TESIS

Presentada el día 4 de julio de 2014 en la

Universidad de la República, UdelaR
Programa de Desarrollo de las Ciencias Básicas
PEDECIBA, Área Informática

para obtener el título de

MAGISTER EN INFORMÁTICA

para

Gabriel Francisco BAYÁ MANTANI

Instituto de Investigación : INCO.
Componentes universitarios :
UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Título de la tesis :

Diseño Topológico de Redes
Caso de Estudio:
Capacitated m Two-Node Survivable Star Problem

Comité de examinadores:

Dr. Antonio	MAUTTONE	Director de Tesis
Dr. Franco	ROBLEDO	Director de Tesis
Dra. Simone	LIMA MARTINS	Revisora
Dr. Pablo	RODRIGUEZ-BOCCA	
Dr. Marcos	VIERA	

Agradecimientos

Cuando realizamos un trabajo de esta naturaleza, sin duda nos vemos influenciados por el entorno y las personas que han participado del mismo. Quiero agradecer a mis directores de tesis el Dr. Antonio Mauttone quien ha sido un guía permanente y al Dr.Franco Robledo quien con su entusiasmo, bonhomía y don de gente me convenció que podía transistar por esta maestría y terminarla exitosamente. Sin su participación, este trabajo seguramente no existiría.

Hago extensivo este agradecimiento a la Dra. Alejandra Beghelli y al Dr. Renato Cabrera de la Universidad Adolfo Ibañez en Viña del Mar, quienes pusieron todo a mi disposición e hicieron que en mi estadía allí lograra finalizar una de las partes mas complejas de este trabajo.

Quiero participar también al Prof. Dr. Roberto Baldacci de la Universidad de Bologna quien me cedió gentilmente todos los casos de prueba de su problema relacionado (CmRSP) y que me permitieron mejorar y medir la performance de mi trabajo, al MSc. Alexis Rodríguez quien me aportó sus bibliotecas de grafos y muchas horas de su tiempo, y al Dr. Claudio Risso del LPE quien me brindó su apoyo con algunos ejemplos que valorizaron el resultado de esta tesis.

Finalmente debo agradecer a mi familia, a mi esposa Elena quien con su apoyo incondicional me ayudo a sobrellevar los momentos difíciles, a mi hija Claudia, y fundamentalmente a mi hijo Rodrigo, quien me ayudo a lidiar con la depuración del código y la mejora de la eficiencia de los algoritmos.

Índice general

Resumen

Hoy en día prácticamente todas las actividades inherentes al ser humano tienen un soporte de hardware y software que las gestiona o las controla. Los elementos de comunicaciones cumplen un papel vital en este entramado. El diseño de topologías de redes tolerantes a fallos ha pasado a tener una importancia fundamental en la infraestructura de estas redes y constituye un elemento crítico a la hora de mantener la comunicación ante la falla de un enlace o alguno de sus nodos. El problema tratado en esta tesis intenta resolver eficientemente el diseño de una red con conexiones redundantes utilizada a menudo por los operadores telefónicos y servicios de internet. Dicha red conecta un nodo principal con clientes y establece algunas normas que modelan su construcción, tales como cantidad de clientes, número de componentes y tipos de enlaces, con el fin de satisfacer restricciones técnicas y necesidades operativas.

Se considera en este trabajo un problema de optimización combinatoria denominado CmTNSSP (Capacitated m Two-Node-Survivable Star Problem), una relajación del problema CmRSP (Capacitated m Ring Star Problem). La variante relajada consiste en que los anillos ($rings$) del problema CmRSP pueden ser componentes 2-nodo-conexas, las cuales no necesariamente deben ser ciclos.

El objetivo de esta tesis consta de la especificación de un modelo de programación matemática del problema a tratar, la resolución exacta mediante un lenguaje algebraico y un solver, y la resolución aproximada del mismo a través de una metaheurística que alterna búsquedas locales que obtienen soluciones incrementalmente mejores y búsquedas locales de resolución exacta basadas en modelos de programación matemática, en particular Programación Lineal Entera.

Los resultados computacionales obtenidos por los algoritmos desarrollados muestran la robustez y competitividad de los mismos, comparados con las instancias benchmark publicadas en la literatura. Así mismo los experimentos muestran la relevancia de considerar la variante concreta del problema estudiado en esta tesis.

Palabras Clave: Diseño Topológico de Redes, Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS), Metaheurísticas.

Capítulo 1

Introducción

1.1. Motivación

A medida que las poblaciones comenzaron a crecer y las distancias entre ellas eran cada vez mayores, la necesidad de los seres humanos de comunicarse efectivamente comenzó a desarrollarse. Con el advenimiento del telégrafo y fundamentalmente del teléfono -que fue una vía de comunicación directa y a la distancia entre dos personas sin necesidad de codificar y decodificar la información-, comenzó a generarse una infraestructura subyacente de comunicaciones que debía funcionar eficientemente e ir incorporando gradualmente cada vez más clientes.
Esto trajo aparejado un crecimiento en las redes telefónicas que no fue planificado correctamente debido a la necesidad de manejar una demanda en alza y hacer que el floreciente negocio fuera sustentable.

A medida que transcurrieron los años, las redes telefónicas se extendieron y comenzaron a formar parte de servicios esenciales dentro las comunidades. Esto generó una dependencia cada vez mayor en las comunicaciones y apenas se comenzó a tomar conciencia de la importancia que estos servicios cumplían en la sociedad.

Más adelante, con la evolución de las comunicaciones telefónicas, empezó el desarrollo de la informática y la transmisión digital de datos. Para comunicar dos computadores distantes entre si se utilizó nuevamente la red telefónica como medio de transmisión. Esto volvió a generar una cantidad de servicios asociados que se asentaban en una infraestructura de comunicaciones que había crecido sin suficiente planificación.

Es así que ocurren algunos hechos, cuyas consecuencias devastadoras están directamente ligadas a esta falta de planificación, tales como el incendio de una central telefónica en un suburbio de Chicago en mayo de 1988, que dejó sin comunicación a 35.000 abonados locales, afectó 120.000 lineas troncales de larga distancia, comprometió el funcionamiento del tráfico aéreo del aeropuerto O'Hare y puso fuera de servicio el número de emergencias 911, tal cual se detalla en el informe *Keeping the Phone Lines Open* de Zorpette [1].

Estos accidentes ponen de manifiesto entre otras cosas, la necesidad de una correcta planificación de las redes telefónicas y de transmisión de datos. Mas allá de todas las acciones preventivas que puedan tomarse para evitar accidentes como el citado anteriormente, un elemento fundamental para mitigar el impacto de los mismos es el correcto diseño de las redes de telecomunicaciones. El estudio de la estructura, la introducción de niveles de conectividad mínimos entre sus nodos y redundancia son elementos cruciales para evitar catástrofes ante una falla.

La principal motivación de estudiar el diseño topológico de redes es su aplicación en el área de las telecomunicaciones [2]. Básicamente el objetivo es obtener estructuras con el nivel de redundancia deseado, que toleren la falla en algunos de sus nodos o de sus enlaces y permitan el ahorro en los costos de construcción.

1.2. Definición del Problema CmTNSSP

El problema que describiremos a continuación es un ejemplo acerca de la planificación que deberá seguirse para construir redes tolerantes a fallos que cumplan con ciertas necesidades operativas y restricciones técnicas. Trataremos en este capítulo una descripción verbal del problema CmTNSSP y una definición formal del mismo.

1.2.1. Descripción del Problema

Sea un grafo simple no dirigido $G = (V, E)$, queremos obtener un subgrafo (red) que cumpla con cierta topología, la cual describiremos a continuación.

Consideremos dicho grafo inicial G. Dentro del conjunto de vértices V existe un nodo distinguido "d" que llamaremos depósito o depot.

En todo el alcance de esta tesis el término "nodo" se utilizará para referirnos a cualquier vértice perteneciente al conjunto de vértices de cualquiera de los grafos definidos. Ambos términos serán usados indistintamente.

Al conjunto de vértices restantes $V \setminus \{d\}$ lo vamos a particionar en dos conjuntos disjuntos, uno de ellos llamado conjunto de los nodos terminales T, y el otro conjunto de los nodos auxiliares o de Steiner W. Los nodos terminales son aquellos que deberán estar obligatoriamente presentes en la red a construir y los auxiliares o de Steiner participarán de la misma solamente si su inclusión mejora los costos de construcción de dicha red.

Una solución factible de la red que proponemos construir estará compuesta por un cierto número m de subgrafos conexos, que solamente tendrán en común el nodo d, de modo que si quitamos este nodo el grafo resultante quedaría dividido en m componentes conexas.

Cada componente conecta el depósito d con un cierta cantidad de nodos terminales que no podrá ser mayor a un parámetro de capacidad Q dado. Este parámetro acota la cantidad de nodos de cada componente respondiendo a restricciones de conexión y de latencia en las comunicaciones.

Los nodos terminales presentes en cada una de estas m componentes conexas, o bien pertenecen a una estructura conexa con redundancia que forma parte de la componente, o bien están unidos a dicha estructura mediante una arista. En dicha estructura conexa con redundancia, todo par de vértices están conectados mediante dos caminos independientes.

Los nodos de Steiner, si se incluyen, solamente podrán pertenecer a las estructuras conexas con redundancia mencionadas en el párrafo anterior. Dichos nodos no podrán estar unidos a estas estructuras mediante una arista.

El grafo inicial G tiene asociadas dos matrices de costos. Una de ellas determina el costo de conectar cada par de vértices si ambos forman parte de la estructura conexa con redundancia (costos de ruteo) y la otra determina el costo de conectar un par de vértices si uno de ellos está unido a la estructura mediante una arista (costos de conexión). Normalmente en el diseño de redes el costo de los routers del core es mayor al costo de los los routers de acceso, por lo tanto se contempla esta situación mediante la definición de costos diferenciales.

Nuestro problema consiste en obtener un subgrafo del grafo inicial, que sea de costo mínimo y construido bajo las premisas anteriores. Al problema así definido le denominaremos *Capacitated m Two Node Survivable Star Problem* (CmTNSSP).

1.2.2. Definición Formal

Para dar una definición formal del problema CmTNSSP estableceremos algunas definiciones y convenciones con las cuales trabajaremos de aquí en más.

Definiciones Previas

Los problemas de diseño de redes con requisistos de conectividad o supervivencia, pueden definirse de dos formas diferentes:

- Con respecto a la cantidad de aristas (links) que pueden fallar en la red sin que queden desconexos ciertos pares de nodos terminales. Estos se traducen en requisitos de caminos arista-disjuntos entre pares de nodos terminales.
- Con respecto a la cantidad de nodos que pueden fallar (juntos con sus aristas incidentes) sin que queden desconexos ciertos pares de nodos terminales. Estos se traducen en requisitos de caminos nodo-disjuntos entre pares de nodos terminales.

DEFINICIÓN **1.1** *Sea $G = (V, E)$ un grafo no dirigido donde V es un conjunto de vértices y E un conjunto de aristas, se dice que una pareja de nodos $(i, j) \in V$ tiene k-arista-conectividad*

o está k- arista-conectada en G, cuando existen al menos k caminos arista-disjuntos (que no comparten ninguna arista tomados dos a dos) que conectan i con j.

Esta definición es equivalente a afirmar que todo corte del grafo para los nodos i, j contiene al menos k aristas.

DEFINICIÓN **1.2** *Se dice que un grafo $G = (V, E)$ es k-arista-conexo cuando para toda pareja de nodos $(i, j) \in V$, dicha pareja está k-arista-conectada.*

En forma análoga se definen los conceptos para nodo-conectividad.

