiene la siguiente información:
  - *grupo1*: es un entero que identifica a uno de los grupos que se encuentra conectado.
  - *grupo2*: es un entero que identifica al otro grupo que se encuentra conectado.

- *demanda*: es un entero que indica la demanda entre ambos grupos, o sea el tráfico entre ambos grupos.
- *lista_tecnologias*: es una variable de tipo ***listaTecnologías***. La estructura l***istaTecnologías*** consiste de una lista que contiene información sobre las tecnologías asignadas a las aristas caño que conectan a los dos grupos. Cada nodo de esta lista tiene la siguiente información:
  - *idTecnologia*: es un entero que representa el índice del arreglo de tecnología, es decir que indica la tecnología a usar para las aristas caño incluidas en ese nodo.
  - *nroAristasCaño*: es un entero que indica la cantidad de aristas caño que tienen que tener esa tecnología asociada.
  - *nodosCaño*: es un arreglo que guarda la información de todas las aristas caño entre los dos grupos, que tienen asociada la tecnología *idTecnología*. El tamaño del arreglo es *nroAristasCaño* y cada celda tiene dos enteros que son:
    - *idNodoOrigen*: identificador del nodo de salida origen de la arista caño.
    - *idNodoDestino*: identificador del nodo de salida destino de la arista caño.

  Esta lista se ordena de mayor a menor en la capacidad de las tecnologías. Esto simplifica el algoritmo, ya que al momento de elegir los nodos que serán nodos de salida de cada grupo, los pares que tienen menor distancia en transporte son seleccionados primero y se asignan a las aristas caño de mayor capacidad que se encuentran primero en esta lista por encontrarse ordenada.

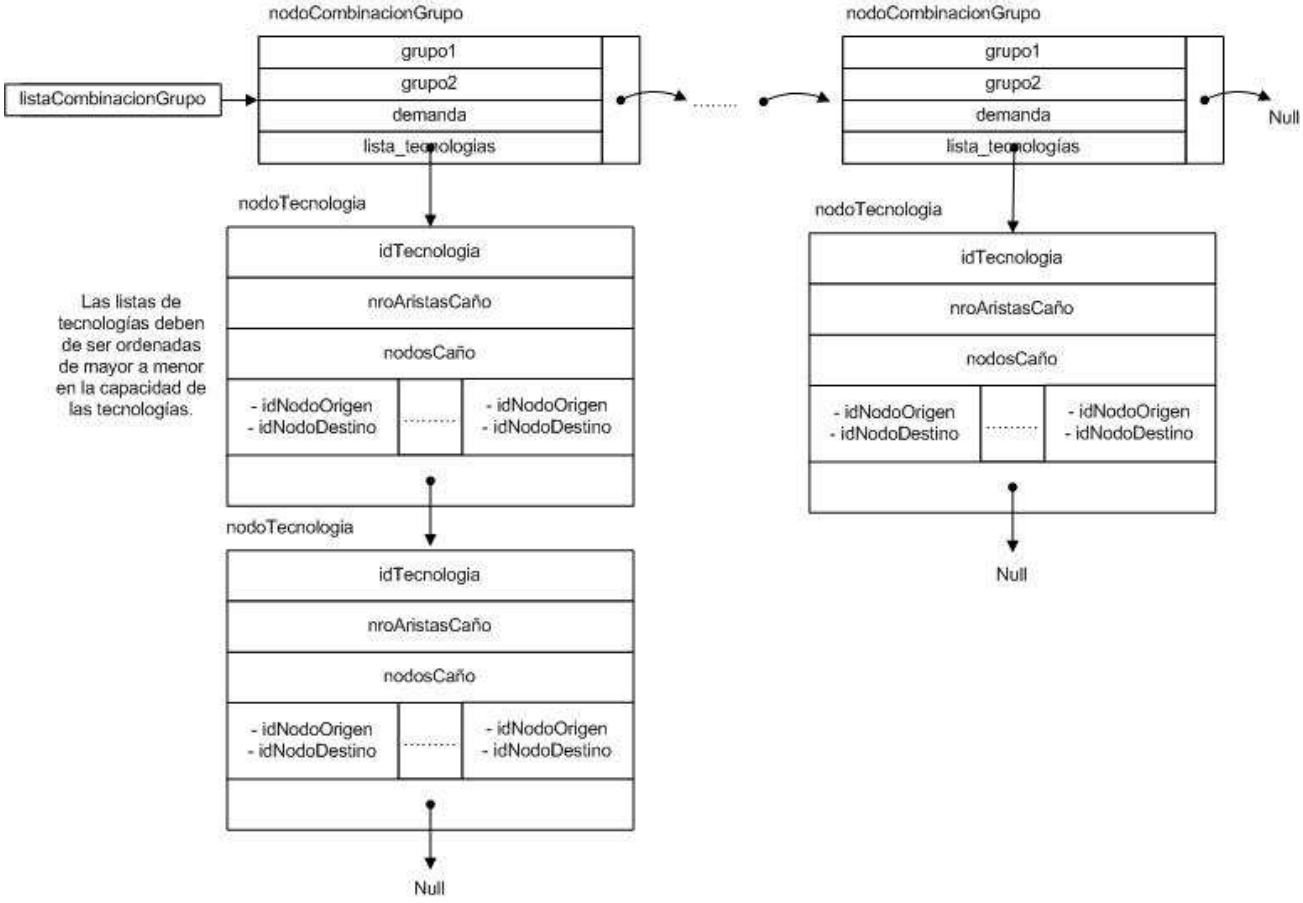

Figura 3.2.4.4: Descripción de la estructura *listaCombinacionGrupo* de la solución aproximada 2.

- Estructura ***MatrizCamino***: es un tipo de dato utilizado para varios propósitos, entre ellos se destacan los siguientes:
  a) Cargar los resultados procesados de la tabla de Dijkstra con los caminos más cortos entre cada par de nodos del grafo de transporte.
  b) Almacenar valores temporales de caminos mas cortos y tecnologías
  c) Guardar los resultados finales para calcular la solución objetivo utilizando sólo dicha estructura

  Esta estructura es una matriz cuadrada de tamaño la cantidad de nodos de datos y almacena en cada una de sus celdas $m(i, j)$ la siguiente información:
  - *distancia*: es la distancia del camino en transporte que va desde el $tns(i)$ al $tns(j)$.
  - *camino*: es una variable de tipo ***ListaDeNodos***. La estructura ***ListaDeNodos*** es una lista que contiene los nodos de transporte que forman parte del camino entre $tns(i)$ y $tns(j)$. Para cada nodo del camino se guarda:
    - *nodo*: identificador del nodo de transporte que forma parte del camino.

- *contador*: distancia acumulada que lleva el camino desde el origen hasta ese nodo.

- *demandaUtilizada:* se utiliza para almacenar la demanda que se va asignando a la arista caño y a las aristas que comparten ciclos durante el algoritmo. Dicha demanda se va incrementando a medida que los nodos de los grupos son procesados y no puede superar la capacidad de la tecnología indiceTecnArista.
- *indiceTecnArista:* es un entero que representa el índice del arreglo de tecnología, es decir que indica la tecnología utilizada por la arista *(i, j)*.

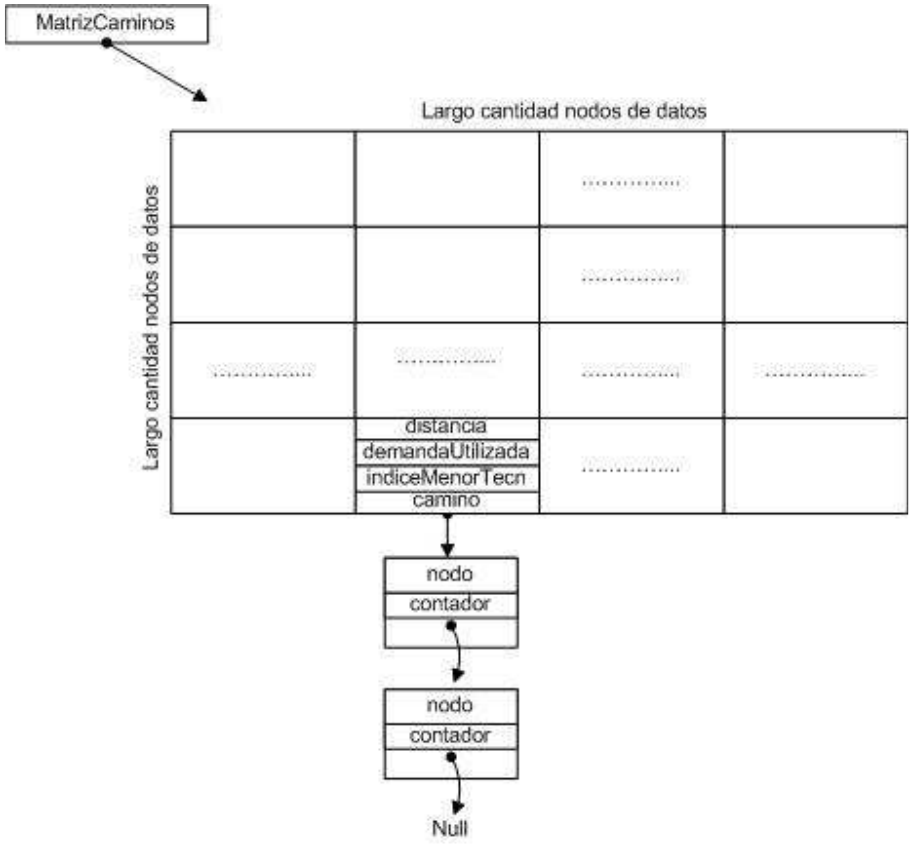

Figura 3.2.4.5: Descripción de la estructura *MatrizCamino* de la solución aproximada 2.

- Estructura ***listaMatrizCamino***: corresponde a una lista de estructuras *MatrizCamino*. En cada nodo de la lista (*nodoMatrizCaminos*) se guarda la matriz camino y un identificador de la misma asociado a una única punta. Esta estructura se utiliza en el paso seis del algoritmo, cuando se construyen los ciclos internos dentro de los grupos, para guardar de forma temporal los caminos en transporte de las distintas variantes de ciclos que se pueden construir.

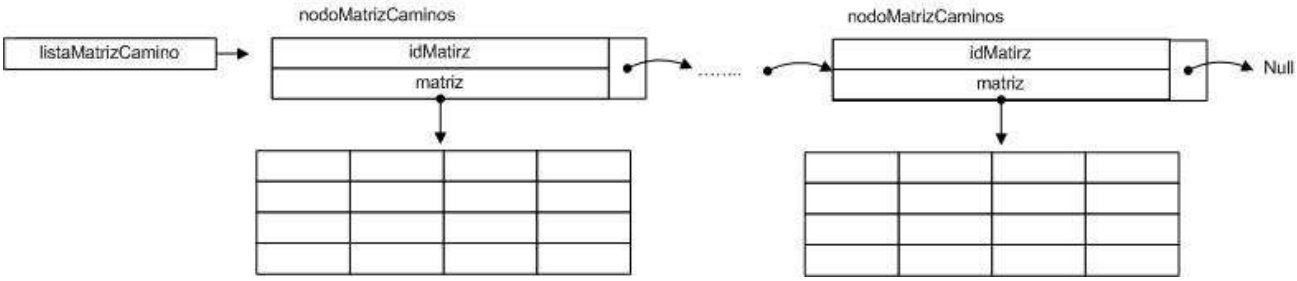

Figura 3.2.4.6: Descripción de la estructura *listaMatrizCamino* de la solución aproximada 2.

- Estructura ***listaPuntas***: esta estructura se utiliza en el paso seis del algoritmo para armar los ciclos internos dentro de cada grupo. El término punta se utiliza para identificar una cadena de nodos dentro de un grupo que se encuentran conectados, las puntas se conectan luego para armar ciclos internos dentro de los grupos. Esta estructura es una lista de puntas, donde cada punta (*nodoPunta*) tiene la siguiente información:
  - *puntaIzq*: es el identificar de uno de los nodos extremo de la cadena de nodos.
  - *puntaDer*: es el identificador del otro extremo de la cadena de nodos.
  - *estaEnSalida*: es un booleano que indica si alguno de los nodos de la cadena es un nodo de salida del grupo (valor true), o si no hay nodos de salida en la cadena (valor false).
  - *id*: es el identificador de la matriz camino que guarda los caminos en transporte para las aristas de datos que componen la cadena de nodos de la punta. Dado este *id*, para obtener la matriz camino correspondiente a esta cadena, se recorre la *listaMatrizCamino* (estructura descrita anteriormente) buscando el nodo de la lista que tenga *idMatriz* igual a *id*.
  - *demandasGrupos*: es un arreglo de números de tamaño la cantidad de grupos total mas uno. En cada posición se guarda, la suma de las demandas de todos los nodos de la cadena, hacia el grupo dado por el índice de esa posición
  - *camino*: es una variable de tipo ***listaCamino***. La estructura ***listaCamino*** guarda la lista de aristas de datos que componen la cadena. Cada arista es representada por un nodo donde se guardan los identificadores de los nodos extremos de la arista (campos *izq* y *der* de *nodoCamino*). El extremo izquierdo del primer nodo de la lista debe ser igual a la

punta izquierda de la cadena (campo *puntaIzq*) y el extremo derecho del último nodo de esta lista debe ser igual a la punta derecha de la cadena (campo *puntaDer*).

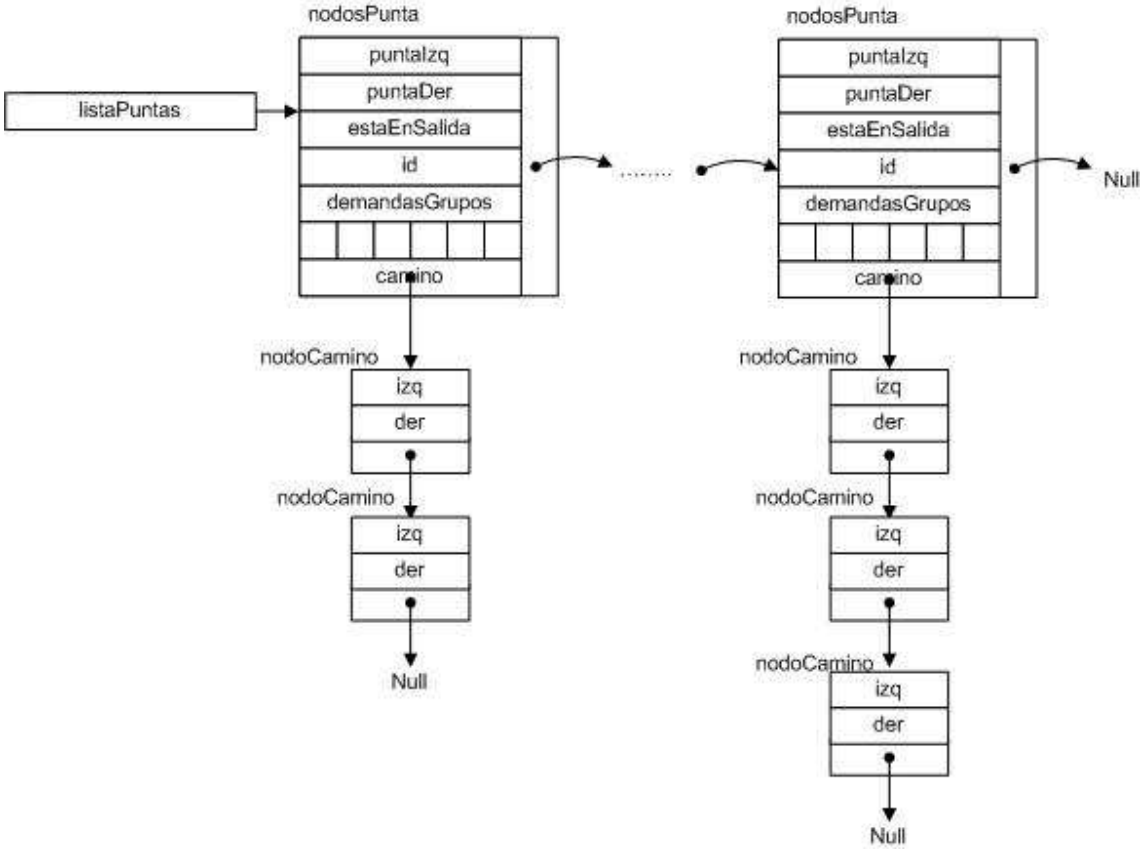

Figura 3.2.4.7: Descripción de la estructura *listaPuntas* de la solución aproximada 2.

- Estructura ***listaAristaYDestinoUsados***: es una estructura que se sirve para recordar tecnologías y destinos de cada nodo que ya fueron utilizados para no repetirlos.

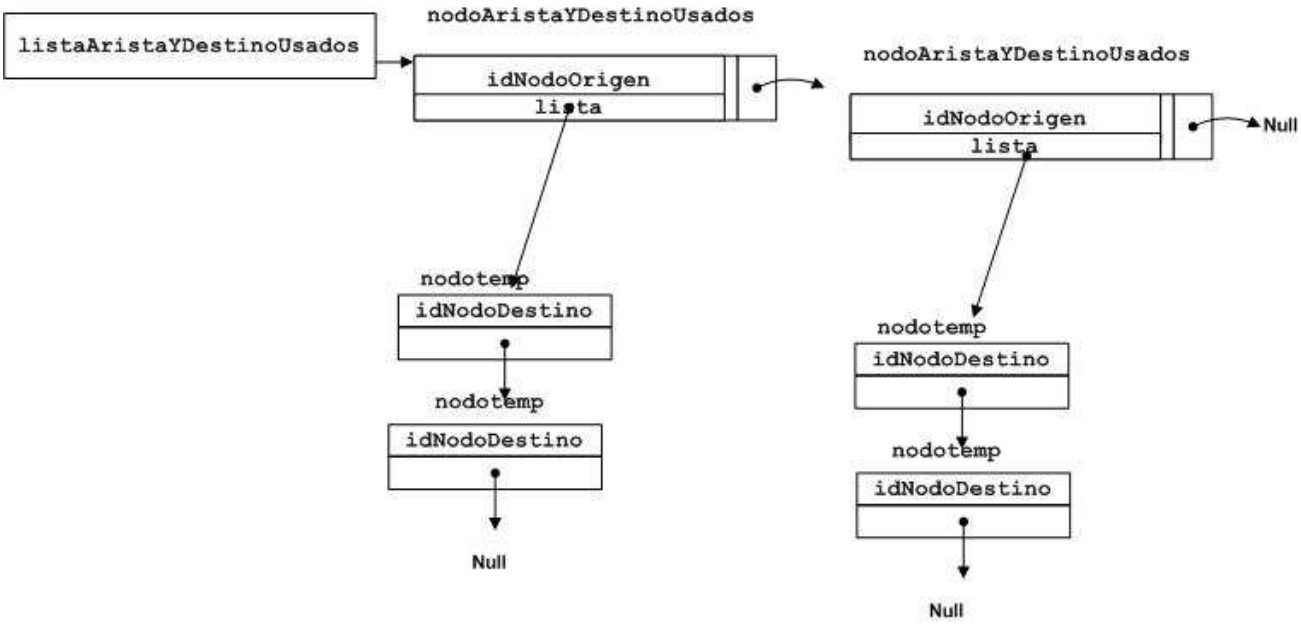

Figura 3.2.4.8: Descripción de la estructura *listaAristaYDesinoUsados* de la solución aproximada 2.

- Estructura ***arrNodosSalidaUsados***: es una estructura que sirve para guardar en un arreglo los nodos de salida que ya use para no sumarlos 2 veces en las aristas caño.

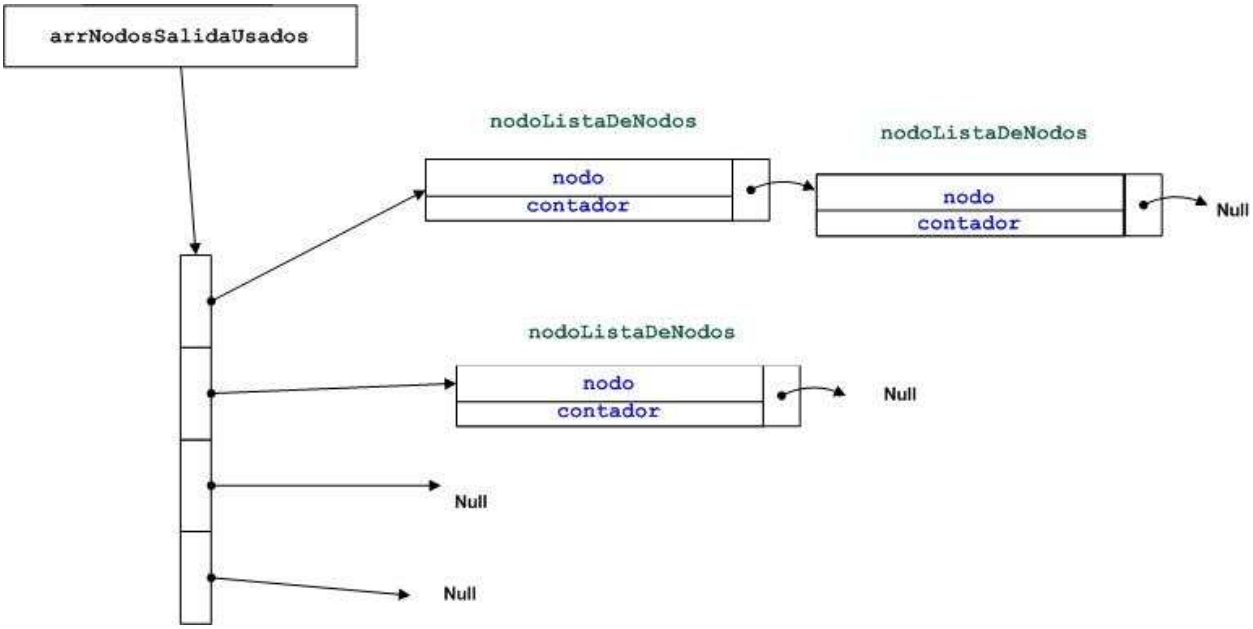

Figura 3.2.4.9: Descripción de la estructura *arrNodosSalidaUsados* de la solución aproximada 2.

- Estructura ***MatrizHolgurasPuntaSalida***: es una estructura que sirve para ir almacenando la holgura disponible que tienen las aristas dentro de un mismo grupo cuyos extremos son ambos nodos de salida.

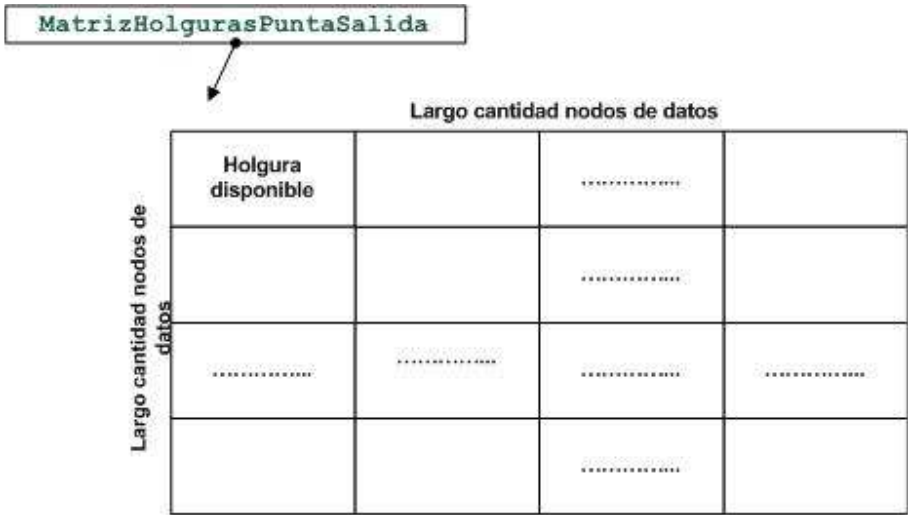

Figura 3.2.4.10: Descripción de la estructura *MatrizHolgurasPuntaSalida* de la solución aproximada 2.

3.2.5. Descripción de cómo se implementa cada paso del algoritmo

- **Paso 1**

Objetivo: Calcular los caminos más cortos entre todo par de nodos del grafo de transporte.

Para cumplir con el objetivo se ejecuta el algoritmo de Dijkstra en el grafo de transporte. Los resultados se guardan en la estructura *MatrizCaminos*.

- **Paso2**

Objetivo: Separar los nodos del grafo de datos en grupos, de acuerdo a su demanda, de forma que los nodos con demanda entre si se encuentren en grupos separados. Análogo al problema de coloración de grafo [7] [12] [13].

El problema de separar los nodos en la mínima cantidad de grupos manteniendo los criterios descriptos y la definición de grupo (los nodos pertenecientes a un mismo grupo no tienen demanda entre ellos) es similar al problema de coloración de grafos. Este es un problema NP-Hard [15] por lo que en este proyecto de grado no encontramos una solución polinomial exacta al mismo. Buscamos una solución exacta y la encontrada fue un backtracking de tiempo exponencial [14], por lo que se optó por una solución polinomial del tipo voraz [15] [16]. A continuación se brinda un pseudocódigo del algoritmo que realizamos, que da una solución aproximada a este problema:

```
If (listaGrupo == null) {
        /* todavía no se creo ningún grupo, creamos 2 grupos y le cargamos a cada uno el
numero del grupo, luego creamos el grafo del grupo inicializándolo en -1 */
}

Para todo nodo v en el grafo de datos {
        //obtengo los nodos hacia los que v tiene demanda
        lista_nodos_con_demanda = MatrizDemandas[v];
    If (lista_nodos_con_demanda != NULL){
        If (arrIdGrupos(v) == 0) {
          //El nodo v aún no esta asignado a ningún grupo, en caso contrario no hay nada para
          //hacer
                ld = creoListaDeGruposDisponibles();
                /* ld va a tener todos los id de los grupos creados hasta el momento y que son
                disponibles para colocar el nodo v */
```

```
                Para todo nodo w perteneciente a la lista lista_nodos_con_demanda {
                        //recorro la lista de nodos hacia los que v tiene demanda
                        If (arrIdGrupos(w) != 0){
                                /*w ya esta en algún grupo, por lo que debo sacar ese grupo de ld,
                                ya que v no puede estar en el mismo grupo que w, por tener
                                demanda entre si*/
                                ld.sacoGrupo(arrIdGrupos(w));
                        }
                } Fin de nodo w perteneciente a la lista lista_nodos_con_demanda

                If (ld.notEmpty()) {
                        /*tomamos el primer grupo de los disponibles para agregar el nodo i
                        agrego el nodo a la lista de nodos del grupo
                        agrego el nodo al grafo asociado al grupo
                        le asigno el grupo al que pertenece el nodo procesado en el
                        arregloIdGrupo*/
                }
                Else{
                      /*tengo que crear un nuevo grupo porque el nodo no puede ponerse en los
                       Demás
                            gn = creounGrupoNuevo();
                       agrego el nodo a la lista de nodos del grupo
                            gn.agrego(v);
                       agrego el nodo al grafo asociado al grupo
                       le asigno el grupo al que pertenece el nodo procesado en el
                       arregloIdGrupo*/
                }
        } Fin If
    } Fin If
    Else{
        /*agrego el nodo i en la lista de nodos inválidos (nodos que no tienen demanda con
          ningún otro nodo)
          indico en el arregloIdGrupo que dicho nodo no es valido (no tiene demanda hacia
          nadie, poniendo un -2 en la posición del índice correspondiente al id de nodo en el
         arreglo)*/
    }
}// fin del for para todos los nodos v de datos

//Para cada grupo de la lista de grupos resultante, en sus grafos correspondientes, se carga con -2
//todos los nodos inválidos en dicho grafo.
```

A continuación, agregamos los siguientes comentarios a modo aclarativo:

Cuando agregamos un nodo *v* con demanda, a un grupo *g*, realizamos lo siguiente:

1. En la estructura *arrIdGrupos* cargo el número de grupo *g* en la posición del arreglo correspondiente al nodo *v*.
2. En el nodo *Tipo A* que corresponde al grupo *g*, agrego *v* a la lista de nodos que pertenecen al grupo, también agrego *v* en el grafo del grupo colocando en null su lista de adyacentes, quitando el menos uno de la inicialización.

Si el nodo *w* que estamos procesando es un nodo no válido, por no tener una lista de demanda asociada en la estructura *MatrizDemandas*, se realiza lo siguiente:
Se agrega el *id* de *w* en una lista de nodos inválidos, que será usada en forma temporal. Cuando se hayan procesado todos los nodos de *MatrizDemandas*, se recorren todos los grupos y para cada uno se setean con valor menos dos los nodos del grafo asociado que pertenecen a la lista de nodos inválidos.

También se setea con menos dos en la estructura *arrIdGrupos*, en la posición *w*, para indicar que dicho nodo no es valido.

Paso 3

Objetivo: Calcular la demanda que hay entre cada par de grupos.

Para alcanzar este objetivo realizamos lo siguiente:

1. Para todas las combinaciones de grupos *g1*, *g2*, obtenemos la lista de nodos de cada grupo (usando los correspondientes nodos *Tipo A*), sean estas listas *ln1* y *ln2* respectivamente.
2. Para cada nodo *v*, miembro de *ln1*, ver la lista de nodos con los que tiene demanda (usando *MatrizDemandas*), sea esta lista *ld1*.
3. Para cada nodo de la lista de demandas *ld1*, ver a que grupo pertenece (usando *arrIdGrupos*). Si pertenece a *g2*, sumar la demanda a una variable (la demanda se obtiene también de la estructura *matriz_Demandas*).
4. Cuando se hayan procesado todos los nodos de *ln1* tenemos calculada la demanda desde *g1* hacia *g2*.

Paso 4

Objetivo: Calcular la cantidad de aristas caño necesarias entre cada par de grupos (de acuerdo a la demanda entre los mismos) y las tecnologías asociadas a las mismas.

Como el cálculo de las aristas caño entre grupos se realiza para todos los combinaciones de grupos posibles, se necesita tener una estructura para almacenar dichos cálculos, esta estructura es *listaCombinacionGrupo*.

Anteriormente ordenamos el arreglo de tecnologías de menor a mayor en cuanto a capacidad.
Para calcular la cantidad de aristas caño y las tecnologías a asignar a las mismas realizamos los siguientes pasos:

Calculo la cantidad de *aristas_caño* y las tecnologías a asignar a las mismas de la siguiente manera:

Tomo la menor *tecnologíaA* que cubre *Dtotal*(*Gi*, *Gj*) ó la más grande si no hay ninguna que cubra
Cantidad _ caños = 0;
While demanda – *tecnologíaA* > 0
sumo una *arista_caño Caño_i*(*Gi*, *Gj*) y le asigno *tecnologíaA*
demanda = demanda – *tecnologíaA*
cantidad _ caños++;
si demanda > 0
Tomo la menor *tecnologíaB* que cubra la demanda_resultado calculada de la siguiente forma:
Obtengo aquel nodo (nodoi) del grupo Gi que tiene una mayor demanda hacia el grupo Gj
Obtengo aquel nodo (nodoj) del grupo Gj que tiene una mayor demanda hacia el grupo Gi
Demanda_resultado se calcula como el Max{nodoi, nodoj, demanda}
sumo una *arista_caño Caño_i*(*Gi*, *Gj*) y le asigno *tecnologíaB*
cantidad _ caños++;
sumo una *arista_caño Caño_i*(*Gi*, *Gj*) y le asigno la tecnología mayor de todas las anteriores (cubro caída del algún caño)
cantidad _ caños++;

Ahora que ya conocemos la cantidad de aristas caño entre los grupos y las tecnologías asignadas a las mismas, guardamos esta información en la estructura *listaCombinacionGrupo*.

Para cada par de grupos que se relacionan (representados por nodos de la *listaCombinacionGrupo*) agregamos un nodo por cada tecnología diferente que tengan las aristas caño entre los mismos.

En este nodo tecnología guardamos el identificador de la tecnología, la cantidad de aristas caño entre esos grupos que tienen esa tecnología e inicializamos en menos uno el arreglo con los identificadores de los nodos de salida de cada grupo (que son extremo de las aristas caño), para indicar que todavía no tienen valor (ya que los nodos de salida se calculan en el paso cinco).

La lista de tecnologías para cada par de grupos que se relacionan, se ordena de mayor a menor en la capacidad de las tecnologías. Esto simplifica el paso cinco, donde se eligen los nodos de salida de cada grupo, los pares de nodos que tienen menor distancia en transporte son seleccionados primero y se asignan a las aristas caño de mayor capacidad que se encuentran primero en esta lista, por encontrarse ordenada.

Paso 5

Objetivo: Para cada arista caño fijar los nodos de salida de cada grupo que serán sus extremos y fijar el ruteo que tendrá en el grafo de transporte.

Para cumplir con el objetivo se realizan los siguientes pasos:

1. Para cada par de grupos *g1* y *g2* que se relacionan (nodos de la *listaCombinacionGrupo*).
2. Se lanza la ejecución de un algoritmo Dijkstra en el grafo de transporte, los resultados con todos los caminos se guardan en una matriz temporal (*matriz_camino_tmp*) del tipo *MatrizCaminos*.
3. Sea *lt* la lista de tecnologías asociada a los grupos *g1* y *g2*. Cada nodo de *lt* representa una tecnología y tiene la cantidad de aristas caño que se necesitan de esa tecnología.
4. Realizo una búsqueda en *matriz_camino_tmp* de la menor distancia entre los nodos de *g1* y *g2*. Sean *n1* y *n2* los nodos que encuentro.
5. *n1* y *n2* pasan a ser los nodos de salida extremos de una de las aristas caño que se requieren para esa tecnología. Cargo los identificadores *n1* y *n2* en el arreglo *nodosCaño* del nodo de *lt* que estamos procesando.
6. Copio el camino entre *n1* y *n2* en una matriz del tipo *MatrizCaminos* que llamaremos resultado. En dicha matriz también cargamos la distancia del camino, y la tecnología correspondiente a la arista caño con extremos *n1* y *n2.*
7. Luego tiramos los link en transporte que utiliza el camino entre *n1* y *n2* y volvemos a repetir desde el paso dos, hasta cubrir todas las aristas caño que son necesarias entre *g1* y *g2*, es decir recorriendo toda la lista de tecnologías asociada a *g1* y *g2* y dentro de cada tecnología todas las aristas caño que se requieren. Es necesario tirar los link ya que las aristas caño entre un mismo par de grupos deben ser link disjuntas en transporte, para soportar el escenario de falla.

Nota 1

En el paso cuatro, para obtener entre que nodos comparar la distancia, vamos al nodo *Tipo A* del grupo *g1* y del nodo *Tipo A* del grupo *g2* y obtenemos de cada uno, la lista de nodos que forman parte del grupo, sean estas listas *ln1* y *ln2* respectivamente. A continuación se incluye un pseudocódigo de cómo se buscan los nodos que tienen menor distancia entre los grupos *g1* y *g2*.

```
BuscarMenorDistancia(){
        min = infinito;
        Mientras (ListaNodoGrupo ln1 <> null){
                Mientras (ListaNodoGrupo ln2 <> null){
                        If (matriz_camino_tmp[nodo_ln1, nodo_ln2].distancia < min){
                                /* guardo id de ambos nodos, el de nodo_ln1 y el de nodo_ln2 y
                                actualizo min con el nuevo mínimo*/
                        }
                }
        }
        Retorno los id de los nodos de menor distancia
}
```

Nota 2

El par (n1, n2) hallado en el paso 4 debe ser único. Se pueden repetir n1 o n2, pero no ambos a la vez porque cada arista caño tiene que tener extremos distintos. A su vez, el par (n1, n2) elegido no pueden tener una demanda entre ellos que supere la tecnología asignada a la arista caño.

Hasta aquí hemos calculado las aristas caño, hemos asignado los nodos entre ellas y además hemos calculado caminos disjuntos entre ellas, finalmente hemos asignado una tecnología a cada arista caño.

Paso 6

Objetivo: armar ciclos dentro de cada grupo con la mayor cantidad de nodos dentro de cada ciclo, con la menor distancia en su contorno y procesar todos los nodos no procesados del grupo. Este problema es analogo al problema de TSP (Travel Salesman Problem) [7][9][10][11] [17].

En primera instancia se cargan los nodos de salida de cada grupo, en el arreglo de listas de nodos de salida *arrNodosSalida* que tiene cada grupo en la estructura *listaGrupo*. Para ello se realizan los siguientes pasos:

Parte A

1. Inicializo los nodos de salida de cada grupo.
2. Recorro la lista de grupos, sea el grupo actual *grupo1*.
3. Recorrer los nodos que pertenecen a *grupo1*, sea el nodo actual *nodo1*.
4. Recorro la lista de combinaciones de grupos.
5. Para cada nodo combinación donde esté *grupo1*, sea *grupo2* el grupo con el que se relaciona. Llamemos *nodoComb* a este nodo.
6. Recorrer la lista de tecnologías de *nodoComb*.
7. Recorrer la lista de aristas caño de cada tecnología.
8. Ver si *nodo1* es el origen o el destino de una de esas aristas
9. Si es así entonces agregar *n1* a la lista de nodos de salida del *grupo1* en la posición del arreglo dada por *grupo2*.

Luego de realizar los pasos anteriores se tiene cada grupo con sus nodos de salida.

Parte B

Primeramente unimos los nodos de salida que tienen aristas caño hacia el mismo grupo usando la función *UnirNodosSalidaRespaldo*

Posteriormente se crean puntas de cadenas dentro de cada grupo, a partir de los nodos de salida. Estas cadenas luego se unirán cumpliendo la condición de mantenerse link disjuntas con las aristas caño asociadas a los nodos de salida correspondientes.

Los pasos para cumplir con dicho objetivo se pueden resumir de la siguiente manera:

1. Recorro la lista de grupos, sea el grupo actual *grupo1*.
2. Recorro todos los índices del arreglo del grupo y para cada índice me fijo si hay algún nodo de salida hacia dicho grupo.
3. Para cada nodo de cada lista de cada índice del arreglo, cargo cual es su correspondiente destino y la menor tecnología de las aristas caño que le llegan al nodo.
4. Ordenamos cada lista de cada índice del arreglo por la tecnología que tiene la arista caño correspondiente a ese nodo de salida de mayor a menor.
5. Armamos una lista con los primeros nodos de cada lista del arreglo (que son los de mayor tecnología)
6. Ordenamos la lista de los nodos que tienen la mayor tecnología (la generada en el paso anterior) con los demás grupos de mayor a menor.
7. Clono la lista de nodos de salida de *grupo1*, sea esta *listaClonadaSalida*.
8. Recorro los nodos de *listaClonadaSalida* y cargo en *listaMayorTecnología* los nodos de salida que tienen la misma tecnología y es la mayor.
9. Elimino los nodos de *listaMayorTecnologia* de *listaClonadaSalida*.

10. Recorro la lista *listaMayorTecnología* para tratar de unir primero los nodos de mayor tecnología.
11. Creo una punta por cada nodo de *listaMayorTecnología* y las agrego a una lista de puntas (*lista_puntas*).

Unión de puntas de la *lista_puntas*:

12. Comparamos todas las puntas de *lista_puntas* de a pares, buscando los caminos en transporte de las aristas caño asociadas a cada punta.
13. Tiro estos caminos en el grafo de transporte.
14. Ejecuto el algoritmo de Dijkstra en el grafo de transporte
15. Busco camino entre las dos puntas.
16. Si (*distancia camino encontrado* < *min*) y (*esPosibleUnirCadena()*)
    min = distancia
    guardo el camino de links
17. Guardo que extremos voy a unir. Ya que en la primera ejecución se trata de unir puntas unitarias pero en las siguientes iteraciones se unen cadenas, por lo que puede haber cuatro posibilidades para unir, izquierda *cadena1* con izquierda *cadena2*, izquierda *cadena1 con* derecha *cadena2*, derecha *cadena1* con izquierda *cadena2* y derecha *cadena1* con derecha *cadena2*. Se elegirá la que tenga menor distancia y sea posible unir las cadenas.