DEFINICIÓN **1.3** *Se dice que una pareja de nodos (i, j) tiene k-nodo-conectividad o está k-nodo-conectada en un grafo dado, cuando existen al menos k caminos nodo disjuntos (o sea, que no comparten ningún nodo salvo i, j) que conectan i con j.*

DEFINICIÓN **1.4** *Se dice que un grafo es k-nodo-conexo cuando toda pareja de nodos i, j del mismo está k-nodo-conectada.*

Nótese que si dos caminos con los mismos extremos i, j son nodo-disjuntos, entonces también lo son arista-disjuntos; pero no así recíprocamente, es decir, dos caminos arista-disjuntos pueden en general ser o no nodo-disjuntos.

DEFINICIÓN **1.5** *Dado un grafo $G = (V, E)$ y un vértice $i \in V$ llamamos grado de i y lo anotamos $\delta(i)$ a la cantidad de aristas incidentes al nodo i.*

Una vez especificadas estas definiciones, pasemos ahora a la definición formal del problema CmTNSSP.

Sea $G = (V, E)$ un grafo donde V es el conjunto de vértices, E es el conjunto de aristas y $d \in V$ es un nodo distinguido de dicho conjunto de vértices V que llamaremos $depot$. De aquí en mas utilizaremos indistintamente vértice o nodo para referirnos a los elementos del conjunto V.

Sea $T \subseteq V \setminus \{d\}$ un conjunto de nodos, que denominaremos nodos terminales del grafo G.

Sea $\hat{T} = T \cup \{d\}$ el conjunto de terminales con el nodo $depot$ incluido.

Sea $W = V \setminus \hat{T}$ el conjunto de los nodos opcionales o de Steiner del grafo G.

Se desea construir un grafo H de la forma:

$$H = H_1 \cup H_2 \cup H_3 ..... \cup H_m \tag{1.1}$$

donde cada componente H_i se define como:

$$H_i = G'_i \cup S_i \quad i = 1, ...m \tag{1.2}$$

y cumple que:

- $G'_i = (U'_i, E'_i) \qquad U'_i \subseteq V$ y $E'_i \subseteq E, \qquad i = 1, ...m \qquad$ son grafos 2-nodo-conexos.
- $S_i = (\bar{V}_i \cup \bar{U}_i, \bar{E}_i), \quad \bar{U}_i \subseteq U'_i, \quad \bar{V}_i \subset T, \quad \bar{V}_i \cap U'_i = \phi,$
  $\bar{E}_i = \{(u_i, v_i)\}, \quad u_i \in \bar{U}_i, \quad v_i \in \bar{V}_i, \quad \bar{E}_i \subset E$
  $\delta(v_i) = 1 \quad \forall v_i \in \bar{V}_i \qquad i = 1, ...m$.

Sea $T(H_i)$ el conjunto de nodos terminales de la componente i-ésima del grafo H, existe una restricción de capacidad tal que:

$$|T(H_i)| \leq Q \tag{1.3}$$

Dado d el nodo distinguido de G definido anteriormente como *depot* se cumple que:

$$d = H_1 \cap H_2 \cap H_3 ..... \cap H_m \tag{1.4}$$

Definimos $C = \{c_{ij}\}_{i,j \in V}$ como la matriz de costos de ruteo del grafo, es decir los costos de una cierta arista (i, j) cuando dicha arista pertenece a algún G'_k, con $k = 1...m$.

Definamos ahora $D = \{d_{ij}\}_{i,j \in V}$ como la matriz de costos de conexión del grafo, es decir los costos de la arista (i, j) cuando dicha arista pertenece a S_k, con $k = 1...m$.

El grafo H a construir definido en 1.2.2 deberá ser de costo mínimo, donde el costo incluye los costos de ruteo y los costos de conexión, es decir que deberá ser la solución óptima del CmTNSSP.

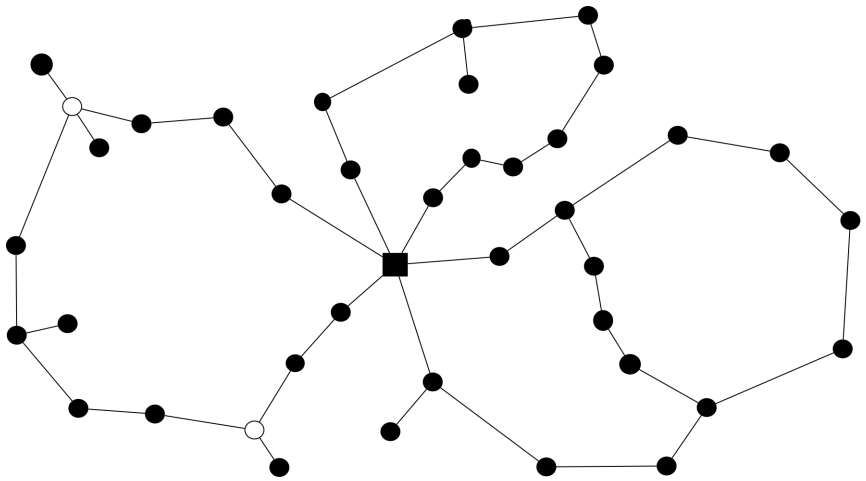

Figura 1.1: Solución factible del problema CmTNSSP.

El la Figura 1.1 se puede apreciar la topología de una solución factible del problema CmTNSSP. En dicha solución los nodos en color negro corresponden al conjunto de terminales, los nodos en color blanco son los nodos opcionales o de Steiner y el nodo representado con forma cuadrangular es el nodo $depot$.

1.2.3. Caracterización del CmTNSSP

PROPOSICIÓN 1.1 *El CmTNSSP pertenece a la clase de problemas $\mathcal{NP}$-Difícil.*

DEMOSTRACIÓN: Dado un grafo no dirigido $G = (V, E)$ el Minimum-Weight Two-Connected Spanning Network [3] es un caso particular del CmTNSSP con $m = 1, \quad Q = |V|, \quad W = \phi$, y $\bar{V}_1 = \phi$. Esta última condición podemos obligarla haciendo que los elementos de la matriz de costos de conexión D, especificada en la definición del CmTNSSP [1.2.2] sean lo suficientemente grandes.

Como el Minimum-Weight Two-Connected Spanning Network [3] es un problema $\mathcal{NP}$-Difícil queda así demostrado que el CmTNSSP también es $\mathcal{NP}$-Difícil.

1.3. Trabajos Relacionados

En la revisión de la literatura no hemos encontrado referencias al *Capacitated m Two-Node-Survivable Star Problem* como tal. El trabajo relacionado desarrollado por Baldacci et al. [4] trata la resolución exacta del *Capacitated m Ring Star Problem*. Dicho problema es ligeramente diferente al tratado en esta tesis. En el problema CmRSP tratado por Baldacci et al. las estructuras 2-nodo-conexas son exclusivamente ciclos, mientras que en nuestro problema (CmTNSSP) las estructuras 2-nodo-conexas pueden no ser ciclos. El CmTNSSP es por lo tanto una relajación del CmRSP. Toda solución factible del CmRSP será solución del CmTNSSP pero no a la inversa.

En [4] se consideran dos formulaciones matemáticas del problema CmRSP, la formulación de dos índices *Two-index formulation*) y la formulación de dos flujos de productos (*Two-commodity flow formulation*) y se resuelve el problema CmRSP de manera exacta comparando el resultado de la implementación de ambas formulaciones para un conjunto de casos de prueba prestablecidos.

Algunos autores también tratan el CmRSP y su resolución exacta [5] y otros autores lo hacen mediante métodos de resolución aproximados. Como ejemplo podemos citar a Naji-Azimi et al. [6] y Mauttone et al. quienes utilizan una metaheurística GRASP, resolviendo instancias de hasta 101 nodos.

Existen también algunos trabajos relacionados que sirven de fundamento al trabajo de Baldacci et al. tales como el problema de la localización de ciclos medios en redes. Dicho problema es un caso particular del CmRSP y es tratado por Labbé et al. [7]. En dicho problema se busca

construir una red que consta de un ciclo principal y nodos colgantes cuyo costo total, compuesto de los costos de las aristas que pertenecen al ciclo (costos de ruteo) más los costos de conexión de las aristas con incidencia en nodos colgantes, es mínimo. Aquí se acota el costo total de conexión a un valor p dado. En [8] los mismos autores resuelven el RSP (Ring Star Problem), un problema muy semejante al del ciclo medio en redes, sin imponer restricciones de costos en las aristas que no pertenecen al ciclo. Solo se mencionan en este problema restricciones de servicio tales como cantidad de nodos colgantes conectados a un mismo nodo perteneciente al ciclo. En [8] también se resuelve el RSP en forma exacta.

Otros problemas de similar objetivo pero con diferencias en las estructuras consideradas se tratan en [9]. Tanto en el CmRSP como en el problema CmTNSSP tratado en esta tesis las estructuras buscadas son ciclos o estructuras 2-conexas, mientras que en estos otros problemas las estructuras buscadas son conexas simples sin redundancia tales como caminos o árboles.

En este trabajo proponemos una variante hasta donde sabemos no estudiada para diseñar soluciones de bajo costo 2-nodo-conexas, de utilidad en el contexto de construcción de redes de telecomunicaciones con cierto nivel de supervivencia.

1.4. Estructura del documento

El Capítulo 2 hace referencia al marco teórico y el estudio de propiedades estructurales de los grafos 2-conexos sobre las cuales basamos la definición del problema tratado y que utilizamos en la construcción de los algoritmos.

El Capítulo 3 trata el modelo de programación matemática del problema CmTNSSP.

Los Capítulos 4 y 5 abarcan la resolución del problema mediante una metaheurística, detallando la metodología utilizada.

En el Capítulo 6 se detalla un estudio experimental consistente en el armado de los casos de prueba, ejecución de los algoritmos y evaluación de los resultados.

El Capítulo 7 trata las conclusiones de la tesis y las líneas de trabajo futuro que se pueden seguir a partir del estudio del problema CmTNSSP.

Capítulo 2

Marco Teórico

2.1. Definiciones previas

Estableceremos a continuación algunos conceptos sobre grafos conexos con una mínima redundancia, que servirán como una introducción de las propiedades estructurales de 2-conectividad y su uso posterior en la resolución del problema tratado en esta tesis, tanto en su resolución exacta como en su resolución mediante el uso de métodos aproximados. De estas propiedades podemos extraer resultados importantes, que diferencian los problemas estudiados hasta el momento con el problema CmTNSSP que vamos a resolver en el desarrollo de esta tesis.

2.1.1. Introducción

Consideremos un conjunto de vértices V y una función de distancia $d(.)$ definida sobre el producto cartesiano $V \times V$ tal que $d(.)$ cumpla con las siguientes propiedades:

$$
\begin{aligned}
&d(u,u) = 0 \qquad \forall u \in V \\
&d(u,v) \geq 0 \qquad \forall u, v \in V \\
&d(u,v) = d(v,u) \qquad \forall u, v \in V \\
&d(u,w) \leq d(u,v) + d(v,w) \qquad \forall u, v, w \in V
\end{aligned}
$$

Definimos $d(u,v)$ como el peso o largo de la arista (u,v). El conjunto de vértices V y un subconjunto $E \subseteq V \times V$ definen el grafo $G = (V,E)$ cuyo peso total es la suma de los pesos de cada una de las aristas.

$$D(E) = \sum_{(u,v)\in E} d(u,v)$$

Consideraremos ahora el problema de construir una red de cubrimiento de los vértices V tal que $|V| \geq 3$ que sea 2-conexa y de costo mínimo.

Un ciclo Hamiltoniano (ciclo que contiene todos los vértices del conjunto V) es solución factible de este problema.

Un ciclo Hamiltoniano de costo mínimo es solución factible y óptima en el problema conocido como TSP ($Travelling\ Salesman\ Problem$) ampliamente tratado desde hace varias décadas en la literatura [10] [11] [12].

2.2. Propiedades estructurales.

Citaremos en extenso algunas propiedades estructurales de los grafos de cubrimiento 2-conexos (tanto 2-nodo-conexos como 2-arista-conexos), que nos permitirán fundamentar más adelante algunos de los algoritmos utilizados, y que constituyen en buena medida una herramienta para la búsqueda de soluciones óptimas a partir de la consideración de dichas propiedades.

DEFINICIÓN **2.1** *Dado un grafo $G = (V, E)$ definimos* punto de articulación *a todo vértice cuya remoción aumenta el número de componentes conexas del grafo G. Análogamente definimos* arista puente *a toda arista del grafo cuya remoción incrementa el número de componentes del mismo.*

DEFINICIÓN **2.2** *Dado un grafo conexo $G = (V, E)$ decimos que el mismo es 2-nodo conexo si ninguno de sus vértices es punto de articulación. Análogamente decimos que un grafo conexo es 2-arista conexo si ninguna de sus aristas es una arista puente.*

De las definición 2.2 se desprende que si un grafo es 2-nodo-conexo también es 2-arista-conexo. El recíproco no es válido, un grafo puede ser 2-arista-conexo y no ser 2-nodo-conexo.

LEMA **2.1** [Berge] *Dado un grafo $G = (V, E)$ los enunciados siguientes son equivalentes:*

a) *G es 2-nodo conexo.*
b) *Dos vértices cualesquiera de G pertenecen a un mismo ciclo.*
c) *Dos aristas cualesquiera de G pertenecen a un mismo ciclo.*
d) *Un vértice y una arista cualesquiera de G pertenecen a un mismo ciclo.*

Estas equivalencias se mantienen para grafos 2-arista conexos cambiando "ciclo" por "circuito".

Demostración. Trivial [3].

LEMA **2.2** *Sea G=(V,E) un grafo 2-conexo, consideremos $G' = (V', E')$ un subgrafo de G inducido por V'. Si remplazamos E' en G por un conjunto de aristas E'' definidas sobre V' donde $G'' = (V', E'')$ es un grafo 2-conexo, entonces el grafo $G^* = (V, (E \setminus E') \cup E'')$ es 2-conexo.*

Demostración. Supongamos que G^* no sea un grafo 2-conexo y consideremos un vértice v que es un punto de articulación de G^*. Dado que G es un grafo 2-conexo cada componente de $G^*(V \setminus \{v\})$ deberá contener un vértice de V'. Como G' es 2-conexo podemos encontrar un camino in G^* que no contenga al vértice v en una de las componentes, lo cual contradice la definición de v como punto de articulación [3].

TEOREMA 2.3 *Para cualquier conjuntos de vértices V con función de distancia $d(.)$ definida en $V \times V$ existe un grafo $G = (V, E)$ 2-conexo de costo mínimo que satisface las siguientes condiciones:*

a) *Todo vértice en G tiene grado 2 o 3.*
b) *Si removemos una o dos aristas en G obtenemos una arista puente en alguna de las componentes conexas resultantes de G.*

Demostración. Esta demostración es bastante extensa y no la transcribiremos. La misma fue desarrollada por Monma et al. y se encuentra en la bibliografía [3].

TEOREMA 2.4 *Sea $G = (V, E)$ un grafo que satisface las siguientes condiciones:*

a) *G es 2-conexo*
b) *Todo vértice de G tiene grado 2 o 3.*
c) *G es arista minimal.*
d) *Removiendo un par de aristas en G deja una arista puente en una de las componentes conexas resultantes*

Se cumple entonces que G es el único grafo 2-conexo de cubrimiento de costo mínimo para la función de distancia canónica.

Demostración. De igual forma que en el Teorema 2.3 no incluiremos la demostración encontrándose el detalle de la misma en [3].

COROLARIO 2.5 *Si $G = (V, E)$ es un grafo 2-conexo que no sea un ciclo y satisface las condiciones del Teorema 2.3 entonces dicho grafo contiene el grafo mostrado en la Figura 2.1 como grafo nodo inducido.*

Demostración. Dado que el grafo G es 2-conexo contendrá un ciclo C. Por la condición b) del Teorema 2.3 C contiene al menos 3 vértices. Dado por hipótesis que G no es un ciclo, y C forma parte de G, existirá un vértice v que no pertenece a C. Por el Lema 2.1 v esta incluido en el mismo ciclo que los otros vértices de C. Por la condición a) del Teorema 2.3 existen dos vértices x e y en C extremos de un camino que pasa por el vertice v y no comparte ningún otro vértice de C. La condición b) del Teorema 2.3 implica que el grafo nodo inducido de esta manera, debe ser necesariamente como el que se muestra en la Figura 2.1 [3].

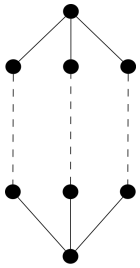

Figura 2.1: Grafo 2-conexo nodo inducido.

Nota: Las líneas punteadas corrresponden a caminos de una o más aristas.

2.3. Comparación con el TSP

El TSP (Traveling Salesman Problem) es un problema ampliamente tratado en la literatura y se ha trabajado intensamente en heurísticas que obtienen ciclos de costo mínimo[13][14][15][16][17].

Sabemos por el Teorema 2.4 que un ciclo no necesariamente es una solución optimal 2-conexa que cubre el conjunto de nodos V del grafo $G = (V, E)$. Este resultado es una de las motivaciones fundamentales en el estudio del problema de esta tesis, dado que podrían hallarse soluciones de mejor costo que las obtenidas en problemas semejantes tales como el CmRSP [4], citado en la sección 1.3.

Definiremos a continuación algunos conceptos y notaciones necesarios para establecer una relación entre los costos de soluciones 2-conexas cíclicas y no cíclicas.

DEFINICIÓN **2.3** *Un matching perfecto de $W \subseteq V$ en un grafo G=(V,E) es un subconjunto de aristas $M \subseteq E$ cuyos vértices en W se encuentran exactamente una vez.*

Para todo conjunto de vértices V y una función de distancias $d(.)$ denotaremos $C_{opt}(V)$ como el ciclo hamiltoniano óptimo en V y $TC_{opt}(V)$ como la solución de cubrimiento 2-conexa óptima en V y $d(C_{opt}(V))$ y $d(TC_{opt}(V))$ sus respectivos costos.

LEMA **2.6** *En todo grafo $G = (V, E)$ cúbico (grafo donde todo sus vértices tienen grado 3), 2-arista-conexo y con función de distancia c(.) no negativa, existe un matching perfecto M tal que $c(M) \leq \frac{1}{3}c(E)$.*

Supongamos que $G = (V, E)$ es una solución óptima 2-conexa para una cierta función de distancia $d(.)$. Por desigualdad triangular se cumple que $d(u, v)$ es a lo sumo el largo del camino más corto entre u y v en G. Si consideramos el peor caso podemos asumir que dicha distancia es igual para todas las aristas $(u, v) \notin E$. Si asumimos que la función distancia toma valores

racionales de la forma $\frac{m_i}{n_i}$ con $i = 1, 2, \cdots \frac{(|V|)(|V|-1)}{2}$ existiría un entero (por ejemplo el mínimo común múltiplo de los n_i) tal que multiplicado por cada una de las distancias $\frac{m_i}{n_i}$ nos daría como resultado una función de distancia entera.

Finalmente todas las aristas de largo $h > 1$ se podrían sustituir por un camino de h aristas de largo 1 sin modificar el costo total de la solución y el mejor ciclo en este problema aumentado no puede ser de costo menor que en el problema original. Podremos entonces definir una cota superior para la relación $d(C_{opt}(V))$ / $d(TC_{opt}(V))$ para la función de distancia canónica.

TEOREMA 2.7 Monma et al. [3]. *Dado un conjunto cualesquiera de vértices V y una función de distancia $d(.)$ definida para dicho conjunto se cumple que:*

$$\frac{d(C_{opt}(V))}{d(TC_{opt}(V))} \leq \frac{4}{3} \tag{2.1}$$

Podemos agregar además que esta relación puede ser arbitrariamente aproximada a la cota superior de $\frac{4}{3}$ para la clase de grafos mostrados en la Figura 2.1 y con una función de distancia $d(.)$ canónica.

Demostración. Sea $G = (V, E)$ un grafo 2-conexo minimal que cumple con las condiciones del Teorema 2.3. Asumimos que G no es un ciclo y que $d(.)$ es una función de distancia canónica. Sea $\bar{V}$ el conjunto de los vértices de grado tres del grafo G, entonces G es una división del grafo cúbico sin aristas puente $\bar{G} = (V, \bar{E})$ obtenido contrayendo repetidamente las aristas de los vértices de grado dos, y cada $(u, v) \in \bar{E}$ corresponde a un camino maximal en G cuyos vértices internos tienen grado dos. Consideremos para cada $(u, v) \in \bar{E}$ una distancia $\bar{d}(u, v)$ igual a la cantidad de aristas en el camino (u, v), se cumple entonces que $\bar{d}(\bar{E}) = d(E)$. Por el Lema 2.6 $\bar{G}$ incluye un matching perfecto M con costo de a los sumo $\frac{1}{3}\bar{d}(\bar{E})$. Duplicando las aristas correspondientes a los caminos formados por las propias aristas de M en G obtenemos un multigrafo de Euler $G' = (V, E')$ tal que $d(E') \leq \frac{1}{3}d(E)$. Dado que la función de distancia d satisface la desigualdad triangular podemos construir un circuito hamiltoniano sobre V tal que $d(C) \leq d(E') \leq \frac{4}{3}d(TC_{opt}(V))$. La cota de $\frac{4}{3}$ es estrecha considerando el grafo de la figura 2.1, en los que cada par de vértices de grado tres s y t tienen k vertices internos. Bajo la distancia canónica el mejor ciclo tiene distancia $4k + 2$. Dado que el grafo tiene $3k + 3$ aristas esto da una razón de $(4k + 2)/(3k + 3)$ la cual podemos aproximar a $\frac{4}{3}$ cuando $k \to \infty$ [3].

Definamos ahora el modelo matemático del TSP.

Sea V un conjunto de vértices y $d(.)$ una función de distancia que cumple con la desigualdad triangular representada por $C = \{c_{ij}\}$, matriz de costos simétrica, definimos el Travelling Salesman Problem (TSP) como:

$$min \sum_{i,j \in V} c_{ij} * x_{ij} \tag{2.2}$$

s.a.

$$\sum_{v \in V \setminus \{u\}} x_{uv} = 2 \qquad \forall\, u \in V, \tag{2.3}$$

$$\sum_{u \in S, v \in V \setminus S} x_{uv} \geq 2 \qquad \forall\, S \subset V \text{ tal que } |S| \geq 2, \tag{2.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in V. \tag{2.5}$$

Para este modelo la restricción 2.3 nos asegura que el grado de todos los nodos es dos y la restricción 2.4 impone que no existan subtours (es decir ciclos no hamiltonianos). La restricción 2.5 indica que la variable x_{ij} es binaria, lo que implica que la arista (i, j) pertenece o no a la solución.

Consideremos la relajación al problema de programación lineal del TSP expresado anteriormente, es decir cambiando la restricción 2.5 por la siguiente:

$$0 \leq x_{ij} \leq 1 \qquad x_{ij} \in \mathbb{R}, \forall\, i, j \in V. \tag{2.6}$$

Al problema definido de esta manera se le denomina *subtour polytope* [3]. La solución óptima de este problema de programación lineal la denotaremos S_{opt} y su costo $d(S_{opt})$.