18. Si ya hay un camino fijado en transporte que una dos extremos de cadena:

    a. Utilizo ese camino ya que no debe cambiar.
    b. Uno esta combinación encontrada de puntas. Llamémosle *unirCadena()* a la función que se encarga de unir las mismas. Dentro de esta función se indica el camino fijado con anterioridad y la distancia de dicho camino.
    c. Le asigno la tecnología a la punta obteniéndola de la tecnología asignada anteriormente para ese camino.

19. Si no hay un camino en transporte para la unión de los dos extremos de las puntas:

    a. Asigno temporalmente dicho camino en transporte.
    b. Uno esta combinación encontrada de puntas. Llamémosle *unirCadena()* a la función que se encarga de unir las mismas.
    c. Le asigno la tecnología a la punta asignándole la menor tecnología de todas las tecnologías que tienen las aristas caño de los extremos.

20. Guardo en *lista_puntas* la cadena generada.
21. Si *lista_puntas* tiene mas de un elemento vuelvo al paso 12

Por lo que hasta este punto se tienen unidos los nodos de salida de cada grupo. Resta comenzar a generar ciclos que incluyan todos los nodos no marcados de cada grupo.

Parte C

A continuación se describen los pasos para generar ciclos con los nodos no marcados:

1. Si hay nodos internos sin marcar, obtengo todos los nodos internos sin marcar del grupo y los guardo en una lista llamada *lnsNoMarcados*.
   Si hay nodos de salida sin marcar, obtengo todos los nodos de salida que no hayan sido procesados guardándolos en la misma lista.
   En caso de que todos los nodos del grupo estén marcados voy al paso 19 de Parte C.
2. Para cada nodo de *lnsNoMarcados* intento verificar que pueda generar un ciclo con la punta *lista_puntas* de la siguiente manera*:*
   a. Transformo dicho nodo en una punta mediante la función *CrearCadena().* Llamémosle *prueba1*.
   b. Verifico si es posible unir la punta *prueba1* con *lista_puntas* mediante la función *EsPosibleUnirCadena()*
   c. En el caso de que sea posible la unión hago una preunión entre ambas mediante la función *preUnión()*
   d. Usando la función *cerrarCiclo()* verifico si es posible cerrar un ciclo luego de hacer la *preUnión*.
3. Si todos los nodos de *lnsNoMarcados* pueden formar un ciclo con *lista_puntas* voy al paso 4, de lo contrario se deberá hacer lo siguiente:
   a. Obtenemos el camino en transporte (el conjunto de links) que tiene asociada la punta *lista_puntas*. Llamémosle a dicho conjunto de links *linksPunta.* Dicha lista queda fija y se va iterando sobre ella. En el caso de que se vuelva a este paso nuevamente solo debo avanzar la lista y en el caso de que no existan más links a tirar no existe solución factible al problema.
   b. Para cada link de *linksPunta* se tira el link en el grafo de transporte dejando levantados los demás links, y se vuelve al paso 7 de la Parte B.
4. Obtengo todos los nodos que no son de salida (nodos sin marcar y nodos internos) del grupo, los guardo en una lista, sea ésta *listaNodosNoSalida*.
5. Creo una punta con cada uno de los nodos de *listaNodosNosSalida.*
6. Agrego estas puntas a *lista_puntas*, por lo que ahora esta lista tiene no solo las cadenas generadas hasta el momento sino también cada nodo no marcado, llamémosle a esta nueva lista *lista_puntas_general*.
7. A partir de *lista_puntas_general* generar ciclos con los nodos no marcados, mediante la función *generarCicloConNodosNoSalida*.
8. Cargo el grafo asociado al grupo en la estructura *listaGrupo* con los nuevos ciclos generados.
9. Asigno la tecnología al ciclo
10. Marco los nodos procesados.

11. Cargo la demanda ocupada en las aristas caños correspondientes.
12. Cargo el camino definitivo en la estructura del resultado final
13. Verifico si existen nodos no marcados internos, si no existen entonces terminé de procesar los nodos internos de ese grupo (voy al paso 14 de parte C), si existen entonces seguir con el siguiente paso.
14. Creo un arreglo *arregloOrdenado* donde ordeno de mayor a menor las demanda de los nodos hacia un mismo grupo. Es decir que es un arreglo donde cada posición representa un grupo destino y en cada posición guardo los nodos internos sin procesar que tienen demanda hacia ese grupo, ordenados de mayor a menor demanda.
15. Tomo un nodo no marcado y me fijo su demanda hacia los otros grupos (sea este *nodoNoMarc1*).
16. Para cada grupo con el que *nodoNoMarc1* tiene demanda selecciono usando *arregloOrdenado* los nodos de salida a los que lo voy a conectar. Para cada grupo destino, recorro la lista de nodos de salida asociada a la posición de ese grupo destino. Detengo la recorrida de cada lista, cuando encuentro un nodo de salida que tiene la arista caño asociada con la suficiente holgura para que pase la demanda de *nodoNoMarc1*. La recorrida comienza desde el principio de la lista, por lo que al estar ordenada, siempre trato de hacer pasar los nuevos nodos por las aristas caño de mayor tecnología posible e ir llenando las holguras. Vuelvo al paso 7 de la Parte B
17. Verifico si todos los nodos del grupo están marcados. Si existen nodos sin marcar se crea una nueva lista de nodos de salida para volver al paso 7 de la Parte B de la siguiente forma:
    a. Obtengo todos los nodos que no están marcados del grupo que son de salida
    b. Para cada grupo obtengo la lista de nodos de salida hacia los demás grupos, y con la lista de nodos no marcados verifico si dichos nodos pertenecen a la lista de nodos de salida hacia los grupos correspondientes, si es así , selecciono dicho nodo y paso a evaluar el siguiente grupo (vuelvo a b con el próximo grupo). En caso de no encontrar ningún nodo entre los no marcados en la lista de nodos de salida hacia el grupo destino sigo con el paso c.
    c. Creo un arreglo *arregloOrdenado* donde ordeno de mayor a menor las demanda de los nodos hacia un mismo grupo. Es decir que es un arreglo donde cada posición representa un grupo destino y en cada posición guardo los nodos no marcados sin procesar que tienen demanda hacia ese grupo, ordenados de mayor a menor demanda.
    d. Tomo un nodo no marcado y me fijo su demanda hacia los otros grupos (sea este *nodoNoMarc1*
    e. Para cada grupo con el que *nodoNoMarc1* tiene demanda selecciono usando *arregloOrdenado* los nodos de salida a los que lo voy a conectar. Para cada grupo destino, recorro la lista de nodos de salida asociada a la posición de ese grupo destino. Detengo la recorrida de cada lista, cuando encuentro un nodo de salida que tiene la arista caño asociada con la suficiente holgura para que pase la demanda de *nodoNoMarc1*. La recorrida comienza desde el principio de la lista, por lo que al estar ordenada, siempre trato de hacer pasar los

nuevos nodos por las aristas caño de mayor tecnología posible e ir llenando las holguras. Vuelvo al paso 7 de la Parte B.

18. Vuelvo al paso 1 de la Parte C.
19. Vuelvo a inicializar las holguras de las aristas caño del grupo que estoy iterando para que cuando itere en el próximo grupo las holguras estén completas. Esto se hace porque no se considera que la arista caño sea full duplex.
20. Avanzo la cantidad de grupos y vuelvo al paso 1 de la Parte B.

Paso 7

Calculo el resultado final de la siguiente forma:

i. Recorro la estructura utilizada que fue utilizada en todos los pasos anteriores para guardar los resultados de forma permanente. Llamémosle a esta estructura *Matriz Resultado*, y al valor objetivo como *resultado.*
ii. Inicializo *resultado* en 0
iii. Recorro la *Matriz Resultado* de forma triangular superior ya que los datos están guardados en forma simétrica y en su diagonal los valores son 0.
iv. Para cada celda recorrida de la matriz, calculo el valor objetivo sumándole al *resultado* el costo de la tecnología de la arista * la distancia en transporte de dicha arista) en cada iteración. Dichos valores son obtenidos de la celda correspondiente en *Matriz Resultado.*

Funciones citadas en los pasos del algoritmo

generarCicloConNodosNoSalida:

i. Mientras el tamaño de las listas fusionadas sea mayor que uno o tenga que salir, comparo todas las puntas en la lista contra todas las demás
ii. Busco la menor distancia entre sus extremos y verifico que se pueda realizar la unión.
iii. Si es posible realizar la conexión genero una preunión entre ambas cadenas y verifico que sea posible unir con la cadena de salida generada anteriormente.
iv. Finalmente genero otra preunión entre la cadena resultante anteriormente y la cadena de salida, y verifico si es posible unir sus extremos para estar seguro que se pueda realizar la conexión de todo el ciclo, si esto es positivo guardo las dos cadenas iniciales que probé, y sus distancias mínimas de unión junto con las puntas a unir.
v. Sigo iterando hasta que ya no sea posible seguir uniendo cadenas.
vi. Luego de esa iteración busco en la lista de cadenas generadas cuales cadenas tienen mas nodos no procesados y con esto genero una lista que finalmente

vuelvo a filtrar, probando la unión de cada cadena de la lista con mas nodos no procesados, con la cadena de nodos de salida generada anteriormente en este paso. Verifico todas las combinaciones midiendo la que tiene menor perímetro, y finalmente selecciono esa cadena que tiene menor perímetro al unirla con la cadena de salida y genero la unión real cerrando el ciclo y lo retorno.

vii. Es importante que se le da mas relevancia al hecho de tener mas nodos sin procesar en la cadena, y luego la distancia total del perímetro en el ciclo, como forma heurística elegida, ya que el bucle principal del paso seis se detiene cuando todos los nodos internos están marcados porque fueron usados en algún ciclo.

UnirNodosSalidaRespaldo:

Para cada grupo hago lo siguiente:

i. Recorro todos los indices del arreglo de nodos de salida del grupo y para cada indice ordeno la lista que contiene de menor a mayor tecnología.

ii. Para cada indice del arreglo de nodos de salida del grupo:

1. Clono la lista de nodos de salida del grupo actual, sea esta *listaClonadaSalida*.
2. Recorro los nodos de *listaClonadaSalida* y cargo en *listaMayorTecnología* los nodos de salida que tienen la misma tecnología y es la mayor.
3. Elimino los nodos de *listaMayorTecnologia* de *listaClonadaSalida*.
4. Recorro la lista *listaMayorTecnología* para tratar de unir primero los nodos de mayor tecnología.
5. Creo una punta por cada nodo de *listaMayorTecnología* y las agrego a una lista de puntas (*lista_puntasTemp*).
6. Mientras el tamaño de *lista_puntasTemp* sea mayor que uno y sea posible unir:

   a. Comparo todas las puntas en la lista contra todas las demás y lo hago para cada extremo de cada punta.
   b. Si no hay un camino asignado previamente entre los extremos de las dos puntas que estoy comparando actualmente:
      - Busco para ambos extremos que estoy comparando de cada punta cuales son las aristas caño en común hacia el mismo grupo. De dicha aclaración se desprende que si un extremo de una punta tiene una arista caño hacia un grupo y el otro extremo de la misma punta no tiene aristas caños hacia el mismo grupo, no se considera como arista caño en común.

- Se tiran en el grafo de transporte las aristas caño encontradas en común.
- Se tira Dijkstra en el grafo resultante.
- Si la distancia encontrada entre los extremos evaluados de las puntas a unir es menor que el minimo encontrado hasta el momento y ademas es posible unir las puntas mediante la funcion *esPosibleUnirCadena()* guardo los extremos de las puntas a unir, el camino correspondiente y actualizo el minimo.

c. En el caso de que ya exista un camino previamente asignado y la distancia de dicho camino es menor que el minimo actual, guardo los extremos de las puntas a unir, el camino correspondiente y actualizo el minimo.

d. Luego de probar todas las combinaciones de todos los extremos de cada punta hago lo siguiente:

- Si los extremos que resultaron elegidos para unir las dos puntas candidatas ya tenían un camino asignado previamente, se le asigna ese camino y se hace la unión de las puntas mediante la funcion *unirCadena()*.
- En caso contrario se guarda el camino y la distancia hallada en *resultado*, se hace la unión de las puntas mediante la funcion *unirCadena()* y se la asigna la tecnología a la punta resultante como la mayor tecnología entre todas las aristas caño en común hacia los mismos grupos.

7. Cargo la demanda ocupada en las aristas caño utilizadas por los extremos de las puntas que uní.

iii. Inicializo la demanda ocupada de las aristas caño.

unirCadena:

i. Se crea una nueva cadena para guardar la unión entre cadena1 y cadena2. Llamemosle a dicha cadena *resultado*.

ii. Se cargan en resultado los atributos de la cadena de la siguiente forma:

a. Inicializo la demanda de la cadena hacia los demas grupos como la suma de las demandas hacia los demas grupos que tienen ambas cadenas a unir
b. Cargo *estaEnSalida* en verdadero si alguna de las dos cadenas a unir tiene dicho atributo en verdadero.

c. Incremento un contador del sistema y lo asigno como el identificador de la punta *resultado* y de su matriz correspondiente.
d. Se colocan los dos extremos que no se van a unir de las dos puntas como puntaIzq y puntaDer de la punta resultado. En el caso de que alguna de las dos puntas tenga un solo nodo, entonces se toma dicho nodo como extremo a unir y como extremo de la punta.

iii. Si ambas cadenas tienen un solo elemento y no se le indicó ningun camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.

iv. Si la cadena1 a unir tiene un elemento, la cadena2 mas de uno y la unión se hace mediante el extremo derecho de la cadena2, hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
- La cadena resultado se forma de la siguiente manera:

  Se debe de crear un nodo nuevo colocando (puntader, puntaIzq) en este orden o sea donde sea izq se coloca el der y simetrico. Luego, dicho nodo se coloca al final de cadena2, por lo que el resultado queda: cadena2 + nodo (puntader, puntaizq)

v. Si la cadena1 a unir tiene un elemento, la cadena2 mas de uno y la unión se hace mediante el extremo izquierdo de la cadena2, hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
- La cadena resultado se forma de la siguiente manera:

  Se debe de crear un nodo nuevo colocando (puntaIzq, puntaDer) en este orden. Luego, de dicho nodo colgamos cadena2, por lo que el resultado queda: nodo (puntaIzq, puntaDer) + cadena2

vi. Si la cadena1 a unir tiene mas de un elemento, la cadena2 un solo elemento y la unión se hace mediante el extremo izquierdo de la cadena1, hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta

resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.

- La cadena resultado se forma de la siguiente manera:

  Se debe de crear un nodo nuevo colocando (puntader, puntaIzq) en este orden o sea donde sea izq se coloca el der y simetrico. Luego, de dicho nodo se cuelga cadena1, por lo que el resultado queda: nodo (puntader, puntaizq) + cadena1.

vii. Si la cadena1 a unir tiene más de un elemento, la cadena2 un solo elemento y la unión se hace mediante el extremo derecho de la cadena1, hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
- La cadena resultado se forma de la siguiente manera:

  Se debe de crear un nodo nuevo colocando (puntaIzq, puntaDer) en este orden. Luego, dicho nodo lo colgamos al final de cadena1, por lo que el resultado queda: cadena1 + nodo (puntaIzq, puntaDer)

viii. Si cadena1 y cadena2 tienen más de un elemento y se intenta unir el extremo izquierdo de la cadena1 con el extremo derecho de la cadena2 hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
- La cadena resultado se forma de la siguiente manera:

  Se debe de crear un nodo nuevo colocando (puntader, puntaIzq) en este orden o sea donde sea izq se coloca el der y simetrico. Luego, dicho nodo se coloca al final de cadena2, por lo que el resultado queda: cadena2 + nodo (puntader, puntaizq) + cadena1.

ix. Si cadena1 y cadena2 tienen más de un elemento y se intenta unir el extremo derecho de la cadena1 con el extremo derecho de la cadena2 hacemos lo siguiente:

- Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
- La cadena resultado se forma de la siguiente manera:

Para la unión en este caso se debe de crear un nodo nuevo colocando (puntaizq, puntader) en este orden. Luego simetrizo todos los nodos de la cadena2 , ej: si el nodo es (1,3) lo transformo en (3,1) y finalmente le realizo el inverso total a la lista de esta cadena. Coloco la cadena1 como está, luego coloco el nodo creado y finalmente la cadena2 con el proceso mostrado anteriormente. O sea que queda: cadena1 + nodo (puntaIzq, puntaDer) + Invertida (simetrizada(cadena2))

x. Si cadena1 y cadena2 tienen más de un elemento y se intenta unir el extremo izquierdo de la cadena1 con el extremo izquierdo de la cadena2 hacemos lo siguiente:
   - Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
   - La cadena resultado se forma de la siguiente manera:
     Para la unión en este caso se debe de crear un nodo nuevo colocando (puntaizq, puntader) en este orden. Luego simetrizo todos los nodos de la cadena1 (si el nodo es (1,3) lo transformo en (3,1)) y finalmente le realizo el inverso total a la lista de esta cadena. Por último realizamos la unión de la cadena 1 con el proceso mostrado anteriormente y luego coloco el nodo creado al comienzo de la cadena2 como esta inicialmente. O sea que queda: Invertida (simetrizada(cadena 1)) + nodo (puntaIzq, puntaDer) + cadena2

xi. Si cadena1 y cadena2 tienen mas de un elemento y se intenta unir el extremo derecho de la cadena1 con el extremo izquierdo de la cadena2 hacemos lo siguiente:
   - Si no se le indicó ningún camino para unirlas entonces se busca un camino usando Dijkstra entre los dos extremos a unir, guardandose el camino encontrado con su respectiva distancia en la matriz asociada a la punta resultado. En caso contrario, se guarda en la matriz resultado el camino y la distancia indicadas.
   - La cadena resultado se forma de la siguiente manera:
     Para la unión en este caso se debe de crear un nodo nuevo colocando (puntaizq, puntader) en este orden. Luego coloco dicho nodo al final de la cadena1 y por último ubico la cadena2 al final del nodo creado. O sea que queda: cadena1 + nodo (puntaIzq, puntaDer) + cadena2.

xii. Coloco la punta creada en la lista de puntas del sistema y la matriz asociada a la punta a la lista de matrices del sistema.

preUnión:

Esta función es similar a UnirCadena pero no destruye las cadenas de entrada.
En lugar de devolver la lista de puntas con las dos puntas unidas en la lista general quitando las dos puntas de entrada de la misma y la lista de matrices con la matriz resultado quitando las dos matrices asociadas, solo devuelve una punta unión y su matriz correspondiente.
Además devuelve cuales son los links que utiliza esa punta y su distancia correspondiente.

crearCadena:

i. Creo una punta y a dicha punta le cargo el identificador del nodo como extremos izquierdo y derecho
ii. Se crea el camino de la punta que va a tener al nodo como unico extremo del camino.
iii. Creamos la matriz correspondiente a la punta para que guarde el camino en trasporte de cada arista que se encuentre en el camino de la punta.
iv. Se incrementa un contador y se lo asigna a la matriz y a la punta para poder crear una relación única entre ellas.
v. Se agrega la punta a la lista de puntas del sistema
vi. Se agrega la matriz asociada a la punta, a la lista de matrices del sistema.

cerrarCiclo:

i. Para cada arista que forma parte de la punta a cerrar el ciclo se tiran sus respectivos links en transporte en el grafo de transporte (G)
ii. Se corre Dijkstra en G.
iii. Si no existe un camino entre los extremos de la punta dada significa que no es posible cerrar un ciclo, por lo que se retorna false, de lo contrario se realiza lo siguiente:
    a. Se guarda dicho camino en la matriz asociada a la punta
    b. Se agrega la nueva arista encontrada a la punta uniendo sus extremos y formando un ciclo.
    c. Se retorna true.

esPosibleUnirCadena:

i. Se calcula las demandas de las puntas a unir. Llamémosle a dichas demandas *demanda_punta1* y *demanda_punta2*.
ii. Si la punta1 tiene algún nodo de salida:
    a. Para cada arista de la punta1 verifico si ambos extremos son de salida

b. En dicho caso obtengo la menor holgura de las aristas que forman parte de la punta.
c. Si *demanda_punta1* + *demanda_punta2* > capacidad de la máxima tecnología disponible retorno falso.
d. En caso contrario: Si *demanda_punta1* + *demanda_punta2* > la menor holgura de las aristas que forman parte de la punta retorno falso.

iii. Se hace lo mismo que en el paso ii pero para la punta2.
iv. Si punta1 y punta2 no tienen nodos de salida: Si *demanda_punta1* + *demanda_punta2* > capacidad de la máxima tecnología disponible retorno falso.
v. De lo contrario hago lo siguiente:
   a. Si la suma de las demandas que tiene cada punta hacia el resto de los grupos > la holgura de la arista caño elegida hacia cada grupo retorno falso.
   b. Tiro los links de cada una de las puntas
   c. Ejecuto Dijkstra sobre el grafo con los links tirados.
   d. Si no encuentro un camino entre los dos extremos a unir de las dos puntas retorno falso.
   e. Verifico si se puede cerrar el ciclo. Para ello hacemos lo siguiente:
      - tiro los links del camino hallado en el paso v parte c
      - Si ambas puntas a unir tienen un solo elemento retorno verdadero
      - Si alguna de las dos puntas tiene mas de un elemento y la otra solo uno y ademas puedo cerrar el ciclo retorno verdadero, de lo contrario retorno falso.
      - Si las dos puntas tienen mas de un elemento entonces, si existe un camino entre los extremos que no son los que se van a unir (se hace para ver si puedo cerrar el ciclo) entonces retorno verdadero, en caso contrario retorno falso.

3.3. Comparación de las dos heurísticas implementadas

Como se ha descripto, el problema fue resuelto por dos heurísticas diferentes. Nos parece interesante compararlas, ya que reflejan diferentes momentos en este proyecto de grado, instancias que nos dejaron una gran experiencia en resoluciones de este tipo.
En esta sección nos referiremos a heurística 1 cuando hablemos de la primera solución implementada y a heurística 2 cuando hablemos de la segunda.

Experiencia previa: En primer lugar y como diferencia importante interesa resaltar la familiarización que había con el problema a la hora de implementar las heurísticas. La heurística 1 fue implementada al poco tiempo de comenzar con el proyecto, en ese momento ni siquiera contábamos con los datos del caso real y fundamental a resolver lo cual fue una debilidad al momento de implementar. Por el contrario, la implementación de la heurística 2 fue el producto final del proyecto, es decir que teníamos una gran maduración en el entendimiento del problema.

Fortaleza de los resultados: Vinculado al punto anterior está la calidad de la solución. Nos resulta claro que en problemas que deben ser atacados por métodos heurísticos para hallar buenas soluciones aproximadas, es fundamental algunos aspectos, por un lado entender bien el problema y por otro conocer los datos para ajustar la posible resolución. En la heurística 1 se había entendido el problema y enseguida se comenzó con la implementación, por otra parte no se conocían los datos del caso real a resolver. Además el objetivo de la heurística 1 era proporcionar una cota superior para la solución utilizando Cplex. Por todos estos motivos se obtuvo una solución que descuida aspectos importantes del problema para hallar un buen resultado. En el caso de la heurística 2, la maduración del entendimiento del problema era mayor, por otro lado ya habíamos analizado los datos del problema real. Fue así que obtuvimos una solución más precisa, que ataca el problema en su globalidad y que además tenía el objetivo de ser el resultado final.

Detalle de la solución: A pesar de encontrar una solución peor, la heurística 1 otorga en su solución más datos que la heurística 2, concretamente, además de encontrar una red de datos solución y los ruteos en transporte de cada enlace de datos, la heurística 1 también propone el ruteo de la información en la red de datos para el escenario sin fallas y para cada escenario de falla. Esto se debe a que en un principio pensamos que era necesario pero luego se nos aclaró que no era parte del alcance de este proyecto.

Costo de la solución: Para el juego de datos reales, la heurística 2 encontró una solución de la mitad de precio que la encontrada por la heurística 1.

Complejidad de la solución: Si bien las dos heurísticas plantean ideas bastante accesibles de entender, a la hora de implementarlas presentaron diferencias muy grandes que nos permiten afirmar que la heurística 2 es mucho más compleja que la 1.

Uno de esos aspectos es el **tiempo que nos llevo implementar las soluciones**, a grandes rasgos podemos decir que la heurística 1 llevó la tercera parte de tiempo que la heurística 2 (dos y seis meses). En detalle la diferencia es aun mayor ya que cuando se implemento la heurística 2 se dedicaron más horas por mes.
Una diferencia importante en la complejidad de estos algoritmos, son las **estructuras de datos** que se presentaron en las secciones anteriores, la heurística 1 tiene nueve mientras que la heurística 2 tiene dieciséis.
Como otro indicador de complejidad están las **líneas de código** que para la heurística 1 fueron aproximadamente mil y para la heurística 2 aproximadamente seis mil.
Una gran diferencia en la complejidad, es el **tipo de lógica** que utiliza cada una de las heurísticas, es decir, mientras la heurística 1 utiliza una lógica secuencial y con poco uso de memoria, la heurística 2 resuelve en su implementación, sub problemas que son NP-Hard, con soluciones aproximadas ya conocidas (problema de coloración de grafos y problema del viajero), además utiliza lógica de fuerza bruta que conlleva a una gran demanda de memoria.
A pesar de esta diferencia en la lógica se destaca que el **tiempo de ejecución** de las heurísticas es muy aceptable para el juego de datos real. La heurística 1 encuentra el resultado casi de inmediato y la heurística 2 demora alrededor de doce segundos.

Generalidad: A pesar de que para ambas soluciones se tuvieron que hacer retoques para encontrar una solución al juego de datos real, las soluciones implementadas son generales, es decir, que si se varían los datos de entradas como por ejemplo la red de transporte, las dos heurísticas seguirían funcionando en la generalidad de los casos. Cabe aclarar que la heurística 2 no puede ejecutar con un juego de datos pequeño en sus dimensiones como ya fue mencionado.

3.4. Diseño de algoritmo Branch and Bound

En esta sección se describe un diseño de algoritmo Branch and Bound, que se realizó con el propósito de hallar una solución óptima al problema partiendo del modelo lineal binario que lo representa (especificado en el Anexo I). Una vez completado este diseño, pasamos a investigar como podíamos implementarlo utilizando el software Cplex. Allí nos dimos cuenta que el software Cplex ya contaba con operaciones que implementaban algoritmos Branch and Bound (mas precisamente Branch and Cut, que es una variante de Branch and Bound descrita en el capítulo dos), por lo que optamos por utilizar estas operaciones en lugar de implementar desde cero nuestro diseño de Branch and Bound. La implementación utilizando Cplex se describe en la sección 3.5.

Pasemos ahora a describir el algoritmo Branch and Bound que diseñamos.
A partir de la solución factible hallada con alguna de las soluciones aproximadas (denominada de aquí en más *solAprox*), pensamos tres posibles variantes para hallar el óptimo. Estas variantes utilizan el método Branch and Bound.
Antes de iniciar cualquiera de las variantes aplicamos el método simplex a todo el problema a fin de tener una cota inferior del óptimo a hallar. Denominamos a este valor como ***optCotaInferior***.
Por otro lado definimos ***optCotaSuperior*** como el valor de la función objetivo evaluada en la solución aproximada *solAprox.*
Por lo que la solución a hallar se encontrará entre ***optCotaInferior*** y ***optCotaSuperior*.**

1. Primera variante de Branch and Bound.

   Aplicamos un algoritmo Branch and Bound con las siguientes características:

   Criterio para hallar la cota inferior en cada nodo: método simplex.
   Criterio para seleccionar la variable para la separación: dentro del conjunto de variables que no dieron enteras al aplicar el método simplex, utilizar alguna de las siguientes variantes o combinaciones de ellas.
   - Elegir la variable con la mayor parte fraccionaria
   - Elegir la variable con mayor coeficiente en al función objetivo.
   - Elegir la variable que tenga mayor influencia sobre el resto de las variables. Por ejemplo una arista en transporte influye mucho sobre las otras variables, si hay muchos caminos en datos que la utilizan para el ruteo de la información.
   - Elegir la variable que esta a menor distancia de un valor entero, en este caso cero o uno.
   - Elegir randómicamente.

   Criterio para elegir la rama a recorrer primero: tomamos la rama que le da valor cero a la variable, ya que se trata de un problema de minimización y todos los coeficientes de las variables son positivos (por ser productos de distancia por costo de tecnología). Por lo que si la variable vale cero el valor de la función objetivo será menor que si vale uno.

Criterio para descartar ramas: sea ***optRelajado*** el óptimo solución del problema relajado en un nodo intermedio dado. Si ***optRelajado*** >= ***optCotaSuperior*** se descarta la rama, ya que el mejor valor que se puede lograr en ella es *optRelajado* y es mayor que el valor que ya se tiene con otra solución entera (*optCotaSuperior*).
Si en una hoja dada, se tiene una solución entera con optimo menor que ***optCotaSuperior,*** este optimo pasara a ser la nueva cota superior (es decir se sobrescribe *optCotaSuperior* con este valor).
La solución final es la correspondiente a la última cota superior hallada.

2. Segunda variante de Branch and Bound.

   Aplicamos un algoritmo Branch and Bound utilizando los criterios dados por el algoritmo de Balas (descrito en el capítulo dos). Lo optimizamos utilizando estructuras para manejar dependencias entre variables, a fin de evitar nodos infactibles. Cuando fijamos el valor de una variable, analizamos sus dependencias, seteando las variables que queden definidas, evitando llegar a nodos infactibles. La idea es que al fijar una variable, queden definidas muchas, de forma de poder bajar rápido en el árbol.

3. Tercera variante de Branch and Bound.

   Aplicamos un algoritmo Branch and Bound con las siguientes características:

   a. Realizamos los cambios de variables necesarios para que todos los coeficientes de la función objetivo sean positivos. Ordenamos las variables de acuerdo a los coeficientes de mayor a menor.
   b. Partimos de la hoja correspondiente a la solución inicial aproximada hallada con alguna de las heurísticas (*solAprox*). Para esta hoja el valor de la función objetivo es una cota superior del óptimo a hallar, la definida como ***optCotaSuperior***.
   c. Aplicamos Branch and Bound subiendo niveles desde esta hoja hacia la raíz.

   Para el Branch and Bound se pueden utilizar tanto los criterios definidos para el punto uno, como para el punto dos.
   Notar que la diferencia del punto tres con los puntos uno y dos, es que utilizamos la solución inicial aproximada, no solo para podar ramas, sino también para iniciar el Branch and Bound en la hoja dada por esta solución. Esto lo realizamos considerando que la solución óptima debería estar cercana de la solución inicial, por lo que el algoritmo sería más eficiente ya que descartaría más ramas.

3.5. Solución utilizando Cplex

En esta sección se describe el algoritmo Branch and Cut que se implementó utilizando el software Cplex. Lo denominaremos *"Solución utilizando Cplex"*.

Esta solución esta compuesta por dos etapas:

1. La primera consiste en generar un archivo de datos con el modelo lineal binario que representa el problema a resolver.
2. La segunda consiste en resolver el modelo lineal binario a partir del archivo de datos generado en el paso uno.

La primera etapa de la solución se puede sintetizar en los siguientes pasos:

1. ***Especificar los datos de entrada al problema en un archivo***: se utiliza el mismo archivo de entrada descrito para la primera solución aproximada. Recordar que en este archivo se especifica la red de transporte (nodos, enlaces, distancias de los enlaces), la red de datos (nodos, demandas entre nodos en escenarios con y sin falla), la función tns (correspondencia de nodos de la red de datos con nodos de la red de transporte) y las tecnologías disponibles para los enlaces de la red de datos (capacidad y costo de cada una).

2. ***Cargar las estructuras necesarias para generar el archivo de entrada para Cplex***: se procesa el archivo de entrada generado en el paso uno, y a partir de los datos leídos, se cargan las estructuras que se utilizan para guardar la información de los grafos, necesaria para generar el archivo de entrada para Cplex. Estas estructuras son: *nt_incidentes*, *arista_transporte_nodos*, *nd_incidentes* y *arista_datos_nodos*, las cuales se explican más adelante en la sección 3.5.1.2.

3. ***Generar archivo de entrada para Cplex***: a partir de las estructuras cargadas en el paso dos y del modelo lineal binario que representa el problema a resolver, se genera el archivo de entrada para Cplex. El archivo de entrada para Cplex no es otra cosa que el modelo lineal binario escrito en un formato que Cplex entiende. Dentro de los posibles formatos de archivo de entrada para Cplex, se eligió el formato denominado *lp*, por considerarlo de fácil interpretación, ya que es muy similar al modelo matemático. A continuación se incluye un ejemplo de un archivo con formato *lp* para un modelo cualquiera.

El siguiente es un ejemplo de archivo para describir un modelo utilizando el formato *lp* de Cplex. Primero se indica si hay que minimizar o maximizar la función objetivo (en el ejemplo es el segundo caso). Luego se escribe la función objetivo de manera similar a como se hace con símbolos matemáticos (previamente se incluye el prefijo *obj* seguido de dos puntos). A continuación se escribe *Subjet To* para indicar que empiezan las restricciones, las cuales se escriben de la misma forma que la función objetivo, cada una precedida por un identificador incremental (en el ejemplo es *ci* donde *i* es el numero de la restricción). Posteriormente viene la sección *Bounds* que es opcional y se utiliza para indicar las variables que están acotadas inferior o superiormente o ambas. La sección *General* también es opcional y se utiliza para indicar las variables que son enteras (en el caso del ejemplo solo la x4 es entera). También se pude incluir una sección *Binary* para indicar las variables que son binarias (esta sección se utiliza en los archivos generados en este proyecto de grado, ya que como se ha mencionado anteriormente, el modelo que representa nuestro problema es lineal binario).
Tanto la sección *General* como la sección *Binary*, en caso de estar presentes, deben ir a continuación de la sección *Bounds*, si es que esta existe, en caso contrario deben ir a continuación de la sección *Subjet To*. Por último se termina el archivo escribiendo *End* [8].

```
Maximize
 obj: x1 + 2 x2 + 3 x3 + x4
Subject To
 c1: - x1 + x2 + x3 + 10 x4 <= 20
 c2: x1 - 3 x2 + x3 <= 30
 c3: x2 - 3.5 x4 = 0
Bounds
 0 <= x1 <= 40
 2 <= x4 <= 3
General
 x4
End
```

La segunda etapa de la solución consiste en resolver el modelo lineal binario dado por el archivo *lp* (generado en la primera etapa), mediante un algoritmo Branch and Cut. Este algoritmo Branch and Cut ya se encuentra implementado en el software Cplex (más precisamente en la clase *IloCplex*), y se puede controlar la recorrida del árbol mediante el seteo de parámetros.

Se utiliza el método solve de la clase *IloCplex*, aplicado al objeto que contiene el modelo. Como se trata de un MIP, el método *solve*() utiliza un procedimiento Branch and Cut para resolverlo, el cual puede sintetizarse de la siguiente manera [8]:

Cplex resuelve la relajación de varios subproblemas. Para manejar estos subproblemas eficientemente construye un árbol donde cada subproblema es un nodo. La raíz del árbol es la relajación del MIP original. Si la solución de esta relajación tiene una o mas variables fraccionarias

Cplex tratará de encontrar cortes. Los cortes son restricciones que eliminan áreas de la región factible del problema relajado que contienen soluciones fraccionarias. Si la solución del subproblema relajado tiene una o más variables enteras con valor fraccional, Cplex vuelve a ramificar en una de las variables que tiene valor fraccional, generando dos nuevos subproblemas, cada uno con una restricción más en la variable elegida para ramificar. En este caso que se trata de variables binarias, un nodo fijara la variable en cero y el otro en uno.
La solución del problema relajado en cada nodo puede resultar en una solución con todas las variables cero o uno, una solución infactible u otra solución fraccional. Si la solución es fraccional Cplex repite el procedimiento.

Se fija el parámetro *CutUp* con el costo resultado de la primera solución aproximada descrita en este capitulo. Con esto se logra que se descarten los nodos que tengan un óptimo del problema relajado superior al valor de *CutUp*. Es decir no seguir recorriendo el árbol, en nodos que el mejor caso, darán una solución entera cuyo costo esta por encima del costo de la solución entera que ya se ha calculado.

Se utiliza el parámetro *TiLim* para controlar la finalización del algoritmo, este parámetro se carga con la cantidad de segundos máximos de ejecución. Si el algoritmo termina antes de *TiLim* segundos, devuelve la solución óptima hallada. Si pasaron *TiLim* segundos y no terminó, finaliza la ejecución y devuelve la mejor solución encontrada hasta el momento.

En las secciones siguientes, 3.5.1 y 3.5.2 se dan más detalles sobre la primera y segunda etapa de esta solución respectivamente.

3.5.1. Primera etapa: generar archivo de datos con el modelo

3.5.1.1. Organización en módulos

La solución se encuentra organizada de manera muy similar a la primera solución aproximada, en los siguientes tres módulos: ***Grafo***, ***LeerArchivo*** y ***Escritura***.

El módulo ***Grafo*** contiene estructuras para manipular los grafos de datos y transporte y las demandas entre los nodos del grafo de datos. También contiene operaciones para trabajar con estas estructuras. No tiene dependencias con los otros módulos.

El módulo ***LeerArchivo*** contiene operaciones para leer un archivo con los datos de entrada al problema y utilizarlos para cargar las estructuras. Se relaciona con el módulo Grafo para el acceso a estas estructuras.
Recordar que los datos de entrada son: red de transporte (nodos, enlaces, distancias de los enlaces), red de datos (nodos, demandas entre nodos en escenarios con y sin falla), función tns (correspondencia de nodos de la red de datos con nodos de la red de transporte) y tecnologías disponibles para los enlaces de la red de datos (capacidad y costo de cada una).
El formato del archivo de entrada es el mismo que se describió para la primera solución aproximada en la sección 3.1.3.

El módulo ***Escritura*** tiene como cometido generar un archivo con formato *lp*, para el modelo lineal binario que representa el problema. El archivo se genera a partir de las estructuras del módulo Grafo (cargadas con operaciones del módulo LeerArchivo), y del modelo lineal binario que representa el problema a resolver (descrito en el Anexo I). Por lo mencionado anteriormente este módulo se relaciona con los módulos Grafo y LeerArchivo.
Este archivo en formato *lp* será utilizado por Cplex para resolver el modelo, en la segunda parte de esta solución.

En la figura 3.5.1.1.1 se encuentra un diagrama que muestra las dependencias entre los módulos descritos anteriormente.

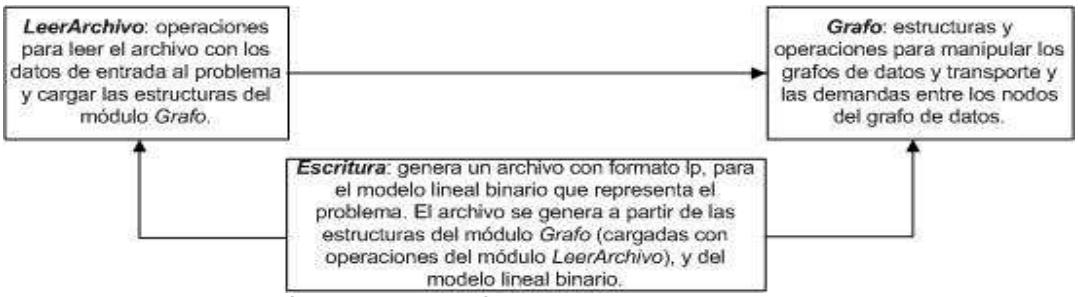

Figura 3.5.1.1.1: Módulos de la solución utilizando Cplex y como se relacionan entre si.

3.5.1.2. Estructuras de datos más importantes

Las siguientes son las estructuras de datos más importantes que se utilizaron para esta primera parte de la solución:

- Estructura ***nt_incidentes***.
- Estructura ***arista_transporte_nodos***.
- Estructura ***nd_incidentes***.
- Estructura ***arista_datos_nodos***.