Trivialmente sabemos que toda relajación de un problema de programación entera a un problema de programación lineal, tiene un valor de objetivo “mejor” al problema original. Por lo tanto sabemos que:

$$d(S_{opt}) \leq d(C_{opt}). \tag{2.7}$$

TEOREMA 2.8 [Cunningham] [3] *Dado un conjunto cualesquiera de vértices V y una función de distancia $d(.)$ definida para dicho conjunto se cumple que:*

$$d(S_{opt}(V)) \leq d(TC_{opt}(V)) \tag{2.8}$$

Demostración. Sea H un grafo de cubrimiento 2-conexo mínimo de V. Si agregamos una arista duplicada por cada arista del grafo H, obtendremos un multigrafo H'. Cada vértice del grafo H' tendrá al menos grado cuatro y es cuatro arista-conexo. El costo de dicho grafo será $2d(H)$. Si elegimos un vértice v cuyo grado sea mayor a cuatro y consideramos dos aristas incidentes cualesquiera (v, u) y (v, w) y las sustituimos por la arista (u, w), preservaremos la cuatro arista-conexidad del grafo debido al resultado de Lovász [18] y no incrementaremos el costo del mismo debido a la propiedad de desigualdad triangular. Realizando esta transformación para todos los vértices de grado mayor a cuatro obtenemos un grafo regular con todos sus vértices de grado cuatro y cuatro arista-conexo con costo de a lo sumo $2d(H)$. Definiendo $\bar{x}(u, v)$ como 0.5 por la cantidad de aristas existentes entre u y v, $\bar{x}$ pertenece al *subtour polytpe* por lo tanto $d(S_{opt}) \leq d\bar{x} \leq d(TC_{opt})$. Este resultado nos da una cota inferior para el TC_{opt}, a través de la resolución de un problema de programación lineal, el cual es menos costoso en recursos computacionales.

2.4. Generalización

Generalizando los problemas de construcción de redes 2-conexas podemos obtener soluciones factibles considerando nodos de Steiner. Dichos nodos serán utilizados en la solución solamente si mejoran su costo.

Dado un conjunto de vértices V y un conjunto $D \subseteq V$ definiremos $STC_{opt}(D, V)$ a la solución óptima del grafo de cubrimiento 2-conexo, considerando el conjunto D como los vértices que deberán formar parte de la solución obligatoriamente y $V \setminus D$ al conjunto de vértices opcionales o de Steiner que serán incluidos en la solución si y solo si mejoran el costo de la misma.

TEOREMA 2.9 *[3] Dado un conjunto cualesquiera de vértices V, un subconjunto $D \subseteq V$ y una función de distancia $d(.)$ se cumple que:*

$$\frac{d(TC_{opt}(D))}{d(STC_{opt}(D, V))} \leq \frac{4}{3} \tag{2.9}$$

Demostración. Sea $G = (W, E)$ una solución óptima 2-conexa de Steiner con $W = D \cup U$, $U \subseteq V \setminus D$, por lo tanto $D \subseteq W$.

Sabemos (por resultados anteriores la primera de las desigualdades y trivialmente la segunda dado que $D \subseteq W$) que:

$$d(TC_{opt}(D)) \leq d(C_{opt}(D)) \leq d(C_{opt}(W)) \tag{2.10}$$

y además por la definición de la solución óptima que establecimos:

$$d(STC_{opt}(D,V)) = d(STC_{opt}(D,W)) = d(TC_{opt}(W)) \tag{2.11}$$

Esto implica que:

$$\frac{d(TC_{opt}(D))}{d(STC_{opt}(D,V))} \leq \frac{d(C_{opt}(W))}{d(TC_{opt}(W))} \tag{2.12}$$

y sabemos que el segundo miembro de la desigualdad 2.12 tiene por cota superior $\frac{4}{3}$ por el Teorema 2.7, por lo tanto queda demostrada la hipótesis.

TEOREMA 2.10 *Dado un conjunto cualesquiera de vértices V, un subconjunto $D \subseteq V$ y una función de distancia $d(.)$ se cumple que:*

$$\frac{d(C_{opt}(D))}{d(STC_{opt}(D,V))} \leq \frac{16}{9} \tag{2.13}$$

Demostración.

$$\frac{d(C_{opt}(D))}{d(STC_{opt}(D,V))} = \frac{d(TC_{opt}(D))}{d(STC_{opt}(D,V))} \quad \frac{d(C_{opt}(D))}{d(TC_{opt}(D))} \tag{2.14}$$

Por Teorema 2.7 se cumple que:

$$\frac{d(C_{opt}(D))}{d(TC_{opt}(D))} \leq \frac{4}{3}$$

y por Teorema 2.9 se cumple que:

$$\frac{d(TC_{opt}(D))}{d(STC_{opt}(D,V))} \leq \frac{4}{3}$$

con lo cual queda demostrada la desigualdad 2.13.

2.5. Conclusiones

La descripción de las propiedades de los grafos 2-conexos que hemos desarrollado en este capítulo y la obtención de cotas que indican una mejora sobre las soluciones 2-conexas cíclicas, fundamentan el estudio del problema tratado en esta tesis y abre las puertas a la búsqueda de soluciones de mejor costo que las del CmRSP para problemas de interconexión de redes.

Capítulo 3

Formulación del Modelo Matemático del CmTNSSP.

3.1. Introducción

En este capítulo estudiaremos en base a formulaciones de programación matemática el problema CmTNSSP, mostrando la similitud con los modelos utilizados para definir otros problemas de topología similar que incluyen requerimientos de conexión mínimos y la adaptación de dichos modelos al problema tratado en esta tesis.

3.2. El Problema de Steiner Generalizado

Antes de pasar a la definición del modelo de nuestro problema, daremos algunas definiciones previas necesarias y definiremos el Problema de Steiner Generalizado (GSP por sus siglas en inglés) dado que el concepto de conectividad aplicado en ese modelo es fuertemente utilizado en la definición del CmTNSSP.

El Problema de Steiner Generalizado y su formulación matemática fue tratado extensamente por varios autores [19] [20].

Consiste en encontrar redes de costo mínimo que cumplan con ciertos requerimientos de conectividad entre cada par de nodos pertenecientes a dicha red.

El diseño de redes con conectividad simple consiste en encontrar un camino entre todo par de nodos de la red. Si dicha red no contiene nodos opcionales el problema consiste en hallar el árbol de cubrimiento mínimo (MST por su sigla en inglés). Dicho problema tratado por Kruskal en 1956 es bien conocido y es de resolución exacta con orden polinomial. [21].

Cuando el árbol de cubrimiento mínimo puede utilizar nodos opcionales o de Steiner el problema se denomina MSTP (Minimum Steiner Tree Problem)[22]. Este problema no es de orden polinomial, encontrándose dentro de la categoría de problemas NP-Completos. El MSTP es

un caso particular del GSP donde el requerimiento de conectividad entre todo par de nodos terminales es uno.

A medida que se introducen requerimientos de conectividad entre cada par de nodos el problema crece en complejidad, puede haber pares de nodos que requieran conectividad múltiple, es decir que habrá k caminos disjuntos entre dicho par de nodos, y también puede haber nodos con conectividad mínima cero, es decir que puede haber nodos opcionales o de Steiner.

En el problema CmTNSSP no hay requerimientos de conectividad definidos a priori. Dado un par de nodos terminales pertenecientes a una misma componente H_k, la conectividad entre ellos podrá ser dos si ambos terminales se encuentran en la estructura 2-nodo-conexa, o uno si alguno de estos terminales es un nodo colgante de acuerdo a los visto en la subsección 1.2.1.

Análogamente dado un par de nodos i, j con $i \in H_k$ y $j \in H_l$ con $k \neq l$ la nodo-conectividad entre dicho par de nodos será uno, dado que hay un único camino nodo disjunto entre i y j, y este pasa por el nodo d.

3.2.1. Definición del Problema de Steiner Generalizado

Dado un grafo conexo $G = (V, E)$, una matriz $C = \{c_{ij}\}_{(i,j)\in E}$ de costos positivos asociados a las aristas del grafo, un subconjunto de nodos $T \subseteq V$ denominados nodos terminales, tal que $|T| > 2$, y una matriz de requerimientos de conexión entre nodos terminales $R = \{r_{ij}\}_{i,j\in T}$ donde los r_{ij} son enteros no negativos. El Problema de Steiner Generalizado consiste en encontrar un subgrafo $G' = (V', E')$ del grafo G tal que $T \subseteq V'$, cada par de nodos terminales $i, j \in T, i \neq j$ tenga r_{ij}-conectividad en G' y tal que el costo total de G' sea mínimo. La conectividad puede ser especificada como nodo-conectividad o arista-conectividad.
Con respecto a los nodos que no pertenecen al conjunto T no hay requisitos de conectividad pudiéndose asumir conectividad 0. Estos nodos, se definen como nodos opcionales o nodos de Steiner y pueden formar parte de la solución si su inclusión mejora los costos de la solución.

Sean las siguientes variables de decisión:

$$x_{ij} = \begin{cases} 1 & \text{si la arista } (i,j) \text{ pertenece a la solución} \\ 0 & \text{o.c.} \end{cases} \tag{3.1}$$

$$y_{(i,j)}^{(u,v)} = \begin{cases} 1 & \text{si la arista } (i,j) \text{ es usada en el camino que va de } u \text{ a } v \text{ en el sentido de } i \text{ a } j \\ 0 & \text{o.c.} \end{cases} \tag{3.2}$$

3.2.2. Formulacion Matemática del GSP (versión arista-conectividad)

$$min \sum_{i,j\in V} c_{ij} * x_{ij} \tag{3.3}$$

s.a.

$$\sum_{(u,j)\in E} y_{(u,j)}^{(u,v)} \geq r_{uv} \qquad \forall u, v \in T, u \neq v \tag{3.4}$$

$$\sum_{(i,v)\in E} y_{(i,v)}^{(u,v)} \geq r_{uv} \qquad \forall u, v \in T, v \neq u \tag{3.5}$$

$$\sum_{(i,p)\in E} y_{i,p}^{(u,v)} - \sum_{(p,i)\in E} y_{(p,i)}^{(u,v)} \geq 0 \quad \forall u, v \in T \quad \forall p \in V \setminus u, v \tag{3.6}$$

$$y_{(i,j)}^{(u,v)} + y_{(j,i)}^{(u,v)} \leq x_{i,j} \quad \forall u, v \in T, u \neq v, \quad \forall (i,j) \in E \tag{3.7}$$

3.3. El Problema CmTNSSP

Basándonos en alguno de los aspectos de las restricciones de conexión del Problema de Steiner Generalizado y adaptando estas restricciones a la topología de nuestro problema expresaremos a continuación un modelo matemático para el CmTNSSP.

3.3.1. Definiciones Previas

De acuerdo a la definición del CmTNSSP dada en la subsección 1.2.2 consideremos un grafo $G = (V, E)$ donde V es el conjunto de vértices del grafo, E es el conjunto de aristas y d el depot.

Dado $T \subseteq V \setminus \{d\}$ el conjunto de nodos terminales del grafo G.

Dado $\hat{T}$= $T \cup \{d\}$ el conjunto de terminales con el nodo *depot* incluido.

Dado $W = V \setminus \hat{T}$ el conjunto de los nodos de Steiner del grafo G.

Dado m la cantidad de componentes del grafo G.

Definamos el conjunto de los nodos adyacentes al nodo $i \in V$ como $Adj(i)$ de manera que:

$$Adj(i) = \{j \in V : (i,j) \in E\} \tag{3.8}$$

El grafo H a construir será de la forma:

$$H = H_1 \cup H_2 \cup H_3 ..... \cup H_m$$

donde cada componente H_i la habíamos definido como:

$$H_i = G'_i \cup S_i \quad i = 1, ...m$$

cumpliéndo con las condiciones especificadas en 1.2, 1.3 y 1.4.

También habíamos definido $C = \{c_{ij}\}_{i,j \in V}$ como la matriz de costos de ruteo del grafo, es decir los costos de una cierta arista (i, j) cuando dicha arista pertenece a algún G'_k, con $k = 1...m$.

Y habíamos definido $D = \{d_{ij}\}_{i,j \in V}$ como la matriz de costos de conexión del grafo, es decir los costos de la arista (i, j) cuando dicha arista pertenece a S_k, con $k = 1...m$.

Definamos ahora las variables de decisión del modelo:

$$X_i^k = \begin{cases} 1 & \text{si el nodo } i \in V \text{ está en } G'_k \text{ (estructura 2-conexa de la subred } H_k) \\ 0 & \text{o.c.} \end{cases} \tag{3.9}$$

$$Y_i^k = \begin{cases} 1 & \text{si el nodo } i \in T \text{ es un nodo colgante de } G'_k \text{ (estructura 2-conexa de la subred } H_k) \\ 0 & \text{o.c.} \end{cases} \tag{3.10}$$

$$Y_{i,j}^k = \begin{cases} 1 & \text{si } i \in T \text{ y } j \in V \text{ están conectados mediante la arista } (i, j) \in E, \\ & \text{siendo } i \text{ un nodo colgante de } G'_k \text{ (estructura 2-conexa de la subred } H_k) \\ 0 & \text{o.c.} \end{cases} \tag{3.11}$$

$$y_{(i,j)}^{(u,v,k)} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es usada en el camino que va de } u \text{ a } v \\ & \text{en el sentido de } i \text{ a } j \text{ dentro de la componente } H_k \\ 0 & \text{o.c.} \end{cases} \tag{3.12}$$

$$X_{i,j}^k = \begin{cases} 1 & \text{si hay un camino entre los nodos } i \text{ y } j \text{ dentro de la componente } H_k \\ 0 & \text{o.c.} \end{cases} \tag{3.13}$$

$$x_{i,j} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es usada en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{3.14}$$

$$w_{i,j} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es arista colgante usada en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{3.15}$$

$$z_i = \begin{cases} 1 & \text{si el nodo colgante } i \text{ es usado en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{3.16}$$

3.3.2. Formulación Matemática del CmTNSSP

$$min \sum_{k=1}^{m} (\sum_{i,j \in V} c_{ij}(x_{ij} - w_{ij}) + \sum_{i,j \in V} d_{ij} w_{ij}) \tag{3.17}$$

sujeto a:

$$\sum_{k=1}^{m} X_i^k + Y_i^k = 1 \qquad \forall i \in T \tag{3.18}$$

La restricción 3.18 nos asegura que cualquier nodo terminal, o bien pertenece a la estructura 2-conexa de una única componente H_k o bien está colgado de esta.

$$X_i^k + Y_i^k \leq 1 \qquad \forall i \in T,\ \forall k \in 1 \cdots m \tag{3.19}$$

La restricción 3.19 implica que cualquier nodo terminal, pertenece a la estructura 2-conexa de alguna componente o está colgado de esta.

$$\sum_{k=1}^{m} X_d^k = m \tag{3.20}$$

El nodo distinguido d (depot) pertenece a las m componentes 2-conexas (Restricción 3.20).

$$\sum_{i=1}^{m} X_i^k \leq 1 \qquad \forall i \in W \tag{3.21}$$

La ecuación 3.21 asegura que los nodos de Steiner pertenecen a las estructuras 2-conexas de las componentes H_k o no participan de la solución.

$$\sum_{i \in T} (X_i^k + Y_i^k) \leq Q \qquad \forall k \in 1 \cdots m \tag{3.22}$$

La ecuación 3.22 es la restricción de capacidad, cada componente H_k puede tener hasta Q nodos terminales sin contar el nodo d.

$$Y_{ij}^k \leq x_{ij} \qquad \forall i \in T,\ \ \forall j \in V,\ \ \forall k \in 1 \cdots m \tag{3.23}$$

La restricción 3.23 implica que si i y j están conectados en la componente H_k siendo i un nodo colgante, entonces la arista (i, j) forma parte de la solución.

$$Y_i^k = \sum_{j \in Adj(i)} Y_{ij}^k \qquad \forall i \in T \quad \forall k \in 1 \cdots m \tag{3.24}$$

La ecuación 3.24 asegura que si el nodo i cuelga de la estructura 2-conexa de la componente H_k entonces cuelga a través de una sola arista.

$$\sum_{(u,j) \in E} y_{(u,j)}^{(u,v,k)} \geq 2X_{(u,v)}^k - Y_u^k \qquad \forall u, v \in \hat{\mathbf{T}}, u \neq v, \forall k \in 1 \cdots m \tag{3.25}$$

$$\sum_{(i,v) \in E} y_{(i,v)}^{(u,v,k)} \geq 2X_{(u,v)}^k - Y_v^k \qquad \forall u, v \in \hat{\mathbf{T}}, v \neq u, \forall k \in 1 \cdots m \tag{3.26}$$

Las restricciones 3.25 y 3.26 son las restricciones de 2-conexidad que nos aseguran que para todo vértice de la estructura 2-conexa de la componente H_k, salen y llegan al menos 2 caminos. Si alguno de los extremos es un nodo colgante de la componente entonces los segundos miembros de la desigualdad en 3.25 y 3.26 son 1, por lo tanto las restricciones mencionadas exigirán para este caso un solo camino.

$$\sum_{(i,p) \in E} y_{i,p}^{(u,v,k)} - \sum_{(p,i) \in E} y_{(p,i)}^{(u,v,k)} \geq 0 \quad \forall u, v \in \hat{\mathbf{T}}, \quad \forall p \in V \setminus u, v \quad \forall k \in 1 \cdots m \tag{3.27}$$

La ecuación 3.27 es una ecuación de balance para los nodos internos del camino.

$$y_{(i,j)}^{(u,v,k)} + y_{(j,i)}^{(u,v,k)} \leq x_{i,j} \quad \forall u, v \in \hat{\mathbf{T}}, u \neq v, \quad \forall (i,j) \in E \quad \forall k \in 1 \cdots m \tag{3.28}$$

La ecuación 3.28 es la restricción normativa para que los caminos sean arista disjuntos. Para nuestro caso donde aplicamos 2-nodo-conexidad debemos transformar previamente el grafo inicial, utilizando la técnica descripta en 3.3.2.1.

$$\sum_{i \in V} X_i^k \geq 1 \quad \forall k \in 1 \cdots m \tag{3.29}$$

La restricción 3.29 obliga a que cada componente H_k tenga al menos un nodo (terminal o de Steiner) en la estructura 2-conexa. Con esta restricción y con las ecuaciones 3.25, a la 3.28 nos aseguramos que la estructura de cada componente H_k es por lo menos un ciclo que incluye un par de nodos cualesquiera (terminales u opcionales) y el nodo d (depot).

Restricciones de conexidad simple:

$$X_i^k + X_j^k \leq 1 + X_{i,j}^k \qquad \forall i \in V \quad \forall j \in V \quad \forall k \in 1 \cdots m \tag{3.30}$$

$$X_i^k + Y_j^k \leq 1 + X_{i,j}^k \qquad \forall i \in V \quad \forall j \in T \quad \forall k \in 1 \cdots m \tag{3.31}$$

$$Y_i^k + Y_j^k \leq 1 + X_{i,j}^k \qquad \forall i \in T \quad \forall j \in T \quad \forall k \in 1 \cdots m \tag{3.32}$$

Las ecuaciones 3.30, 3.31 y 3.32 indican que si los nodos están presentes en la solución (en la misma componente H_k) entonces están conectados.

$$2X_{i,j}^k \leq X_i^k + X_j^k + Y_i^k + Y_j^k \qquad \forall i \in V \quad \forall j \in V \quad \forall k \in 1 \cdots m \tag{3.33}$$

La restricción 3.33 determina que i y j están conectados en una componente H_k entonces se cumple que dichos nodos pertenecen a la solución y forman parte de la estructura 2-conexa de la componente H_k o son nodos colgantes de esta.

$$\sum_{k=1}^{m} X_{i,j}^k \leq 1 \qquad \forall i, j \in V \tag{3.34}$$

La ecuación 3.34 asegura que si los nodos i y j están conectados, solo lo hacen en una componente.

$$\sum_{k=1}^{m} Y_i^k \leq z_i \qquad \forall i \in T \tag{3.35}$$

$$\sum_{j \in Ady[i]} x_{i,j} - 1 \leq M(1 - z_i) \qquad \forall i \in T \quad M \in \mathbb{Z}^+, M \geq max(\delta_i) \;\; i = 1 \cdots |V| \tag{3.36}$$

Las ecuaciones 3.35 y 3.36 garantizan que los nodos colgantes tengan grado 1. Si el nodo es colgante estará obligado a ser de grado 1 y si no lo es, el grado será menor o igual al grado del nodo con mayor grado del grafo original.

$$\sum_{k=1}^{m} Y_{i,j}^{k} = w_{i,j} \qquad \forall i \in T \quad j \in Ady[i] \tag{3.37}$$

La ecuación 3.37 determina que si el nodo i cuelga del nodo j entonces la arista (i, j) es una arista colgante.

$$w_{i,j} \leq x_{i,j} \qquad \forall i \in T \quad j \in Ady[i] \tag{3.38}$$

La ecuacion 3.38 asegura que si una arista es colgante entonces pertenece a la solución.

$$Y_{i,j}^{k} \leq X_{j}^{k} \qquad \forall i \in T \quad \forall j \in Ady[i] \quad \forall k \in 1 \cdots m \tag{3.39}$$

La ecuación 3.39 obliga a que todo nodo colgante de una componente H_k lo hace de un nodo perteneciente a la estructura 2-conexa de dicha componente G'_k.

$$Y_{i}^{k} = 0 \qquad \forall i \in W \quad \forall k \in 1 \cdots m \tag{3.40}$$

La restricción 3.40 determina que los nodos de Steiner no pueden ser colgantes.

$$(\sum_{i \in Ady[j]} x_{j,i} - \sum_{i \in Ady[j]} Y_{i,j}^{k}) \geq 2X_{j}^{k} \qquad \forall j \in V \setminus T \quad \forall k \in 1 \cdots m \tag{3.41}$$

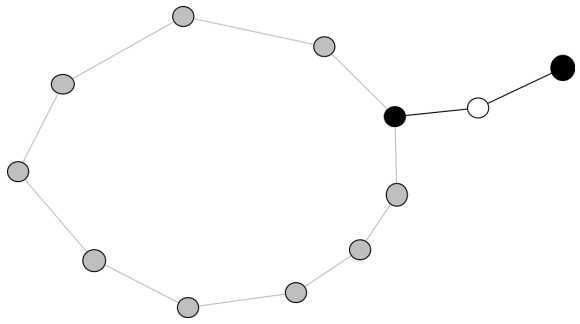

Figura 3.1: Forma de solución no factible evitada por la restricción 3.41

La restricción 3.41 fuerza que para todo nodo de Steiner que pertenece a la solución, su grado considerando solamente los nodos adyacentes que no son colgantes debe ser mayor o igual a 2. Esta restricción evita situaciones del tipo mostrado en la Figura 3.1

$$2y_{i,j}^{(u,v,k)} \leq X_{i,j}^{k} + X_{u,v}^{k} \quad \forall u, v \in \hat{\mathrm{T}}, \quad \forall i, j \in V, u \neq v \quad \forall k \in 1 \cdots m \tag{3.42}$$

La restricción 3.42 nos garantiza que si se usa la arista (i, j) para ir de u a v entonces i con j y u con v están conectados en la componente H_k

Hemos definido así el modelo de programación matemática del CmTNSSP. Dicho modelo es de naturaleza lineal entera, con cantidad polinomial de variables y restricciones respecto al tamaño del grafo.

3.3.2.1. Separación de vértices

Las restricciones expresadas anteriormente corresponden a una versión arista-conexa del problema tal como se detalla en la restricción 3.28. Como nosotros debemos resolver el CmTNSSP, el cual exige 2-nodo-conexidad se debe transformar el grafo $G = (V, E)$ en un grafo $\hat{G} = (\hat{V}, \hat{E})$ utilizando la técnica de separación de vértices ($Splitting\ Vertex$ en inglés)[23], que explicaremos a continuación.

Sea un grafo dirigido $G' = (V, E')$ tal que para cada arista $(i, j) \in E$ existe una arista dirigida $(i \to j) \in E'$ y una arista dirigida $(j \to i) \in E'$.