A continuación se describe cada una de ellas:

- Estructura ***nt_incidentes***: es un arreglo de listas de aristas. Para cada nodo del grafo de transporte se guarda la lista de las aristas incidentes en él. El índice del arreglo corresponde al identificador del nodo.

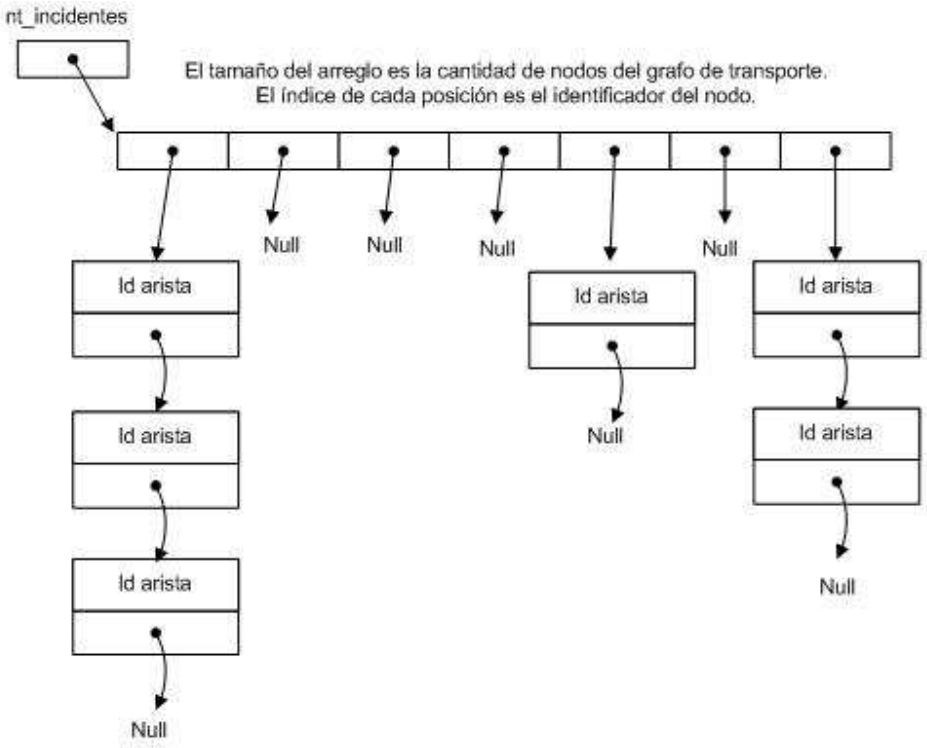

Figura 3.5.1.2.1: Descripción de la estructura *nt_incidentes* de la Solución utilizando Cplex.

- Estructura ***arista_transporte_nodos***: es un arreglo de pares de enteros. Para cada arista del grafo de transporte se guardan los *id* de los nodos que la forman. El índice del arreglo corresponde al identificador de la arista.

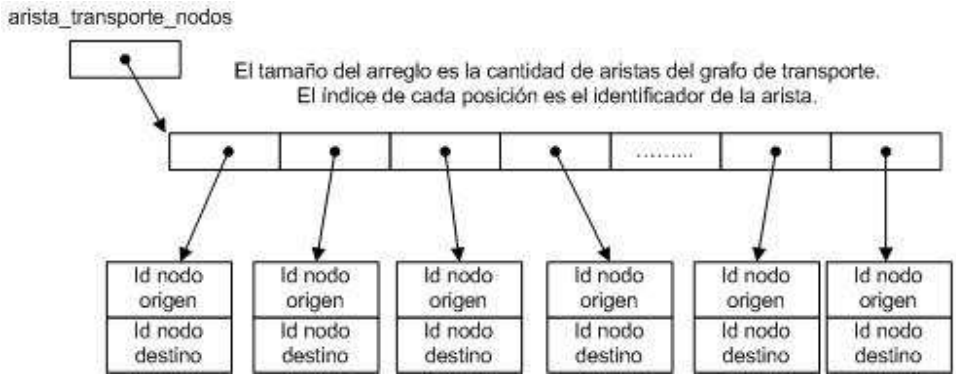

Figura 3.5.1.2.2: Descripción de la estructura *arista_transporte_nodos* de la Solución utilizando Cplex.

- Estructura ***nd_incidentes***: es un arreglo de listas de aristas. Para cada nodo del grafo de datos se guarda la lista de las aristas incidentes en él. El índice del arreglo corresponde al identificador del nodo. Análoga a la estructura *nt_incidentes* pero para el grafo de datos.

- Estructura ***arista_datos_nodos***: es un arreglo de pares de enteros. Para cada arista del grafo de datos se guardan los *id* de los nodos que la forman. El índice del arreglo corresponde al identificador de la arista. Análoga a la estructura *arista_transporte_nodos* pero para el grafo de datos.

3.5.2. Segunda etapa: resolver el modelo

Esta solución se basa en el ejemplo *ilolpex2.cpp* que viene incluido en el software Cplex. Este ejemplo toma un archivo de entrada con el modelo que se quiere resolver (primer parámetro) e invoca a las operaciones de la clase *IloCplex* para resolverlo mediante un algoritmo Branch and Cut. Mediante una letra (segundo parámetro) permite seleccionar el algoritmo que se utilizara para hallar una cota inferior en cada nodo.

La solución utilizando Cplex lee el problema de optimización de un archivo y lo resuelve con una opción de optimización específica. El archivo que contiene el problema y la opción de optimización se pasan al programa como parámetros en la invocación.

Ejemplo de invocación:

> ***solCplex_resuelvoModelo.exe*** *modelo.lp* *p*

En caso se esta pidiendo resolver el problema de optimización descrito en el archivo *modelo.lp* y la opción de optimización es la *p*, que indica que se utiliza el algoritmo simplex primal para evaluar la cota inferior del Branch and Cut en cada nodo.

A continuación se incluyen los pasos más importantes de esta solución [8].

1. Lee el modelo de un archivo: El modelo se crea leyendo el archivo especificado como primer argumento de la línea de comando. Esto se hace usando el método *importModel* aplicado a un objeto de la clase *IloCplex* (*objeto Cplex*). Los objetos *obj*, *var* y *rng* se pasan a *importModel* para que posteriormente las variables del modelo puedan ser accedidas. La invocación a *importModel* no extrae el modelo en el objeto Cplex, esto se hace mas tarde con la invocación a *Cplex.extract*(*model*).

2. Selecciona un optimizador: El optimizador se elige con el segundo argumento ingresado en la línea de comando.
   La invocación *setParam*(*IloCplex::RootAlg*, *alg*), *alg* selecciona el algoritmo deseado, uno de la clase *IloCplex::Algorithm.*

   Las siguientes son las opciones de optimizador que se pueden elegir:

   Letra *o*: Algoritmo *IloCplex::AutoAlgm*, *IloCplex* selecciona automáticamente un algoritmo.
   Letra *p*: Algoritmo *IloCplex::Primal*, *IloCplex* selecciona el algoritmo simplex primal.
   Letra *d*: Algoritmo *IloCplex::Dual*, *IloCplex* selecciona el algoritmo simplex dual.
   Letra *b*: Algoritmo *IloCplex::Barrier*, *IloCplex* selecciona un algoritmo de barrera.

Letra *n*: Algoritmo *IloCplex::Concurrent*, *IloCplex* selecciona varios algoritmos concurrentemente.

Para la ejecución de nuestros casos de prueba elegimos el algoritmo simplex primal (letra *p*), ya que se puede utilizar para problemas lineales y estamos familiarizados con su procedimiento de resolución.

3. Resolver el modelo: Se utiliza el método solve de la clase *IloCplex*, aplicado al objeto que contiene el modelo. En este caso *Cplex.solve*(). Como se trata de un MIP el método *solve*() utiliza un procedimiento Branch and Cut para resolverlo. Al comienzo de la sección 3.5 se dan mas detalles sobre como opera este algoritmo Branch and Cut.

4. Accede a la información básica: Acceder a los resultados, imprimir el valor de cada variable.

5. Calcula medidas de calidad: Finalmente el programa imprime la máxima infactibilidad primal de la solución. Esto se debe a la precisión finita de los cálculos numéricos que se realizan en la computadora, Cplex permite cierta tolerancia que podría permitir que alguna condición de óptimalidad no se cumpla.

En la figura 14.1 del Anexo IX, se incluye una imagen de un ejemplo de archivo de salida de esta solución, y posteriormente se describe la información contenida en el mismo.

3.6. Ambiente de desarrollo y ejecución del software

3.6.1. Ambiente de desarrollo

Como se mencionó en las secciones anteriores se implementaron dos heurísticas para obtener una solución aproximada al problema. La primera de ellas un algoritmo sencillo que buscaba lograr una solución aproximada de forma rápida. En la segunda se buscó refinar el procedimiento de cálculo de manera de lograr una solución más cercana al optimó. Estas dos soluciones fueron implementadas utilizando la plataforma de desarrollo Eclipse, versión 3.3.2, con el compilador de C/C++ MinGW 5.1.6 y el debugger gdb 6.6.

El algoritmo Branch and Cut se implementó utilizando las operaciones del software Cplex. La versión de Cplex utilizada fue ILOG Cplex 12.2. Este algoritmo consta básicamente de dos partes, la primera consiste en generar un archivo con el modelo lineal binario correspondiente al problema a resolver. Este archivo es entrada de la segunda parte, que resuelve el modelo descrito en el mismo con un algoritmo Branch and Cut, que utiliza las operaciones de Cplex. La primera parte se implementó utilizando la misma plataforma que para las soluciones aproximada. La segunda se implementó con la plataforma Microsoft Visual Studio 2008 Professional, ya que presentaba una integración fácil con el software Cplex.

3.6.2. Ejecución del software

Para ejecutar la heurística 1 o la 2 se debe invocar el ejecutable correspondiente, heuristica1.exe o heuristica2.exe respectivamente, teniendo en cuenta que el archivo con los datos de entrada se encuentre en el mismo directorio y que tenga el nombre ENTRADA.txt. El formato del archivo de entrada se encuentra en la sección 3.1.3. Se debe tener en cuenta que el formato es poco flexible en cuanto a la cantidad de espacios y saltos de línea que debe haber entre cada sección, se recomienda partir de uno de los archivos de entrada proporcionados. La ejecución de estas heurísticas genera el archivo SALIDA.txt, con los formatos que fueron descritos en las secciones 3.1.3 y 3.2.3.

La primera parte de la solución utilizando Cplex se ejecuta de la misma manera, se invoca el ejecutable solCplex_generoModelo.exe teniendo en cuenta que el archivo con los datos de entrada se encuentre en el mismo directorio y tenga el nombre ENTRADA.txt. Esta ejecución genera el archivo modelo.lp, el cuál se pasa como entrada al segundo ejecutable de esta solución solCplex_resuelvoModelo.exe, el cual se invoca desde línea de comando de la siguiente manera:

> ***solCplex_resuelvoModelo.exe*** *modelo.lp* *p*

El significado del parámetro p se detalla en la sección 3.5.2.

4. Casos de prueba y sus resultados

4.1. Breve descripción de los casos de prueba

En el siguiente cuadro se indican las características básicas de cada caso de prueba ejecutado. Posteriormente se describe cada uno detalladamente.

Nro. de caso de prueba	**Cant. nodos grafo transporte**	**Cant. links grafo transporte**	**Cant. tecnologías**	**Cant. nodos grafo datos**	**Cant. demandas entre nodos grafo datos** (1)	**Cant. filas matriz problema lineal** (2)	**Cant. columnas matriz problema lineal** (2)	**Cant. entradas no nulas en matriz prob. lineal** (2)
1	5	6	3	5	6	1258	856	3752
2	7	8	4	7	8	4159	2772	13242
3	9	13	8	9	4	14635	7084	41774
4 (3)								

Cuadro 4.1.1: Características básicas de los casos de prueba ejecutados.

(1) Es la cantidad de entradas distintas de cero de la matriz de demandas.
(2) Estas cantidades corresponden al problema lineal reducido por Cplex, el cual elimina filas o columnas redundantes.
(3) El caso de prueba 4, corresponde a los datos brindados por ANTEL, por motivos de confidencialidad no se pueden presentar sus valores.

4.2. Resumen de resultados

A continuación detallamos las dos PCs y el servidor utilizados para las pruebas:

a. PC: Pentium 4 CPU 3.00 GHz, con 1,24 GB de memoria RAM.
b. PC: Intel Core 2 duo 2.00 GHz, con 3 GB de memoria RAM.
c. Servidor: Intel Core i7-975, 16 GB (8M Cache, 3.33 GHz, 6.40 GT/s Intel QPI). Este servidor se encuentra en la Facultad de Ingeniería.

En el cuadro 4.2.1 se presentan los resultados obtenidos luego de la ejecución de cada una de las soluciones implementadas, con cada caso de prueba. A la derecha de cada resultado se indica la PC o servidor se utilizo para la ejecución.

Nro. de caso de prueba	**Resultado solución aproximada 1** (1)	**Resultado solución aproximada 2**	**Tiempo ejecución sol. aprox. 2 (en seg.)**	**Resultado solución Cplex** (2)	**Tiempo ejecución Cplex (en seg.)**	**¿Cplex terminó la recorrida del árbol?**
1	2795 (a)	No aplica	-	1365 (a)	15	SI
2	314800 (a)	No aplica	-	194300 (c)	854.73	SI
3	756800 (a)	652500 (b)	< 1	488400 (c)	4814.24	NO (3)
4			12	(4)	(4)	(4)

Cuadro 4.2.1: Resumen de los resultados obtenidos para cada solución, con cada uno de los casos de prueba ejecutados.

(1) El tiempo de ejecución de la solución aproximada 1 para todos los casos de prueba es instantáneo.

(2) Para todos los casos de prueba la invocación a la solución utilizando Cplex se realizó con la opción p, para que utilice el algoritmo simplex primal en la evaluación de la cota inferior del Branch and Cut en cada nodo (ver ejemplo de invocación en la sección 3.5.2.1).
Al principio se realizaron varias ejecuciones de esta solución en la PC a), que tardaron para los casos de prueba 2 y 3, 72 horas continuas de ejecución, sin lograr la finalización de la recorrida del árbol. Para estas ejecuciones se fijó como cota superior del valor de la función objetivo, los resultados de la solución aproximada 1. Posteriormente surgió la posibilidad de realizar pruebas en el servidor c) de la facultad, donde para el caso de prueba 2 se logro la solución óptima, y para el caso de prueba 3 se logró una solución factible mejor que la obtenida con la PC a), en un tiempo mucho menor (80 minutos).

(3) Para estos casos el resultado de Cplex corresponde a la mejor solución encontrada luego de cumplir un tiempo de ejecución fijado (el indicado en la columna Tiempo ejecución Cplex). Para esto se utilizan los parámetros ClockType y TiLim de la clase IloCplex, los cuales se cargan utilizando la función setParam de la misma clase. El parámetro ClockType se carga con valor dos para indicar que el tiempo se mide de acuerdo a un “reloj de pared” y el parámetro TiLim se carga con la cantidad de segundos máxima de ejecución.

(4) El caso de prueba 4, corresponde a los datos brindados por ANTEL, por motivos de confidencialidad no se pueden presentar sus valores. Dadas las grandes dimensiones del mismo no se logró iniciar la ejecución del Branch and Cut que realiza Cplex, por lo que no se tiene un resultado para esta solución.
La dimensión del archivo de entrada con el modelo para este caso es de 404 MB y el algoritmo que lo genera (primera parte de la "Solución utilizando Cplex") tardó seis días en terminar su ejecución.
Como ya se mencionó, la segunda parte de la "Solución utilizando Cplex" correspondiente al algoritmo Branch and Cut no logra iniciar su ejecución ya que se obtienen el mensaje de error de Cplex correspondiente a que se ha quedado sin memoria "Cplex Error 1001: Out of memory.".
Se probaron variaciones de los siguientes parámetros de Cplex, obteniendo siempre el mismo mensaje de error: MemoryEmphasis (para conservar memoria donde sea posible), WorkDir y NodeFileInd (para grabar los nodos en disco), TreLim (máximo tamaño del árbol en MB), WorkMem y ReInv (cota superior en megabytes para la cantidad de memoria central que Cplex use antes de hacer swapping) y NodeSel (para seleccionar la estrategia con la que se elijen los nodos a recorrer) y VarSel (para seleccionar la variable con la que se ramifica).
En cuanto al parámetro NodeSel se probó sin éxito con la estrategia búsqueda primero en profundidad (DFS, depth-first search, en ingles) ya que se considera que requiere menos memoria que otras opciones, porque recorre el árbol a través de una rama hasta llegar a una hoja por lo que no genera una lista muy grande de nodos no explorados, como si lo hacen otras opciones que saltan en su recorrida de una rama a otra.
En cuanto al parámetro VarSel se probó sin éxito con el valor tres para que elija en cada nodo la mejor variable para ramificar. Esta opción tiene como desventaja que requiere mas esfuerzo computacional en cada nodo para determinar la mejor variable. Como ventaja se tiene que genera menos nodos y por tanto consume menos memoria.
Por otra parte se intentó compilar la aplicación en un sistema operativo y hardware de 64 bits, para así tener un mayor direccionamiento de memoria disponible. Esto no fue posible ya que si bien se contaba con el hardware y sistema operativo de 64 bits, no se pudo conseguir la versión del software ILOG Cplex con las librerías para 64 bits.
Las pruebas se realizaron tanto en la PC a) como en el servidor c) detallados al inicio de esta sección.

4.3. Porcentaje de reducción del costo total entre las distintas soluciones

Nro. de caso de prueba	Porcentaje de reducción del costo total entre sol. aprox. 1 y sol. aprox. 2	Porcentaje de reducción del costo total entre sol. aprox. 2 y sol. Cplex	Porcentaje de reducción del costo total entre sol. aprox. 1 y sol. Cplex
1	-	-	51,16 %
2	-	-	38,28 %
3	13,78 %	25,15%	35,47 %
4	49,22 %	-	-

Cuadro 4.3.1: Porcentajes de reducción del costo total entre las distintas soluciones.

En el cuadro 4.3.1 se muestran para cada caso de prueba, los porcentajes de reducción del costo total entre las distintas soluciones implementadas. Las celdas con un guión (-) indican que no se puede efectuar la comparación, ya que la ejecución de alguna de las soluciones involucradas en la misma, no aplicó para ese caso de prueba.
El porcentaje de reducción se calculó de la siguiente manera: supongamos que para un caso de prueba cualquiera el resultado de la solX es valor_x y el resultado de la solY es valor_y, donde solX y solY son cualquiera de las soluciones implementadas. Supongamos que valor_x es mayor que valor_y y queremos calcular cuanto se redujo el costo.
El porcentaje de reducción del costo total es igual a ((valor_x – valor_y)*100/valor_x). Es decir calculamos cuál es la proporción del costo total disminuido, respecto al mayor de los dos valores.

En la primera columna se presenta el porcentaje de reducción del costo total entre la solución aproximada uno y la solución aproximada dos. Estos porcentajes se calcularon para los casos de prueba tres y cuatro, ya que para los casos uno y dos la ejecución de la heurística dos no aplica.
Para el caso de prueba tres la reducción fue del 13,78% y para el caso de prueba cuatro la reducción fue bastante mayor alcanzando el 49,22%.

En la segunda columna se presenta el porcentaje de reducción del costo total entre al solución aproximada dos y la solución utilizando Cplex. Este porcentaje se calculó únicamente para el caso de prueba tres ya que es el único que cuenta con los dos resultados. Se obtuvo un porcentaje de reducción del 25,15%.

En la tercera columna se presenta el porcentaje de reducción del costo total entre la solución aproximada uno y la solución utilizando Cplex. Estos porcentajes se calcularon para todos los casos de prueba a excepción del cuatro, para el que no se tiene un resultado de la solución utilizando Cplex. Para el caso de prueba uno se obtuvo un porcentaje del 51,16 %, para el dos del 38,28 % y para el tres del 35,47 %.

4.4. Descripción detallada de los casos de prueba y sus resultados

4.4.1. Caso de prueba 1

4.4.1.1. Descripción

- **Grafo de transporte:** compuesto por 5 nodos y 6 links, con las distancias que se muestran en la siguiente figura.

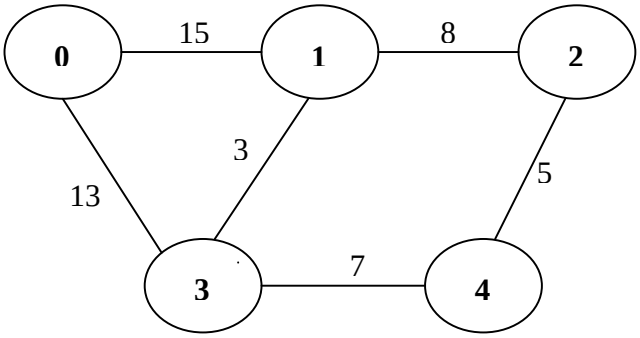

Figura 4.4.1.1.1: Grafo de transporte del caso de prueba 1.

- **Grafo de datos:** compuesto por 5 clientes, con las demandas que se muestran en la siguiente figura. Las demandas son las mismas para todos los escenarios de falla.

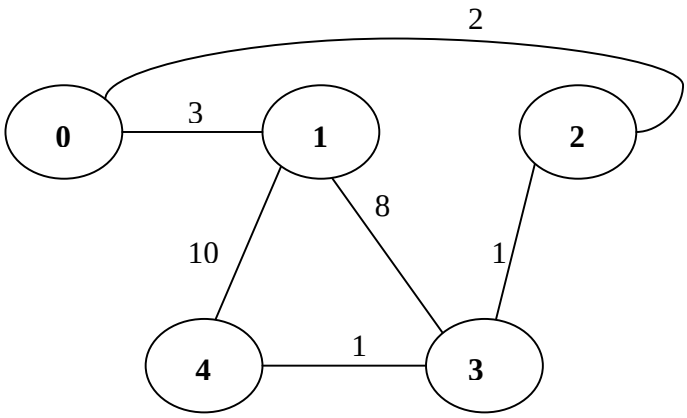

Figura 4.4.1.1.2: Grafo de datos del caso de prueba 1.

- **TNS:** para todo cliente i se cumple que su correspondiente nodo en el grafo de transporte es el que tiene la misma numeración, es decir que $TNS(i) = i \quad \forall i = 0..4$

- **Tecnologías:** se tienen dos tecnologías, con las características que se muestran en el cuadro a continuación. A los efectos del algoritmo se agrega una tecnología de capacidad 0 y costo 0 que se asigna a las aristas que no forman parte del grafo resultado.

Tecnología	Capacidad	Costo
1	8	20
2	25	35

Cuadro 4.4.1.1.3: Tecnologías del caso de prueba 1.

En el Anexo III se incluye el archivo de entrada para los algoritmos, correspondiente a este caso de prueba.

4.4.1.2. Resultados

- **Resultado solución aproximada 1:** se obtuvo un costo total de 2795. En el Anexo III se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

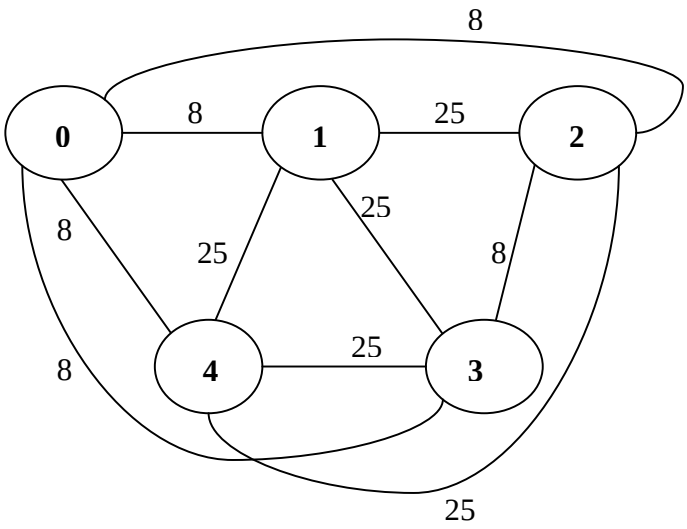

Figura 4.4.1.2.1: Grafo de datos resultado de la solución aproximada 1 para el caso de prueba 1.

- **Resultado solución aproximada 2**: No aplica por tratarse de un ejemplo muy pequeño para ejecutar la heurística correspondiente.

- **Resultado Cplex**: obtuvo un costo total de 1365. Correctamente este costo total es menor que el obtenido con la primera solución aproximada (se reduce en un 51%). En el Anexo III se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

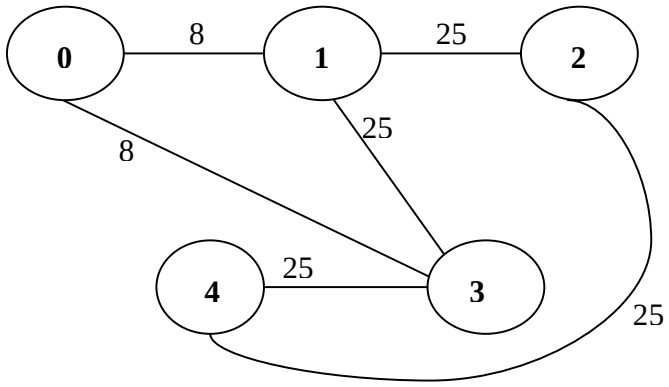

Figura 4.4.1.2.2: Grafo de datos resultado de la solución utilizando Cplex para el caso de prueba 1.

4.4.2. Caso de prueba 2

4.4.2.1. Descripción

- **Grafo de transporte:** compuesto por 7 nodos y 8 links, con las distancias que se muestran en la siguiente figura.

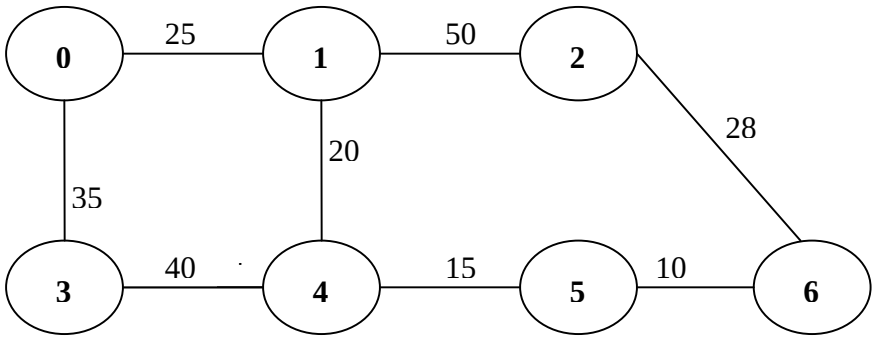

Figura 4.4.2.1.1: Grafo de transporte para el caso de prueba 2.

- **Grafo de datos:** compuesto por 7 clientes, con las demandas que se muestran en la siguiente figura. Las demandas son las mismas para todos los escenarios de falla.

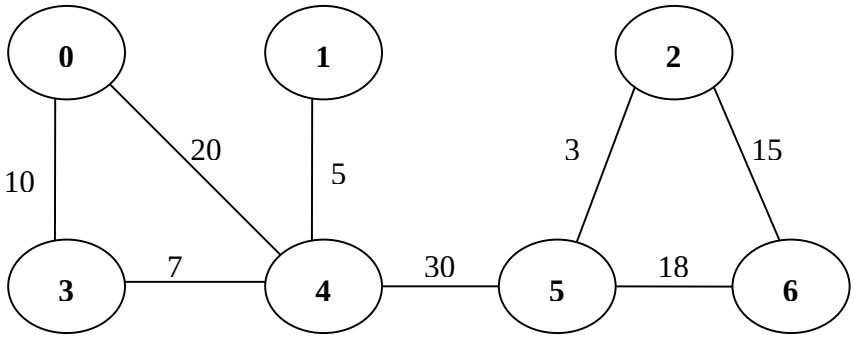

Figura 4.4.2.1.2: Grafo de datos para el caso de prueba 2.

- **TNS:** para todo cliente i se cumple que su correspondiente nodo en el grafo de transporte es el que tiene la misma numeración, es decir que $TNS(i) = i \quad \forall i = 0..6$

- **Tecnologías:** se tienen tres tecnologías, con las características que se muestran en el cuadro a continuación. A los efectos del algoritmo se agrega una tecnología de capacidad 0 y costo 0 que se asigna a las aristas que no forman parte del grafo resultado.

Tecnología	Capacidad	Costo
1	20	500
2	40	800
3	80	1100

Cuadro 4.4.2.1.3: Tecnologías del caso de prueba 2.

En el Anexo IV se incluye el archivo de entrada para los algoritmos, correspondiente a este caso de prueba.

4.4.2.2. Resultados

- **Resultado solución aproximada 1**: se obtuvo un costo total de 314800. En el Anexo IV se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

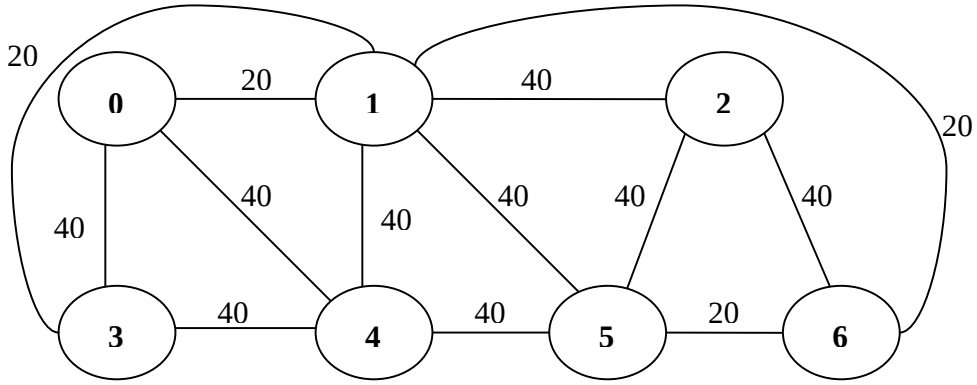

Figura 4.4.2.2.1: Grafo de datos resultado de la solución aproximada 1 para el caso de prueba 2.

- **Resultado solución aproximada 2**: No aplica por tratarse de un ejemplo muy pequeño para ejecutar la heurística correspondiente.

- **Resultado Cplex**: se obtuvo un costo total de 194300. Correctamente este costo total es menor que el obtenido con la primera solución aproximada (se reduce en un 38,3%). En el Anexo IV se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

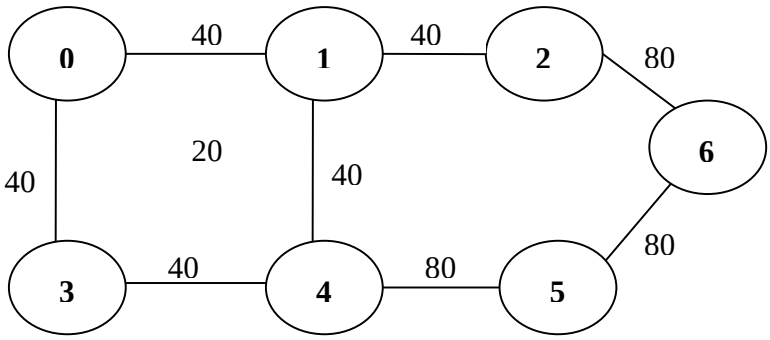

Figura 4.4.2.2.2: Grafo de datos resultado de la solución utilizando Cplex para el caso de prueba 2.

4.4.3. Caso de prueba 3

4.4.3.1. Descripción

- **Grafo de transporte:** compuesto por 9 nodos y 13 links, con las distancias que se muestran en la siguiente figura.

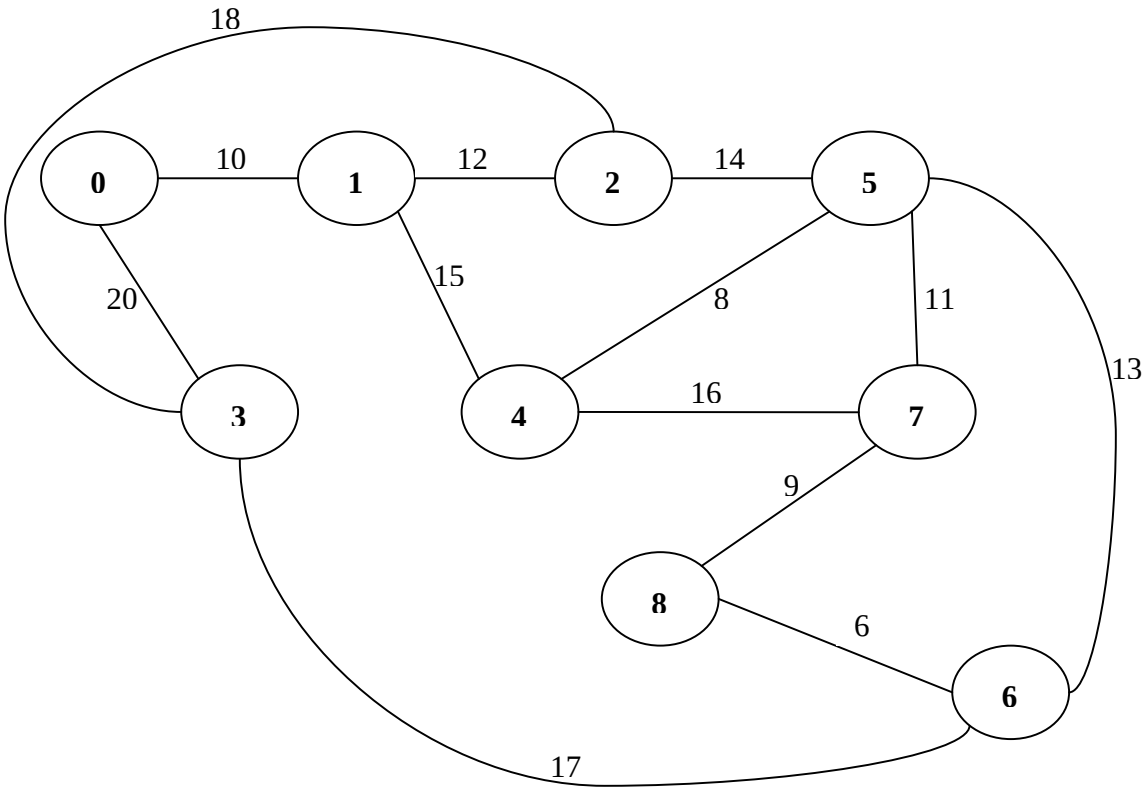

Figura 4.4.3.1.1: Grafo de transporte del caso de prueba 3.

- **Grafo de datos:** compuesto por 9 clientes, con las demandas que se muestran en la siguiente figura. Las demandas son las mismas para todos los escenarios de falla.

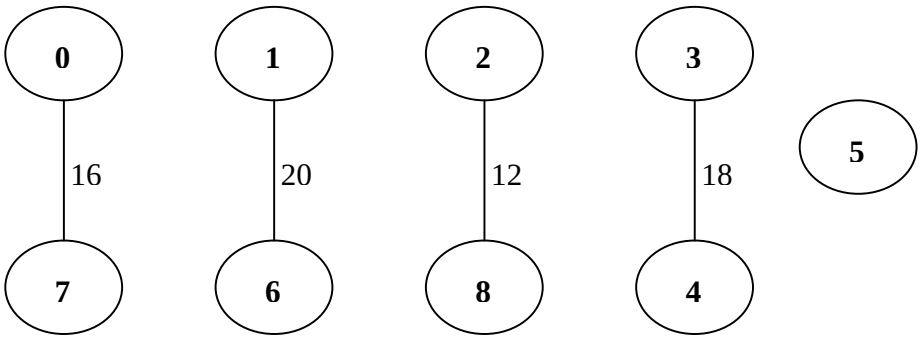

Figura 4.4.3.1.2: Grafo de datos del caso de prueba 3.

- **TNS:** para cada cliente i se cumple que su correspondiente nodo en el grafo de transporte es el que tiene la misma numeración, es decir que $TNS(i) = i \quad \forall i = 0..8$

- **Tecnologías:** se tienen siete tecnologías, con las siguientes características. A los efectos del algoritmo se agrega una tecnología de capacidad 0 y costo 0 que se asigna a las aristas que no forman parte del grafo resultado.

Tecnología	Capacidad	Costo
1	1	100
2	2	180
3	5	450
4	10	800
5	20	1600
6	40	3000
7	60	3500

Cuadro 4.4.3.1.3: Tecnologías del caso de prueba 3.

En el Anexo V se incluye el archivo de entrada para los algoritmos, correspondiente a este caso de prueba.

4.4.3.2. Resultados

- **Resultado solución aproximada 1**: se obtuvo un costo total de 756800. En el Anexo V se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

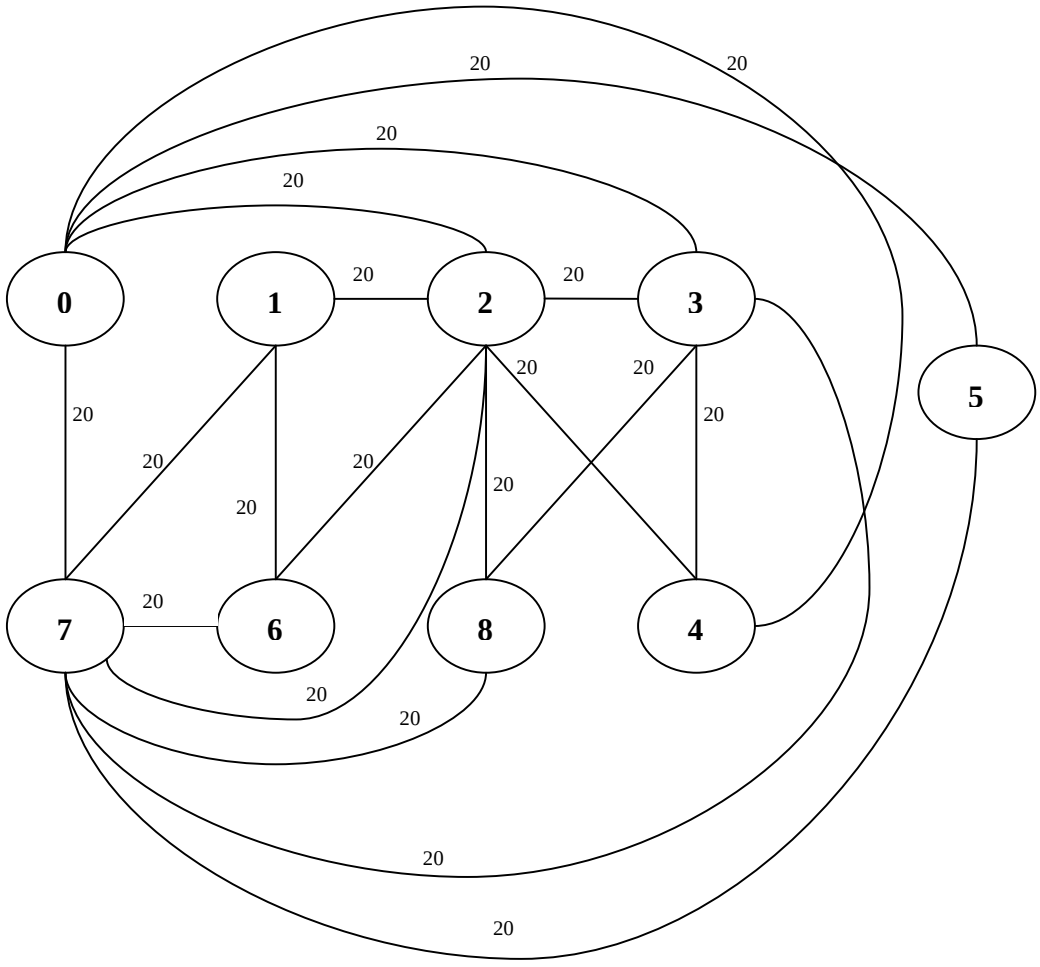

Figura 4.4.3.2.1: Grafo de datos resultado de la solución aproximada 1 para el caso de prueba 3.