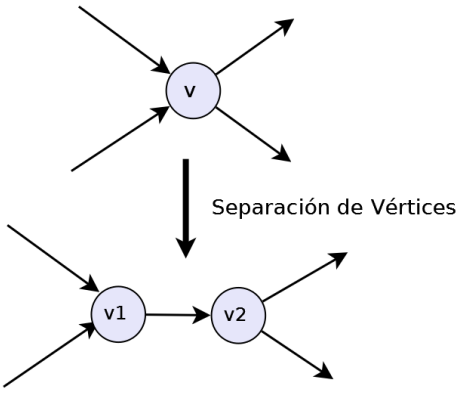

Figura 3.2: Separación de Vértices en un nodo v cualquiera del grafo G'.

En un grafo 2-nodo-conexo, dado cualquier par de vértices existen al menos 2 caminos nodo-disjuntos que los unen. Para imponer que el grafo solución del CmTNSSP cumpla con esta propiedad, lo transformaremos utilizando la técnica de separación de vértices. En dicha técnica imponemos que dado un vértice v en el grafo original G', este es sustituido por dos vértices

v_1 y v_2 en el grafo resultante $\hat{G}$, de forma tal que cada arista que llega a v llegará al nuevo vértice v_1 y cada arista que sale de v saldrá del vértice v_2. A su vez, v_1 y v_2 son unidos a través de una arista adicional con dirección $(v_1 \rightarrow v_2)$ y con costo cero. En la Figura 3.2 se representa gráficamente esta técnica.

Una vez que transformamos el grafo $G' = (V, E')$ en el grafo $\hat{G} = (\hat{V}, \hat{E})$ se aplica el modelo del CmTNSSP detallado en 3.3.2, obteniendo un subgrafo 2-arista-conexo solución óptima .

Esta solución tiene la propiedad, que dado cualquier vértice de alguna de las estructuras 2-conexas existen al menos 2 caminos que los unen sin repetir aristas. Por lo tanto, las aristas del tipo (v_1, v_2) (Figura 3.2)que fueron agregadas por la técnica de separación de vértices no se repiten en dichos caminos, logrando así, que tampoco se repitan los vértices determinados por estas aristas.

Realizando la transformación inversa a la separación de vértices, es decir, uniendo los vértices que fueron separados, se obtiene caminos que son 2-nodo-disjuntos, logrando de esta manera un grafo 2-nodo-conexo.

Capítulo 4

GRASP para el CmTNSSP

Este capítulo trata sobre el abordaje aproximado del problema con metaheurísticas. Primero se presenta un panorama general de las mismas, luego GRASP y más adelante el procedimiento de construcción de soluciones para el CmTNSSP en el contexto de GRASP.

4.1. Acerca de las metaheurísticas

4.1.1. Introducción

Considerando la naturaleza del problema a tratar y su complejidad, vamos a abordar la resolución aproximada del mismo utilizando una metaheurística.

Metaheurística deriva del griego, $meta$ "más allá de o en nivel superior a" y $heurisken$ "la técnica de descubrir nuevas estrategias para resolver problemas". El término "metaheurística" fue introducido por Glover [24] en su paper *"Future paths for integer progamming and links to artificial inteligence"*.

Metaheurística se puede definir como "los métodos de alto nivel que pueden ser usados como guías en el diseño de heurísticas subyacentes para resolver problemas específicos de optimización" [25].

Dos importantes características de las metaheurísticas son la intensificación y la diversificación [26]. Intensificación tiene por objeto realizar una búsqueda intensiva en un entorno local, mientras que diversificación tiene por cometido que el algoritmo explore el espacio de soluciones globalmente y en lo posible lo haga de manera eficiente. El término intensificación es llamado también explotación (por su traducción del inglés exploitation) y típicamente busca alrededor de las mejores soluciones (óptimos locales) con el fin de encontrar soluciones aun mejores. La diversificación, tambień llamada exploración recorre el espacio de búsqueda eficientemente a menudo utilizando procesos estocásticos.

El balance entre intensificación y diversificación es un elemento muy importante a tener en

cuenta a la hora de lograr eficiencia y performace en un algoritmo. Una escasa exploración del espacio de soluciones y demasiada intensificación podría dar como resultado que el algoritmo caiga en óptimos locales de los cuales sería dificultoso escapar y como consecuencia sería muy improbable o virtualmente imposible hallar el óptimo global. A la inversa, un exceso de diversificación unido a escasa intensificación generará una lenta convergencia hacia los óptimos y una degradación de la performance en general. El balance adecuado de estas dos características es una de las tareas más importantes en el diseño de una metaheurística.

4.1.2. Clasificación

Existen varias formas de clasificación de metaheurísticas de acuerdo a cada una de las características o propiedades mediante las cuales pueden ser diferenciadas.

Inspiradas o no en la naturaleza.

Esta clasificación está basada en el origen del algoritmo. Algunos algoritmos toman los comportamientos de diferentes especies animales que son exitosos a la hora de mantener la evolución o encontrar alimento de forma eficiente. Algunos ejemplos de estos son los Algoritmos Genéticos, (GA) [27], Colonia de Hormigas (ACO) [28], Cuckoo Search (CS) y Firefly Algorithm (FA) [29], y Enjambre de Partículas (PSO)[30].

Dentro de los no inspirados por la naturaleza se encuentran el Greedy Randomized Adaptative Search Procedure (GRASP) [31], Tabu Search (TS) [32] [33]) e Iterated Local Search (ILS) [34].

Esta clasificación puede volverse un tanto ambigua en algunos casos, dado que que existen metaheurísticas que pueden combinar algoritmos basados en la naturaleza con otros que no lo son.

De trayectoria o basada en poblaciones.

Otra característica usualmente utilizada para la clasificación de una metaheurística está basada en el número de soluciones que se consideran a la vez. Los algoritmos que trabajan con una solucion a la vez son llamados de trayectoria. Dentro de este tipo se encuentran las metaheurísticas que utilizan búsqueda locales tales como GRASP, TS, ILS, ,Variable Neighborhood Search (VNS) [35] y Variable Neighborhood Descendant (VND) [25].

Las metaheurísticas basadas en poblaciones obtienen un conjunto de soluciones en cada iteración que se irán combinando de alguna forma para obtener otro conjunto de soluciones mejoradas. Dentro de este tipo se encuentran los Algoritmos Genéticos (GA).

Función objetivo estática o dinámica.

Otra clasificación posible de las metaheurísticas está dada por la forma de utilizar la función

objetivo. Muchos de los algoritmos consideran la función objetivo tal cual se define en el problema mientras que otros, tales como Guided Local Search (GLS) [36] alteran la función objetivo en cada paso de las iteraciones con el fin de escapar de óptimos locales modificando la forma en que se recorre el espacio de búsqueda.

Uno o varios vecindarios.

Una gran cantidad de metaheurísticas trabajan con una sola estructura de vecindario, mientras que otras tales como Variable Neighborhood Search (VNS) utilizan más de uno a fin de aplicar una mejor diversificación saltando entre los diferentes vecindarios. Esto es particularmente útil cuando el óptimo local en un vecindario no coincide con el óptimo local en otro.

Con o sin uso de memoria.

Esta clasificación es importante y divide los algoritmos entre los que ejecutan un proceso markoviano, es decir que para avanzar a un estado siguiente solo consideran el estado anterior y los que consideran otros movimientos realizados anteriormente. Existen algoritmos que consideran unos pocos movimientos y otros una cantidad considerable (short term vs. longterm memory). Dentro de este tipo de algoritmos con memoria se encuentra la Búsqueda Tabú. (TS).

Hibridación.

Una técnica utilizada durante los últimos años para mejorar la performace de las metaheurísticas es la hibridación [37]. Podríamos utilizar esta característica para realizar una clasificación de las metaheurísticas en híbridas o no híbridas pero de hecho no aparece explícitamente en la literatura.

Según Blum et al. [38] existen varias formas de hibridación de una metaheurística. Alguna de ellas puede ser la inclusión de programación lineal entera (como utilizaremos nosotros en esta tesis), programación con restricciones, solución del problema relajado mediante método exacto y luego aplicación de la metaheurística para resolver el problema original y utilización de programación dinámica dentro de las búsquedas locales para la solución exacta de subproblemas, entre otras.

4.1.3. Elección de la metaheurística

Definimos para la resolución del CmTNSSP la utilización de un procedimiento GRASP. Es un procedimiento versátil en el cual alternaremos la fase de construcción y la búsqueda local que resulte más adecuada.

La elección fue considerada debido a la potencia de la metodología para la resolución de problemas de optimización combinatoria y especialmente a los buenos resultados de este algoritmo en problemas de construcción de redes utilizando propiedades estructurales de los grafos (en nuestro caso de los grafos 2-nodo-conexos) [5] [6].

GRASP

De acuerdo a las clasificaciones hechas en los párrafos anteriores podemos situar al GRASP que utilizaremos como una metaheurística de trayectoria, no bio inspirada, sin uso de memoria e híbrida. Otras clasificaciones pueden depender de la búsqueda local seleccionada.

GRASP es un proceso iterativo donde cada iteración se compone de dos fases: la fase de construcción y la búsqueda local. La fase de construcción obtiene una solución factible del problema y la búsqueda local mejora esta solución. La mejor solución encontrada sobre todas las iteraciones GRASP realizadas se devuelve como el resultado.

Algoritmo 1 Secuencia de ejecución del algoritmo GRASP.

```
1: procedure grasp
2: input Instance, ListSize, MaxIter
3: for (k = 1 to MaxIter) do
4:    FeasibleSol ← Construct_Greedy_Rand_Solution(Instance;ListSize; Seed)
5:    Local_Search_Solution ← Local_Search(FeasibleSol)
6:    if cost(Local_Search_Solution) < cost(Best_Solution_Found) then
7:       Best_Solution_Found ← Local_Best_Solution
8:    end if
9: end for
10: return Best_Solution_Found
```

El algoritmo 1 detalla un pseudocódigo del GRASP que describe las fases del proceso. GRASP consta de tres parámetros principales: La cantidad total de iteraciones $MaxIter$, el tamaño de la Restricted Candidate List (RCL), $ListSize$ que veremos a continuación y la semilla inicial $Seed$ para la inicialización del generador de números pseudo-aleatorios.

Fase de Construcción.

La misma parte de una solución vacía y va agregando candidatos de acuerdo a un cierto criterio. En los algoritmos golosos (greedy) este criterio está basado en consideraciones de costo, es decir que la solución parcial se va construyendo agregando los candidatos que generen la mejor solución en cada iteración.

En el GRASP ocurre una variación respecto a un algoritmo puramente $greedy$, al introducir un elemento llamado Lista de Candidatos Restringidos (RCL por su nombre en inglés). En esta lista se incluyen de acuerdo a un criterio prestablecido, los candidatos que pueden formar parte de la solución en la iteración considerada, los cuales serán seleccionados aleatoriamente entre los elementos de esta lista. Con el fin de asegurar que la solución sea de buena calidad los elementos que formarań parte de la RCL deberán ser elegidos cuidadosamente.

Existen dos criterios básicos para restringir los candidatos:

Definamos c_i = c(e_i), el costo de agregar el elemento e_i a la solución en construcción.

Criterio basado en cardinalidad

Este criterio considera la construción de la RCL con los k elementos que generan soluciones de mejor costo incremental entre todos los posibles. Ordenamos en forma decreciente los c_i y tomamos los primeros k e_i correspondientes para agregar a la RCL.

Criterio basado en valores

Este criterio considera la selección de elementos que van a formar la RCL teniendo en cuenta a los mejores dentro de un rango dado.

Sea c^{min} el menor costo para el conjunto de elementos considerado y c^{max} el mayor, contruiremos la lista de candidatos restringidos de la siguiente forma:

$$RCL = \left\{e_i : c_i \leq c^{min} + \alpha(c^{max} - c^{min})\right\} \text{ con } \alpha \in [0, 1] \tag{4.1}$$

Cuando α toma el valor 0 estamos ante un procedimiento greedy dado que el elemento a incluir será el que genere la solución incremental de mejor costo.

En el algoritmo 2 se detalla el funcionamiento del procedimiento de construcción.

Algoritmo 2 Secuencia de ejecución de la fase de construcción

```
1: procedure Construct_Greedy_Rand_Solution
2: input Instance, ListSize
3: G_Sol ← φ
4: Evaluación de los costos incrementales c_i de todos los candidatos e_i
5: repeat
6:    Construir RCL de tamaño ListSize
7:    e_i = Random(RCL) // Selección de un elemento de la RCL al azar
8:    if G_Sol ∪ e_i ∈ F siendo F la región factible then
9:       G_Sol ← G_Sol ∪ e_i
10:   end if
11: until Solucion Completa
12: return G_Sol
```

Fase de Búsqueda Local.

Una vez construida en la primera fase una solución factible, dicha solución se mejora mediante una búsqueda local. El algoritmo GRASP puede ir acompañado de diferentes búsquedas locales. La especificación de esta fase la detallaremos en el Capítulo 5 donde trataremos extensamente las búsquedas locales elegidas y sus características. Como reseña podemos adelantar que utilizaremos como estrategia de búsqueda local un esquema de VND [37], donde algunas de ellas se resuelven mediante métodos exactos con la aplicación de programación lineal entera, utilizando un modelo matemático y su implementación a través de llamadas a un solver.
En la siguiente sección haremos una reseña de algunos algoritmos previos necesarios para el proceso de construcción.

4.2. Algoritmos previos utilizados

Trataremos en esta sección algunos algoritmos básicos utilizados para construir una solución factible de buena calidad. Dichos algoritmos nos permiten entre otras funcionalidades, identificar los nodos más "lejanos" entre sí y construir caminos nodo disjuntos de menor costo entre dichos nodos y el depot, siendo fundamental su inclusión en los procedimientos de construcción de soluciones factibles para el CmTNSSP.

4.2.1. Algoritmo de Bhandari

Para la construcción de k caminos nodo disjuntos entre un par de nodos (u, v) de un grafo, cuyo costo total sea mínimo (que utilizaremos como parte del algoritmo constructor de las componentes 2-nodo-conexas) nos basamos en un algoritmo diseñado por Bhandari [39]. Existen otros algoritmos para la construcción de dos caminos nodo disjuntos tales como los tratados por Suurballe y Tarjan [40] pero consideramos que el algoritmo de Bhandari es más adecuado para introducir el grado deseado de aleatorización a fin de cumplir con las especificaciones del GRASP en la construcción de soluciones factibles. La metodología del algoritmo de Bhandari se basa en la construcción de k caminos nodo disjuntos a partir de $k-1$ caminos nodo disjuntos. El algoritmo comienza con el paso base buscando un camino de costo mínimo entre dos vértices del grafo (ej. A y Z), aplicando una ligera variante del Algoritmo SPF (Shortest Path First) de Dijkstra [41].

Esta variante del algoritmo de Dijkstra permite aristas con costos negativos (no presentes en el paso base pero sí en los siguientes) sabiendo de antemano (por construcción del grafo) que no se van formar ciclos cuyo costo sea negativo, lo cual invalidaría el algoritmo. A consecuencia de permitir esto último, los nodos marcados no son permanentes y pueden volver a ser usados para determinar el camino de costo mínimo. Una vez determinado dicho camino de costo mínimo entre los puntos A y Z del grafo de la Figura 4.1 (A,B,C,D,Z) se procede de la siguiente manera: Se sustituyen las aristas del camino A-Z por arcos desde Z hacia A con el mismo costo de la arista original en valor absoluto, pero negativos. Esto hace que el grafo, que originalmente no era dirigido, comience a serlo.

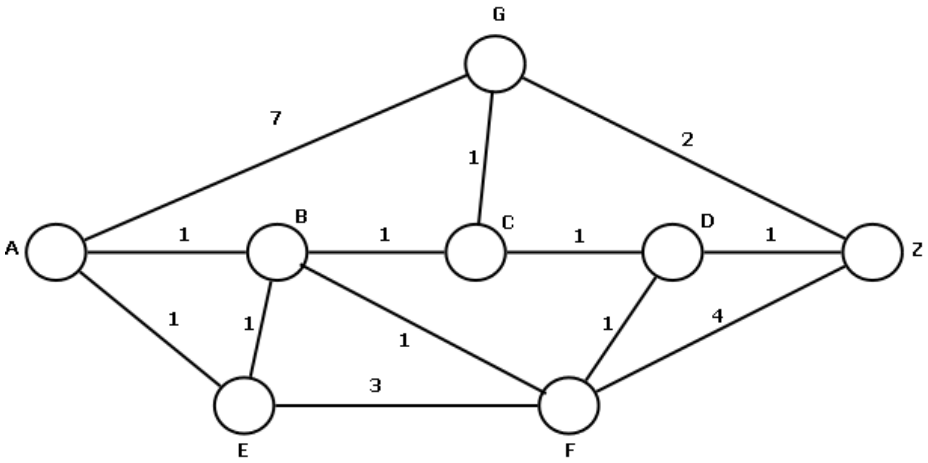

Figura 4.1: Grafo inicial

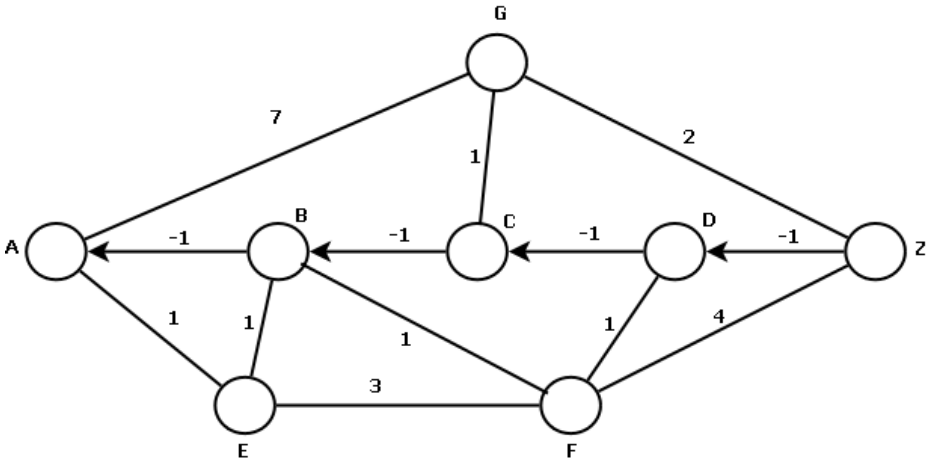

Figura 4.2: Grafo dirigido con costos negativos

A continuación al grafo de la Figura 4.2 que contiene el camino marcado desde Z hacia A y costos negativos, se le hace un $split$ de todos los nodos del camino A-Z (salvo el A y el Z) de la siguiente manera: Llamemos O_i al nodo original y S_i al nodo splitted de cada par de nodos original-splitted del camino. En nuestro ejemplo los nodos O_i son B, C y D y los nodos S_i son los splitted Bs, Cs, Ds. Se agrega un arco desde el nodo S_i al nodo O_i con costo 0. Las aristas incidentes al nodo O_i (que no pertenecen al camino) se sustituyen por arcos con destino O_i e igual costo y se agregan arcos con el mismo costo desde los nodos adyacentes a O_i (que no pertenecen al camino) hacia S_i.

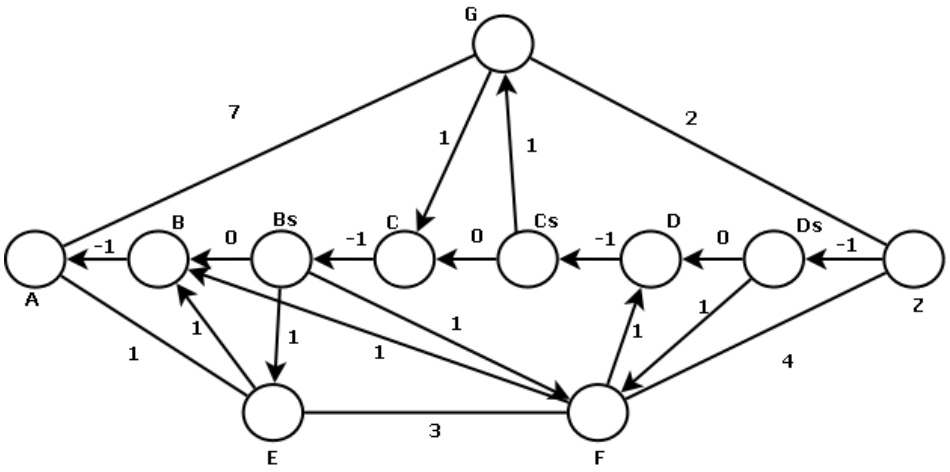

Figura 4.3: Grafo con nodos del camino A,B,C,D,Z y sus correspondientes en la transformación (split)

Esto hace que los nodos O_i tengan solamente arcos entrantes salvo el arco saliente que forma parte del camino desde Z hacia A, y los nodos S_i tengan solamente arcos salientes salvo el arco entrante que forma parte del camino Z-A. Esta construcción evita que se formen circuitos de costo negativo que harían inviable la aplicación del algoritmo de Dijkstra. Este procedimiento se hace para todos los pares de nodos (original-splitted) del camino salvo los propios A y Z.

Al grafo resultante de este procedimiento que podemos ver en la Figura 4.3 se le aplica el algoritmo de Dijkstra modificado explicado en el parráfo anterior, obteniéndose un nuevo camino de costo mínimo 7 (A, E, F, D, Cs, G, Z). Luego se deshace la operación de splitted anterior, colapsando los pares de nodos (O_i,S_i) y cambiando los arcos por aristas (eliminando aristas múltiples). Como resultado de este procedimiento se obtiene el camino sin los nodos splitted que en nuestro grafo original es el (A,E,F,D,C,G,Z) Las aristas comunes a este nuevo camino y al camino obtenido en el paso base (A,B,C,D,Z) (en nuestro ejemplo solo la arista (C,D)) se eliminan.

De esta manera nos quedan dos caminos nodos disjuntos (A,B,C,G,Z) y (A,E,F,D,Z) (Figura 4.4) cuyo costo total es mínimo. Este algoritmo nos evita suboptimalidad y falsas ausencias de caminos nodo disjuntos.

Para obtener tres caminos nodos disjuntos entre A y Z procedemos de manera análoga repitiendo el procedimiento anterior para los dos caminos obtenidos hasta el momento, es decir se dirigen los caminos con sentido Z - A cambiando los costos por los opuestos y realizando la transformación de split a los nodos de cada camino segun las reglas expuestas anteriormente. A este grafo así construido se le vuelve a correr el algoritmo de Dijkstra y se obtiene un nuevo camino. Luego se colapsan los nodos a los cuales se les transformó mediante el splitting, se eliminan aristas múltiples y se modifican los costos por los originales. Se eliminan las aristas

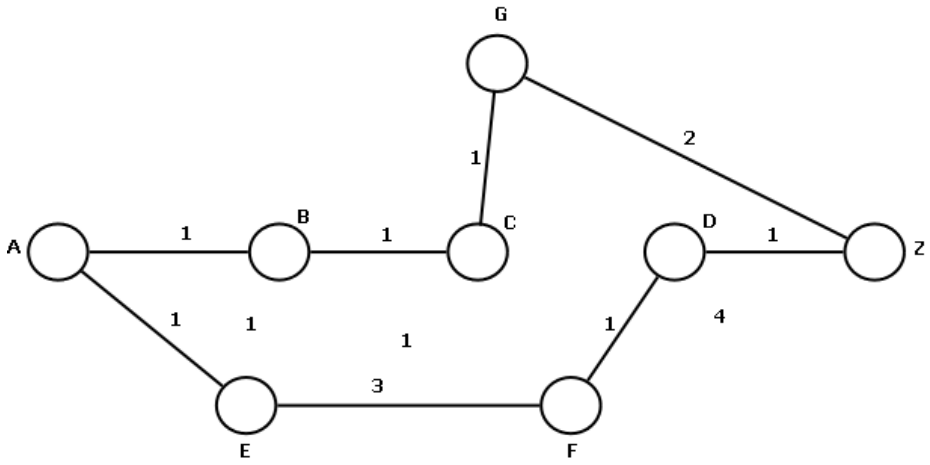

Figura 4.4: Caminos nodo disjuntos de costo mínimo

comunes a los 3 caminos (o a dos de ellos) y obtendremos tres caminos nodo disjuntos de costo mínimo. El algoritmo aplicado de esta manera nos permite hallar k caminos nodo disjuntos a partir de los $k-1$ caminos nodo disjuntos hallados en el paso anterior. El algoritmo de Dijkstra en la k-ésima iteración puede no obtener ningún camino entre A y Z en el grafo modificado, en cuyo caso no existiría k-nodo-conexión entre los vértices A y Z.