- **Resultado solución aproximada 2:** se obtuvo un costo total de 652500. En el Anexo V se incluye el archivo de salida completo. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

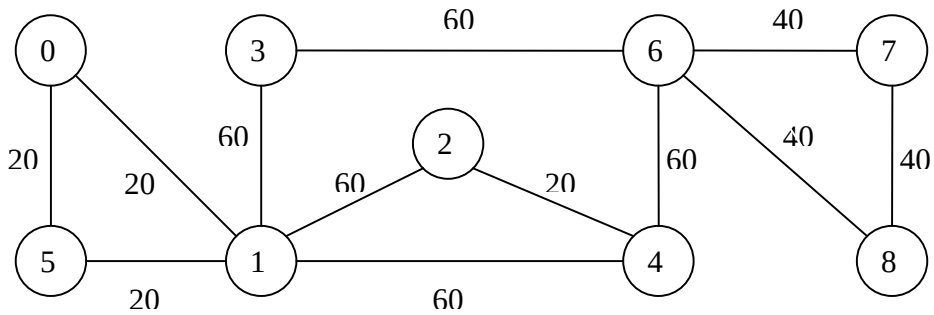

Figura 4.4.3.2.2: Grafo de datos resultado de la solución aproximada 2 para el caso de prueba 3.

- **Resultado Cplex**: se obtuvo un costo total de 488400. En el Anexo V se incluyen los archivos de salida de cada ejecución. A continuación se encuentra un diagrama con el grafo de datos resultado (para cada arista se indica la capacidad de la tecnología asignada).

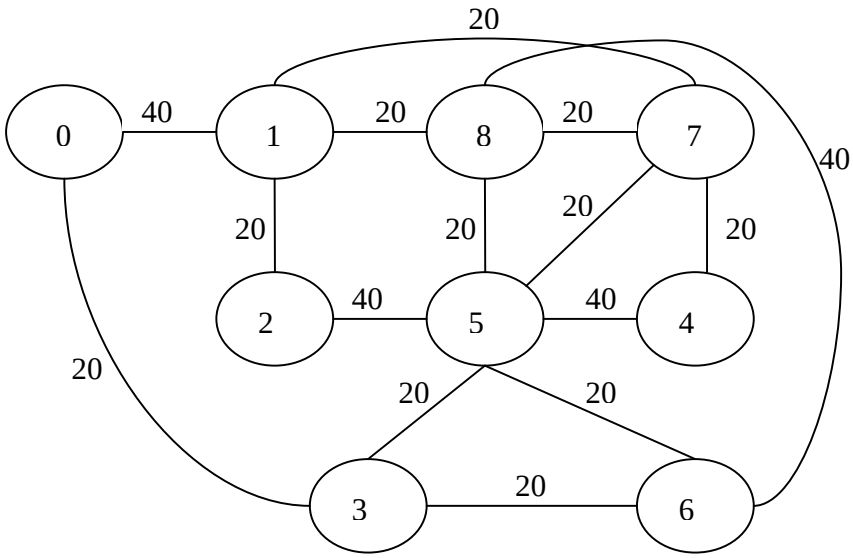

Figura 4.4.3.2.3: Grafo de datos resultado de la solución utilizando Cplex para el caso de prueba 3.

4.4.4. Caso de prueba 4

Este caso de prueba corresponde al problema real a solucionar por lo cual representa nuestro foco de atención y uno de los motivos fundamentales para el desarrollo de este proyecto. Por motivos de confidencialidad no se pueden presentar sus valores.

4.5. *Verificación de los resultados esperados*

En esta sección se incluyen las pruebas realizadas para los sistemas construidos.
Cabe mencionar en términos generales que el nivel de pruebas en la etapa de desarrollo fue bajo y que el esfuerzo de verificación se volcó a pruebas funcionales una vez implementadas las herramientas. La excepción fue la solución utilizando Cplex que por contener la transcripción de un modelo de programación lineal, necesitó reiteradas revisiones para garantizar que el modelo era codificado de forma correcta.

4.5.1. Heurística 2

Como se mencionó las pruebas fuertes fueron de carácter funcional, es decir que para la salida que el software otorga, se reviso: que la red encontrada tuviera la forma y características planteadas en el seudocódigo de la heurística, que la red resultado fuera factible, es decir, que permita las demandas especificadas y que sea robusta respecto a una falla simple y que el costo de la solución estuviera bien calculado. Se utilizaron los casos de prueba tres y cuatro descriptos anteriormente en este capítulo, los casos uno y dos no se utilizaron por ser ejemplos muy pequeños para la aplicación de esta heurística.

Para verificar que el algoritmo se comportaba como habíamos diseñado comparamos cada paso del seudocódigo con lo que el programa iba construyendo al ejecutar paso a paso. Estás pruebas fueron de vital importancia ya que además de encontrar errores de correctitud, es decir, algunas equivocaciones de implementación respecto a lo especificado, se encontraron particularidades en los casos de prueba que obligaron decisiones de rediseño (se explica en la sección 3.2) para poder aplicar la idea.

Una vez que la heurística pudo ejecutar para los dos casos de prueba y actuar de la manera esperada, pasamos a revisar que la solución encontrada fuera factible. Aunque este análisis se había realizado en etapa de diseño, es decir, que la construcción era en base a obtener una solución factible, es la verificación más importante. Se debía repetir para cubrir cualquier equivocación posible.
La lista a verificar fue la siguiente:

- Ver que los caminos en transporte indicados para cada arista en datos, existen y que la distancia de los mismos corresponde con la indicada por el software.
- Ver que la red de datos es capaz de transportar todas las demandas entre los nodos, es decir, que la capacidad de las aristas no es desbordada y que existe un camino en la red de datos para cada demanda.
- Ver que el punto anterior se cumple en cualquier escenario de falla simple.
- Ver que el costo de la solución sea bien calculado.

Para el primer ítem se reviso cada uno de los caminos indicados en la solución, tomando como ayuda un diagrama de la red de transporte para comprobar que los caminos existían y que además medían lo que el algoritmo indicaba.

Por motivos de confidencialidad no se presenta una tabla con las aristas de la red de datos afectadas ante la caída de cada link de transporte, así como tampoco un ejemplo de como queda la red de datos ante la caída de un link de transporte en particular que influye en muchas aristas.

4.5.2. Solución utilizando Cplex

Para la solución utilizando Cplex se revisó en primera instancia que el modelo generado para ser resuelto por Cplex, coincidiera con el modelo que nos pasaron los tutores. Es decir que estuviera correctamente implementado, el programa que toma los datos del caso de prueba (a partir de un archivo de entrada) y genera un archivo con la función objetivo y restricciones del modelo, el cual es entrada para el programa que lo resuelve utilizando el algoritmo Branch and Cut de Cplex.

En segunda instancia se revisó la salida del Branch and Cut de Cplex para cada cada caso de prueba ejecutado (casos de prueba uno a tres). Para la mejor solucion encontrada (en los casos de prueba uno y dos fue la optima), la salida incluye los valores asignados a cada variable (un ejemplo de salida se puede ver en el Anexo IX). Se revisaron las variables que definen la red de datos solución (aristas incluídas y tecnología asignada a cada una) y los caminos en transporte para cada arista de la misma. A partir de estas variables construímos en papel la red de datos resultado, anotando para cada arista la tecnología asignada y el camino en transporte correspondiente. Por otra parte anotamos también los ruteos en la red de datos para el escenario sin falla y para cada escenario de falla. Se plantea como una posible extensión a este proyecto de grado, el contar con un programa que dibuje toda esta información (red de datos resultado, caminos en transporte, ruteos en la red de datos) a partir de los valores de las variables, ya que esto permitiría visualizar rapidamente los resultados, faciliando su comprensión y verificación.

A partir de los resultados realizamos dos verificaciones:

1. Revisamos que las tecnologías asigndas a las aristas de la red de datos soportaran las demandas requeridas entre los nodos, para el escenario sin fallas y para cada escenario de falla.
2. Por otro lado verificamos que el cálculo de la función objetivo fuera correcto, para esto realizamos el cálculo manual a partir de los resultados obtenidos y lo comparamos con el resultado dado por Cplex.

En una primera instancia realizamos ejecuciones en la PC a) indicada en la sección 4.2. En estas pruebas se dejó corriendo el programa primero 12 horas y luego 72 horas, no logrando finalizar la recorrida del arbol para los casos de prueba dos y tres. De todas formas fijando el parámetro de tiempo límite de ejecución se logró obtener la mejor solución hasta el momento.
Verifcamos estas soluciones y para los casos de prueba dos y tres (donde no termino la recorrida del arbol) fallaban las verificaciones uno y dos indicadas anteriormente. Para el caso de prueba uno ambas verificaciones dieron correctas.
Por sospechar pudiera tratarse de un problema con al PC a), el cotutuor nos ofreció la posibilidad de correr el programa (para los casos de prueba dos y tres), en el servidor de la facultad (su especificación se encuentra en la sección 4.2, opción c)). Tras una primera

ejecución en este servidor, para los dos casos la verificacion uno dio correcta pero la verificación dos no dio correcta. Se corrigió el error que provocaba la falla de la verificación dos y se pidió al cotutor que probara nuevamente el programa con estos casos de prueba. Tras esta segunda ejecución en el servidor, las verificaciones uno y dos dieron correctas, por lo que podemos afirmar que se obtuvieron soluciones factibles en ambos casos y también verificar que la solución utilizando Cplex es funcionalmente correcta.

5. Conclusiones y trabajo a futuro

5.1. Evaluación de resultados alcanzados

La evaluación de los resultados alcanzados es ampliamente satisfactoria ya que se logró resolver el problema real con dos soluciones propias. Por otro lado se investigó el software Cplex desarrollando una solución que resuelve nuestro problema de forma exacta, para casos de prueba de dimensiones pequeñas.

En un principio comenzamos estudiando métodos heurísticos para resolver nuestro problema pero nos encontramos con la dificultad de que al no tener experiencia en los mismos, la curva de aprendizaje era muy elevada. Fue así que nos enfocamos en resolver el problema con un método exacto, concretamente, Branch and Bound que tomara como cota superior para el costo de la solución, un valor aproximado calculado con una heurística sencilla.
Se logró implementar la heurística sencilla y el algoritmo exacto, pero nos encontramos con el inconveniente de que este último no podía iniciar su ejecución para el problema real por sus dimensiones.

En ese momento se cambio el foco de resolución dejando el algoritmo exacto y apostando a mejorar la implementación de la heurística sencilla para lograr una buena solución aproximada.
Por haber sido implementada con el fin de ser una cota superior para el algoritmo exacto, la heurística no contemplaba muchos aspectos importantes a minimizar para lograr una buena solución. Por tal razón se confeccionó un nuevo diseño para la implementación de una nueva heurística.

A pesar de muchas dificultades de implementación se logró una segunda heurística que resuelve el caso real en un tiempo de computo aceptable (aproximadamente doce segundos) y con un costo de solución de la mitad del encontrado por la primera heurística.

Como se puede ver, hubo dificultades importantes y varios cambios en los enfoques de resolución del problema. Por eso es que consideramos satisfactorios los resultados alcanzados.

5.2. Posibles extensiones al trabajo

- Dibujar el grafo de datos resultado de las diferentes soluciones utilizando alguna biblioteca gráfica.
- Profundizar en el conocimiento de los distintos tipos de cortes que puede generar Cplex en su implementación de Branch and Cut. Investigar los parámetros a utilizar para controlar más eficientemente la recorrida del árbol.
- Conseguir la versión de ILOG Cplex para 64 bits y compilar la solución utilizando Cplex en 64 bits. Volver a probar el caso de prueba 4 con este nuevo ejecutable y verificar si se sigue obteniendo o no el mensaje de Cplex indicando que se ha quedado sin memoria.
- Realizar análisis de sensibilidad aplicado por ejemplo a los costos o a las capacidades de las tecnologías, para ver cuanto se pueden variar los mismos sin afectar el resultado de la función objetivo. Lo mismo si se agrega algún nodo en la red de datos, si se modifica alguna demanda o varía algún nodo o link de la red de transporte.
- Optimizar el modelo lineal matemático, entrada de la solución utilizando Cplex. Reducir restricciones y variables logrando un modelo equivalente.
- Como simplificación en el problema a resolver en este Proyecto de Grado se considera un único tráfico suma del tráfico comprometido y eventual para todo para de nodos de la red de datos, por lo que una posible extensión es considerar el tráfico comprometido y eventual por separado.
- Como simplificación en el problema a resolver en este Proyecto de Grado, se considera que el factor de calidad z_Q vale 1 para todos los casos, por lo que una posible extensión es considerar que este factor pueda tener un valor distinto de 1.
- Estudiar posibles optimizaciones para la heurística 2 que permitan disminuir el costo de la solución.
- Como una posible optimización para la heurística 2 se plantea evaluar la conveniencia de minimizar la cantidad de grupos (paso dos del algoritmo).
- Comparar los resultados obtenidos con los logrados por otro equipo de trabajo.

- Realizar una interfaz que permita pasar la solución obtenida con alguno de las heurísticas (solución aproximada 1 o 2) a la solución utilizando Cplex para que inicie la recorrida del algoritmo Branch and Cut desde la hoja del árbol correspondiente a esta solución inicial. De esta forma lograr que el algoritmo Branch and Cut tarde menos tiempo en terminar. En el siguiente diagrama se ilustra en negro la comunicación que se buscaría lograr entre las soluciones:

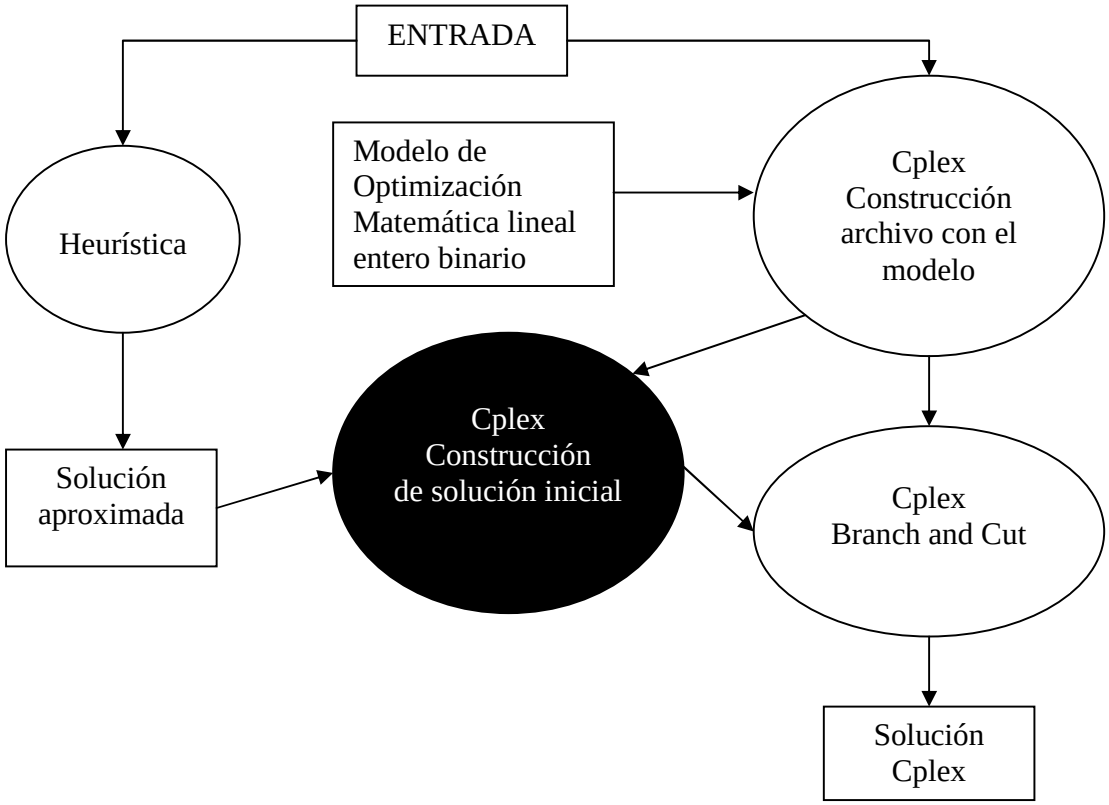

Figura 5.2.1: Diagrama ilustrando las distintas soluciones implementadas (con sus entradas y salidas) y como se relacionan entre si, destacando en negro la interfaz que se plantea como trabajo a futuro.

- Una vez hecha la interfaz que permita a raíz de la solución hallada por las heurísticas inicializar una solución en formato Cplex, agregar instrucciones para indicar si la solución cargada es factible o no y de esta manera se obtendría una forma de probar la factibilidad de las soluciones halladas. Además de esta forma se establece un estándar para la salida de cualquier heurística que intente resolver el problema, sabiendo que si se sigue el estándar se puede chequear el trabajo realizado.

5.3. Conclusiones

Transcurrido un año y medio del comienzo de este proyecto, nos quedan varias conclusiones.

En lo que refiere al tipo de problema resuelto, entendemos que es de alta complejidad por varios motivos. Uno de ellos es que se trata de un problema real y por tal razón, se deben manipular datos y complejidad reales y fijos. Otro aspecto es la complejidad computacional que corresponde al tipo de problemas NP-completo. También resultó difícil atacar el problema por la dificultad que presenta aplicar los métodos y estrategias de resolución existentes, a nuestro caso particular. Relacionado a lo anterior y como complejidad fundamental aparece la poca experiencia del equipo en la aplicación de métodos aproximados.

Como fue mencionado, los resultados obtenidos fueron de nuestra satisfacción dado que al comparar las dos soluciones aproximadas implementadas, la segunda logra una solución con un costo de la mitad que el de la primera. Sin embargo resta comparar la solución encontrada con otra hallada por otro equipo de trabajo.
Se considera valioso tener formas de comparar las soluciones encontradas por las heurísticas así como formas de validar la factibilidad de las mismas. Nos resultó de gran utilidad tener el problema resuelto de forma exacta para caso de pruebas pequeños, con la solución utilizando Cplex, para comparar con los resultados de las heurísticas.

Nos sentimos satisfechos con el trabajo realizado, sin embargo la finalización tardó más de lo previsto. Esto se debió a diversos factores, algunos inherentes a la complejidad del problema, la demora en el acceso a la información de Antel debido a la confidencialidad de los datos, los diversos cambios de estrategia para resolverlo y a la falta de experiencia del grupo en la resolución de proyectos de esta envergadura.

6. Referencias

[1] Claudio Risso, Franco Robledo.
Presentación *"Diseño óptimo de Redes Multioverlay Robustas"*.
Laboratorio de Probabilidad y Estadística - Facultad de Ingeniería – UDELAR
Febrero de 2008

[2] Graciela Ferreira.
Material del curso *"Extensiones de Programación Lineal y Entera"*.
También nos brindó asesoramiento en cuanto a estrategias para resolver el problema.
Facultad de Ingeniería – UDELAR
Junio de 2009

[3] Sebastián Laborde, Sebastián Reís, Álvaro Rivoir.
Proyecto de grado *"Diseño de Topologías de Red Confiables"*
Departamento de Investigación Operativa - Instituto de Computación
Facultad de Ingeniería - UDELAR
2006

[4] Cecilia Gonzáles, Javier Regusci, Sebastián Santos
Proyecto de grado *"Generación de Prescripciones para la Aplicación de Insumos en la Producción Agrícola"*
Facultad de Ingeniería - UDELAR.
Abril de 2008

[5] Adrián Delfino, Sebastián Rivero, Marcelo San Martín
Proyecto Final de Carrera "Ingeniería de Tráfico en Redes MPLS"
Instituto de Ingeniería Eléctrica - Facultad de Ingeniería - UDELAR
Agosto 2005

[6] Carleton University Universidad Capital de Canadá
http://www.sce.carleton.ca/dept/index.shtml
Algoritmo Aditivo de Balas
http://www.sce.carleton.ca/faculty/chinneck/po/Chapter13.pdf
Ultima fecha de ingreso 30 de mayo de 2010.

[7] Wikipedia la enciclopedia libre
www.wikipedia.org.
Redes Overlay – Problema TSP – Problema de coloración de grafos.
Ultima fecha de ingreso 9 de octubre de 2010.

[8] Documentación ILOG CPLEX 10.1
Se instala junto con el software Cplex.
Solución utilizando Cplex.
Ultima fecha de ingreso 01 de agosto de 2010.

[9] Georgia Tech
www.gatech.edu
Problema TSP
http://www.tsp.gatech.edu/
http://www.tsp.gatech.edu/methods/papers/index.html
Ultima fecha de ingreso 9 de octubre de 2010.

[10] Universidad de Illinois
www.illinois.edu
Problema TSP
http://valis.cs.uiuc.edu/~sariel/research/CG/applets/tsp/TspAlg.html
Ultima fecha de ingreso 9 de octubre de 2010.

[11] Libros de Google
www.books.google.com.uy
"The Traveling Salesman Problem. A Computacional Study"
http://books.google.com.uy/books?id=nmF4rVNJMVsC&printsec=frontcover&dq=Traveling+salesman+problem&source=bl&ots=7eD0srDlMh&sig=FYYRuIUqpqdpsnGIheX3YFsEtbs&hl=es&ei=CruwTOnMKIT48AaOg8mdCQ&sa=X&oi=book_result&ct=result&resnum=5&ved=0CEMQ6AEwBA#v=onepage&q&f=false
Ultima fecha de ingreso 9 de octubre de 2010.

[12] Documentos de Google
www.docs.google.com
"Coloración de Grafos"
http://docs.google.com/viewer?a=v&q=cache:BdXEhyh6phAJ:www-ma4.upc.edu/~fiol/grafs/Coloracion.pdf+coloracion+de+grafos&hl=es&gl=uy&pid=bl&srcid=ADGEESgD-i5VZrpaiK_KpEg2ec6gcwLZhepBmKoM74yKYFTA_eUtTGjinU7Ftcqimbes9ZTeVbTVF90Ik21aIZ1kakzOIjDupL06oXfJX_Hj0-MS_CYP6Zdbt_0IuE-ay2RlJul3vEMg&sig=AHIEtbSF91vUtFZokgeYnwalvoOdBaVE3A
Ultima fecha de ingreso 9 de octubre de 2010.

[13] Portal de enseñanza Algovidea
www.algovidea.cl
Coloración de Grafos
http://www.algovidea.cl/index.php?option=com_content&view=article&id=52&Itemid=55
Ultima fecha de ingreso 9 de octubre de 2010.

[14] Universidad de Buenos Aires
www.uba.ar
Coloración de Grafos (Solución exponencial exacta)
http://www-2.dc.uba.ar/materias/aed3/2010-02/Documents/algo3_coloreo.pdf
Ultima fecha de ingreso 25 de octubre de 2010.

[15] Universidad Nacional de Trujillo
http://www.seccperu.org
Coloración de Grafos (demostración NP-Hard y algoritmos)
http://www.seccperu.org/files/paper.pdf
Ultima fecha de ingreso 25 de octubre de 2010.

[16] Universidad de Málaga
http://www.uma.es/
Coloración de Grafos (algoritmo voráz)
http://www.matap.uma.es/profesor/magalan/MatDis/material/GrafosTema5_2_MatDiscreta.pdf
Ultima fecha de ingreso 25 de octubre de 2010.

[17] Universidad central de Venezuela
http://www.ucv.ve/
TSP (Traveling Salesperson Problem)
http://ccg.ciens.ucv.ve/~ernesto/nds/CotoND200302.pdf
Ultima fecha de ingreso 25 de octubre de 2010.

7. Anexo I

Modelo de Optimización Matemática lineal que representa el problema planteado

MODELO DE PROGRAMACIÓN ENTERA BINARIA PARA OPTMIZACIÓN DE REDES MULTI-OVERLAY

CARLOS TESTURI-CECILIA PARODI-EDUARDO CANALE-FRANCO ROBLEDO

1. Datos del Problema

- $G_D = \langle V_D, E_D \rangle$ y $G_T = \langle V_T, E_T \rangle$: dos grafos simples que modelan la Red de Datos y Red de Transporte respectivamente.
- En G_D existen 2 tipos de nodos de datos $V_D = V_A \cup V_E$ donde se cumple $V_A \cap V_E = \emptyset$, que corresponden a los *access nodes* y *edge nodes* respectivamente.
- Existe un conjunto V_F de nodos fijos, que deben estar presentes en cualquier red planificada por ser éstos donde termina (se inicia) el tráfico. Todo el resto de los nodos $V_S = V_D \backslash V_F$ son opcionales o de *Steiner*. Es claro que los nodos de acceso forman parte de los fijos ($V_A \subseteq V_F$), ya que la existencia de los primeros sólo se justifica en la medida que haya tráfico para terminar.
- Siendo $V_D = \{v_1, v_2, \ldots, v_{h-1}, v_h\}$ ($|V_D| = h$) existe $\bar{M} = [m_{i,j}]_{1 \leq i,j \leq h}$ una matriz de vectores de tráfico entre estos nodos, donde $m_{i,j} = \{(\dot{m}_{i,j}, \ddot{m}_{i,j}) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+, \forall v_i, v_j \in V_D\}$. $\dot{m}_{i,j}$ representa el tráfico a garantizar entre v_i y v_j mientras que $\ddot{m}_{i,j}$ corresponde al tráfico de pico. Supondremos que $\dot{m}_{i,j} = \dot{m}_{j,i}$ y $\ddot{m}_{i,j} = \ddot{m}_{j,i}$. Es claro que $m_{i,j} \neq \vec{0}$ solo cuando $v_i, v_j \in V_F$.
- Cada link $(i,j) \in E_D$ ($(i,j) = v_i v_j$) debe ser dimensionado con una capacidad a determinar perteneciente al conjunto $\hat{B} = \{\hat{b}_0, \hat{b}_1, \hat{b}_2, \ldots, \hat{b}_{\bar{B}}\}$, $0 = b_0 < b_1 < \cdots < b_{\bar{B}}$ donde b_0 corresponde conceptualmente al caso en que se elige no incluir en la solución a las aristas dimensionadas con este valor. $\hat{B}$ es un dato conocido en el problema. El dimensionamiento de los links implica la existencia de una función de asignación de capacidades $b : E_D \rightarrow \hat{B}$, denotaremos $b_{i,j} = b(i,j)$. Como parte de la solución al problema hay que determinar $B = \{b_{i,j}, \forall (i,j) \in E_D\}$.
- Sea $V_T = \{t_1, t_2, \ldots, t_{k-1}, t_k\}$ ($|V_T| = k$). Asumiremos que la red física consta de un conjunto de estaciones *(Network Stations)* donde en cada estación hay exactamente un nodo de V_T. Los nodos V_D pueden o no estar presentes en una estación. Sea entonces $ns : (V_D \bigcup V_T) \times (V_D \bigcup V_T) \rightarrow \{0,1\}$, donde $ns(u,v) = 1 \Leftrightarrow$ tanto u como v están en la misma estación. Es obvio que $ns(u,v) = 0 \, \forall \, u, v \in V_T$ y que $\forall v \in V_D$, $\exists t \in V_T \, / \, ns(v,t) = 1$. De las observaciones se desprende que está bien definida la función $tns : V_D \rightarrow V_T \, / \, tns(v) = t$ cuando $ns(v,t) = 1$.
- Deberá existir una función $f_{G_D} : E_T \rightarrow E_D^*$, donde $f_{G_D}(l), l \in E_T$ retorna el conjunto de links de la red de datos que se ven afectados por la falla del enlace l de la red de transporte.

Date: 28 de Marzo de 2009.

2. Problema a Atacar

En el Modelo de Programación Entera a presentar, se asume que se ha resuelto vía pre-procesamiento las conexiones acceso-edge, por lo tanto se asume que cada nodo de acceso ha sido conectado a un nodo edge. Introducimos la siguiente notación:

- $\hat{V}_D$ es el conjunto de nodos edge luego de realizada la fase de asignación acceso-edge.
- $\bar{M}^0 = \{(\dot{m}^0_{ij}, \ddot{m}^0_{ij})\}_{i,j \in \hat{V}_D}$ es la matriz de flujos punto a punto entre nodos de $\hat{V}_D$ en el escenario sin fallas.
- $\bar{M}^f = \{(\dot{m}^f_{ij}, \ddot{m}^f_{ij})\}_{i,j \in \hat{V}_D}$ es la matriz de flujos punto a punto entre nodos de $\hat{V}_D$ en el escenario de falla $f \in E_T$.
- $\hat{E}_D = E_D(\hat{V}_D)$ conjunto de conexiones factibles de la red de datos inducidas por el conjunto $\hat{V}_D$.
- $G_M = (\hat{V}_D, E_M)$ grafo de demandas en el escenario sin fallas: $ij \in E_M$ sii $(\dot{m}^0_{ij}, \ddot{m}^0_{ij}) \neq (0,0)$.
- $G_{Mf} = (\hat{V}_D, E_{Mf})$ grafo de demandas en el escenario de falla f: $ij \in E_{Mf}$ sii $(\dot{m}^f_{ij}, \ddot{m}^f_{ij}) \neq (0,0)$.

El problema a atacar será el de diseñar una red de datos $\mathcal{H} \subseteq \hat{G}_D = (\hat{V}_D, \hat{E}_D)$ tal que:

- cubra $\hat{V}_D$,
- el dimensionamiento de sus enlaces soporte la matriz $\bar{M}^0$ cuando no hay fallas en el transporte,
- el dimensionamiento de sus enlaces soporte la matriz $\bar{M}^e$ cuando falla el enlace $e \in E_T$ en la red de transporte,
- para cada link $(i,j) \in \mathcal{H}$ existe en G_T un camino entre $tns(i)$ y $tns(j)$,

todo esto a costo mínimo.

3. Variables del Problema

- $x^{ef}_g = \begin{cases} 1, & \text{Si el link } g \in \hat{E}_D \text{ en la Red de Datos es utilizado para} \\ & \text{enviar datos entre los vértices de } e \in E_{Mf} \text{ en el escenario } f \in E_T; \\ 0, & \text{Si no.} \end{cases}$
- $x^e_g = \begin{cases} 1, & \text{Si el link } g \in \hat{E}_D \text{ en la Red de Datos es utilizado para} \\ & \text{enviar datos entre los vértices de } e \in E_M \text{ en el escenario sin falla;} \\ 0, & \text{Si no.} \end{cases}$
- $y^e_g = \begin{cases} 1, & \text{Si el link } g \in E_T \text{ en la Red de Transporte es} \\ & \text{utilizado para el ruteo de } e \in \hat{E}_D; \\ 0, & \text{Si no.} \end{cases}$
- $w^t_e = \begin{cases} 1, & \text{Si } e \in \hat{E}_D \text{ utiliza la capacidad } \hat{b}_t \in \hat{B}; \\ 0, & \text{Si no.} \end{cases}$
- $\eta^{et}_g = y^e_g \cdot w^t_e$.

Aclaración:

- $\sim$ = "incidente a";
- e_1 (e_2) primer (segundo)vértice de e.

4. Formulación Completa

/ * Antigua función objetivo: * /

$$\text{(OPT\_MPLS) min} \sum_{e \in \hat{E}_D} \sum_{g \in E_T} \sum_{\hat{b}_t \in \hat{B}} r_T(g) \cdot y_g^e \cdot w_e^t \cdot T(\hat{b}_t)$$

/ * Nueva función objetivo: * /

$$\text{(OPT\_MPLS) min} \sum_{e \in \hat{E}_D} \sum_{g \in E_T} \sum_{\hat{b}_t \in \hat{B}} r_T(g) \cdot T(\hat{b}_t) \cdot \eta_g^{et} \tag{4.1}$$

s.a.

/ * Linealización de los productos $y_g^e \cdot w_e^t$ por medio de las η_g^{et} * /

$$y_g^e + w_e^t - 1 \leq \eta_g^{et}, \qquad \forall e \in \hat{E}_D, \forall g \in E_T, \ \forall \hat{b}_t \in \hat{B}, \tag{4.2}$$

$$\eta_g^{et} \leq y_g^e, \qquad \forall e \in \hat{E}_D, \forall g \in E_T, \ \forall \hat{b}_t \in \hat{B}, \tag{4.3}$$

$$\eta_g^{et} \leq w_e^t, \qquad \forall e \in \hat{E}_D, \forall g \in E_T, \ \forall \hat{b}_t \in \hat{B}, \tag{4.4}$$

/ * Toda arista $e \in \hat{E}_D$ tiene a lo sumo una tecnología asignada * /

$$\sum_{\hat{b}_t \in \hat{B}} w_e^t \leq 1, \forall e \in \hat{E}_D, \tag{4.5}$$

/ * Existirá el link e ($\chi_e = 1$) en la red solución sii se le asignó una tecnología * /

$$\chi_e = \sum_{\hat{b}_t \in \hat{B}} w_e^t, \forall e \in \hat{E}_D, \tag{4.6}$$

/ * Existirá un camino en G_T asociado a cada link de datos en la solución * /

$$\sum_{f \sim tns(e_1)} y_f^e = \chi_e, \quad \forall e \in E_D \tag{4.7}$$

$$\sum_{f \sim tns(e_2)} y_f^e = \chi_e, \quad \forall e \in E_D \tag{4.8}$$

$$\sum_{f \sim p} y_f^e = 2\gamma_p^e, \quad \forall e \in E_D \, \forall p \in V_T \setminus \{tns(e_1), tns(e_2)\} \tag{4.9}$$

/* $2\gamma_p^e$ es el grado de p en el ruteo de e que puede ser 0 o 2 */

/ * Restricción de integridad * /

$$y_g^e \leq \chi_e, \ \forall e \in \hat{E}_D, \ \forall g \in E_T, \tag{4.10}$$

/ * Restricción de capacidad en el escenario de falla f * /

$$\sum_{e \in E_{Mf}} x_g^{ef} \cdot \dot{m}_e^f + z_{Q_1} \Big(\sum_{e \in E_{Mf}} x_g^{ef} \cdot \ddot{m}_e^f \Big) \leq \sum_{\hat{b}_t \in \hat{B}} w_g^t \cdot \hat{b}_t, \forall f \in E_T, \forall g \in \hat{E}_D \tag{4.11}$$

/ * Ruteo en la red de datos en el escenario de falla f * /

$$\dot{m}_e^f \cdot \left(\sum_{g \sim e_1} x_g^{ef} - 1 \right) \geq 0, \quad \forall f \in E_T, \forall e \in E_{M'} \tag{4.12}$$

$$\dot{m}_e^f \cdot \left(\sum_{g \sim e_2} x_g^{ef} - 1 \right) \geq 0, \quad \forall f \in E_T, \forall e \in E_{M'} \tag{4.13}$$

$$\dot{m}_e^f \cdot \left(\sum_{g \sim p} x_g^{ef} - 2\gamma_p^{ef} \right) = 0, \ \forall f \in E_T, \forall e \in E_{M'}, \forall p \in (\hat{V}_D \setminus \{e_1, e_2\}) \tag{4.14}$$

/ * Relación de links de transporte sobre links de datos * /

$$x_g^{ef} \leq 1 - y_f^g, \quad \forall f \in E_T, \forall g \in \hat{E}_D, \forall e \in E_{M'} \tag{4.15}$$

/ * Restricción de capacidad en el escenario sin falla * /

$$\sum_{e \in E_M} x_g^e \cdot \dot{m}_e^0 + z_{Q_0} \Big(\sum_{e \in E_M} x_g^e \cdot \ddot{m}_e^0 \Big) \leq \sum_{\hat{b}_t \in \hat{B}} w_g^t \cdot \hat{b}_t, \ \forall g \in \hat{E}_D, \tag{4.16}$$

/ * Ruteo en la red de datos en el escenario sin falla * /

$$\dot{m}_e^0 \cdot \left(\sum_{g \sim e_1} x_g^e - 1 \right) \geq 0, \ \forall e \in E_M, \tag{4.17}$$

$$\dot{m}_e^0 \cdot \left(\sum_{g \sim e_2} x_g^e - 1 \right) \geq 0, \ \forall e \in E_M, \tag{4.18}$$

$$\dot{m}_e^0 \cdot \left(\sum_{g \sim p} x_g^e - 2\gamma_p^e \right) = 0, \ \forall e \in E_M, \forall p \in (\hat{V}_D \setminus \{e_1, e_2\}), \tag{4.19}$$

$$w_g^t, x_g^{ef}, x_g^e, y_g^e, \eta_g^{et}, \gamma_p^{ef}, \gamma_p^e \in \{0,1\} \quad \forall t, e, f, g. \tag{4.20}$$

Donde el dimensionamiento b_e en términos de las variables de decisión del problema, vendría dado por la igualdad:

$$b_e = \sum_{\hat{b}_t \in \hat{B}} w_e^t \cdot \hat{b}_t,$$

ya las w_e^t se instancian en 1 a lo sumo para una tecnología.

- A partir de las restricciones (4.9) puede ocurrir que haya variables y_g^e con valor 1, tales que a $e \in \hat{E}_D$ no se le haya asignado ninguna tecnología. Sin embargo, son tenidos en cuenta en la función objetivo, por lo que se agregó un nueva restricción (4.10) para solucionar esto. Si no se le asignó ninguna tecnología al link $e \in \hat{E}_D$, entonces, y_g^e deberá ser 0 para todos los $g \in E_T$.

8. Anexo II

Conceptos básicos generales

En este Anexo se definen conceptos básicos de optimización que son muy importantes para entender este proyecto.

- **Problemas de Optimización:** son problemas de asignación óptima de recursos limitados, para cumplir un objetivo dado que representa la meta del decisor. Los recursos pueden corresponder a personas, materiales, dinero, etc. Entre todas las asignaciones de recursos admisibles, se quiere encontrar la/s que maximiza/n o minimiza/n alguna cantidad numérica tal como ganancias o costos, etc. [2].

- **Modelo de Optimización Matemática:** es una representación matemática de una cierta "realidad" y consiste en una función objetivo y un conjunto de restricciones en la forma de un sistema de ecuaciones o inecuaciones (es decir un problema de optimización). Los problemas de optimización son muy comunes en el modelado matemático de sistemas reales en un amplio rango de aplicaciones: economía, finanzas, química, astronomía, física, medicina, computación, etc... Los modelos presentan limitaciones como información incompleta, simplificaciones y errores numéricos, tecnología limitada [2].

- **Componentes de problemas de optimización:**

  - **Ejemplo de problema de optimización**:

    Min $3x_1+2x_2+x_3+2x_4+2x_5$ /
    Sujeto a:

    $$x_1-x_2+2x_3-x_4+x_5+2x_6 = 1$$
    $$-x_1+2x_2+x_3-2x_4-x_5+x_6 = 3$$
    $$2x_1+x_2-x_3+x_4-2x_5+x_6 = 2$$
    $$x_i \geq 0 \ \forall i=1\ldots6$$

  - **Función objetivo:** es la función que se quiere optimizar. La mayoría de los problemas de optimización tienen una función objetivo pero hay casos de varias funciones objetivo (programación multiobjetivo) o incluso puede no tener función objetivo (interesan soluciones factibles solamente) [2]. En el caso del ejemplo es $3x_1+2x_2+x_3+2x_4+2x_5$.

  - **Variables**: Entradas controlables son las variables de decisión, cuyos valores afectan la función objetivo y restricciones [2]. En el caso del ejemplo son $x_i \geq 0 \ \forall i=1\ldots6$.

- **Parámetros:** son entradas no controlables que se fijan a priori y dependen del problema particular, a veces se requiere un análisis de sensibilidad (para ver como afecta al valor de la función objetivo variaciones en los mismos) o uno paramétrico (para los casos en que no se conoce el valor exacto de los mismos) [2]. Son todos los coeficientes de la función objetivo y los coeficientes y términos independientes de todas las restricciones.

- **Restricciones:** son relaciones entre parámetros y variables de decisión. En el ejemplo corresponde a las tres ecuaciones a las que se encuentra sujeto el problema [2].

- **Clasificación de los problemas de optimización [2]**

  - **Según el tipo de variables**
    - Continua (todas las variables pertenecen a los reales).
    - Discreta (todas las variables pertenecen a los enteros).
    - Binaria (todas las variables pueden tomar solo el valor cero o uno)
    - Mixta (tienen variables de diferente tipo, de los vistos anteriormente).

  - **Según el tipo de funciones objetivo y restricciones**
    - Lineal (tanto la función objetivo como las restricciones son todas lineales).
    - No lineal (la función objetivo o alguna de las restricciones no es lineal).

- **Programación lineal (PL):** aborda una clase de problemas de programación donde tanto la función objetivo a optimizar como todas las relaciones entre las variables correspondientes a los recursos son lineales. Hoy en día, esta técnica se aplica con éxito para resolver problemas de presupuestos de capital, diseño de dietas, conservación de recursos, juegos de estrategias, predicción de crecimiento económico, sistemas de transporte, etc. [2].

- **Relajación de un problema de optimización [2]**
  Dado el siguiente problema de optimización: (G) Min $f(x)$ / $x \in F$
  Definimos: (GL) Min $fL(x)$ / $x \in FL$
  GL es una **relajación** de G si:
  1 - $FL \supseteq F$
  2 - $fL(x) \leq f(x)\ \forall x \in F$

- **Relajación Lagrangeana [2]**
  Dado el siguiente problema de optimización: (P) Min $f(x)$ / $g(x) \leq 0\ \forall x \in X$
  Definimos: (P_λ) Min $f(x) + \Sigma \lambda_j * g_j(x)$ / $x \in X$
  P_λ es una **relajación Lagrangeana** de P
  Observación: toda relajación Lagrangeana con $\lambda \geq 0$ de un problema (P) es una relajación de (P)

- **Problema dual [2]**

  Dados los problemas P y P_λ , se define el **dual** D de P como:
  (D) Max $\Phi(\lambda)$ / $\lambda \geq 0$, donde $\Phi(\lambda)$= Min($f(x)$ + $\Sigma\ \lambda j * gj(x)$) / $x \in X$

- **Gap de dualidad [2]**

  Sean d = Max $\Phi(\lambda)$ / $\lambda \geq 0$ y p = Min $f(x)$ / $g(x) \leq 0\ \forall x \in X$
  Donde $\Phi(\lambda)$ es la definida para el problema dual.
  El **gap de dualidad** denominado con la letra Δ es igual a p-d.
  NOTA: $p \geq d$

- **Método Simplex [2]**

  El método Simplex fue creado por el matemático norteamericano George Bernard Dantzig en 1947, es una técnica para dar soluciones numéricas a problema de programación lineal.
  Es un procedimiento iterativo que permite ir mejorando la solución a cada paso. El proceso concluye cuando no es posible seguir mejorando más dicha solución. Partiendo del valor de la función objetivo en un vértice cualquiera de la región factible, el método consiste en buscar sucesivamente otro vértice que mejore al anterior. La búsqueda se hace siempre a través de los lados del polígono (o de las aristas del poliedro, si el número de variables es mayor).

  Sea (P) el siguiente problema de programación lineal:
  (P) Min $c^T x$ /$Ax = b$ $x \geq 0$
  Donde $c \in R_n$, $b \in R_m$ y A es una matriz de rango completo mxn con n>m.

  Sea la *región factible* $S = \{x \in R_n, / Ax = b, x \geq 0\}$
  y su *interior relativo* $S_0 = \{x \in R_n / Ax = b, x>0\}$

  Sea (D) el problema dual asociado a (P):
  (D) Max $b^T w$ / $A^T w + z = c$, $z \geq 0$
  Donde las variables $z \in R_m$ son las variables de holgura.

  Sea cualquier par (x, z) donde $x \in S$ y (w, z) es una solución factible para (D) para algún $w \in R_m$, se define el *gap de dualidad* (Δ) como: $\Delta = c^T x - b^T w = x^T z$

  Teorema: un par de soluciones primal (x) y dual (z) factibles son óptimas sii cuando una de estas soluciones factibles tiene una holgura estrictamente positiva, el valor de la variable asociada en el dual es cero, o sea $x^T z = 0 = \Delta$.
  En cada iteración el método simplex encuentra una solución primal factible y mantiene Δ=0. El algoritmo se detiene cuando se encuentra una solución dual factible.

  Soluciones básicas:
  - Si A_{mn}, de rango completo y m<n, las bases tiene dimensión m y una solución básica x=(x_1...x_n) correspondiente a una base contiene m componentes positivas y n-m nulas.
  - Corresponden a los puntos extremos del poliedro de restricciones.

- **Problemas de optimización combinatoria y entera**: maximizan o minimizan funciones de varias variables sujeto a restricciones de integridad sobre todas o algunas variables.
  Ejemplos de aplicación: distribución de productos o mercaderías, secuenciamiento de tareas en problemas de producción, diseño de redes de comunicación, etc. [2].

- **Programación lineal entera mixta (MIP) [2]**
  Sea (*P*) el siguiente problema de programación lineal:
  Min $cx + hy / Ax + Gy \leq b$, con $x \in Z_n^+$, $y \in R_n^+$

  Región factible *S*:
  $S = \{(x, y) / x \in Z_n^+ , y \in R_n^+ , Ax + Gy \leq b\}$

  Decimos que es un problema lineal entero mixto (MIP, en ingles mix integer problem) ya que algunas variables pertenecen a los enteros Z_n^+ y otras a los reales R_n^+ .

  Solución óptima y valor óptimo:
  Una solución factible (x_0, y_0) es *óptima* si $\forall (x, y) \in S$, $cx+hy \geq cx_0+hy_0$ (para un problema de minimización). Se define el valor óptimo como el resultado de cx_0+hy_0. Un MIP puede ser no factible (no existe ninguna solución factible), factible (existe al menos una solución factible) o no acotado (región factible no acotada y la función objetivo no alcanza el mínimo en alguno de los vértices, si los hay).

- **Problema lineal entero (ILP) [2]**
  Sea (*P*) el siguiente problema de programación lineal:
  Min $cx / Ax \leq b, x \in Z_n^+$

  Decimos que es un problema entero (IP, en ingles integer problem) ya que todas las variables pertenecen a los enteros Z_n^+ . Si además la función objetivo y las restricciones son lineales decimos que un problema lineal entero ILP (en ingles integer lineal problem).

  Los problemas continuos e IP son casos particulares de MIP.

  Dentro de los IP se encuentran los problemas binarios donde todas las variables pueden tomar únicamente los valores cero y uno (muchas veces se usan para representar relaciones lógicas).
  O sea $x \in B_n^+$, B = {0,1}. El modelo que representa el problema a resolver en este proyecto de grado es lineal entero y binario.

- **Métodos de corte [2]**

  Sea el problema de optimización:
  (ILP, integer lineal problem, problema lineal y entero)
  Min $c^T x$ / $Ax=b$, $x \geq 0$, x entero (A, c y b enteros)

  Sea (LP, lineal problem, problema lineal) una relajación de (ILP):
  (LP) Min $c^T x$ / $Ax=b$, $x \geq 0$ (A, c y b reales)
  Se cumple que óptimo (ILP) ≥ óptimo (LP).

  - **Ideas básicas en los métodos de corte**
    - Ir agregando a LP restricciones, de a una por vez, de forma de excluir el óptimo no entero del problema relajado
    - Estas nuevas restricciones no deben excluir ningún punto entero de la región factible

  - **Plano de corte**
    Un plano de corte es una restricción lineal tal que no excluye ningún punto entero del conjunto de soluciones factibles (soluciones enteras al problema).

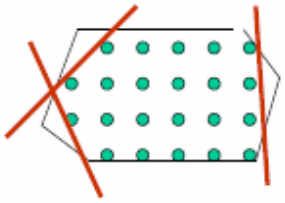

Diagrama para representar los planos de corte de una región factible. El polígono de líneas negras representa la región factible, los puntos verdes son las soluciones enteras factibles y las líneas rojas son las restricciones que representan los planos de corte que se agregan.

Complejidad de los problemas de decisión [2]

- **Codificación**
  Permite asociar a todo enunciado de un problema una palabra *x* definida sobre un alfabeto *X* ($x \in X^*$). A todo problema de decisión se le asocia un lenguaje $L \subseteq X^*$, que está formado por un conjunto de palabras para las cuales la respuesta es *SI*.

- **Clase *NP***
  La clase *NP* es el conjunto de lenguajes reconocidos polinomialmente por una MTND (Máquinas de Turing no determinísticas).
  $L \in NP$ si existe una MTND y un polinomio *P* tal que: $x \in L$ sii la MTND termina en el estado *SI* para el dato *x* en tiempo finito y $T(x) \leq P(x)$ ($T(x)$=duración del cálculo en la MTND). Los problemas de decisión clásicos pertenecen a *NP*. Para verificar que un problema de decisión es *NP*, una vez fijado los valores de una potencial solución, alcanza con que la verificación se realice en tiempo polinomial. Uno de los problemas de decisión clásicos es el denominado *SAT* (Problema de satisfacibilidad booleana).

  Problema *SAT*: Dado un conjunto *V* de variables booleanas y un conjunto *C* de cláusulas sobre *V*. Se pregunta si existe una asignación de las variables que haga verdadera todas las cláusulas. Por ejemplo, una instancia de *SAT* sería el saber si existen valores para x_1, x_2, x_3, x_4 tales que la expresión: $(x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$ es verdadera.

- **Problemas *NP-completos***
  Stephen Cook demostró que todo problema de *NP* se reduce polinomialmente a *SAT*, entonces *SAT* es un problema más difícil que todo problema de *NP*.
  Como *SAT* pertenece a *NP*, *NP* contiene una subclase de problemas más difíciles, los *NP-completos*. Los problemas de decisión clásicos son *NP-completos*.

- **Problemas *NP-difíciles***
  Un problema π es *NP-difícil* si es más difícil que un problema *NP-completo*, o sea existe un problema *NP-completo* que se reduce polinomialmente por la reducción de Turing a π. Los problemas de optimización lineales enteros ILP, son *NP-difíciles*.

9. Anexo III

En este Anexo se incluye el archivo de entrada con los datos del caso de prueba 1, así como también el archivo de salida de cada una de las soluciones implementadas, resultado de su ejecución para este caso de prueba. Debido a lo extenso de estos archivos no se incluyen en su totalidad en este informe, se agregó una línea de puntos en los lugares donde se suprimieron líneas.

Caso de prueba 1

1. Archivo de entrada

```
-------------------------------------------
- DATOS_GRAFO_TRANSPORTE                   -
-------------------------------------------


CANT_NODOS_TRANSPORTE: 5
CANT_LINKS_TRANSPORTE: 6

-------------------------------------------
- LINKS_CON_DISTANCIAS_ASOCIADAS           -
-------------------------------------------

LINK: 0 1 15.0
LINK: 0 3 13.0
LINK: 1 2 8.0
LINK: 1 3 3.0
LINK: 2 4 5.0
LINK: 3 4 7.0

-------------------------------------------
- TECNOLOGÍAS                             -
-------------------------------------------

CANT_TECNOLOGIAS: 3
TECNOLOGIA: 0 0
TECNOLOGIA: 8 20
TECNOLOGIA: 25 35

-------------------------------------------
- CLIENTES                                -
-------------------------------------------

CANT_CLIENTES: 5
TNS_0: 0
TNS_1: 1
TNS_2: 2
TNS_3: 3
TNS_4: 4

-------------------------------------------
- Matrices                                -
-------------------------------------------

LINK1: 0 1

0 3 2 0 0
0 0 8 10
```

```
0 1 0

0 1
0

LINK1: 0 3

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0

LINK1: 1 2

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0

LINK1: 1 3

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0

LINK1: 2 4

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0

LINK1: 3 4

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0

0 3 2 0 0
0 0 8 10
0 1 0
0 1
0
```

2. Archivo de salida de la solución aproximada 1

```
****** Ruteos en transporte en el escenario sin fallas ******
Arista (0,1):  0      1
Arista (0,2):  0      1      2
Arista (0,3):  0      3
Arista (0,4):  0      3      4
Arista (1,2):  1      2
Arista (1,3):  1      3
Arista (1,4):  1      3      4
Arista (2,3):  2      1      3
Arista (2,4):  2      4
Arista (3,4):  3      4

****** Ruteos en datos en escenarios de falla ******
Falla link (0,1):
       arista (0,2) es:      0      3      2
       arista (0,1) es:      0      3      1
```

```
Falla link (0,3):
       arista (0,4) es:      0      1      4
       arista (0,3) es:      0      1      3
Falla link (1,2):
       arista (2,3) es:      2      4      3
       arista (1,2) es:      1      4      2
       arista (0,2) es:      0      4      2
Falla link (1,3):
       arista (2,3) es:      2      4      3
       arista (1,4) es:      1      2      4
       arista (1,3) es:      1      2      4      3
Falla link (2,4):
       arista (2,4) es:      2      1      4
Falla link (3,4):
       arista (3,4) es:      3      2      4
       arista (1,4) es:      1      2      4
       arista (0,4) es:      0      2      4

****** Tecnologías asignadas a las aristas y distancias en transporte ******
Arista (0,1):
       tecnologia 8.000000
       Costo Tec. 20.000000
       distancia 15.000000
Arista (0,2):
       tecnologia 8.000000
       Costo Tec. 20.000000
       distancia 23.000000
Arista (0,3):
       tecnologia 8.000000
       Costo Tec. 20.000000
       distancia 13.000000
Arista (0,4):
       tecnologia 8.000000
       Costo Tec. 20.000000
       distancia 20.000000
Arista (1,2):
       tecnologia 25.000000
       Costo Tec. 35.000000
       distancia 8.000000
Arista (1,3):
       tecnologia 25.000000
       Costo Tec. 35.000000
       distancia 3.000000
Arista (1,4):
       tecnologia 25.000000
       Costo Tec. 35.000000
       distancia 10.000000
Arista (2,3):
       tecnologia 8.000000
       Costo Tec. 20.000000
       distancia 11.000000
Arista (2,4):
       tecnologia 25.000000
       Costo Tec. 35.000000
       distancia 5.000000
Arista (3,4):
       tecnologia 25.000000
       Costo Tec. 35.000000
       distancia 7.000000


RESULTADO DE LA FUNCION OBJETIVO = 2795.000000


Desperdicio total de la capacidad de la red = 82.000000
```

3. Archivo de salida de la solución utilizando Cplex

```
Tried aggregator 1 time.
MIP Presolve eliminated 52 rows and 1 columns.
MIP Presolve modified 282 coefficients.
Reduced MIP has 1258 rows, 856 columns, and 3752 nonzeros.
Presolve time =    0.02 sec.
Clique table members: 1450.
MIP emphasis: balance optimality and feasibility.
Root relaxation solution time =    0.02 sec.

        Nodes                                         Cuts/
   Node  Left      Objective  IInf  Best Integer    Best Node    ItCnt     Gap         Variable B
Parent  Depth

      0     0        0.0000   107                    0.0000      136
                    81.6667   218                 Cuts:  374      391
                   275.0000   267                 Cuts:  428      659
                   420.0000   328                 Cuts:  246      841
                   420.0000   311                 Cuts:  137      955
                   438.0000   304                  Cuts:  86     1088
                   472.5000   291                  Cuts:  58     1145
                   472.5000   261                  Cuts:  46     1205
*     0+    0                    0     1365.0000      472.5000     1205   65.38%
                   472.5000   258     1365.0000     Cuts:  90     1279   65.38%
                   480.0000   265     1365.0000     Cuts:  35     1315   64.84%
                   480.0000   307     1365.0000     Cuts:  41     1422   64.84%
    100     26       945.0000   315      1365.0000       840.0000      6865   38.46%           wt1_e3 U
8      9
    200     49      1260.0000   215      1365.0000       980.0000     13751   28.21%          xe6g4_e5 U
198      7
    300      8         cutoff             1365.0000      1280.0000     18556    6.23%    gamaDFe5g4_p0 U
156     12

GUB cover cuts applied:  132
Clique cuts applied:  85
Cover cuts applied:  692
Implied bound cuts applied:  190
Gomory fractional cuts applied:  10
Solution status = Optimal
Solution value  = 1365
Variables:
ne0t0_g0 = -0.000000 ne0t1_g0 = -0.000000 ne0t2_g0 = -0.000000 ne0t0_g1 = -0.000000
ne0t1_g1   =   1.000000   ne0t2_g1   =   -0.000000   ne0t0_g2   =   -0.000000   ne0t1_g2   =   -0.000000
.....................................................................................................................
ne8t1_g5 = -0.000000 ne8t2_g5 = -0.000000 ne9t0_g0 = -0.000000 ne9t1_g0 = -0.000000
ne9t2_g0 = -0.000000 ne9t0_g1 = -0.000000 ne9t1_g1 = -0.000000 ne9t2_g1 = -0.000000
ne9t0_g2 = -0.000000 ne9t1_g2 = -0.000000 ne9t2_g2 = -0.000000 ne9t0_g3 = -0.000000
ne9t1_g3 = -0.000000 ne9t2_g3 = -0.000000 ne9t0_g4 = -0.000000 ne9t1_g4 = -0.000000
ne9t2_g4   =   0.000000   ne9t0_g5   =   -0.000000   ne9t1_g5   =   -0.000000   ne9t2_g5   =   1.000000
.....................................................................................................................
ye8_g5 = -0.000000 ye9_g0 = -0.000000 wt0_e9 = -0.000000 wt1_e9 = 0.000000
wt2_e9 = 1.000000 ye9_g1 = -0.000000 ye9_g2 = 0.000000 ye9_g3 = 0.000000
ye9_g4 = 0.000000 ye9_g5 = 1.000000 Se0 = 1.000000 Se1 = 0.000000
Se2 = 1.000000 Se3 = 0.000000 Se4 = 1.000000 Se5 = 1.000000
Se6 = -0.000000 Se7 = 0.000000 Se8 = 1.000000 Se9 = 1.000000
gamaTe0_p2 = -0.000000 gamaTe0_p3 = 0.000000 gamaTe0_p4 = -0.000000 gamaTe1_p1 = 0.000000
gamaTe1_p3 = 0.000000 gamaTe1_p4 = 0.000000 gamaTe2_p1 = 0.000000 gamaTe2_p2 = -0.000000
gamaTe2_p4 = -0.000000 gamaTe3_p1 = 0.000000 gamaTe3_p2 = 0.000000 gamaTe3_p3 = 0.000000
gamaTe4_p0 = 0.000000 gamaTe4_p3 = 0.000000 gamaTe4_p4 = 0.000000 gamaTe5_p0 = -0.000000
gamaTe5_p2 = -0.000000 gamaTe5_p4 = -0.000000 gamaTe6_p0 = 0.000000 gamaTe6_p2 = 0.000000
.....................................................................................................................
xe0g0_e3 = -0.000000 xe1g0_e3 = -0.000000 xe5g0_e3 = -0.000000 xe6g0_e3 = 0.000000
xe7g0_e3 = 0.000000 xe9g0_e3 = 0.000000 xe0g0_e4 = 1.000000 xe1g0_e4 = 1.000000
xe5g0_e4 = -0.000000 xe6g0_e4 = 0.000000 xe7g0_e4 = 1.000000 xe9g0_e4 = -0.000000
xe0g0_e5 = 1.000000 xe1g0_e5 = -0.000000 xe5g0_e5 = 1.000000 xe6g0_e5 = 1.000000
xe7g0_e5 = 1.000000 xe9g0_e5 = -0.000000 xe0g0_e6 = -0.000000 xe1g0_e6 = 0.000000
xe5g0_e6 = -0.000000 xe6g0_e6 = 0.000000 xe7g0_e6 = -0.000000 xe9g0_e6 = -0.000000
```

```
xe5g4_e6 = 0.000000 xe6g4_e6 = 0.000000 xe7g4_e6 = -0.000000 xe9g4_e6 = 0.000000
xe0g4_e7 = -0.000000 xe1g4_e7 = -0.000000 xe5g4_e7 = 0.000000 xe6g4_e7 = -0.000000
xe7g4_e7 = 0.000000 xe9g4_e7 = 0.000000 xe0g4_e8 = -0.000000 xe1g4_e8 = 0.000000
xe5g4_e8 = -0.000000 xe6g4_e8 = 0.000000 xe7g4_e8 = 0.000000 xe9g4_e8 = 0.000000
```

```
gamaDFe5g2_p0 = 0.000000 gamaDFe5g2_p2 = 1.000000 gamaDFe5g2_p4 = 1.000000 gamaDFe6g2_p0 = -0.000000
gamaDFe6g2_p2 = 1.000000 gamaDFe6g2_p3 = 0.000000 gamaDFe7g2_p0 = 0.000000 gamaDFe7g2_p1 = 0.000000
gamaDFe7g2_p4 = 1.000000 gamaDFe9g2_p0 = -0.000000 gamaDFe9g2_p1 = 0.000000 gamaDFe9g2_p2 = 0.000000
gamaDFe0g3_p2 = -0.000000 gamaDFe0g3_p3 = -0.000000 gamaDFe0g3_p4 = -0.000000 gamaDFe1g3_p1 = 0.000000
gamaDFe1g3_p3 = 1.000000 gamaDFe1g3_p4 = 1.000000 gamaDFe5g3_p0 = -0.000000 gamaDFe5g3_p2 = 0.000000
gamaDFe5g3_p4 = 0.000000 gamaDFe6g3_p0 = -0.000000 gamaDFe6g3_p2 = 0.000000 gamaDFe6g3_p3 = 1.000000
```

```
gamaDFe1g5_p3 = 0.000000 gamaDFe1g5_p4 = 0.000000 gamaDFe5g5_p0 = 0.000000 gamaDFe5g5_p2 = 0.000000
gamaDFe5g5_p4 = 0.000000 gamaDFe6g5_p0 = -0.000000 gamaDFe6g5_p2 = 1.000000 gamaDFe6g5_p3 = 0.000000
gamaDFe7g5_p0 = 1.000000 gamaDFe7g5_p1 = 1.000000 gamaDFe7g5_p4 = 0.000000 gamaDFe9g5_p0 = 0.000000
gamaDFe9g5_p1 = 1.000000 gamaDFe9g5_p2 = 1.000000 xe0_e0 = 1.000000 xe1_e0 = 0.000000
xe5_e0 = 0.000000 xe6_e0 = 0.000000 xe7_e0 = 1.000000 xe9_e0 = -0.000000
xe0_e1 = 0.000000 xe1_e1 = 0.000000 xe5_e1 = -0.000000 xe6_e1 = 0.000000
xe7_e1 = -0.000000 xe9_e1 = 0.000000 xe0_e2 = -0.000000 xe1_e2 = 1.000000
xe5_e2 = -0.000000 xe6_e2 = -0.000000 xe7_e2 = 1.000000 xe9_e2 = 0.000000
```

```
xe5_e8 = 1.000000 xe6_e8 = 1.000000 xe7_e8 = -0.000000 xe9_e8 = -0.000000
xe0_e9 = -0.000000 xe1_e9 = 1.000000 xe5_e9 = 1.000000 xe6_e9 = 0.000000
xe7_e9 = -0.000000 xe9_e9 = 1.000000 gamaDe0_p2 = 0.000000 gamaDe0_p3 = -0.000000
gamaDe0_p4 = -0.000000 gamaDe1_p1 = 0.000000 gamaDe1_p3 = 1.000000 gamaDe1_p4 = 1.000000
gamaDe5_p0 = 0.000000 gamaDe5_p2 = 1.000000 gamaDe5_p4 = 1.000000 gamaDe6_p0 = 0.000000
gamaDe6_p2 = 1.000000 gamaDe6_p3 = 0.000000 gamaDe7_p0 = 1.000000 gamaDe7_p1 = 1.000000
gamaDe7_p4 = -0.000000 gamaDe9_p0 = 0.000000 gamaDe9_p1 = 0.000000 gamaDe9_p2 = 0.000000

Maximum bound violation = 3.55271e-015
```

10. Anexo IV

En este Anexo se incluye el archivo de entrada con los datos del caso de prueba 2, así como también el archivo de salida de cada una de las soluciones implementadas, resultado de su ejecución para este caso de prueba. Debido a lo extenso de estos archivos no se incluyen en su totalidad en este informe, se agregó una línea de puntos en los lugares donde se suprimieron líneas.

Caso de prueba 2

1. Archivo de entrada

```
-------------------------------------------
- DATOS_GRAFO_TRANSPORTE                   -
-------------------------------------------

CANT_NODOS_TRANSPORTE: 7
CANT_LINKS_TRANSPORTE: 8

-------------------------------------------
- LINKS_CON_DISTANCIAS_ASOCIADAS           -
-------------------------------------------

LINK: 0 1 25
LINK: 0 3 35
LINK: 1 2 50
LINK: 1 4 20
LINK: 2 6 28
LINK: 3 4 40
LINK: 4 5 15
LINK: 5 6 10

-------------------------------------------
- TECNOLOGÍAS                             -
-------------------------------------------

CANT_TECNOLOGIAS: 4
TECNOLOGIA: 0 0
TECNOLOGIA: 20 500
TECNOLOGIA: 40 800
TECNOLOGIA: 80 1100

-------------------------------------------
- CLIENTES                                -
-------------------------------------------

CANT_CLIENTES: 7
TNS_0: 0
TNS_1: 1
TNS_2: 2
TNS_3: 3
TNS_4: 4
TNS_5: 5
TNS_6: 6

-------------------------------------------
- Matrices                                -
-------------------------------------------

LINK1: 0 1

0 0 0 10 20 0 0
0 0 0 5 0 0
0 0 0 3 15
0 7 0 0
```

```
0 30 0
0 18
0

LINK1: 0 3

0 0 0 10 20 0 0
0 0 0 5 0 0
0 0 0 3 15
0 7 0 0
0 30 0
0 18
0
```

.....

```
0 0 0 10 20 0 0
0 0 0 5 0 0
0 0 0 3 15
0 7 0 0
0 30 0
0 18
0
```

2. Archivo de salida de la solución aproximada 1

```
****** Ruteos en transporte en el escenario sin fallas ******
Arista (0,1):  0      1
Arista (0,2):  0      1      2
Arista (0,3):  0      3
Arista (0,4):  0      1      4
Arista (0,5):  0      1      4      5
Arista (0,6):  0      1      4      5      6
Arista (1,2):  1      2
Arista (1,3):  1      4      3
Arista (1,4):  1      4
Arista (1,5):  1      4      5
Arista (1,6):  1      4      5      6
Arista (2,3):  2      6      5      4      3
Arista (2,4):  2      6      5      4
Arista (2,5):  2      6      5
Arista (2,6):  2      6
Arista (3,4):  3      4
Arista (3,5):  3      4      5
Arista (3,6):  3      4      5      6
Arista (4,5):  4      5
Arista (4,6):  4      5      6
Arista (5,6):  5      6

****** Ruteos en datos en escenarios de falla ******
Falla link (0,1):
       arista (0,6) es:      0      3      6
       arista (0,5) es:      0      3      5
       arista (0,4) es:      0      3      4
       arista (0,2) es:      0      3      2
       arista (0,1) es:      0      3      1
Falla link (0,3):
       arista (0,3) es:      0      1      3
Falla link (1,2):
       arista (1,2) es:      1      4      2
       arista (0,2) es:      0      4      2
```

.....

```
Falla link (5,6):
       arista (5,6) es:      5      1      2      6
       arista (4,6) es:      4      1      2      6
       arista (3,6) es:      3      0      2      6
       arista (2,5) es:      2      1      5
```

```
       arista (2,4) es:      2      1      4
       arista (2,3) es:      2      1      3
       arista (1,6) es:      1      2      6
       arista (0,6) es:      0      2      6

****** Tecnologías asignadas a las aristas y distancias en transporte ******
Arista (0,1):
       tecnologia 20.000000
       Costo Tec. 500.000000
       distancia 25.000000
Arista (0,2):
       tecnologia 0.000000
       Costo Tec. 0.000000
       distancia 75.000000
Arista (0,3):
       tecnologia 40.000000
       Costo Tec. 800.000000
       distancia 35.000000
.....................................................................................................
Arista (5,6):
       tecnologia 20.000000
       Costo Tec. 500.000000
       distancia 10.000000


RESULTADO DE LA FUNCION OBJETIVO = 314800.000000


Desperdicio total de la capacidad de la red = 111.000000
```

3. Archivo de salida de la solución utilizando Cplex

```
Tried aggregator 1 time.
MIP Presolve eliminated 250 rows and 151 columns.
MIP Presolve modified 508 coefficients.
Reduced MIP has 4159 rows, 2772 columns, and 13242 nonzeros.
Reduced MIP has 2772 binaries, 0 generals, 0 SOSs, and 0 indicators.
Probing time =    0.02 sec.
Tried aggregator 1 time.
MIP Presolve eliminated 4 rows and 14 columns.
Reduced MIP has 4155 rows, 2758 columns, and 13220 nonzeros.
Reduced MIP has 2758 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time =    0.08 sec.
Probing time =    0.01 sec.
Clique table members: 9623.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time =    0.02 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0        0.0000   330                      0.0000      286
      0     0        0.0000   508                  Cuts: 454      602
      0     0        0.0000   464                  Cuts: 266      833
      0     0        0.0000   609                  Cuts: 585     1234
*     0+    0                        823100.0000        0.0000     1234  100.00%
      0     2        0.0000   499   823100.0000        0.0000     1234  100.00%
Elapsed real time =   0.70 sec. (tree size =  0.01 MB, solutions = 1)
*    13+   13                        791200.0000        0.0000     3131  100.00%
*    62+   62                        786300.0000        0.0000     5353  100.00%
     62    64   199100.0000   309   786300.0000        0.0000     5353  100.00%
*   102+  102                        670900.0000        0.0000     7179  100.00%
*   103+  103                        620500.0000        0.0000     7203  100.00%
*   226+  222                        459100.0000        0.0000    13222  100.00%
*   226+  222                        402600.0000        0.0000    13222  100.00%
```

```
     358   327    55000.0000   588   402600.0000        0.0000    18965  100.00%
     408   373    47000.0000   446   402600.0000        0.0000    23992  100.00%
     503   456   190551.8519   375   402600.0000        0.0000    35687  100.00%
*    514+  463                     390100.0000        0.0000    36260  100.00%
*    515+  464                     382100.0000        0.0000    36351  100.00%
*    516+  465                     381700.0000        0.0000    36446  100.00%
*    518+  297                     356300.0000    87543.9144   104022   75.43%
*    518+  197                     350300.0000    90054.0258   108977   74.29%
*    518+  130                     320200.0000    91713.9324   114136   71.36%
*    518+   86                     306300.0000    96322.1389   127082   68.55%
*    518+   57                     271800.0000    97562.3823   131077   64.11%
     518    58   100616.8441   949   271800.0000   100616.8441   142634   62.98%
     519    59   102408.0599   845   271800.0000   100616.8441   143880   62.98%
     520    60   105553.8920   842   271800.0000   102489.2758   146116   62.29%
     523    63   107138.1912   816   271800.0000   102597.9993   148057   62.25%
     526    63   108056.6719   834   271800.0000   102597.9993   151758   62.25%
     538    70   109273.6122   900   271800.0000   102597.9993   164351   62.25%
Elapsed real time =  85.61 sec. (tree size =  0.36 MB, solutions = 15)
     551    79   146146.0007   717   271800.0000   102597.9993   177797   62.25%
     566    90   116476.9913   923   271800.0000   102597.9993   192150   62.25%
     615   117   133066.4232   824   271800.0000   102597.9993   208554   62.25%
     711   191   206395.1731   674   271800.0000   102597.9993   224567   62.25%
     756   232   108821.1022  1103   271800.0000   103303.7002   236552   61.99%
     876   335   110475.9580  1121   271800.0000   103303.7002   263647   61.99%
     895   350   113411.7627  1026   271800.0000   103303.7002   282002   61.99%
     936   390   125126.2445   919   271800.0000   103303.7002   302744   61.99%
     990   444   141657.5886   915   271800.0000   103303.7002   319688   61.99%
    1062   511   166489.3806   632   271800.0000   103303.7002   339938   61.99%
Elapsed real time = 107.28 sec. (tree size = 14.72 MB, solutions = 15)
    1169   607   193454.3376   550   271800.0000   103303.7002   364333   61.99%
    1175   610   171325.7116  1205   271800.0000   103303.7002   371713   61.99%
    1181   616   176245.0604  1064   271800.0000   107154.4733   378661   60.58%
    1269   699   202095.3595   984   271800.0000   107154.4733   413654   60.58%
    1324   745   221572.7973   949   271800.0000   107154.4733   435351   60.58%
    1378   790   247126.8068   870   271800.0000   107154.4733   463848   60.58%
    1424   828   135588.8450  1040   271800.0000   107154.4733   479621   60.58%
    1481   879   144004.8387   969   271800.0000   109538.0503   499724   59.70%
    1525   919   166154.9917   945   271800.0000   109538.0503   517199   59.70%
    1639  1025   205679.6664   577   271800.0000   109538.0503   546644   59.70%
Elapsed real time = 130.82 sec. (tree size = 39.36 MB, solutions = 15)
    1686  1062   117116.4429  1064   271800.0000   109538.0503   569293   59.70%
    1694  1070   140282.6864   910   271800.0000   110744.4862   580938   59.26%
*   1715+ 1089                     262800.0000   110744.4862   602338   57.86%
    1719  1064   158572.7512   887   262800.0000   110744.4862   604395   57.86%
    1747  1088   189596.9097   743   262800.0000   110744.4862   621273   57.86%
    1814  1152   206651.7092   540   262800.0000   110744.4862   643384   57.86%
    1864  1191   168645.0257  1191   262800.0000   110744.4862   662864   57.86%
    1893  1214   180700.5567  1000   262800.0000   111125.8411   684249   57.71%
    1918  1239   191783.1647   931   262800.0000   111125.8411   699332   57.71%
    1958  1279   199658.7922   560   262800.0000   111125.8411   718913   57.71%
    2036  1351   230106.7787   558   262800.0000   111125.8411   763674   57.71%
Elapsed real time = 153.42 sec. (tree size = 52.84 MB, solutions = 16)
    2092  1395   165729.2206   929   262800.0000   111125.8411   796766   57.71%
    2122  1422   177612.1113   744   262800.0000   113495.9340   816867   56.81%
    2171  1458   200080.9582   462   262800.0000   113495.9340   847866   56.81%
    2191  1473   208432.8480  1021   262800.0000   113495.9340   853936   56.81%
    2224  1502   240628.6116   869   262800.0000   115171.9757   880222   56.18%
    2251  1523   159500.9530  1088   262800.0000   115171.9757   889391   56.18%
    2291  1558   170452.7566  1092   262800.0000   115669.0093   912347   55.99%
    2314  1576   177141.4979  1047   262800.0000   115669.0093   930269   55.99%
    2342  1596   184791.9031   885   262800.0000   115669.0093   949745   55.99%
    2378  1631   191547.1524   788   262800.0000   115669.0093   973648   55.99%
Elapsed real time = 170.94 sec. (tree size = 63.12 MB, solutions = 16)
    2430  1668   215513.8664   453   262800.0000   115669.0093  1000567   55.99%
    2465  1696   171368.9385  1071   262800.0000   115669.0093  1022391   55.99%
    2499  1724   180984.1045   956   262800.0000   115669.0093  1038496   55.99%
    2511  1736   193509.1669   806   262800.0000   117116.4429  1047106   55.44%
    2567  1787   215621.1465   758   262800.0000   117116.4429  1071043   55.44%
    2625  1838   258499.4314   450   262800.0000   117116.4429  1094483   55.44%
*   2660+ 1862                     252800.0000   117530.2323  1106630   53.51%
    2668  1872   131152.0049  1055   252800.0000   117530.2323  1116302   53.51%
```

```
   2688  1892   171629.9547   917   252800.0000   117530.2323  1133820   53.51%
*  2762+ 1865                       237800.0000   117530.2323  1171246   50.58%
   2773  1878   162989.7143   285   237800.0000   117530.2323  1185889   50.58%
   2804  1906   209975.4997  1205   237800.0000   117530.2323  1221572   50.58%
Elapsed real time = 195.33 sec. (tree size = 74.06 MB, solutions = 17)
   2876  1970   207705.9266   957   237800.0000   117530.2323  1240717   50.58%
   2889  1802   226066.6456  1107   237800.0000   118072.8753  1267930   50.35%
   2913  1820   217566.2579   976   237800.0000   118072.8753  1287322   50.35%
   2955  1850   222856.0316  1022   237800.0000   119172.5031  1318056   49.89%
   2960  1855   156876.2146   938   237800.0000   119172.5031  1328046   49.89%
   2972  1865   164920.6266   973   237800.0000   119172.5031  1345454   49.89%
   2991  1874   234253.7743   959   237800.0000   119172.5031  1366263   49.89%
   3047  1914   164359.2931   686   237800.0000   120409.4302  1414027   49.37%
   3075  1934   194633.5508   920   237800.0000   120409.4302  1429565   49.37%
   3113  1963   192251.4654  1099   237800.0000   120409.4302  1443855   49.37%
Elapsed real time = 216.47 sec. (tree size = 75.91 MB, solutions = 18)
   3123  1973   199596.8775  1121   237800.0000   121810.9225  1463285   48.78%
   3131  1980   204092.5518   987   237800.0000   121810.9225  1477730   48.78%
   3151  1998   211704.8915   754   237800.0000   121810.9225  1496334   48.78%
   3191  2018   219751.9492   474   237800.0000   121810.9225  1526993   48.78%
   3235  2051   153912.4489   827   237800.0000   121810.9225  1560568   48.78%
   3273  2081   228290.0917   820   237800.0000   121810.9225  1581403   48.78%
   3302  2103   205623.1337   957   237800.0000   121810.9225  1602179   48.78%
   3325  2126   224000.5916   801   237800.0000   121810.9225  1613358   48.78%
   3368  2167   221099.0174   932   237800.0000   122391.0313  1638137   48.53%
   3381  2174   171479.3593  1064   237800.0000   122391.0313  1651736   48.53%
Elapsed real time = 237.57 sec. (tree size = 87.30 MB, solutions = 18)
   3401  2192   179506.6006   951   237800.0000   122391.0313  1671941   48.53%
   3429  2217   193896.0477   851   237800.0000   122391.0313  1691566   48.53%
   3474  2262   236614.8148   462   237800.0000   123886.4575  1735758   47.90%
   3494  2278   164734.4524  1097   237800.0000   123886.4575  1756190   47.90%
   3526  2306   196969.1421   728   237800.0000   123886.4575  1777866   47.90%
   3582  2346   212286.1923   759   237800.0000   123886.4575  1834682   47.90%
   3666  2422   219816.0041   454   237800.0000   123886.4575  1870217   47.90%
   3703  2442   218636.1015  1030   237800.0000   126013.7580  1887948   47.01%
   3723  2449   230447.8942   906   237800.0000   126013.7580  1900744   47.01%
   3740  2454   157479.6487   956   237800.0000   126013.7580  1913860   47.01%
Elapsed real time = 259.30 sec. (tree size = 94.68 MB, solutions = 18)
   3750  2460   166141.7575   989   237800.0000   126013.7580  1928117   47.01%
   3772  2482   187433.6392   707   237800.0000   126731.1131  1946651   46.71%
   3816  2514   207605.0746   766   237800.0000   126731.1131  1974336   46.71%
   3881  2575   186025.3652   692   237800.0000   126731.1131  2004769   46.71%
*  3911+ 2603                       234800.0000   126731.1131  2015795   46.03%
   3921  2615   234947.1131   388   234800.0000   126731.1131  2019095   46.03%
   3989  2669   202432.0386   910   234800.0000   126731.1131  2051014   46.03%
   4006  2675   166822.9051  1016   234800.0000   126731.1131  2062845   46.03%
   4024  2629   185305.5294   916   234800.0000   127680.9285  2071143   45.62%
   4044  2640   200149.8570   796   234800.0000   127680.9285  2082566   45.62%
   4186  2740   186469.4598   788   234800.0000   128849.9071  2152026   45.12%
Elapsed real time = 284.28 sec. (tree size = 123.98 MB, solutions = 19)
*  4321+ 2843                       222800.0000   128849.9071  2228536   42.17%
   4350  2868    infeasible         222800.0000   128849.9071  2245034   42.17%
   4481  2555   200839.2327   820   222800.0000   129730.2684  2323480   41.77%
   4635  2673   190604.2102   793   222800.0000   130580.0922  2418315   41.39%
   4710  2735   183892.7115  1173   222800.0000   130580.0922  2486470   41.39%
   4832  2820   209076.5345  1126   222800.0000   131631.8138  2585434   40.92%
   4904  2865   150532.7594   729   222800.0000   131631.8138  2641355   40.92%
*  4955+ 2904                       215300.0000   133090.8052  2666899   38.18%
   4985  2925   158498.9644  1150   215300.0000   133090.8052  2695937   38.18%
*  5093+ 3009                       201800.0000   133090.8052  2769288   34.05%
   5169  3066   195981.4472   663   201800.0000   133090.8052  2849613   34.05%
   5254  2037   171942.5628   862   201800.0000   133668.9114  2969342   33.76%
   5341  2072   158442.7455  1220   201800.0000   133668.9114  3046858   33.76%
Elapsed real time = 365.70 sec. (tree size = 83.02 MB, solutions = 23)
*  5389+ 2095                       194300.0000   136595.5463  3109815   29.70%
   5393  2101   175330.8317  1125   194300.0000   136595.5463  3115533   29.70%
   5461  2142   153644.2700  1123   194300.0000   136595.5463  3173840   29.70%
   5577  1754   183484.5983   803   194300.0000   138097.3932  3274108   28.93%
   5637  1775   187584.2805  1015   194300.0000   138097.3932  3328381   28.93%
   5740  1815   187828.2820  1085   194300.0000   139158.3702  3414479   28.38%
```

```
   5794  1838   184448.0397   953   194300.0000   139158.3702  3503394   28.38%
   5891  1897   166371.5172   921   194300.0000   141687.7590  3602815   27.08%
   5975  1936   151725.0309  1245   194300.0000   141687.7590  3693530   27.08%
   6079  1995   183073.6862   843   194300.0000   143849.9012  3801997   25.97%
   6125  2013   168818.8250   957   194300.0000   143849.9012  3855420   25.97%
Elapsed real time = 451.55 sec. (tree size = 90.10 MB, solutions = 24)
   6202  2043   168354.8619  1049   194300.0000   143849.9012  3942262   25.97%
   6252  2056   188456.3236   982   194300.0000   145812.7751  4003353   24.95%
   6300  2081   187226.8397   957   194300.0000   145812.7751  4077217   24.95%
   6368  2096   191052.3425   990   194300.0000   145812.7751  4170845   24.95%
   6444  2112   172108.0354   792   194300.0000   148628.0996  4239747   23.51%
   6524  2146   178032.6890   904   194300.0000   148628.0996  4322675   23.51%
   6615  2162   159818.5373  1167   194300.0000   148628.0996  4398277   23.51%
   6684  2172   192432.1206  1140   194300.0000   150945.3781  4470661   22.31%
   6768  2178   193736.2827  1077   194300.0000   150945.3781  4548006   22.31%
   6869  2199   177024.1987  1053   194300.0000   150945.3781  4633374   22.31%
Elapsed real time = 532.62 sec. (tree size = 99.04 MB, solutions = 24)
   6933  2208   192307.5349  1088   194300.0000   153201.8296  4744090   21.15%
   7017  2228   171746.3871  1109   194300.0000   153201.8296  4846314   21.15%
   7053  2242   187859.9585  1004   194300.0000   153201.8296  4895523   21.15%
   7149  2237   191604.1749   914   194300.0000   156288.6627  5025732   19.56%
   7213  2224   166532.2832   956   194300.0000   156288.6627  5069784   19.56%
   7297  2241   192981.2751   901   194300.0000   156288.6627  5134792   19.56%
   7390  2268   185849.8234  1093   194300.0000   159063.1310  5220666   18.14%
   7481  2286   172694.1164   825   194300.0000   159063.1310  5295831   18.14%
   7561  2288   165924.0949  1324   194300.0000   159063.1310  5353734   18.14%
   7646  2277   189105.8807   662   194300.0000   161108.5940  5424679   17.08%
Elapsed real time = 621.20 sec. (tree size = 104.34 MB, solutions = 24)
   7702  2301   192462.8488   927   194300.0000   161108.5940  5494012   17.08%
   7759  2294   188598.8063  1266   194300.0000   161108.5940  5558663   17.08%
   7849  2279   186510.3394   952   194300.0000   161108.5940  5645884   17.08%
   7931  2272   176406.3823   781   194300.0000   164994.6487  5798444   15.08%
   7981  2249   191109.4668   636   194300.0000   164994.6487  5850312   15.08%
   8073  2269   193476.3622   841   194300.0000   164994.6487  5961465   15.08%
   8256  2253   192524.2030   826   194300.0000   167235.1237  6124083   13.93%
   8326  2218   192336.8966  1248   194300.0000   167235.1237  6173034   13.93%
   8358  2214   182508.8698  1252   194300.0000   167235.1237  6211556   13.93%
   8527  2189   180082.7236   799   194300.0000   169349.5766  6337338   12.84%
Elapsed real time = 715.23 sec. (tree size = 90.71 MB, solutions = 24)
   8610  2191        cutoff         194300.0000   169349.5766  6445390   12.84%
   8652  2171   193029.0227  1196   194300.0000   169349.5766  6492786   12.84%
   8739  2152   186431.2614  1127   194300.0000   169349.5766  6613980   12.84%
   8911  2067   191510.6144   857   194300.0000   172776.9338  6771384   11.08%
   8962  2042   190355.8609  1131   194300.0000   172776.9338  6822660   11.08%
   9030  1999   188378.3241  1072   194300.0000   172776.9338  6855189   11.08%
   9098  1978   192879.2431   924   194300.0000   172776.9338  6908964   11.08%
   9290  1860   182145.1856  1083   194300.0000   172776.9338  7051084   11.08%
   9348  1850        cutoff         194300.0000   175590.7478  7110666    9.63%
   9544  1745   193777.2398   953   194300.0000   175590.7478  7253909    9.63%
Elapsed real time = 805.49 sec. (tree size = 68.57 MB, solutions = 24)
   9704  1637        cutoff         194300.0000   175590.7478  7350759    9.63%
   9828  1582   191217.1071   991   194300.0000   175590.7478  7453612    9.63%
  10145  1359   190776.3972   938   194300.0000   182654.2388  7650342    5.99%
  10505  1033        cutoff         194300.0000   182654.2388  7779283    5.99%
  11140   428        cutoff         194300.0000   189406.4841  7907729    2.52%

GUB cover cuts applied:  1176
Clique cuts applied:  398
Cover cuts applied:  5238
Implied bound cuts applied:  4
Zero-half cuts applied:  680
Gomory fractional cuts applied:  41

Root node processing (before b&c):
  Real time             =    0.61
Parallel b&c, 4 threads:
  Real time             =  854.12
  Sync time (average)   =   17.11
  Wait time (average)   =   38.30
                          -------
Total (root+branch&cut) =  854.73 sec.
```

```
Solution status = Optimal
Solution value  = 194300
Variables:
ne0t0_g0 = 0.000000 ne0t1_g0 = 0.000000 ne0t2_g0 = 0.000000 ne0t3_g0 = 0.000000
ne0t0_g1 = 0.000000 ne0t1_g1 = 0.000000 ne0t2_g1 = 1.000000 ne0t3_g1 = 0.000000
ne0t0_g2 = 0.000000 ne0t1_g2 = 0.000000 ne0t2_g2 = 0.000000 ne0t3_g2 = 0.000000
ne0t0_g3 = 0.000000 ne0t1_g3 = 0.000000 ne0t2_g3 = 0.000000 ne0t3_g3 = 0.000000
ne0t0_g4 = 0.000000 ne0t1_g4 = 0.000000 ne0t2_g4 = 0.000000 ne0t3_g4 = 0.000000
ne0t0_g5 = 0.000000 ne0t1_g5 = 0.000000 ne0t2_g5 = 0.000000 ne0t3_g5 = 0.000000
ne0t0_g6 = 0.000000 ne0t1_g6 = 0.000000 ne0t2_g6 = 0.000000 ne0t3_g6 = 0.000000
ne0t0_g7 = 0.000000 ne0t1_g7 = 0.000000 ne0t2_g7 = 0.000000 ne0t3_g7 = 0.000000
ne1t0_g0 = 0.000000 ne1t1_g0 = 0.000000 ne1t2_g0 = 0.000000 ne1t3_g0 = 0.000000
ne1t0_g1 = 1.000000 ne1t1_g1 = 0.000000 ne1t2_g1 = 0.000000 ne1t3_g1 = 0.000000
ne1t0_g2 = 0.000000 ne1t1_g2 = 0.000000 ne1t2_g2 = 0.000000 ne1t3_g2 = 0.000000
ne1t0_g3 = 1.000000 ne1t1_g3 = 0.000000 ne1t2_g3 = 0.000000 ne1t3_g3 = 0.000000
ne1t0_g4 = 0.000000 ne1t1_g4 = 0.000000 ne1t2_g4 = 0.000000 ne1t3_g4 = 0.000000
ne1t0_g5 = 0.000000 ne1t1_g5 = 0.000000 ne1t2_g5 = 0.000000 ne1t3_g5 = 0.000000
ne1t0_g6 = 0.000000 ne1t1_g6 = 0.000000 ne1t2_g6 = 0.000000 ne1t3_g6 = 0.000000
ne1t0_g7 = 0.000000 ne1t1_g7 = 0.000000 ne1t2_g7 = 0.000000 ne1t3_g7 = 0.000000
ne2t0_g0 = 0.000000 ne2t1_g0 = 0.000000 ne2t2_g0 = 1.000000 ne2t3_g0 = 0.000000
ne2t0_g1 = 0.000000 ne2t1_g1 = 0.000000 ne2t2_g1 = 0.000000 ne2t3_g1 = 0.000000
ne2t0_g2 = 0.000000 ne2t1_g2 = 0.000000 ne2t2_g2 = 0.000000 ne2t3_g2 = 0.000000
ne2t0_g3 = 0.000000 ne2t1_g3 = 0.000000 ne2t2_g3 = 0.000000 ne2t3_g3 = 0.000000
ne2t0_g4 = 0.000000 ne2t1_g4 = 0.000000 ne2t2_g4 = 0.000000 ne2t3_g4 = 0.000000
ne2t0_g5 = 0.000000 ne2t1_g5 = 0.000000 ne2t2_g5 = 0.000000 ne2t3_g5 = 0.000000
ne2t0_g6 = 0.000000 ne2t1_g6 = 0.000000 ne2t2_g6 = 0.000000 ne2t3_g6 = 0.000000
ne2t0_g7 = 0.000000 ne2t1_g7 = 0.000000 ne2t2_g7 = 0.000000 ne2t3_g7 = 0.000000
ne3t0_g0 = 0.000000 ne3t1_g0 = 0.000000 ne3t2_g0 = 0.000000 ne3t3_g0 = 0.000000
ne3t0_g1 = 1.000000 ne3t1_g1 = 0.000000 ne3t2_g1 = 0.000000 ne3t3_g1 = 0.000000
ne3t0_g2 = 1.000000 ne3t1_g2 = 0.000000 ne3t2_g2 = 0.000000 ne3t3_g2 = 0.000000
ne3t0_g3 = 0.000000 ne3t1_g3 = 0.000000 ne3t2_g3 = 0.000000 ne3t3_g3 = 0.000000
ne3t0_g4 = 0.000000 ne3t1_g4 = 0.000000 ne3t2_g4 = 0.000000 ne3t3_g4 = 0.000000
ne3t0_g5 = 0.000000 ne3t1_g5 = 0.000000 ne3t2_g5 = 0.000000 ne3t3_g5 = 0.000000
ne3t0_g6 = 0.000000 ne3t1_g6 = 0.000000 ne3t2_g6 = 0.000000 ne3t3_g6 = 0.000000
ne3t0_g7 = 0.000000 ne3t1_g7 = 0.000000 ne3t2_g7 = 0.000000 ne3t3_g7 = 0.000000
ne4t0_g0 = 0.000000 ne4t1_g0 = 0.000000 ne4t2_g0 = 0.000000 ne4t3_g0 = 0.000000
ne4t0_g1 = 0.000000 ne4t1_g1 = 0.000000 ne4t2_g1 = 0.000000 ne4t3_g1 = 0.000000
ne4t0_g2 = 0.000000 ne4t1_g2 = 0.000000 ne4t2_g2 = 0.000000 ne4t3_g2 = 0.000000
ne4t0_g3 = 0.000000 ne4t1_g3 = 0.000000 ne4t2_g3 = 0.000000 ne4t3_g3 = 0.000000
ne4t0_g4 = 0.000000 ne4t1_g4 = 0.000000 ne4t2_g4 = 0.000000 ne4t3_g4 = 0.000000
ne4t0_g5 = 0.000000 ne4t1_g5 = 0.000000 ne4t2_g5 = 0.000000 ne4t3_g5 = 0.000000
ne4t0_g6 = 0.000000 ne4t1_g6 = 0.000000 ne4t2_g6 = 0.000000 ne4t3_g6 = 0.000000
ne4t0_g7 = 0.000000 ne4t1_g7 = 0.000000 ne4t2_g7 = 0.000000 ne4t3_g7 = 0.000000
ne5t0_g0 = 0.000000 ne5t1_g0 = 0.000000 ne5t2_g0 = 0.000000 ne5t3_g0 = 0.000000
ne5t0_g1 = 0.000000 ne5t1_g1 = 0.000000 ne5t2_g1 = 0.000000 ne5t3_g1 = 0.000000
ne5t0_g2 = 0.000000 ne5t1_g2 = 0.000000 ne5t2_g2 = 0.000000 ne5t3_g2 = 0.000000
ne5t0_g3 = 0.000000 ne5t1_g3 = 0.000000 ne5t2_g3 = 0.000000 ne5t3_g3 = 0.000000
ne5t0_g4 = 0.000000 ne5t1_g4 = 0.000000 ne5t2_g4 = 0.000000 ne5t3_g4 = 0.000000
ne5t0_g5 = 0.000000 ne5t1_g5 = 0.000000 ne5t2_g5 = 0.000000 ne5t3_g5 = 0.000000
ne5t0_g6 = 0.000000 ne5t1_g6 = 0.000000 ne5t2_g6 = 0.000000 ne5t3_g6 = 0.000000
ne5t0_g7 = 0.000000 ne5t1_g7 = 0.000000 ne5t2_g7 = 0.000000 ne5t3_g7 = 0.000000
ne6t0_g0 = 0.000000 ne6t1_g0 = 0.000000 ne6t2_g0 = 0.000000 ne6t3_g0 = 0.000000
ne6t0_g1 = 0.000000 ne6t1_g1 = 0.000000 ne6t2_g1 = 0.000000 ne6t3_g1 = 0.000000
ne6t0_g2 = 0.000000 ne6t1_g2 = 0.000000 ne6t2_g2 = 0.000000 ne6t3_g2 = 0.000000
ne6t0_g3 = 0.000000 ne6t1_g3 = 0.000000 ne6t2_g3 = 1.000000 ne6t3_g3 = 0.000000
ne6t0_g4 = 0.000000 ne6t1_g4 = 0.000000 ne6t2_g4 = 0.000000 ne6t3_g4 = 0.000000
ne6t0_g5 = 0.000000 ne6t1_g5 = 0.000000 ne6t2_g5 = 0.000000 ne6t3_g5 = 0.000000
ne6t0_g6 = 0.000000 ne6t1_g6 = 0.000000 ne6t2_g6 = 0.000000 ne6t3_g6 = 0.000000
ne6t0_g7 = 0.000000 ne6t1_g7 = 0.000000 ne6t2_g7 = 0.000000 ne6t3_g7 = 0.000000
ne7t0_g0 = 0.000000 ne7t1_g0 = 0.000000 ne7t2_g0 = 0.000000 ne7t3_g0 = 0.000000
ne7t0_g1 = 0.000000 ne7t1_g1 = 0.000000 ne7t2_g1 = 0.000000 ne7t3_g1 = 0.000000
ne7t0_g2 = 0.000000 ne7t1_g2 = 0.000000 ne7t2_g2 = 0.000000 ne7t3_g2 = 0.000000
ne7t0_g3 = 0.000000 ne7t1_g3 = 0.000000 ne7t2_g3 = 0.000000 ne7t3_g3 = 0.000000
ne7t0_g4 = 0.000000 ne7t1_g4 = 0.000000 ne7t2_g4 = 0.000000 ne7t3_g4 = 0.000000
ne7t0_g5 = 0.000000 ne7t1_g5 = 0.000000 ne7t2_g5 = 0.000000 ne7t3_g5 = 0.000000
ne7t0_g6 = 0.000000 ne7t1_g6 = 0.000000 ne7t2_g6 = 0.000000 ne7t3_g6 = 0.000000
ne7t0_g7 = 0.000000 ne7t1_g7 = 0.000000 ne7t2_g7 = 0.000000 ne7t3_g7 = 0.000000
ne8t0_g0 = 0.000000 ne8t1_g0 = 0.000000 ne8t2_g0 = 0.000000 ne8t3_g0 = 0.000000
ne8t0_g1 = 0.000000 ne8t1_g1 = 0.000000 ne8t2_g1 = 0.000000 ne8t3_g1 = 0.000000
ne8t0_g2 = 0.000000 ne8t1_g2 = 0.000000 ne8t2_g2 = 1.000000 ne8t3_g2 = 0.000000
```

```
ne8t0_g3 = 0.000000 ne8t1_g3 = 0.000000 ne8t2_g3 = 0.000000 ne8t3_g3 = 0.000000
ne8t0_g4 = 0.000000 ne8t1_g4 = 0.000000 ne8t2_g4 = 0.000000 ne8t3_g4 = 0.000000
ne8t0_g5 = 0.000000 ne8t1_g5 = 0.000000 ne8t2_g5 = 0.000000 ne8t3_g5 = 0.000000
ne8t0_g6 = 0.000000 ne8t1_g6 = 0.000000 ne8t2_g6 = 0.000000 ne8t3_g6 = 0.000000
ne8t0_g7 = 0.000000 ne8t1_g7 = 0.000000 ne8t2_g7 = 0.000000 ne8t3_g7 = 0.000000
ne9t0_g0 = 0.000000 ne9t1_g0 = 0.000000 ne9t2_g0 = 0.000000 ne9t3_g0 = 0.000000
ne9t0_g1 = 0.000000 ne9t1_g1 = 0.000000 ne9t2_g1 = 0.000000 ne9t3_g1 = 0.000000
ne9t0_g2 = 0.000000 ne9t1_g2 = 0.000000 ne9t2_g2 = 0.000000 ne9t3_g2 = 0.000000
ne9t0_g3 = 0.000000 ne9t1_g3 = 0.000000 ne9t2_g3 = 0.000000 ne9t3_g3 = 0.000000
ne9t0_g4 = 0.000000 ne9t1_g4 = 0.000000 ne9t2_g4 = 0.000000 ne9t3_g4 = 0.000000
ne9t0_g5 = 0.000000 ne9t1_g5 = 0.000000 ne9t2_g5 = 0.000000 ne9t3_g5 = 0.000000
ne9t0_g6 = 0.000000 ne9t1_g6 = 0.000000 ne9t2_g6 = 0.000000 ne9t3_g6 = 0.000000
ne9t0_g7 = 0.000000 ne9t1_g7 = 0.000000 ne9t2_g7 = 0.000000 ne9t3_g7 = 0.000000
ne10t0_g0 = 0.000000 ne10t1_g0 = 0.000000 ne10t2_g0 = 0.000000 ne10t3_g0 = 0.000000
ne10t0_g1 = 0.000000 ne10t1_g1 = 0.000000 ne10t2_g1 = 0.000000 ne10t3_g1 = 0.000000
ne10t0_g2 = 0.000000 ne10t1_g2 = 0.000000 ne10t2_g2 = 0.000000 ne10t3_g2 = 0.000000
ne10t0_g3 = 0.000000 ne10t1_g3 = 0.000000 ne10t2_g3 = 0.000000 ne10t3_g3 = 0.000000
ne10t0_g4 = 0.000000 ne10t1_g4 = 0.000000 ne10t2_g4 = 0.000000 ne10t3_g4 = 0.000000
ne10t0_g5 = 0.000000 ne10t1_g5 = 0.000000 ne10t2_g5 = 0.000000 ne10t3_g5 = 0.000000
ne10t0_g6 = 0.000000 ne10t1_g6 = 0.000000 ne10t2_g6 = 0.000000 ne10t3_g6 = 0.000000
ne10t0_g7 = 0.000000 ne10t1_g7 = 0.000000 ne10t2_g7 = 0.000000 ne10t3_g7 = 0.000000
ne11t0_g0 = 0.000000 ne11t1_g0 = 0.000000 ne11t2_g0 = 0.000000 ne11t3_g0 = 0.000000
ne11t0_g1 = 0.000000 ne11t1_g1 = 0.000000 ne11t2_g1 = 0.000000 ne11t3_g1 = 0.000000
ne11t0_g2 = 0.000000 ne11t1_g2 = 0.000000 ne11t2_g2 = 0.000000 ne11t3_g2 = 0.000000
ne11t0_g3 = 0.000000 ne11t1_g3 = 0.000000 ne11t2_g3 = 0.000000 ne11t3_g3 = 0.000000
ne11t0_g4 = 0.000000 ne11t1_g4 = 0.000000 ne11t2_g4 = 0.000000 ne11t3_g4 = 0.000000
ne11t0_g5 = 0.000000 ne11t1_g5 = 0.000000 ne11t2_g5 = 0.000000 ne11t3_g5 = 0.000000
ne11t0_g6 = 0.000000 ne11t1_g6 = 0.000000 ne11t2_g6 = 0.000000 ne11t3_g6 = 0.000000
ne11t0_g7 = 0.000000 ne11t1_g7 = 0.000000 ne11t2_g7 = 0.000000 ne11t3_g7 = 0.000000
ne12t0_g0 = 0.000000 ne12t1_g0 = 0.000000 ne12t2_g0 = 0.000000 ne12t3_g0 = 0.000000
ne12t0_g1 = 0.000000 ne12t1_g1 = 0.000000 ne12t2_g1 = 0.000000 ne12t3_g1 = 0.000000
ne12t0_g2 = 0.000000 ne12t1_g2 = 0.000000 ne12t2_g2 = 0.000000 ne12t3_g2 = 0.000000
ne12t0_g3 = 0.000000 ne12t1_g3 = 0.000000 ne12t2_g3 = 0.000000 ne12t3_g3 = 0.000000
ne12t0_g4 = 0.000000 ne12t1_g4 = 0.000000 ne12t2_g4 = 0.000000 ne12t3_g4 = 0.000000
ne12t0_g5 = 0.000000 ne12t1_g5 = 0.000000 ne12t2_g5 = 0.000000 ne12t3_g5 = 0.000000
ne12t0_g6 = 0.000000 ne12t1_g6 = 0.000000 ne12t2_g6 = 0.000000 ne12t3_g6 = 0.000000
ne12t0_g7 = 0.000000 ne12t1_g7 = 0.000000 ne12t2_g7 = 0.000000 ne12t3_g7 = 0.000000
ne13t0_g0 = 0.000000 ne13t1_g0 = 0.000000 ne13t2_g0 = 0.000000 ne13t3_g0 = 0.000000
ne13t0_g1 = 0.000000 ne13t1_g1 = 0.000000 ne13t2_g1 = 0.000000 ne13t3_g1 = 0.000000
ne13t0_g2 = 0.000000 ne13t1_g2 = 0.000000 ne13t2_g2 = 0.000000 ne13t3_g2 = 0.000000
ne13t0_g3 = 0.000000 ne13t1_g3 = 0.000000 ne13t2_g3 = 0.000000 ne13t3_g3 = 0.000000
ne13t0_g4 = 0.000000 ne13t1_g4 = 0.000000 ne13t2_g4 = 0.000000 ne13t3_g4 = 0.000000
ne13t0_g5 = 0.000000 ne13t1_g5 = 0.000000 ne13t2_g5 = 0.000000 ne13t3_g5 = 0.000000
ne13t0_g6 = 0.000000 ne13t1_g6 = 0.000000 ne13t2_g6 = 0.000000 ne13t3_g6 = 0.000000
ne13t0_g7 = 0.000000 ne13t1_g7 = 0.000000 ne13t2_g7 = 0.000000 ne13t3_g7 = 0.000000
ne14t0_g0 = 0.000000 ne14t1_g0 = 0.000000 ne14t2_g0 = 0.000000 ne14t3_g0 = 0.000000
ne14t0_g1 = 0.000000 ne14t1_g1 = 0.000000 ne14t2_g1 = 0.000000 ne14t3_g1 = 0.000000
ne14t0_g2 = 0.000000 ne14t1_g2 = 0.000000 ne14t2_g2 = 0.000000 ne14t3_g2 = 0.000000
ne14t0_g3 = 0.000000 ne14t1_g3 = 0.000000 ne14t2_g3 = 0.000000 ne14t3_g3 = 0.000000
ne14t0_g4 = 0.000000 ne14t1_g4 = 0.000000 ne14t2_g4 = 0.000000 ne14t3_g4 = 1.000000
ne14t0_g5 = 0.000000 ne14t1_g5 = 0.000000 ne14t2_g5 = 0.000000 ne14t3_g5 = 0.000000
ne14t0_g6 = 0.000000 ne14t1_g6 = 0.000000 ne14t2_g6 = 0.000000 ne14t3_g6 = 0.000000
ne14t0_g7 = 0.000000 ne14t1_g7 = 0.000000 ne14t2_g7 = 0.000000 ne14t3_g7 = 0.000000
ne15t0_g0 = 0.000000 ne15t1_g0 = 0.000000 ne15t2_g0 = 0.000000 ne15t3_g0 = 0.000000
ne15t0_g1 = 0.000000 ne15t1_g1 = 0.000000 ne15t2_g1 = 0.000000 ne15t3_g1 = 0.000000
ne15t0_g2 = 0.000000 ne15t1_g2 = 0.000000 ne15t2_g2 = 0.000000 ne15t3_g2 = 0.000000
ne15t0_g3 = 0.000000 ne15t1_g3 = 0.000000 ne15t2_g3 = 0.000000 ne15t3_g3 = 0.000000
ne15t0_g4 = 0.000000 ne15t1_g4 = 0.000000 ne15t2_g4 = 0.000000 ne15t3_g4 = 0.000000
ne15t0_g5 = 0.000000 ne15t1_g5 = 0.000000 ne15t2_g5 = 1.000000 ne15t3_g5 = 0.000000
ne15t0_g6 = 0.000000 ne15t1_g6 = 0.000000 ne15t2_g6 = 0.000000 ne15t3_g6 = 0.000000
ne15t0_g7 = 0.000000 ne15t1_g7 = 0.000000 ne15t2_g7 = 0.000000 ne15t3_g7 = 0.000000
ne16t0_g0 = 0.000000 ne16t1_g0 = 0.000000 ne16t2_g0 = 0.000000 ne16t3_g0 = 0.000000
ne16t0_g1 = 0.000000 ne16t1_g1 = 0.000000 ne16t2_g1 = 0.000000 ne16t3_g1 = 0.000000
ne16t0_g2 = 0.000000 ne16t1_g2 = 0.000000 ne16t2_g2 = 0.000000 ne16t3_g2 = 0.000000
ne16t0_g3 = 0.000000 ne16t1_g3 = 0.000000 ne16t2_g3 = 0.000000 ne16t3_g3 = 0.000000
ne16t0_g4 = 0.000000 ne16t1_g4 = 0.000000 ne16t2_g4 = 0.000000 ne16t3_g4 = 0.000000
ne16t0_g5 = 1.000000 ne16t1_g5 = 0.000000 ne16t2_g5 = 0.000000 ne16t3_g5 = 0.000000
ne16t0_g6 = 1.000000 ne16t1_g6 = 0.000000 ne16t2_g6 = 0.000000 ne16t3_g6 = 0.000000
ne16t0_g7 = 0.000000 ne16t1_g7 = 0.000000 ne16t2_g7 = 0.000000 ne16t3_g7 = 0.000000
ne17t0_g0 = 0.000000 ne17t1_g0 = 0.000000 ne17t2_g0 = 0.000000 ne17t3_g0 = 0.000000
ne17t0_g1 = 0.000000 ne17t1_g1 = 0.000000 ne17t2_g1 = 0.000000 ne17t3_g1 = 0.000000
```

```
ne17t0_g2 = 0.000000 ne17t1_g2 = 0.000000 ne17t2_g2 = 0.000000 ne17t3_g2 = 0.000000
ne17t0_g3 = 0.000000 ne17t1_g3 = 0.000000 ne17t2_g3 = 0.000000 ne17t3_g3 = 0.000000
ne17t0_g4 = 0.000000 ne17t1_g4 = 0.000000 ne17t2_g4 = 0.000000 ne17t3_g4 = 0.000000
ne17t0_g5 = 0.000000 ne17t1_g5 = 0.000000 ne17t2_g5 = 0.000000 ne17t3_g5 = 0.000000
ne17t0_g6 = 0.000000 ne17t1_g6 = 0.000000 ne17t2_g6 = 0.000000 ne17t3_g6 = 0.000000
ne17t0_g7 = 0.000000 ne17t1_g7 = 0.000000 ne17t2_g7 = 0.000000 ne17t3_g7 = 0.000000
ne18t0_g0 = 0.000000 ne18t1_g0 = 0.000000 ne18t2_g0 = 0.000000 ne18t3_g0 = 0.000000
ne18t0_g1 = 0.000000 ne18t1_g1 = 0.000000 ne18t2_g1 = 0.000000 ne18t3_g1 = 0.000000
ne18t0_g2 = 0.000000 ne18t1_g2 = 0.000000 ne18t2_g2 = 0.000000 ne18t3_g2 = 0.000000
ne18t0_g3 = 0.000000 ne18t1_g3 = 0.000000 ne18t2_g3 = 0.000000 ne18t3_g3 = 0.000000
ne18t0_g4 = 0.000000 ne18t1_g4 = 0.000000 ne18t2_g4 = 0.000000 ne18t3_g4 = 0.000000
ne18t0_g5 = 0.000000 ne18t1_g5 = 0.000000 ne18t2_g5 = 0.000000 ne18t3_g5 = 0.000000
ne18t0_g6 = 0.000000 ne18t1_g6 = 0.000000 ne18t2_g6 = 0.000000 ne18t3_g6 = 1.000000
ne18t0_g7 = 0.000000 ne18t1_g7 = 0.000000 ne18t2_g7 = 0.000000 ne18t3_g7 = 0.000000
ne19t0_g0 = 0.000000 ne19t1_g0 = 0.000000 ne19t2_g0 = 0.000000 ne19t3_g0 = 0.000000
ne19t0_g1 = 0.000000 ne19t1_g1 = 0.000000 ne19t2_g1 = 0.000000 ne19t3_g1 = 0.000000
ne19t0_g2 = 0.000000 ne19t1_g2 = 0.000000 ne19t2_g2 = 0.000000 ne19t3_g2 = 0.000000
ne19t0_g3 = 0.000000 ne19t1_g3 = 0.000000 ne19t2_g3 = 0.000000 ne19t3_g3 = 0.000000
ne19t0_g4 = 0.000000 ne19t1_g4 = 0.000000 ne19t2_g4 = 0.000000 ne19t3_g4 = 0.000000
ne19t0_g5 = 0.000000 ne19t1_g5 = 0.000000 ne19t2_g5 = 0.000000 ne19t3_g5 = 0.000000
ne19t0_g6 = 0.000000 ne19t1_g6 = 0.000000 ne19t2_g6 = 0.000000 ne19t3_g6 = 0.000000
ne19t0_g7 = 0.000000 ne19t1_g7 = 0.000000 ne19t2_g7 = 0.000000 ne19t3_g7 = 0.000000
ne20t0_g0 = 0.000000 ne20t1_g0 = 0.000000 ne20t2_g0 = 0.000000 ne20t3_g0 = 0.000000
ne20t0_g1 = 0.000000 ne20t1_g1 = 0.000000 ne20t2_g1 = 0.000000 ne20t3_g1 = 0.000000
ne20t0_g2 = 0.000000 ne20t1_g2 = 0.000000 ne20t2_g2 = 0.000000 ne20t3_g2 = 0.000000
ne20t0_g3 = 0.000000 ne20t1_g3 = 0.000000 ne20t2_g3 = 0.000000 ne20t3_g3 = 0.000000
ne20t0_g4 = 0.000000 ne20t1_g4 = 0.000000 ne20t2_g4 = 0.000000 ne20t3_g4 = 0.000000
ne20t0_g5 = 0.000000 ne20t1_g5 = 0.000000 ne20t2_g5 = 0.000000 ne20t3_g5 = 0.000000
ne20t0_g6 = 0.000000 ne20t1_g6 = 0.000000 ne20t2_g6 = 0.000000 ne20t3_g6 = 0.000000
ne20t0_g7 = 0.000000 ne20t1_g7 = 0.000000 ne20t2_g7 = 0.000000 ne20t3_g7 = 1.000000
ye0_g0 = 0.000000 wt0_e0 = 0.000000 wt1_e0 = 0.000000 wt2_e0 = 1.000000
wt3_e0 = 0.000000 ye0_g1 = 1.000000 ye0_g2 = 0.000000 ye0_g3 = 0.000000
ye0_g4 = 0.000000 ye0_g5 = 0.000000 ye0_g6 = 0.000000 ye0_g7 = 0.000000
.....
ye20_g0 = 0.000000 wt0_e20 = 0.000000 wt1_e20 = 0.000000 wt2_e20 = 0.000000
wt3_e20 = 1.000000 ye20_g1 = 0.000000 ye20_g2 = 0.000000 ye20_g3 = 0.000000
ye20_g4 = 0.000000 ye20_g5 = 0.000000 ye20_g6 = 0.000000 ye20_g7 = 1.000000
Se0 = 1.000000 Se1 = 1.000000 Se2 = 1.000000 Se3 = 1.000000
Se4 = 0.000000 Se5 = 0.000000 Se6 = 1.000000 Se7 = 0.000000
Se8 = 1.000000 Se9 = 0.000000 Se10 = 0.000000 Se11 = 0.000000
Se12 = 0.000000 Se13 = 0.000000 Se14 = 1.000000 Se15 = 1.000000
Se16 = 1.000000 Se17 = 0.000000 Se18 = 1.000000 Se19 = 0.000000
Se20 = 1.000000 gamaTe0_p2 = 0.000000 gamaTe0_p3 = 0.000000 gamaTe0_p4 = 0.000000
gamaTe0_p5 = 0.000000 gamaTe0_p6 = 0.000000 gamaTe1_p1 = 1.000000 gamaTe1_p3 = 0.000000
gamaTe1_p4 = 0.000000 gamaTe1_p5 = 0.000000 gamaTe1_p6 = 0.000000 gamaTe2_p1 = 0.000000
gamaTe2_p2 = 0.000000 gamaTe2_p4 = 0.000000 gamaTe2_p5 = 0.000000 gamaTe2_p6 = 0.000000
.....
xe8g0_e7 = 0.000000 xe13g0_e7 = 0.000000 xe14g0_e7 = 0.000000 xe15g0_e7 = 0.000000
xe18g0_e7 = 0.000000 xe20g0_e7 = 0.000000 xe2g0_e8 = 1.000000 xe3g0_e8 = 1.000000
xe8g0_e8 = 1.000000 xe13g0_e8 = 0.000000 xe14g0_e8 = 0.000000 xe15g0_e8 = 0.000000
.....
gamaDFe3g1_p2 = 0.000000 gamaDFe3g1_p3 = 1.000000 gamaDFe3g1_p5 = 0.000000 gamaDFe3g1_p6 = 0.000000
gamaDFe8g1_p0 = 0.000000 gamaDFe8g1_p2 = 0.000000 gamaDFe8g1_p3 = 0.000000 gamaDFe8g1_p5 = 0.000000
gamaDFe8g1_p6 = 0.000000 gamaDFe13g1_p0 = 0.000000 gamaDFe13g1_p1 = 0.000000 gamaDFe13g1_p3 = 0.000000
.....
xe18_e5 = 0.000000 xe20_e5 = 0.000000 xe2_e6 = 0.000000 xe3_e6 = 0.000000
xe8_e6 = 0.000000 xe13_e6 = 0.000000 xe14_e6 = 0.000000 xe15_e6 = 0.000000
xe18_e6    =    0.000000    xe20_e6    =    0.000000    xe2_e7    =    0.000000    xe3_e7    =    0.000000
.....
gamaDe18_p0 = 0.000000 gamaDe18_p1 = 0.000000 gamaDe18_p2 = 0.000000 gamaDe18_p3 = 0.000000
gamaDe18_p6 = 0.000000 gamaDe20_p0 = 0.000000 gamaDe20_p1 = 0.000000 gamaDe20_p2 = 0.000000
gamaDe20_p3 = 0.000000 gamaDe20_p4 = 0.000000

Maximum bound violation = 0
```

11. Anexo V

En este Anexo se incluye el archivo de entrada con los datos del caso de prueba 3, así como también el archivo de salida de cada una de las soluciones implementadas, resultado de su ejecución para este caso de prueba. Debido a lo extenso de estos archivos no se incluyen en su totalidad en este informe, se agregó una línea de puntos en los lugares donde se suprimieron líneas.

Caso de prueba 3

1. Archivo de entrada

```
-------------------------------------------
- DATOS_GRAFO_TRANSPORTE                   -
-------------------------------------------


CANT_NODOS_TRANSPORTE: 9
CANT_LINKS_TRANSPORTE: 13

-------------------------------------------
- LINKS_CON_DISTANCIAS_ASOCIADAS           -
-------------------------------------------

LINK: 0 1 10
LINK: 0 3 20
LINK: 1 2 12
LINK: 1 4 15
LINK: 2 3 18
LINK: 2 5 14
LINK: 3 6 17
LINK: 4 5 8
LINK: 4 7 16
LINK: 5 6 13
LINK: 5 7 11
LINK: 6 8 6
LINK: 7 8 9

-------------------------------------------
- TECNOLOGÍAS                              -
-------------------------------------------

CANT_TECNOLOGIAS: 8
TECNOLOGIA: 0 0
TECNOLOGIA: 1 100
TECNOLOGIA: 2 180
TECNOLOGIA: 5 450
TECNOLOGIA: 10 800
TECNOLOGIA: 20 1600
TECNOLOGIA: 40 3000
TECNOLOGIA: 60 3500

-------------------------------------------
- CLIENTES                                 -
-------------------------------------------

CANT_CLIENTES: 9
TNS_0: 0
TNS_1: 1
```

```
TNS_2: 2
TNS_3: 3
TNS_4: 4
TNS_5: 5
TNS_6: 6
TNS_7: 7
TNS_8: 8

-----------------------------------------
- Matrices                              -
-----------------------------------------

LINK1: 0 1

0 0 0 0 0 0 0 16 0
0 0 0 0 0 20 0 0
0 0 0 0 0 0 0 12
0 18 0 0 0 0
0 0 0 0 0
0 0 0 0
0 0 0
0 0
0

LINK1: 0 3

0 0 0 0 0 0 0 16 0
0 0 0 0 0 20 0 0
0 0 0 0 0 0 0 12
0 18 0 0 0 0
0 0 0 0 0
0 0 0 0
0 0 0
0 0
0


0 0 0 0 0 0 0 16 0
0 0 0 0 0 20 0 0
0 0 0 0 0 0 0 12
0 18 0 0 0 0
0 0 0 0 0
0 0 0 0
0 0 0
0 0
0
```

2. Archivo de salida de la solución aproximada 1

```
****** Ruteos en transporte en el escenario sin fallas ******
Arista (0,1):  0      1
Arista (0,2):  0      1      2
Arista (0,3):  0      3
Arista (0,4):  0      1      4
Arista (0,5):  0      1      4      5
Arista (0,6):  0      3      6
Arista (0,7):  0      1      4      7
Arista (0,8):  0      3      6      8
Arista (1,2):  1      2
```

```
Arista (2,7):  2      5      7
Arista (2,8):  2      5      6      8
Arista (3,4):  3      6      5      4
Arista (3,5):  3      6      5
Arista (3,6):  3      6
```

```
Arista (3,7):  3      6      8      7
Arista (3,8):  3      6      8
Arista (4,5):  4      5
Arista (4,6):  4      5      6
Arista (4,7):  4      7
Arista (4,8):  4      7      8
Arista (5,6):  5      6
Arista (5,7):  5      7
Arista (5,8):  5      6      8
Arista (6,7):  6      8      7
Arista (6,8):  6      8
Arista (7,8):  7      8

****** Ruteos en datos en escenarios de falla ******
Falla link (0,1):
       arista (1,3) es:      1      2      3
       arista (0,7) es:      0      3      7
       arista (0,5) es:      0      3      5
       arista (0,4) es:      0      3      4
       arista (0,2) es:      0      3      2
       arista (0,1) es:      0      3      2      1
Falla link (0,3):
       arista (1,3) es:      1      2      3
       arista (0,8) es:      0      1      8
       arista (0,6) es:      0      1      6
       arista (0,3) es:      0      2      3
Falla link (1,2):
       arista (1,2) es:      1      4      2
       arista (0,2) es:      0      3      2
.....
Falla link (7,8):
       arista (7,8) es:      7      5      8
       arista (6,7) es:      6      5      7
       arista (4,8) es:      4      5      8
       arista (3,7) es:      3      5      7
       arista (1,8) es:      1      5      8

****** Tecnologías asignadas a las aristas y distancias en transporte ******
Arista (0,1):
       tecnologia 0.000000
       Costo Tec. 0.000000
       distancia 10.000000
Arista (0,2):
       tecnologia 20.000000
       Costo Tec. 1600.000000
       distancia 22.000000
Arista (0,3):
       tecnologia 20.000000
       Costo Tec. 1600.000000
       distancia 20.000000
.....
Arista (7,8):
       tecnologia 20.000000
       Costo Tec. 1600.000000
       distancia 9.000000


RESULTADO DE LA FUNCION OBJETIVO = 756800.000000


Desperdicio total de la capacidad de la red = 58.000000
```

3. Archivo de salida de la solución aproximada 2

```
############# Demanda entre Grupos##############
grupo: 1, grupo: 2, demanda: 66.000000
############# Fin de demanda entre Grupos##############


===================================== Lista de grupos ================================================================
Grupo 1
idNodo 8       demanda con grupo1 = 0.000000 demanda con grupo2 = 12.000000
idNodo 7       demanda con grupo1 = 0.000000 demanda con grupo2 = 16.000000
idNodo 6       demanda con grupo1 = 0.000000 demanda con grupo2 = 20.000000
idNodo 4       demanda con grupo1 = 0.000000 demanda con grupo2 = 18.000000

Grupo 2
idNodo 3       demanda con grupo1 = 18.000000       demanda con grupo2 = 0.000000
idNodo 2       demanda con grupo1 = 12.000000       demanda con grupo2 = 0.000000
idNodo 1       demanda con grupo1 = 20.000000       demanda con grupo2 = 0.000000
idNodo 0       demanda con grupo1 = 16.000000       demanda con grupo2 = 0.000000

===================================== Fin Lista de grupos
============================================================


==================================== Lista de aristas caño
============================================================

Grupo 1


Aristas Caño con grupo 2

(6,3)  con tecnologia 7
(4,1)  con tecnologia 7
(4,2)  con tecnologia 5


Grupo 2

Aristas Caño con grupo 1

(1,4)  con tecnologia 7
(3,6)  con tecnologia 7
(2,4)  con tecnologia 5


==================================== Fin Lista de aristas Caño
============================================================


############################################## Red Resultado
################################################################

------------------------------------------------
i: 0, j: 0
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 0, j: 1
distancia :10.000000
Camino: 0      1
idTecnologia :5
```

```
------------------------------------------------
i: 0, j: 2
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 0, j: 3
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 0, j: 4
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 0, j: 5
distancia :50.000000
Camino: 0      3      6      5
idTecnologia :5

------------------------------------------------
i: 0, j: 6
distancia :0.000000
Camino:
idTecnologia :0
```

.....

```
------------------------------------------------
i: 4, j: 5
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 4, j: 6
distancia :31.000000
Camino: 4      7      8      6
idTecnologia :7

------------------------------------------------
i: 4, j: 7
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 4, j: 8
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 5, j: 0
distancia :50.000000
Camino: 5      6      3      0
idTecnologia :5

------------------------------------------------
i: 5, j: 1
distancia :23.000000
Camino: 5      4      1
idTecnologia :5

------------------------------------------------
i: 5, j: 2
```

```
distancia :0.000000
Camino:
idTecnologia :0

.....

------------------------------------------------
i: 8, j: 5
distancia :0.000000
Camino:
idTecnologia :0

------------------------------------------------
i: 8, j: 6
distancia :6.000000
Camino: 6      8
idTecnologia :6

------------------------------------------------
i: 8, j: 7
distancia :9.000000
Camino: 7      8
idTecnologia :6

------------------------------------------------
i: 8, j: 8
distancia :0.000000
Camino:
idTecnologia :0

################################################ Red Resultado FIN
#########################################################


El costo de la solucion es: 652500.000000
```

4. Archivo de salida de la solución utilizando Cplex

```
IBM ILOG License Manager: "IBM ILOG Optimization Suite for Academic Initiative" is accessing CPLEX 12 with
option(s): "e m b q ".
Tried aggregator 1 time.
MIP Presolve eliminated 341 rows and 113 columns.
MIP Presolve modified 1008 coefficients.
Reduced MIP has 14635 rows, 7084 columns, and 41774 nonzeros.
Reduced MIP has 7084 binaries, 0 generals, 0 SOSs, and 0 indicators.
Probing time =    0.01 sec.
Tried aggregator 1 time.
Presolve time =    0.21 sec.
Probing time =    0.02 sec.
Clique table members: 26980.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time =    0.08 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

      0     0        0.0000   242                      0.0000      214
      0     0        0.0000   464                 Cuts: 371      584
      0     0        0.0000   368                 Cuts: 188      857
      0     0        0.0000   632                 Cuts: 476     1359
*     0+    0                       1615040.0000        0.0000     1359  100.00%
*     0+    0                       1352220.0000        0.0000     1359  100.00%
      0     2        0.0000   384  1352220.0000        0.0000     1359  100.00%
Elapsed real time =    1.64 sec. (tree size =  0.01 MB, solutions = 2)
*    12+   12                       1291950.0000        0.0000     3470  100.00%
     44    46   133000.0000   481  1291950.0000        0.0000     6521  100.00%
    187   189   178500.0000   293  1291950.0000        0.0000     9491  100.00%
    326   328   180000.0000   265  1291950.0000        0.0000    13036  100.00%
    408   408   145600.0000   329  1291950.0000        0.0000    14604  100.00%
    510   508   201600.0000   388  1291950.0000        0.0000    16961  100.00%
*   534+  530                       1275300.0000        0.0000    17543  100.00%
    538   536        0.0000   687  1275300.0000        0.0000    22093  100.00%
    539   537        0.0000   681  1275300.0000        0.0000    22100  100.00%
    540   537        0.0000   709  1275300.0000        0.0000    22131  100.00%
    543   539        0.0000   911  1275300.0000        0.0000    23670  100.00%
    550   542        0.0000   894  1275300.0000        0.0000    28097  100.00%
Elapsed real time =   20.89 sec. (tree size = 91.43 MB, solutions = 6)
    586   566        0.0000   753  1275300.0000        0.0000    30445  100.00%
    708   630        0.0000   702  1275300.0000        0.0000    35056  100.00%
    863   723      400.0000   683  1275300.0000        0.0000    44246  100.00%
*   888+  604                       1268780.0000        0.0000    46359  100.00%
*   920+  540                       1240100.0000        0.0000    47089  100.00%
    999   584     7053.1915   787  1240100.0000        0.0000    51584  100.00%
   1121   647   112011.1111   759  1240100.0000        0.0000    67609  100.00%
*  1218+  671                       1175200.0000        0.0000    82306  100.00%
   1266   695   269466.6667   700  1175200.0000        0.0000    89105  100.00%
*  1296+  711                       1132600.0000        0.0000    92288  100.00%
*  1364+  772                       1018000.0000        0.0000   105858  100.00%
   1364   774    51577.7778   826  1018000.0000        0.0000   105858  100.00%
   1528   922   427600.0000   545  1018000.0000        0.0000   134093  100.00%
   1740  1109    80000.0000   932  1018000.0000        0.0000   156736  100.00%
*  1774+ 1137                       1001900.0000        0.0000   166871  100.00%
   1874  1234   240000.0000   819  1001900.0000        0.0000   185576  100.00%
Elapsed real time =   61.08 sec. (tree size =  4.39 MB, solutions = 12)
   1996  1334    42000.0000   936  1001900.0000        0.0000   208831  100.00%
*  1999+ 1335                        995600.0000        0.0000   209955  100.00%
   2092  1425    52657.7778  1040   995600.0000        0.0000   235230  100.00%
   2185  1512   280500.0000   803   995600.0000        0.0000   253359  100.00%
   2280  1598    28800.0000   882   995600.0000        0.0000   274815  100.00%
   2365  1681    40207.4074   778   995600.0000        0.0000   287902  100.00%
*  2386+ 1700                        970400.0000        0.0000   291734  100.00%
```

```
   2536  1840   447880.0000   703   970400.0000        0.0000   312641  100.00%
   2666  1963    68000.0000   840   970400.0000        0.0000   331670  100.00%
   2742  2029    71888.8889   904   970400.0000        0.0000   345308  100.00%
   2849  2126   285777.7778   631   970400.0000        0.0000   367081  100.00%
   3009  2273    48000.0000   820   970400.0000        0.0000   393206  100.00%
Elapsed real time =  93.48 sec. (tree size = 15.04 MB, solutions = 14)
   3065  2328    48685.7143   880   970400.0000        0.0000   401116  100.00%
*  3149+ 2403                       968000.0000        0.0000   412999  100.00%
*  3162+ 2416                       908500.0000        0.0000   415669  100.00%
   3197  2451   304257.0370   703   908500.0000        0.0000   420751  100.00%
   3296  2537    87500.0000   788   908500.0000        0.0000   434546  100.00%
   3526  2744   157260.0000   713   908500.0000        0.0000   461512  100.00%
   3628  2843   392780.0000   597   908500.0000        0.0000   474850  100.00%
   3750  2958   556128.5380   595   908500.0000        0.0000   493734  100.00%
   3989  3174    73500.0000   914   908500.0000        0.0000   516543  100.00%
   4164  3340   148700.0000   788   908500.0000        0.0000   533220  100.00%
   4259  3425    30000.0000   826   908500.0000        0.0000   543175  100.00%
*  4296+ 3459                       894100.0000        0.0000   546889  100.00%
   4445  3603    62000.0000   883   894100.0000        0.0000   560334  100.00%
Elapsed real time = 126.35 sec. (tree size = 20.34 MB, solutions = 17)
*  4451+ 3607                       834900.0000        0.0000   560993  100.00%
   4718  3840   227455.5556   690   834900.0000        0.0000   583128  100.00%
   4905  4017   595018.1818   597   834900.0000        0.0000   601676  100.00%
   5066  4158    69000.0000   824   834900.0000        0.0000   620875  100.00%
   5135  4221   101333.3333   893   834900.0000        0.0000   632023  100.00%
   5386  4457   698869.7561   496   834900.0000        0.0000   661144  100.00%
   5547  4602   107000.0000   902   834900.0000        0.0000   680340  100.00%
   5595  4643   132636.3636   826   834900.0000        0.0000   687761  100.00%
   5733  4773    69000.0000  1015   834900.0000        0.0000   704033  100.00%
   6094  5121   184622.2222   739   834900.0000        0.0000   732731  100.00%
   6174  5198   534822.2222   708   834900.0000        0.0000   741390  100.00%
Elapsed real time = 167.17 sec. (tree size = 26.93 MB, solutions = 19)
   6428  5436    80500.0000   794   834900.0000        0.0000   760900  100.00%
   6639  5628   265500.0000   642   834900.0000        0.0000   783023  100.00%
*  6665+ 5652                       816400.0000        0.0000   788532  100.00%
   6795  5781    52500.0000   891   816400.0000        0.0000   801409  100.00%
   7011  5925   167522.2222   784   816400.0000        0.0000   821322  100.00%
   7195  6101   432837.2549   662   816400.0000        0.0000   837006  100.00%
   7424  6317    39000.0000   853   816400.0000        0.0000   855893  100.00%
   7675  6560   363400.0000   707   816400.0000        0.0000   880414  100.00%
   7719  6600   425703.7037   712   816400.0000        0.0000   884433  100.00%
   7816  6689    57000.0000   974   816400.0000        0.0000   892917  100.00%
   8080  6935   458981.8182   739   816400.0000        0.0000   924415  100.00%
Elapsed real time = 200.81 sec. (tree size = 33.51 MB, solutions = 20)
   8187  7036        0.0000   964   816400.0000        0.0000   932362  100.00%
   8291  7137   234000.0000   845   816400.0000        0.0000   941549  100.00%
   8578  7412   653800.0000   272   816400.0000        0.0000   961529  100.00%
*  9001  7772      integral     0   814000.0000        0.0000   978828  100.00%
   9083  7846   117000.0000   841   814000.0000        0.0000   985560  100.00%
*  9225+ 7979                       809200.0000        0.0000   998513  100.00%
*  9299+ 8047                       776400.0000        0.0000  1006033  100.00%
   9323  8073   334122.2222   672   776400.0000        0.0000  1009069  100.00%
   9560  8199    42000.0000  1001   776400.0000        0.0000  1028331  100.00%
   9608  8246    42000.0000   905   776400.0000        0.0000  1032696  100.00%
   9722  8354    42000.0000   856   776400.0000        0.0000  1043311  100.00%
*  9822+ 8448                       725200.0000        0.0000  1053056  100.00%
   9934  8561   276000.0000   720   725200.0000        0.0000  1067217  100.00%
Elapsed real time = 234.11 sec. (tree size = 40.02 MB, solutions = 24)
  10191  8791   720600.0000   475   725200.0000        0.0000  1092605  100.00%
  10269  8657   143200.0000   958   725200.0000        0.0000  1099971  100.00%
  10361  8739   337200.0000   822   725200.0000        0.0000  1110301  100.00%
  10476  8847   179200.0000  1060   725200.0000        0.0000  1131879  100.00%
  10603  8967   233700.0000   901   725200.0000        0.0000  1152101  100.00%
* 10634+ 8995                       673400.0000        0.0000  1156680  100.00%
  10673  8835   302700.0000  1059   673400.0000        0.0000  1161945  100.00%
  10773  8933   404858.8889  1003   673400.0000        0.0000  1174983  100.00%
  10905  9047   140200.0000  1033   673400.0000        0.0000  1193476  100.00%
  11072  9209   364000.0000   906   673400.0000        0.0000  1218436  100.00%
  11192  9314    93000.0000   959   673400.0000        0.0000  1238518  100.00%
Elapsed real time = 261.69 sec. (tree size = 43.70 MB, solutions = 26)
* 11350+ 9462                       645800.0000        0.0000  1259712  100.00%
```

```
* 11350+ 9462                        631400.0000        0.0000  1259712  100.00%
  11350  9464    93000.0000  1224   631400.0000        0.0000  1259712  100.00%
  11546  9419   108500.0000   933   631400.0000        0.0000  1280097  100.00%
  11682  9538   247722.2222   893   631400.0000        0.0000  1300363  100.00%
* 11750+ 9596                        613400.0000        0.0000  1308737  100.00%
  11940  9713   110200.0000  1041   613400.0000        0.0000  1329249  100.00%
Elapsed real time = 556.94 sec. (tree size = 752.81 MB, solutions = 35)
Nodefile size = 616.92 MB (380.41 MB after compression)
Elapsed real time = 677.26 sec. (tree size = 912.48 MB, solutions = 35)
Elapsed real time = 1033.26 sec. (tree size = 1759.31 MB, solutions = 35)
Nodefile size = 1602.90 MB (1011.78 MB after compression)
Elapsed real time = 1145.46 sec. (tree size = 2788.92 MB, solutions = 37)
Nodefile size = 2636.92 MB (1674.69 MB after compression)
Elapsed real time = 2321.97 sec. (tree size = 8739.31 MB, solutions = 40)
Nodefile size = 8601.47 MB (5590.64 MB after compression)
```

```
  76552 64724   488250.0000   870   516800.0000    35200.0000 12790950   93.19%
    ..........................................................................................................................
  88017 75029    infeasible         516800.0000    37133.3333 14563172   92.81%
  88125 75123   184666.6667   785   516800.0000    37133.3333 14591728   92.81%
* 88549+75475                     512800.0000    38400.0000 14682758   92.51%
  88577 75500   153600.0000   790   512800.0000    38400.0000 14686354   92.51%
  88842 75727   187289.8551   810   512800.0000    38400.0000 14730403   92.51%
    ..........................................................................................................................
 120020 103907   143000.0000   951   512800.0000    42000.0000 19845755   91.81%
Elapsed real time = 4717.82 sec. (tree size = 19976.30 MB, solutions = 43)
Nodefile size = 19823.16 MB (13204.07 MB after compression)
*120366+104215                     509300.0000    42000.0000 19891875   91.75%
 120584 104424   225700.0000   725   509300.0000    42000.0000 19919303   91.75%
 120703 104181   477200.0000   645   509300.0000    42000.0000 19938636   91.75%
*120853+104318                     503800.0000    42000.0000 19958467   91.66%
*120863+104316                     488400.0000    42000.0000 19963634   91.40%
 121115 104545   164000.0000   893   488400.0000    42000.0000 20019303   91.40%
 121236 102059   179411.1603  1013   488400.0000    42000.0000 20039910   91.40%
 121550 102345   337740.4255   812   488400.0000    42000.0000 20116973   91.40%
 121713 102497   255686.5306   914   488400.0000    42000.0000 20139071   91.40%

GUB cover cuts applied:  449
Clique cuts applied:  137
Cover cuts applied:  10130
Implied bound cuts applied:  250
Zero-half cuts applied:  468
Gomory fractional cuts applied:  84

Root node processing (before b&c):
  Real time             =    2.21
Parallel b&c, 4 threads:
  Real time             = 4812.03
  Sync time (average)   =   90.92
  Wait time (average)   =  273.10
                          -------
Total (root+branch&cut) = 4814.24 sec.
Solution status = Feasible
Solution value  = 488400
Variables:
ne0t0_g0 = 0.000000 ne0t1_g0 = 0.000000 ne0t2_g0 = 0.000000 ne0t3_g0 = 0.000000
ne0t4_g0 = 0.000000 ne0t5_g0 = 0.000000 ne0t6_g0 = 0.000000 ne0t7_g0 = 0.000000
ne0t0_g1 = 0.000000 ne0t1_g1 = 0.000000 ne0t2_g1 = 0.000000 ne0t3_g1 = 0.000000
..........................................................................................................................
ne1t4_g12 = 0.000000 ne1t5_g12 = 0.000000 ne1t6_g12 = 0.000000 ne1t7_g12 = 0.000000
ne2t0_g0 = 0.000000 ne2t1_g0 = 0.000000 ne2t2_g0 = 0.000000 ne2t3_g0 = 0.000000
ne2t4_g0 = 0.000000 ne2t5_g0 = 1.000000 ne2t6_g0 = 0.000000 ne2t7_g0 = 0.000000
..........................................................................................................................
ne7t4_g5 = 0.000000 ne7t5_g5 = 0.000000 ne7t6_g5 = 0.000000 ne7t7_g5 = 0.000000
ne7t0_g6 = 0.000000 ne7t1_g6 = 0.000000 ne7t2_g6 = 0.000000 ne7t3_g6 = 0.000000
ne7t4_g6 = 0.000000 ne7t5_g6 = 0.000000 ne7t6_g6 = 0.000000 ne7t7_g6 = 0.000000
..........................................................................................................................
ne13t0_g10 = 0.000000 ne13t1_g10 = 0.000000 ne13t2_g10 = 0.000000 ne13t3_g10 = 0.000000
ne13t4_g10 = 0.000000 ne13t5_g10 = 0.000000 ne13t6_g10 = 0.000000 ne13t7_g10 = 0.000000
ne13t0_g11 = 0.000000 ne13t1_g11 = 0.000000 ne13t2_g11 = 0.000000 ne13t3_g11 = 0.000000
..........................................................................................................................
ne18t0_g5 = 0.000000 ne18t1_g5 = 0.000000 ne18t2_g5 = 0.000000 ne18t3_g5 = 0.000000
ne18t4_g5 = 0.000000 ne18t5_g5 = 0.000000 ne18t6_g5 = 0.000000 ne18t7_g5 = 0.000000
ne18t0_g6 = 0.000000 ne18t1_g6 = 0.000000 ne18t2_g6 = 0.000000 ne18t3_g6 = 0.000000
..........................................................................................................................
ne24t4_g2 = 0.000000 ne24t5_g2 = 0.000000 ne24t6_g2 = 0.000000 ne24t7_g2 = 0.000000
ne24t0_g3 = 0.000000 ne24t1_g3 = 0.000000 ne24t2_g3 = 0.000000 ne24t3_g3 = 0.000000
ne24t4_g3 = 0.000000 ne24t5_g3 = 0.000000 ne24t6_g3 = 0.000000 ne24t7_g3 = 0.000000
..........................................................................................................................
ne30t4_g11 = 0.000000 ne30t5_g11 = 1.000000 ne30t6_g11 = 0.000000 ne30t7_g11 = 0.000000
ne30t0_g12 = 0.000000 ne30t1_g12 = 0.000000 ne30t2_g12 = 0.000000 ne30t3_g12 = 0.000000
ne30t4_g12 = 0.000000 ne30t5_g12 = 1.000000 ne30t6_g12 = 0.000000 ne30t7_g12 = 0.000000
..........................................................................................................................
ne34t0_g6 = 0.000000 ne34t1_g6 = 0.000000 ne34t2_g6 = 0.000000 ne34t3_g6 = 0.000000
ne34t4_g6 = 0.000000 ne34t5_g6 = 0.000000 ne34t6_g6 = 0.000000 ne34t7_g6 = 0.000000
```

```
ne34t0_g7 = 0.000000 ne34t1_g7 = 0.000000 ne34t2_g7 = 0.000000 ne34t3_g7 = 0.000000
.....
ne35t0_g12 = 0.000000 ne35t1_g12 = 0.000000 ne35t2_g12 = 0.000000 ne35t3_g12 = 0.000000
ne35t4_g12 = 0.000000 ne35t5_g12 = 1.000000 ne35t6_g12 = 0.000000 ne35t7_g12 = 0.000000
ye0_g0 = 0.000000 wt0_e0 = 0.000000 wt1_e0 = 0.000000 wt2_e0 = 0.000000
    .....
Se12 = 1.000000 Se13 = 1.000000 Se14 = 1.000000 Se15 = 1.000000
Se16 = 0.000000 Se17 = 1.000000 Se18 = 0.000000 Se19 = 1.000000
Se20 = 0.000000 Se21 = 1.000000 Se22 = 1.000000 Se23 = 1.000000
.....
gamaTe34_p2 = 0.000000 gamaTe34_p3 = 0.000000 gamaTe34_p4 = 0.000000 gamaTe34_p5 = 0.000000
gamaTe34_p7 = 0.000000 gamaTe35_p0 = 0.000000 gamaTe35_p1 = 0.000000 gamaTe35_p2 = 0.000000
gamaTe35_p3 = 0.000000 gamaTe35_p4 = 0.000000 gamaTe35_p5 = 0.000000 gamaTe35_p6 = 0.000000
.....
xe6g12_e33 = 0.000000 xe12g12_e33 = 0.000000 xe20g12_e33 = 0.000000 xe21g12_e33 = 0.000000
xe6g12_e34 = 0.000000 xe12g12_e34 = 1.000000 xe20g12_e34 = 1.000000 xe21g12_e34 = 0.000000
xe6g12_e35 = 0.000000 xe12g12_e35 = 0.000000 xe20g12_e35 = 0.000000 xe21g12_e35 = 0.000000
gamaDFe6g0_p1 = 1.000000 gamaDFe6g0_p2 = 0.000000 gamaDFe6g0_p3 = 0.000000 gamaDFe6g0_p4 = 0.000000
gamaDFe6g0_p5 = 0.000000 gamaDFe6g0_p6 = 0.000000 gamaDFe6g0_p8 = 0.000000 gamaDFe12g0_p0 = 0.000000
gamaDFe12g0_p2 = 1.000000 gamaDFe12g0_p3 = 0.000000 gamaDFe12g0_p4 = 0.000000 gamaDFe12g0_p5 = 1.000000
.....
xe6_e0 = 1.000000 xe12_e0 = 0.000000 xe20_e0 = 0.000000 xe21_e0 = 0.000000
xe6_e1 = 0.000000 xe12_e1 = 0.000000 xe20_e1 = 0.000000 xe21_e1 = 0.000000
xe6_e2 = 0.000000 xe12_e2 = 0.000000 xe20_e2 = 0.000000 xe21_e2 = 0.000000
.....
gamaDe20_p3 = 0.000000 gamaDe20_p4 = 0.000000 gamaDe20_p5 = 1.000000 gamaDe20_p6 = 0.000000
gamaDe20_p7 = 0.000000 gamaDe21_p0 = 0.000000 gamaDe21_p1 = 0.000000 gamaDe21_p2 = 0.000000
gamaDe21_p5 = 1.000000 gamaDe21_p6 = 0.000000 gamaDe21_p7 = 0.000000 gamaDe21_p8 = 0.000000

Maximum bound violation = 0
```

12. Anexo VI

En este Anexo se incluye el archivo de entrada con los datos del caso de prueba 4, así como también el archivo de salida de cada una de las soluciones implementadas, resultado de su ejecución para este caso de prueba. Debido a lo extenso de estos archivos no se incluyen en su totalidad en este informe, se agregó una línea de puntos en los lugares donde se suprimieron líneas.

Caso de prueba 4

Por motivos de confidencialidad no se pueden presentar esta información.

13. Anexo VII

Descripción de las operaciones incluidas en cada módulo de la Primera Solución Aproximada

1. Módulo Main

A continuación se describen las operaciones del módulo Main. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

1.1. Declaración de estructuras

```
//**** Matriz de demandas
Typedef float** Matriz;

//**** Arreglo de matrices
Typedef Matriz * Matrices;
```

1.2. Descripción de las operaciones

1. **float AsignarMinimaTecnologia**(**float** capacidad, **Tecnologias*** tecnologías, **int** cantTecnologias)

Dada una demanda “capacidad” retorna la mínima tecnología que satisface esa demanda.
PRE: las tecnologías deben estar ordenadas de menor a mayor

2. **float costoDeTecnologia**(**float** capacidad, **Tecnologias** * tecnologías)

Devuelve el costo de la tecnología "capacidad".

2. Módulo LeerArchivo

A continuación se describen las operaciones del módulo LeerArchivo. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

2.1. Declaración de estructuras

```
//**** Costo y capacidad de una tecnología
Typedef struct Tecnologias{
         float capacidad;
         float costo;
         };

//**** Representa una arista en el grafo de transporte
struct link{
       int origen;
       int destino;
   };

//**** Es un arreglo en donde el índice del mismo índica el índice en la
//**** estructura "Matrices" en donde está la matriz de demanda para el link
//**** guardado en este arreglo.
Typedef link * Tabla_link_matrices;
```

2.2. Descripción de las operaciones

1. **void CargarEstructura**(**Grafos** &grafos,**Grafo** &grafo, **string** archivo, **int** &cantNodos, **int** &cantClientes, **Tecnologias*** &tecnologías, **int*** &Tabla_TNS,**Matrices** &matrices,**Tabla_link_matrices** &posEnMatrices, **int** &cantLink, **int** &cantTecnologias)

Dado el archivo de entrada, carga las estructuras de datos.

2. **int darPosicionMij**(**Tabla_link_matrices** &posEnMatrices, **int** i , **int** j, **int** cantLink)

Dado un link (i, j) y la cantidad de aristas que tendría el grafo completo de Transporte, devuelve la posición en la estructura Matrices para el link (i, j).

3. Módulo Grafo

A continuación se describen las operaciones del módulo Grafo. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

3.1. Declaración de estructuras

```
//***** Grafo implementado como una lista de adyacencia
Typedef  struct NodoAdy{
         int nodoAdy;
         float distanciaAdy;
         struct NodoAdy *sigAdy;
         };

Typedef NodoAdy* listaAdy;

struct Nodo{
   listaAdy* arrNodosAdy;
   int cantNodos;
   };

Typedef Nodo* Grafo;

//***** Estructura para almacenar los grafos
Typedef Grafo* Grafos;

//***** Definición auxiliar para almacenar los nodos intermedios del
//***** recorrido de un camino en una lista
struct nodoListaDeNodos{
   int nodo;
   float distanciaAlOrigen;
   float demandaAlSig;
   nodoListaDeNodos *sig;
   };

Typedef nodoListaDeNodos* ListaDeNodos;

//***** Estructura para almacenar los caminos mas cortos entre cada par
//***** de nodos TNS(i) y TNS(j)
Typedef ListaDeNodos * Caminos;
```

```
struct aristaEnDatos{
   int origen; //origen de la arista de la red de datos
   int destino; //destino de la arista de la red de datos
   aristaEnDatos *sig;
   };

Typedef aristaEnDatos * aristas;

Typedef aristas * AristasAfectadas;

//***** Definición de tipo auxiliar para Dijkstra
struct nodoArregloTabla{
   bool conocido;
   float distancia;
   int camino;
   };

struct nodoTabla{
     nodoArregloTabla*  arregloTabla;
     int cantNodos;
     };

Typedef nodoTabla* Tabla;
```

3.2. Descripción de las operaciones

1. **void CrearCaminos** (**Caminos** &c , **int** cantAristas)

Dada la cantidad de aristas del grafo completo de datos, crea la estructura caminos “c”.

2. **void CrearAristasAfectadas** (**AristasAfectadas** &A , **int cantLinks**)

Dado la cantidad de link del grafo completo de transporte, crea el array vacío donde se van a almacenar las aristas afectadas ante la caída de cada link de transporte.

3. **void IniciarTabla**(**Grafo** G, **int** vertice, **Tabla** &t)

Dado un nodo y un grafo inicializa la estructura auxiliar Tabla necesaria para ejecutar el algoritmo de Dijkstra.

4. **void Dijkstra**(**Grafo** G, **Tabla** &t)

Modifica la tabla dejando como resultado el camino más corto desde el nodo con el cual está inicializada la tabla hacia todos los demás.

5. **void ImprimirCamino**(**int** inicial, **Tabla** t, **int** vertice, **ListaDeNodos** &L)

Devuelve la lista de Nodos “L” que representan el camino más corto desde “inicial” a “vértice”.

6. **void imprimirCamino**(**FILE*** &salida, **ListaDeNodos** &L)

Graba en el archivo "salida" el camino "L".

7. **void GuardarCamino**(**Caminos** &caminos, **int** posArreglo, **ListaDeNodos** camino)

Guarda en la posición “posArreglo” de “caminos” la lista “camino”.

8. **void GuardarAfectados**(**ListaDeNodos** &camino, **AristasAfectadas** &aristasAfect , **int** i , **int** j , **int** cantLinksCompleto, **int** cantNodos)

Guarda las aristas afectadas ante la caída del link (“i”,”j”) en la estructura “aristAfect”.

9. **void CrearGrafo** (**Grafo** &G , **int** cantNodos)

Dada la cantidad nodos crea la estructura Grafo “G”.

10. **int pertenece**(**int** clienteOrigen, **int** clienteDestino, **aristas** aristasAfectadasLink)

Devuelve true si la arista (clienteOrigen, clienteDestino) pertenece a la lista de aristas “aristasAfectadasLink”.

11. **float distanciaEntreClientes**(**Caminos** &caminos, **int** clienteOrigen , **int** clienteDestino, **int** cantAristasCompleto,**int** cantClientes)

Devuelve la distancia entre un par de clientes, para eso recorre la lista correspondiente de “caminos” y suma las distancias guardadas.

12. **void AgregarAdy** (**int** n1 , **int** n2 , **float** distancia , **Grafo** &G)

Dados dos nodos (“n1” y “n2”) y un costo asociado a dichos nodos agrega una arista que va de “n1” hacia “n2” al grafo “G”.

13. **void rerutear**(**FILE*** &salida, **aristas** &aristasAfectadas,**Grafo** &grafoFallaE,**Matriz** &matrizFallaE,**int** nodoN,**int** nodoM)

Esta función define los caminos nuevos para las aristas afectadas indicadas en "aristas", aristas afectadas por la caída del link (“nodoN”, “nodoM”), las demandas que deben quedar en las aristas se indican en la “matrizFallaE” que originalmente tiene la matriz de demandas para el escenario de falla E. En “grafoFallaE” está el grafo de datos resultado de tirar las aristas afectadas, sobre este se hará el reruteo. Por último se recibe el archivo “salida” que es donde se imprimirá los nuevos caminos para las aristas afectadas.

14. Anexo VIII

Descripción de las operaciones incluidas en cada módulo de la Segunda Solución Aproximada

1. Módulo Main

A continuación se describen las operaciones del módulo Main. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

1.1. Declaración de estructuras

```
//**** Grafo de la lista de grupos implementado como una lista de Adyacencia ***

Typedef  struct NodoAdyGrupo {
              int nodoAdyGrupo;
              struct NodoAdyGrupo *sigAdyGrupo;
         };

Typedef  NodoAdyGrupo* listaAdyGrupo;

struct NodoGrupo {
   listaAdyGrupo* arrNodosAdy;
    int cantNodos;
};

Typedef  NodoGrupo* GrafoGrupo;


//**** Estructuras para la  Lista de nodos de salida de un grupo ***************

Typedef  struct NodoCanio{
              int nodo;
              //representa el nodo de salida del grupo (nodo origen)
              int nodoDestino;
              //representa el nodo destino de la arista_canio en el otro grupo
              int indiceMenorTecn;
              //representa el índice del arreglo de la menor tecnología que le
              llega al nodo origen
              struct NodoCanio *sig;
```

```
        };

Typedef  NodoCanio* listaNodosSalida;


//**** Estructuras para representar la lista de grupos ************************

Typedef  struct NodoEnGrupo {
              int nodo;
              float* demandas;
              bool marcado;
              struct NodoEnGrupo *sig;
        };

Typedef  NodoEnGrupo* listaNodos;

Typedef  struct Grupo {
              int idGrupo;
              GrafoGrupo grafo;
              listaNodos lista_nodos;
              Grupo *sig;
              listaNodosSalida* arrNodosSalida;
        };

Typedef  Grupo* listaGrupo;


//**** Estructuras para la  Lista Combinación Grupo ***************************

Typedef struct nodosDelCanio{
         int idNodoOrigen;
         int idNodoDestino;
         };

//Lista de tecnologías
Typedef struct nodoTecnologia{
         int idTecnologia;  //índice del arreglo de tecnologías
         int nroAristasCanio;
         nodosDelCanio* nodosCanio;
         nodoTecnologia* sig;
         };
```

```
Typedef nodoTecnologia* listaTecnologias;

//Lista Combinación Grupo
Typedef struct nodoCombinacionGrupo{
         float demanda;
         int grupo1;
         int grupo2;
         listaTecnologias lista_tecnologias;
         nodoCombinacionGrupo* sig;
         };

Typedef nodoCombinacionGrupo* listaCombinacionGrupo;


//**** Estructura para la Matriz Caminos ************************************

Typedef struct celdaCamino{
         float distancia;
         ListaDeNodos camino;
         float demandaUtilizada;
         int indiceTecnArista; // representa la tecnología asignada a la arista
         };

Typedef celdaCamino** MatrizCaminos;


//**** Lista de matrices Camino ********************************************

Typedef struct nodoMatrizCaminos{
         int idMatriz;
         MatrizCaminos matriz;
         nodoMatrizCaminos* sig;
         };

Typedef nodoMatrizCaminos* listaMatrizCamino;


//**** Estructura para recordar tecnologías y destinos de cada nodo que ya
fueron utilizados para no repetirlos ****************************************

struct nodotemp{
   int idNodoDestino;
   nodotemp* sig;
```

```
   };

Typedef nodotemp* listatemp;

struct nodoAristaYDestinoUsados{
   int idNodoOrigen;
   listatemp lista;
   nodoAristaYDestinoUsados* sig;
   };

Typedef nodoAristaYDestinoUsados* listaAristaYDestinoUsados;


//**** Estructura para Armado de ciclos internos de los grupos *****************

struct nodoCamino{
   int izq;
   int der;
   nodoCamino* sig;
   };

Typedef nodoCamino* listaCamino;

struct nodosPunta{
   int puntaIzq;
   int puntaDer;
   float* demandasGrupos;
   listaCamino camino;
   bool estaEnSalida;
   int id;
   nodosPunta* sig;
   };

Typedef nodosPunta* listaPuntas;

//*****************************************************************************

Typedef struct indiceYDestino{
         int indice;
         int destino;
         };
```

1.2. Descripción de las operaciones

1. **void CrearGrafoGrupo** (GrafoGrupo &G , **int** cantClientes)

Crea el grafo de un grupo dado por el parámetro G. El grafo se implementa mediante listas de adyacencia y se crea con la cantidad de nodos dada por cantClientes. Cada nodo se inicializa con una celda dummy con valor -1, indicando que pertenece a otro grupo, es la forma de indicar que el grafo está vacío.

2. **void agregoAdy1Grupo** (**int** n1 , **int** n2 , GrafoGrupo &N)

Agrega el nodo n2 en la lista de nodos adyacentes del nodo n1, en el grafo de grupo N. Lo agrega al comienzo de la lista.

3. **void AgregarAdyGrupo** (**int** n1 , **int** n2 , GrafoGrupo &N)

Invoca dos veces a la función agregoAdy1Grupo para agregar de forma simétrica al nodo n2 como adyacente de n1 y a n1 como adyacente de n2, todo esto en el grafo de grupo N.

4. **void eliminarNodoEnLista**(ListaDeNodos &lista, **int** nodo)

Precondición: el nodo se encuentra en la lista.
Elimina un nodo de la lista de nodos que componen un camino en el grafo de transporte entre un par TNS(i) y TNS(j).

5. listaNodos **eliminarNodoEnListaNodosDelGrupo**(listaNodos lista, **int** nodo)

Precondición: el nodo se encuentra en la lista.
Elimina un nodo del grafo que se encuentra en un grupo.

6. **void asignarNodosInvalidosGrupo**(ListaDeNodos lista_Invalidos, GrafoGrupo &G)

Recorre todos los nodos del grafo de grupo dado por G y marca todos los nodos como inválidos con el valor -2.

7. **int asignarMenorTecnologia**(**float** demanda)

Retorna el índice de la tecnología con menor capacidad que satisface la demanda. Si ninguna tecnología satisface la demanda retorna el índice de la tecnología con mayor capacidad.

8. **int cantidad_Aristas_Canio**(listaTecnologias lt)

La lista de tecnologías parámetro lt, corresponde a las tecnologías asignadas a las aristas caño entre un par de grupos.
Inicializa una variable cantidad en cero. Para cada nodo de lt obtiene la cantidad de aristas caño que se requieren para esa tecnología entre esos dos grupos (campo nroAristasCaño). Suma este valor a la variable cantidad.
Una vez recorridos todos los nodos de lt retorna la variable cantidad.

9. **void imprimirGruposConNodos**(FILE* salida, listaGrupo l, **int** mayor_id_grupo)

Imprime en el archivo salida que se pasa por parámetro, la lista de grupos y los nodos que pertenecen al grupo. Para cada nodo perteneciente al grupo imprime su identificador y la demanda que tiene con nodos de otros grupos.

10. **void imprimirGruposConNodosSalida**(FILE* salida, listaGrupo l, **int** mayorIdGrupo)

Imprime en el archivo salida que se pasa por parámetro, la lista de grupos y los nodos de salida del grupo. Para cada nodo de salida imprime su identificador, el índice de la menor tecnología que soporta la demanda de la arista caño conectada al nodo de salida y el nodo destino en otro grupo que es el otro extremo de la arista caño.

11. **void BuscarMenorDistancia**(**int** &i, **int** &j, **float** &distancia, listaNodos listaGrupo1, listaNodos listaGrupo2, ListaDeNodos &recorrido, MatrizCaminos matriz_Camino)

Parámetros de entrada: listaGrupo1, listaGrupo2 y matriz_camino.
Parámetros de salida: i, j, distancia, recorrido.

listaGrupo1 y listaGrupo2 son las listas de nodos de dos grupos. La función recorre estas listas buscando el par de nodos cuyos tns tienen menor distancia en transporte. Retorna en i, j los identificadores de los nodos con esta menor distancia y el valor de la misma en el parámetro distancia. En recorrido se devuelve la lista de nodos de transporte que forman el camino.

12. **void clonarListaCaminos**(ListaDeNodos listaEntrada, ListaDeNodos &listaClonada)

Realiza una copia de la lista listaEntrada y la devuelve en listaClonada.
ListaEntrada representa un camino de nodos en transporte. Se copian todos los nodos de la lista, es decir que se genera una lista nueva en memoria.

13. **void clonarListaNodosGrupo**(listaNodos listaEntrada, listaNodos &listaClonada)

Realiza una copia de la lista listaEntrada y la devuelve en listaClonada.
ListaEntrada representa la lista de nodos que pertenecen a un grupo. Se copian todos los nodos de la lista, es decir que se genera una lista nueva en memoria.

14. **void clonarListaNodosSalida**(listaNodosSalida listaEntrada, listaNodosSalida &listaClonada)

Realiza una copia de la lista listaEntrada y la devuelve en listaClonada.
ListaEntrada representa la lista de nodos de salida de un grupo. Se copian todos los nodos de la lista, es decir que se genera una lista nueva en memoria.

15. **void crearMatrizCaminos**(MatrizCaminos &matriz_Caminos)

Crea la matriz de caminos que se utiliza para guardar los caminos en transporte, para cada par de nodos del grafo de datos. Esta matriz es cuadrada de dimensión la cantidad de nodos del grafo de datos. La posición i, j de la matriz guarda la información del camino en transporte entre tns(i) y tns(j).
Para cada camino se guarda la distancia total y la lista de nodos que lo componen. También se guarda para la arista i, j en datos el índice de la menor tecnología asignada y la demanda utilizada.

16. **void guardoDijkstraEnMatrizCaminos**(MatrizCaminos &matriz_Caminos, Grafo grafoClonado)

Esta función guarda en matriz_Caminos el resultado de la ejecución del algoritmo de Dijkstra en el grafo de transporte. Para cada camino calculado guarda la distancia total y la lista de nodos que lo componen. Previo a guardar los resultados, invoca a la operación *crearMatrizCaminos(matriz_Caminos)* para crear la matriz.

17. listaNodos **BuscarListaNodosGrupo**(**int** idGrupo, listaGrupo lista)

Retorna la lista de nodos que pertenecen al grupo idGrupo. Si no encuentra el grupo devuelve null.

18. **int ParNodosUsado**(**int** i, **int** j, **int** idGrupo1, **int** idGrupo2, listaCombinacionGrupo lista)

Busca en la lista de combinaciones de grupos parámetro (lista), el nodo correspondiente a la combinación idGrupo1, idGrupo2. Para este nodo recorre la lista de tecnologías asociada y para cada tecnología recorre las aristas caño asignadas. Si encuentra alguna arista caño que tenga extremos i, j devuelve 1, en caso contrario devuelve 0.

19. **void asignarNodosCanio**(**int** i,**int** j,listaTecnologias &lista)

Precondición: la lista de tecnologías debe estar ordenada de mayor a menor en la capacidad de la tecnología.

Asigna los nodos i, j, como extremos de la primera arista caño sin extremos asignados, que encuentra en la lista de tecnologías parámetro.
Cada nodo de la lista de tecnologías tiene la cantidad de aristas caño que se requieren de esa tecnología. Cada nodo tiene también un arreglo de tamaño la cantidad de aristas caño que se requieren para esa tecnología, donde guarda origen y destino de cada arista caño. Esta operación recorre la lista de tecnologías desde el principio hasta encontrar una tecnología que tenga alguna arista a la que le falte definir los extremos. Cuando la encuentra le asigna los extremos i, j.

20. **void tirarLinks**(Grafo &GrafoClonado, ListaDeNodos recorrido)

Esta operación se utiliza para quitar links del grafo de transporte.
Quita de GrafoClonado los links dados por el parámetro recorrido. Para esto quita a cada nodo el adyacente correspondiente en el camino, utilizando la operación QuitarAdy del modulo Grafo.

21. **void imprimirMatrizCaminos**(FILE* salida, MatrizCaminos matriz_Caminos)

Imprime en el archivo dado por el parámetro salida, la información contenida en la matriz parámetro matriz_Camino. Para cada celda de la matriz imprime la correspondiente fila y columna (identificadores de los nodos del grafo de datos), y para el correspondiente camino en transporte entre tns(i) y tns(j) imprime la distancia total y los nodos que forman el camino.

22. **int perteneceListaNodos**(**int** nodo, listaNodos lista)

Busca en la lista dada por el parámetro lista, un nodo con identificador dado por el parámetro nodo. Si lo encuentra devuelve 1, en caso contrario devuelve 0.

23. **int sizeLista**(listaNodos lista)

Devuelve la cantidad de nodos del parámetro lista.

24. **void colocarNodoLista**(**int** idNodo, **int** indiceArreglo, listaGrupo aux)

idNodo: identificador del nodo que se quiere agregar a la lista.
indiceArreglo: posición en el arreglo de nodos de salida del grupo donde se debe agregar el nodo.
aux: apunta al grupo al cual se le quiere agregar el nodo de salida.

Agrega un nodo con identificador idNodo en la lista de nodos de salida del grupo donde se encuentra posicionado aux. Lo agrega como nodo de salida hacia el grupo dado por indiceArreglo (es decir en la lista de nodos de salida de la posición indiceArreglo).

25. listaTecnologias **obtenerListaTecnologias**(**int** idGrupo1, **int** idGrupo2, listaCombinacionGrupo lista)

idGrupo1, idGrupo2: identificadores de los grupos que se relacionan. Representados por nodos de la lista de combinaciones de grupos.
lista: lista de combinaciones de grupos.

Recorre la lista de combinaciones de grupos hasta encontrar el nodo correspondiente a la combinación dada por idGrupo1, idGrupo2. Cuando lo encuentra devuelve la lista de tecnologías asociada al mismo.

26. **void ordenarListaPorTecnologia**(listaNodosSalida &lista)

lista: lista de nodos de salida de un grupo dado hacia otro grupo específico.
Ordena la lista parámetro de forma decreciente en la capacidad de la tecnología asociada a cada nodo (campo del nodo indiceMenorTecn).

27. **int perteneceListaTemp**(listatemp lista, **int** destino)

Busca el destino dado por el parámetro en la lista temporal dada por el parámetro lista. Si lo encuentra devuelve 1, en caso contrario devuelve 0.

28. **int pertenecelistaAristaYDestinoUsados**(**int** origen, **int** destino, listaAristaYDestinoUsados lista)

Busca el origen y destino dados por los parámetros, en la lista de aristas y destinos utilizados dada por el parámetro lista. Si encuentra un nodo que tenga idNodoOrigen igual al parámetro origen y tal que el parámetro destino pertenece a la lista de destinos del nodo, devuelve 1, en caso contrario devuelve 0.

29. listatemp **listaTemporal** (listaAristaYDestinoUsados lista, **int** origen)

Retorna la lista temporal de destinos asociada al nodo dado por el parámetro origen, en la lista de aristas y destinos usados, dada por el parámetro lista.
Si no encuentra al nodo origen devuelve NULL.
listaAristaYDestinoUsados es una estructura para recordar tecnologías y destinos de cada nodo que ya fueron utilizados para no repetirlos.

30. **void agregarlistaAristaYDestinoUsados**(**int** origen, **int** destino, listaAristaYDestinoUsados &lista)

Agrega a la lista de aristas y destinos usados, la información de que el destino dado por el parámetro ya ha sido utilizado por el origen dado por el parámetro.
Si el origen ya se encontraba en la lista parámetro, agrega a la lista de nodos utilizados del nodo origen, el nuevo destino. Si el origen no se encontraba en la lista parámetro, crea el nodo correspondiente con el destino agregado en la lista de destinos utilizados, y lo agrega a la lista parámetro.

listaAristaYDestinoUsados es una estructura para recordar tecnologías y destinos de cada nodo que ya fueron utilizados para no repetirlos.

31. indiceYDestino **obtenerMenorIndiceYDestino**(listaTecnologias lTecn, **int** nodoOrigen, listaAristaYDestinoUsados &lista)

Recorre la lista de tecnologías parámetro, lTecn, y para cada uno de sus nodos recorre su arreglo de aristas caño. Para cada arista caño se fija si el origen o el destino coinciden con el parámetro nodoOrigen. Para los casos en que coincida, se fija si el índice de la tecnología es menor que el encontrado hasta el momento y si el otro extremo de la arista caño no se encuentra en la lista de aristas y nodos utilizados. Si se cumplen estas dos condiciones, actualiza el menor índice de tecnología y carga ese extremo como nodoDestino para retornar.

32. **bool esSalidaCadena**(listaPuntas cadena)

Devuelve true si solo si la cadena dada por el parámetro tiene algún nodo que sea de salida.

33. listaNodos **encontrarNodo**(**int** idNodo, listaGrupo l)

Busca en la lista de grupos parámetro, algún grupo que contenga en su lista de nodos, al nodo idNodo. Si lo encuentra retorna la lista de nodos que lo contiene. Si no lo encuentra retorna NULL.

34. **void crearCadena**(**int** idNodo, **bool** esSalida, listaPuntas &cadena, **int** mayor_id_grupo, listaGrupo l, listaMatrizCamino lmc)

Crea una punta unitaria con valores en punta izquierda y derecha iguales a idNodo, el booleano esSalida igual al parámetro y crea el arreglo demandaGrupos de la punta de tamaño igual al parámetro mayor_id_grupo. Carga en cada posición del arreglo demandaGrupos de la punta, la demanda que tiene el nodo idNodo, con el grupo correspondiente a esa posición.
Crea un nodo del camino con izquierdo y derecho igual a idNodo y lo agrega a la lista de nodos del camino de la punta.
Agrega la punta unitaria creada como primer nodo del parámetro cadena.
Se utiliza la variable global contador (variable que se va incrementando) para identificar la punta con su correspondiente matriz de caminos.

35. **void simetrizarListaCamino**(listaCamino &camino)

Recorre los nodos de la lista dada por el parámetro camino y para cada nodo intercambia los valores de los extremos, es decir que el extremo izquierdo pasa a ser el extremo derecho y el extremo derecho pasa a ser el izquierdo.

36. **void invertirListaCamino**(listaCamino &camino)

Invierte los nodos de la lista dada por el parámetro camino.

37. **void borrarCadenaLista**(listaPuntas cadena, listaPuntas &listaCadena)

Borra la cadena parámetro de la lista de cadenas parámetro.

38. **void borrarMatrizCaminoLista**(**int** id, listaMatrizCamino &lmc)

Borra de la lista de matrices camino dada por el parámetro lmc, el nodo correspondiente a la matriz camino con identificador dado por el parámetro id.

39. **void borrarLista**(listaCamino &camino)

Borra todos los nodos de la lista dada por el parámetro camino.

40. MatrizCaminos **buscarMatrizCamino**(listaMatrizCamino lmc, **int** id)

Dado el id parámetro que identifica a la matriz, busca la matriz en la lista de matrices caminos dada por el parámetro lmc. Si la encuentra retorna un puntero a la misma, en caso contrario retorna NULL.

41. **void CargarCaminoMatriz**(listaPuntas cadena, MatrizCaminos mc_cadena, listaMatrizCamino &mc)

El parámetro mc apunta a la matriz camino que se va a modificar. Se cargan en la matriz apuntada por mc, en las posiciones dadas por los nodos del camino del parámetro cadena, los valores correspondientes de la matriz parámetro mc_cadena.

42. **void tirarLinksPunta**(MatrizCaminos mcPunta1, listaPuntas punta1, Grafo &GrafoClonado)

Para cada nodo del camino asociado al parámetro mcPunta1, obtiene el extremo izquierdo y el derecho. A partir de estos extremos (en el grafo de datos) obtiene los links en transporte que se utiliza para rutear estos extremos.
Luego invoca la operación tirarLinks para tirar estos links del grafo clonado parámetro.

43. **void unirCadena**(listaPuntas cadena1, listaPuntas cadena2, **int** nodoIzq, **int** nodoDer, **int** mayor_id_grupo, listaPuntas &cadena, listaMatrizCamino &lmc, Grafo nodos)

Realiza la unión entre las cadenas parámetro cadena1 y cadena2.
Registra los nuevos caminos en transporte en la matriz de lista de matrices camino parámetro que corresponda.
Los parámetros nodoIzq (sobre la cadena 1) y nodoDer (sobre la cadena 2), son los extremos que quiero unir.
El parámetro cadena es para borrar las cadenas que se unen y agregar la nueva cadena en la lista.
Idem para el parámetro lmc (la lista matriz camino), es para borrar las viejas matrices y dejar solo la nueva matriz.

44. **void obtenerListaMayorTecnologia**(listaNodosSalida listaClonadaSalida, listaNodosSalida &listaMayorTecnologia)

Precondición: La listaClonadaSalida esta ordenada de mayor a menor.
Parámetro de entrada: listaClonadaSalida
Parámetro de salida: listaMayorTecnología

Guarda en una variable (indiceMenorTecn) el índice de la tecnología asignada al primer nodo de listaClonadaSalida, que será la mayor tecnología asignada a los nodos de la lista (ya que está ordenada de mayor a menor). Luego carga en el parámetro de salida listaMayorTecnología, los nodos de listaClonadaSalida que tienen la tecnología con identificador el guardado en la variable indiceMenorTecn.

45. **void eliminarMayorTecnologia**(listaNodosSalida &listaClonadaSalida, listaNodosSalida listaMayorTecnologia)

Precondición: La listaClonadaSalida esta ordenada de mayor a menor.

Parámetro de entrada: listaMayorTecnología
Parámetro de salida: listaClonadaSalida
Como la lista esta ordenada de mayor a menor tecnología, los elementos a eliminar son siempre los del principio. Recorre listaMayorTecnología y va eliminando los nodos correspondientes de listaClonadaSalida.

46. **int tamanio**(listaPuntas lista_Puntas)

Devuelve la cantidad de nodos de la lista de puntas parámetro.

47. **int buscarDestinoNodo**(**int** origen, listaNodosSalida listaNodosSalidaTemp)

Recorre los nodos de salida dados por la lista parámetro listaNodosSalida y busca algún nodo con extremo origen igual al parámetro origen. Si lo encuentra devuelve el identificador del nodo destino de la arista caño correspondiente. Si no lo encuentra devuelve -1.

48. **bool EsNodoSalida**(**int** idNodo, **int** idGrupo, listaGrupo l, **int** mayor_id_grupo)

Devuelve true si solo si el nodo con identificador dado por el parámetro idNodo es un nodo de salida del grupo dado por el parámetro idGrupo. Para esto busca el grupo en la lista de grupos dada por el parámetro l. Cuando encuentra el grupo recorre el arreglo de listas de nodos de salida hacia los restantes grupos. Busca al nodo idNodo en cada una de las listas hasta encontrarlo o hasta que ya no queden listas por revisar. Si lo encuentra retorna true, en caso contrario retorna false.

49. listaNodos **obtenerNodosNoSalidaGrupo**(**int** idGrupo, listaGrupo l, **int** mayor_id_grupo)

Crea una lista de nodos que no son de salida a partir del idgrupo del nodo, busco en la lista de nodos del grupo y verifica que dicho nodo no se encuentre en ninguna lista de nodos de salida del arreglo de nodos de salida del grupo para esto usa la función perteneciendo a la lista.

50. listaPuntas **crearListaPuntas**(listaNodos nodosNoSalida, **int** idgrupo, listaGrupo l, **int** mayor_id_grupo, listaMatrizCamino lmcGeneral)

Crea la lista de puntas partiendo de una lista de nodos, esta lista de puntas esta formada solo por nodos que no son de salida y además que no fueron procesados aun o sea que no están en ningún ciclo de salida lo que hace es recorrer la lista de nodos que recibe, y si no esta marcado lo crea como una cadena y agrega a la lista de puntas.

51. **void cargarGrafoGrupo** (listaPuntas puntas, **int** idGrupo, listaGrupo &l, MatrizCaminos matriz)

Recorro la punta y cargo el grafo del grupo dado.
Pongo la arista en ambos sentidos del grafo.

52. **bool perteneceP untas**(**int** idNodo, listaCamino camino)

Devuelve true si solo si en la lista dada por el parámetro camino, hay algún nodo que tenga como extremo izquierdo o derecho al nodo dado por el parámetro idNodo.

53. **void MarcaNodosCiclo** (listaPuntas ciclo, **int** idGrupo, listaGrupo &l)

Recorro la cadena que ya contiene un ciclo y marco cada nodo del grupo que forma parte del ciclo como procesado.

54. listaNodos **mayorDemanda**(listaNodos* arregloOrdenado,**int** i)

Retorna el mayor nodo de salida (el primero) que esta en la lista de la posición i del arreglo.

55. listaNodosSalida **crearNuevaListaNodosSalida**(**int** idGrupo, listaGrupo l, listaNodos* arregloOrdenado, **int** mayor_id_grupo, MatrizCaminos resultado, Tecnologias* tecnologías)

Crea la nueva lista de salida con los nodos de salida que satisfacen las demandas de todos los nodos no usados.

56. listaNodos **obtenerlistaNodosSalidaNoMarcados**(listaNodos listaNodosTemp, **int** mayor_id_grupo)

Devuelve la lista de nodos que pertenecen al parámetro listaNodosTem y además son nodos de salida y no están marcados.

57. listaPuntas **unirListasPunta**(listaPuntas lPuntas, listaPuntas lista_PuntasTemp)

Devuelve la concatenación de las 2 listas puntas, la que tiene una punta con los nodos de salida (lista_PuntasTemp) con la que tiene una lista de puntas con los nodos no salida con 1 nodo cada punta (lPuntas).

58. **bool verificarNodosMarcados**(**int** idGrupo,listaGrupo l){

Retorna true si todos los nodos del grupo están marcados y false en caso contrario.

59. **void arregloOrdenadolistaNodoIndice**(listaNodos* &arregloOrdenado, listaNodos ln1, **int** i, **int** mayor_id_grupo)

Verifica si efectivamente se modifica la estructura con la referencia.
Agrega ordenadamente un nodo en una lista ordenada en base a la demanda hacia el grupo dado por el índice i.

60. **void unirNodosSalidaRespaldo**(listaGrupo &l, **int** mayor_id_grupo, Grafo nodos, MatrizCaminos &resultado, listaMatrizCamino &lmc)

Une los nodos de salida que van hacia el mismo grupo

61. **float mayorTecnologia**(Tecnologias* tecnologías)

Devuelve la mayor capacidad del arreglo de tecnologías.

62. **bool esPosibleUnirCadena**(listaPuntas punta1, listaPuntas punta2, **int** nodoIzq, **int** nodoDer, **int** mayor_id_grupo, Tecnologias* tecnologías, listaMatrizCamino lmc, Grafo nodos, **float*** holguras)

Controla 4 condiciones para determinar si dos cadenas se pueden unir o no:

1. La demanda hacia un mismo grupo de todos los nodos que van a quedar conectados tiene que ser menor que la holgura de la arista caño correspondiente.

2. La suma de las demandas de los nodos de las dos cadenas no puede ser mayor que la capacidad de la máxima tecnología disponible.
3. Las cadenas a unir tienen que ser link disjuntas en transporte. Asimismo la primera conexión para unir las dos cadenas debe ser también link disjunta con las dos cadenas.
4. Se pueda cerrar el ciclo, es decir que la segunda conexión para cerrar el ciclo también sea link disjunta con las dos cadenas y con la primera conexión.

63. **void cargarArregloHolguras**(listaNodosSalida listaNodosSalidaTemp, **float*** &arregloHolguras, MatrizCaminos resultado, Tecnologias* tecnologías)

Carga el arreglo de holguras con cada holgura (capacidad tecnología - demanda) de la matriz resultado para cada nodo de la lista de entrada.

64. **float demandasNodoSalidaHaciaGrupos**(**int** idNodoSalida, listaNodosSalida listaNodosSalidaTemp, listaGrupo l)

Retorna la suma de las demandas que tiene el nodo de salida hacia los demás grupos.

65. **void asignarTecnologiaCiclo**(listaPuntas puntaSalidaTotal, MatrizCaminos &resultado, listaGrupo l, **int** idGrupo, **int** mayor_id_grupo, listaNodosSalida listaNodosSalidaTemp)

Recorre las aristas del ciclo y le asigna la mínima tecnología que cubre toda la demanda del ciclo menos la demanda de cada nodo de salida hacia el grupo de salida correspondiente. No cambia las tecnologías de las aristas entre nodos de salida ya que a estas ya se le asigno tecnología anteriormente.

66. **void cargarDemandaOcupadaEnAristaCanio**(listaPuntas puntaSalidaTotal, listaNodosSalida listaNodosSalidaTemp, MatrizCaminos &resultado, **int** idGrupo, listaGrupo l, **int** mayor_id_grupo)

Carga el valor de cada demanda hacia las aristas caño para cada lugar del arreglo de las aristas caño. Le sumo la demanda que tienen los nodos del ciclo formado y lo coloco en las aristas caño que tienen los nodos de salida del grupo, sumándole lo que ya tiene cada arista.

67. **float distanciaCadena**(listaPuntas puntasUnidas, MatrizCaminos matriz_Unidas)

Dada una punta (parámetro puntasUnidas) y su matriz correspondiente (parámetro matriz_Unidas), devuelve la distancia sumada de todas las aristas que componen dicha punta.

68. **float distanciaEntreNodos**(**int** nodoOrigen, **int** nodoDestino, Grafo nodos)

Devuelve la distancia que hay entre los nodos del grafo dados por los parámetros nodoOrigen y nodoDestino.

69. **float calcularDistanciaCamino**(ListaDeNodos respaldocamino, Grafo nodos)

Devuelve la suma de las distancias en transporte, de las aristas que componen el camino dado por el parámetro respaldoCamino.

70. **int tamanioCadena**(listaCamino cadena){

Devuelve la cantidad de elementos (nodos) que forman parte de la cadena.

71. **void borrarCamino** (listaCamino &camino)

Destruye el camino dado por el parámetro.

72. **void borrarCadena**(listaPuntas &listaResultado)

Borra la lista de puntas y sus caminos correspondientes.

73. **void clonarCaminoPunta** (listaCamino caminoOriginal, listaCamino &caminoClonado)

Realiza una copia en profundidad del camino, para poder trabajar sobre la copia.

74. **void clonarPunta**(listaPuntas puntaOriginal, listaPuntas &puntaClonada, **int** mayor_id_grupo)

Clona una punta (solo una) junto con su camino correspondiente.

75. listaPuntas **buscarPuntasConMasNodos**(listaPuntas listaPuntasFusionada, **int** mayor_id_grupo)

Retorna una lista de puntas con la punta que tiene mayor cantidad de nodos (si hay mas de una retorna ambas).

76. **void preUnion**(listaPuntas cadena1, listaPuntas cadena2, **int** nodoIzq, **int** nodoDer, **int** mayor_id_grupo, listaMatrizCamino lmc, Grafo nodos, listaPuntas &resultado, MatrizCaminos &mc, ListaDeNodos &recorrido, **float** &distanciaCamino)

Similar a UnirCadena pero no destruye las cadenas de entrada. En lugar de devolver la lista de puntas con las dos puntas unidas en la lista general, quitando las dos puntas de entrada de la misma, y la lista de matrices con la matriz resultado quitando las 2 matrices asociadas; solo devuelve una punta unión y su matriz correspondiente. Además devuelve cuales son los links que utiliza esa punta y su distancia correspondiente.

77. **void preUnionConDistancia**(listaPuntas &puntaSalidaResultado,MatrizCaminos &matrizSalidaResultado, listaPuntas prueba,listaPuntas lista_PuntasSalida , **float** &distancia, **int** mayor_id_grupo, listaMatrizCamino lmc, Grafo nodos)

Dada una punta con nodos internos y otra con los nodos de salida, retorna la punta resultado junto con su matriz asociada, que une ambas puntas por aquellos extremos que generan una distancia menor.

78. **void buscarMenorDistanciaPuntaSalida**(listaPuntas &puntaSalidaResultado,MatrizCaminos &matrizSalidaResultado,listaPuntas lmn, listaPuntas lista_PuntasSalida,listaMatrizCamino lmc , **int** mayor_id_grupo, Grafo nodos)

Retorna una punta, generando un ciclo con la menor distancia de todas las puntas pertenecientes a la lista parámetro lmn.

79. **void** **generarCicloConNodosNoSalida**(listaPuntas lPuntas, listaPuntas lista_PuntasTemp, **int** mayor_id_grupo, Tecnologias* tecnologías, listaMatrizCamino &lmc, Grafo nodos, listaNodosSalida listaNodosSalidaTemp, **float*** arregloHolguras, listaPuntas &puntaSalidaResultado, MatrizCaminos &matrizSalidaResultado, MatrizCaminos resultado, **int** idGrupoOrigen)

El parámetro lPuntas es la lista de puntas con nodos que no son de salida y que no están marcados. El parámetro lista_PuntasTemp es la lista de puntas con nodos de salida. Trata de unir un ciclo con los nodos de salida y los no salida que están en la lista de puntas lPuntasGeneral. Devuelve en puntaSalidaResultado el ciclo formado.

80. **int** **obtenerMenorTecnologiaArista**(**int** nodoIzq, **int** nodoDer, listaNodosSalida listaNodosSalidaTemp)

Recorre la lista de nodos de salida parámetro listaNodosSalidaTemp, buscando al nodo izquierdo (parámetro nodoIzq) y al nodo derecho (parámetro nodoDer) de la arista caño. Para cada uno de estos nodos, obtiene el índice de la menor tecnología asignada a la arista caño. Finalmente devuelve el menor de estos dos índices.

81. **void** **asignarTecnologiaPunta**(listaPuntas lista_PuntasTemp, listaNodosSalida listaNodosSalidaTemp, MatrizCaminos &matrizlista_PuntasTemp)

Carga la tecnología de las aristas, que unen los nodos de salida entre si, de una punta que contiene la unión de los nodos de salida.

2. Módulo Grafo

A continuación se describen las operaciones del módulo Grafo. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

2.1. Declaración de estructuras

```
Typedef float** Matriz;
Typedef Matriz * Matrices;

//*********Implementado como una lista de Adyacencia***************

Typedef  struct NodoAdy{
           int nodoAdy;
           float distanciaAdy;
           struct NodoAdy *sigAdy;
          };

Typedef  NodoAdy* listaAdy;

struct Nodo{
         listaAdy* arrNodosAdy;
         int cantNodos;
         };

Typedef  Nodo* Grafo;

//*****************************************************************

//**********Estructura para almacenar los caminos mas cortos entre cada par
//de TNS(i) y TNS(j)***************

struct nodoListaDeNodos{
         int nodo;
         nodoListaDeNodos *sig;
         int contador;
         };

Typedef nodoListaDeNodos* ListaDeNodos;
```

```
//**********Definición de tipo auxiliar para Dijkstra***************

struct nodoArregloTabla{
        bool conocido;
        float distancia;
        int camino;
        };

struct nodoTabla{
        nodoArregloTabla*  arregloTabla;
        int cantNodos;
        };

Typedef nodoTabla* Tabla;

//*****************************************************************
```

2.2. Descripción de las operaciones

1. **void CrearGrafo** (Grafo &G , **int** cantNodos);

Crea el grafo G con la cantidad de nodos dada por el parámetro cantNodos. La lista de nodos adyacentes de cada uno se inicializa en NULL.

2. **void ClonarGrafo**(Grafo GOrig, Grafo &GClonado);

Clona el grafo completo copiando sus listas en profundidad. Es necesario para tirar los links y no perder el grafo original.

3. listaAdy **ListaAdyacentes** (**int** n , Grafo G);

Dado un nodo devuelve la lista de adyacentes a dicho nodo, o sea la lista de los nodos a los cuales puedo llegar a través de una y solo una arista.

4. **void AgregarAdy** (**int** n1 , **int** n2 , **float** distancia , Grafo &G);

Dados dos nodos (n1 y n2) y una distancia asociada a dichos nodos agrega una arista que va de n1 hacia n2 y de n2 hacia n1.

5. **void QuitarAdy** (**int** n1 , **int** n2 , Grafo &N);

Precondición: n2 es adyacente de n1.
Quita de los adyacentes de n1 a n2.

6. **void ImprimirCamino**(**int** inicial, Tabla t, **int** vertice, ListaDeNodos &L , **float** &distancia);

Parámetros de entrada: inicial, t, vértice.
Parámetros de salida: L, distancia.
Devuelve en L, la lista de Nodos (sin el origen) por los cuales atravesó para ir desde el Nodo de origen (inicial) hacia el Nodo de destino (vértice) a través del costo mínimo. En distancia devuelve la distancia total del camino.

7. **int perteneceLista**(**int** nodo, ListaDeNodos lista);

Devuelve True si el nodo pertenece a la lista y False en caso contrario

8. **void mostrarCamino**(ListaDeNodos &L, FILE* &salida);

Imprime en el archivo de salida el contenido de la lista de Nodos.

9. **void IniciarTabla**(Grafo G, **int** vertice, Tabla &t);

Parámetros de entrada: G, vértice.
Parámetros de salida: t.
Dado un nodo (vértice) y el conjunto de Nodos (G), inicializa la tabla (t) poniendo como Nodo de inicio el Nodo ingresado (poniéndole distancia 0), inicializa en FALSE el resto de los Nodos, inicializa en infinito la distancia para llegar de dicho Nodo a los demás y por último inicializa en -1 (valor arbitrario) el camino desde dicho vértice hacia los demás.

10. **void LimpiarTabla**(Tabla &t);

Libera toda la memoria de t y la iguala a NULL.

11. **int TablaDistanciaCorta**(Tabla t);

Devuelve el Nodo al cual puedo llegar con el menor costo.

12. **void Dijkstra**(Grafo G, Tabla &t);

Modifica la tabla t, dejando como resultado en cada Nodo el costo mínimo para llegar desde el origen hacia dicho Nodo, y además el Nodo anterior por el cual dicho Nodo fue alcanzado.

13. **void imprimirAdyacentes**(listaAdy adyacentes, FILE* salida);

Imprime en el archivo de salida, la lista de nodos adyacentes que tiene un nodo. Para cada nodo adyacente imprime el identificador y la distancia al mismo desde el origen.

14. **void imprimirGrafo**(Grafo G, FILE* salida);

Imprime en el archivo de salida la información del grafo dado por el parámetro G (imprime el identificador del nodo y luego invoca a la función imprimirAdyacentes para imprimir la lista de nodos adyacentes al mismo).

3. Módulo LeerArchivo

A continuación se describen las operaciones del módulo LeerArchivo. Para facilitar su comprensión, se incluyen previamente las declaraciones de las estructuras que éstas utilizan.

3.1. Declaración de estructuras

```
Typedef char *string;

Typedef float** Matriz;
Typedef Matriz * Matrices;

Typedef struct Tecnologias{
               float capacidad;
               float costo;
               };

//**** Arreglo de listas para representar la matriz de demandas***************
//**** Con esta estructura busco para cada nodo, generar una lista con las
demandas que tiene con los demás nodos***************

Typedef    struct NodoDemanda{
               int nodo;
               float demanda;
               struct NodoDemanda *sig;
               };

Typedef  NodoDemanda* listaDemanda;

Typedef listaDemanda* MatrizDemandas;


Typedef int* arrIdGrupos;
// Arreglo dinámico con los Id's de grupo. Este arreglo es utilizado para saber
// a que grupo pertenece cada nodo.
// El 0 en el arreglo significa que dicho nodo no fue procesado aún y -2 que no
// es un nodo válido
```

3.2. Descripción de las operaciones

1. **void CargarEstructura**(FILE* salida,Grafo &grafo, string archivo, **int** &cantNodos, **int** &cantClientes, Tecnologias* &tecnologías, **int*** &Tabla_TNS,Matrices &matrices, **int** &cantLink, **int** &cantTecnologias, arrIdGrupos &arregloIdGrupos, MatrizDemandas &matriz_Demandas);

Parámetros de entrada: archivo
Parámetros de salida: salida, grafo, cantNodos, cantClientes, tecnologías, Tabla_TNS, matrices, cantLink, cantTecnologias, arregloIdGrupos, matriz_Demandas.
Carga las estructuras a partir de procesar el archivo con los datos, cuyo nombre viene dado por el parámetro archivo.

2. **void imprimirListaDemandas** (listaDemanda demandas,FILE* salida)

Imprime en el archivo salida la información contenida en la lista de demandas parámetro. Es decir imprime el identificador del nodo destino y la demanda con el mismo.

3. **void Imprimir_MatrizDemandas** (MatrizDemandas &matriz_Demandas, **int** can_clientes, FILE* salida)

Imprime en el archivo salida la información contenida en la matriz de demandas (matriz_Demandas). Es decir que para cada nodo origen imprime su identificador e invoca a la operación imprimirListaDemandas para imprimir la lista de nodos con los que el origen tiene demanda y los valores de demanda correspondientes.

15. Anexo IX

Descripción del archivo de salida de la Solución utilizando Cplex

En la figura 14.1 se incluye una imagen de un ejemplo de archivo de salida de la *Solución utilizando Cplex*, posteriormente se describe la información contenida en el mismo (ver los números de referencia) [8].

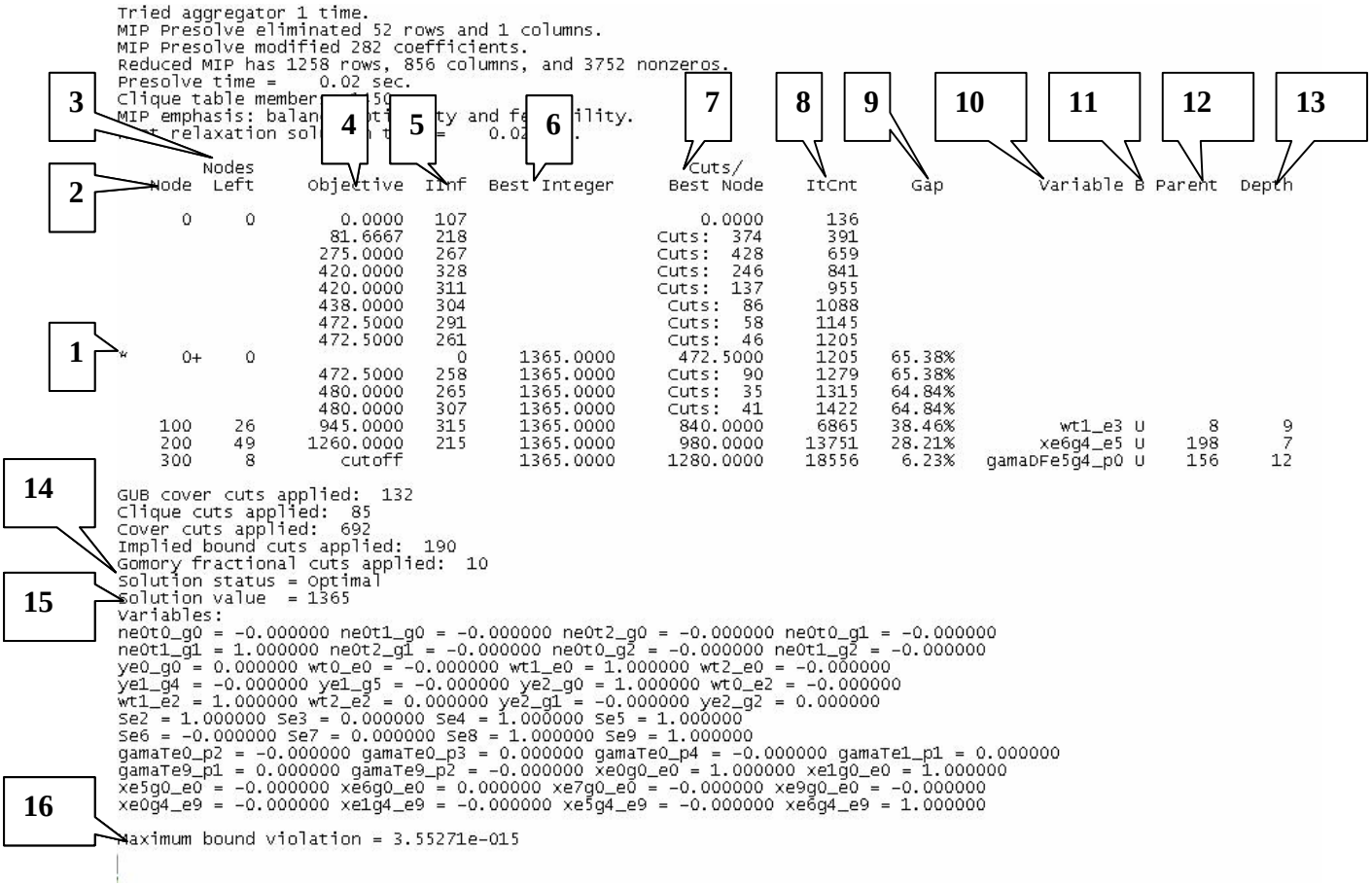

Figura 14.1: Captura de pantalla del archivo de salida de la *Solución utilizando Cplex*.

1. *** en la columna de mas a la izquierda:** Con * se indican las filas donde se encontró una solución entera factible.

2. **Node:** Número de nodo.

3. **Nodes Left:** Cantidad de nodos que restan por explorar.

4. **Objetive:** Es el óptimo del problema relajado en el nodo o indica que ese nodo se descarta lo cual puede ocurrir por 3 razones:
   1. El subproblema asociado al nodo es infactible.
   2. El óptimo del problema relajado asociado al nodo es peor que el óptimo entero hallado hasta el momento.
   3. El nodo provee una solución entera (es una hoja).

5. **IInf:** Cantidad de variables que no son enteras.

6. **Best Integer:** Mejor solución entera hasta el momento.

7. **Cuts/Best Node:** Cuts: indica la cantidad de cortes que se generaron, si aparece el nombre de una variable indica que se generó un corte en esa variable. Cuando aparece un número indica el mejor valor de función objetivo del problema relajado, para los nodos no explorados aún.

8. **ItCnt:** Acumulativo de la cantidad de iteraciones que lleva realizadas el algoritmo que resuelven el problema relajado en cada nodo.

9. **Gap:** Hasta que no se encuentra una solución entera la columna Gap está vacía. Cuando se encuentra una solución entera el gap es una mediada de cuan distante esta la solución entera hallada del óptimo del problema. Se calcula de la siguiente manera:
   |best integer - best node|/(1e-10 + |best integer|).
   Ver la definición de gap de dualidad dada en el Anexo II.

10. **Variable:** Variable por la que se ramificó y por tanto generó este nodo.

11. **B:** Indica la dirección en la que se ramificó:
    - D indica que la variable se seteó con un valor mas bajo
    - U indica que la variable se seteó con un valor mas alto

12. **Parent:** Número de nodo del padre.

13. **Depth:** Profundidad de este nodo en el árbol de Branch and Cut.

14. **Solution Status:** Indica si se encontró una solución óptima.

15. **Solution Value:** Valor de la función objetivo evaluada en la solución óptima hallada.

16. **Variables:** Se muestran las variables con sus respectivos valores correspondientes a la mejor solución entera hallada.

17. **Maximum bound violation:** Debido a la precisión finita de los cálculos computacionales, algunas variables pueden fijarse en cero o uno cuando en realidad se encontraban muy cerca de estos valores pero no exactamente en cero o en uno. Por lo que esta cantidad indica la máxima diferencia entre el valor de una variable y el valor entero que se le dio.