4.2.2. Pseudocódigo del algoritmo de Bhandari

El algoritmo 3 comienza obteniendo el camino más corto entre A y Z mediante el Algoritmo de Disjktra (Línea 3) e inicializa la colección de caminos $minkpaths$ con este único camino (Línea 4).

La función **transform_graph** (Línea 7) toma los caminos $minkpaths$ y transforma las aristas del grafo G correspondientes a estos caminos, en arcos con sentido de Z hacia A, obteniendo el grafo $\bar{G}$.

La función **transform_matrix** (Línea 8) transforma la matriz C de costos del grafo G, en $\bar{C}$ la cual tiene costos negativos en los arcos correpondientes a los caminos $minkpaths$.

La función **split_graph** (Línea 9) transforma el grafo $\bar{G}$, realizando el $split$ de los nodos que integran los caminos $minkpaths$ excepto el nodo A y el nodo Z, cambiando aristas por arcos según lo ejemplificado en la Figura 4.3. Dicha función devuelve la matriz $\hat{C}$ correspondientes a los costos del grafo $\hat{G}$.

A continuación se calcula el camino de menor costo entre los nodos A y Z para el grafo $\hat{G}$ transformado por la operación de $split$ (Línea 10).

Algoritmo 3 Construcción de k caminos de suma mínima según metodología de Bhandari.

```
1:  procedure Bhandari
2:  input G, C, nodes_to_use , k, A, Z
3:  min1path = Disjktra(G, C, A, Z )
4:  minkpaths ← min1path
5:  i = 1
6:  while (min1path ≠ φ and i ≤ k) do
7:     Ḡ = transform_graph(G, minkpaths)
8:     C̄ = transform_matrix(C, minkpaths)
9:     Ĝ = split_graph(Ḡ, C̄, A, Z, minkpaths, output:Ĉ)
10:    min1path = Disjktra(Ĝ, Ĉ, A, Z )
11:    if (min1path ≠ φ) then
12:       G′ = collapse_graph(Ḡ)
13:       minkpaths = get_i_paths(G′, minkpaths, min1path)
14:       i = i + 1
15:    end if
16: end while
17: return minkpaths
```

Si dicho camino de costo mínimo existe (Línea 12) (lo cual indica que hasta la iteración considerada i, se podrán obtener los i caminos nodo-disjuntos cuya suma de costos es mínima) se colapsa los nodos que se habían transformado haciendo el $split$ en el grafo $\hat{G}$ y se calculan los i caminos nodo-disjuntos de costo total mínimo, considerando los $i-1$ caminos anteriores y este último, removiendo las aristas comunes entre ellos.

El algoritmo finaliza cuando se obtuvieron los k caminos solicitados, o el algoritmo no puede obtener el camino i, y por consiguiente devuelve los $i-1$ caminos encontrados hasta el momento.

4.2.3. Algoritmo de Dijkstra modificado

A continuación describiremos el pseudocódigo del algoritmo de Disjkstra modificado que utilizamos para hallar el camino de menor costo entre el nodo fuente A y el nodo pozo Z de un grafo dirigido.

Definiremos previamente algunas convenciones y notación utilizadas en el pseudocódigo.

- $l(i,j)$ es la distancia del nodo i al nodo j.
- $d(i)$ es la distancia del vértice i al nodo fuente A, es decir $l(A,i)$.
- Γ_i el el conjunto de los sucesores del vértice i.

- $P(i)$ es el conjunto de predecesores del nodo i.
- $d(A)$ = 0 por definición de $d(i)$.

Algoritmo 4 Dijsktra modificado. El rescanning puede volver a etiquetar nodos debido a la presencia de arcos negativos.

1: $d(A)$=0
2: **if** i $\in \Gamma_A$ **then**
3: $d(i)$=$l(A, i)$
4: **else**
5: $d(i)$=∞
6: **end if**
7: $P(i)$=A $\forall i \in \Gamma_A$
8: $\bar{S} = \Gamma_A$

9: **Buscar** $j \in \bar{S}$ **tal que** $d(j) = min\, d(i)$ **con** $i \in \bar{S}$
10: $\bar{S} = \bar{S} - \{j\}$
11: **if** j=Z **then**
12: **FIN**
13: **end if**

14: **for all** $i \in \Gamma_j$ **do**
15: **if** $(d(j)$+$l(j, i) < d(i))$ **then**
16: $d(i)$=$d(j)$+$l(j, i)$
17: $P(i)$=j
18: $\bar{S} = \bar{S} \cup \{j\}$
19: **end if**
20: **end for**
21: **goto 9**

Luego de inicializar el Algoritmo 4 en los pasos 1 al 8, éste alterna entre los pasos 9 y 14. En cada iteración el vértice con menor distancia es seleccionado del conjunto $\bar{S}$. El algoritmo busca esta característica realizando un movimiento a la vez y finaliza cuando el vértice seleccionado del conjunto $\bar{S}$ es el vértice pozo Z. En el algoritmo original de Dijkstra cuando un vértice con el camino más corto desde el nodo fuente es seleccionado de la lista de vértices, se dice que el vértice seleccionado es marcado "permanentemente" y ningún rescanning desde ningún otro vértice del grafo puede modificar el etiquetado de este vértice. En el algoritmo modificado que aplicamos, dada la presencia de arcos negativos (ver figura 4.3) el rescanning puede volver a etiquetar un vértice ya etiquetado anteriormente de esta forma.

4.2.4. Selección del primer nodo de cada componente.

El paso inicial de construcción de soluciones factibles para el CmTNSSP, requerirá determinar un nodo inicial en cada una de las m componentes H_k a construir. Para determinar como elegiremos estos m nodos, procedemos según el Algoritmo 5.

Algoritmo 5 Selección de m nodos iniciales

```
1:  procedure Far
2:  input G, C, T, m, n
3:  bestfar ← φ
4:  maxdistance = 0
5:  for i=1 to n do
6:      far ← φ
7:      for i=1 to m do
8:          far[i] ← ExtractRandomNode(T)
9:      end for
10:     distance = 0
11:     for i=1 to m-1 do
12:         for j=i+1 to m do
13:             distance = distance + C_{far[i],far[j]}
14:         end for
15:         if distance > maxdistance then
16:             bestfar ← far
17:             maxdistance = distance
18:         end if
19:     end for
20: end for
21: return bestfar
```

El Algoritmo 5 toma aleatoriamente m nodos terminales y calcula la suma de los costos entre cada par de nodos utilizando la matriz C.

El bloque correspondiente al alcance del **for** (líneas 5 a 18) se ejecuta n veces, y devuelve en una lista ($bestfar$) los nodos seleccionados cuya suma de distancias entre sí haya sido la mayor de todas las obtenidas.

Los nodos terminales seleccionados de esta forma están alejados entre sí y cada uno de ellos formará parte de una componente H_k diferente.

4.3. Construcción de soluciones factibles para el CmTNSSP

Siguiendo la metodología detallada en la subsección 4.1.3 describiremos en esta sección la construcción de soluciones factibles de buena calidad y que cumplan con los requerimientos del GRASP.

Dado un grafo inicial $G = (V, E)$, (vamos a suponer que es un grafo completo para simplificar ideas), definimos T como el conjunto de nodos terminales del grafo G, $T \subseteq V$. Sea $\{0\}$ el nodo distinguido definido como depot.

Definamos $C = \{c_{ij}\}_{i,j \in V}$ como la matriz de costos de ruteo del grafo, es decir los costos de la arista (i, j) cuando dicha arista pertenece a la estructura 2-nodo-conexa de alguna de las componentes.

Sea $m \in \mathbb{Z}^+$ la cantidad de componentes de la red a construir tal que $m \geq 2$.

Describiremos a continuación las etapas del algoritmo de construcción del GRASP.

- **Paso 1**. Procedemos mediante el algoritmo definido en la subsección 4.2.4 a localizar los m primeros nodos terminales a incluir (uno en en cada componente). Dicho algoritmo considera la suma de distancias máxima entre los nodos, luego de una selección aleatoria de dichos m nodos un número establecido de veces que fijamos mediante un parámetro.

- **Paso 2**. Para cada uno de los nodos sorteados en el punto anterior, consideramos los k caminos nodo-disjuntos entre el nodo considerado y el depot, cuya suma total de costos es mínima, utilizando como costos para cada arista (i, j) los especificados en la matriz C definida anteriormente.

- **Paso 3**. Procederemos a continuación a definir la RCL (Restricted Candidate List). El criterio para construir la RCL está basado en cardinalidad. Este entero k es el tamaño de la RCL ($ListSize$ en el algoritmo GRASP). Para obtener los k caminos nodo disjuntos que cumplan con esta condición (suma mínima) utilizamos el $Algoritmo\ de\ Bhandari$ explicado en la subsección 4.2.1. El número de caminos k es un parámetro del constructor. ($k \geq 2$). Consideremos esta lista de k caminos como la RCL del GRASP. De esta lista de k caminos elegimos dos caminos cualesquiera mediante sorteo utilizando una distribución uniforme y los incluimos en la solución. Este proceso lo repetimos m veces, una para cada uno de los nodos más distantes hallados en el punto anterior, los cuales formarán parte de cada una de las m componentes del grafo a construir.

  Pongamos un ejemplo gráfico. Sea V el conjunto de vértices del grafo G mostrado en la Figura 4.5, el conjunto T de nodos terminales (representados en negro) el conjunto de los nodos de Steiner $V \setminus T$ (representados en blanco) y el nodo distinguido depot marcado con 0.

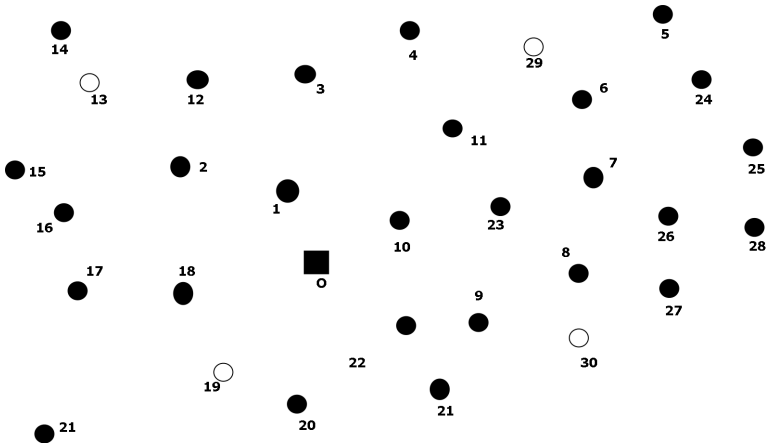

Figura 4.5: Conjunto V de vértices del grafo G.

Supongamos que el número de componentes de la solución factible a construir es $m = 2$, siendo $P = \{5, 21\}$ el conjunto de los nodos más distantes entre sí. Fijemos para el ejemplo $k = 4$ como tamaño de la RCL.

Aplicando el algoritmo de Bhandari (Algoritmo 3) construimos los 4 caminos nodo disjuntos de suma mínima (Figura 4.6) entre cada uno de los nodos de P y el depot. De los cuatro caminos nodo-disjuntos computados por el algoritmo de Bhandari entre los nodos del conjunto P y el depot sorteamos dos, e incluimos estos como parte de la solución en construcción.

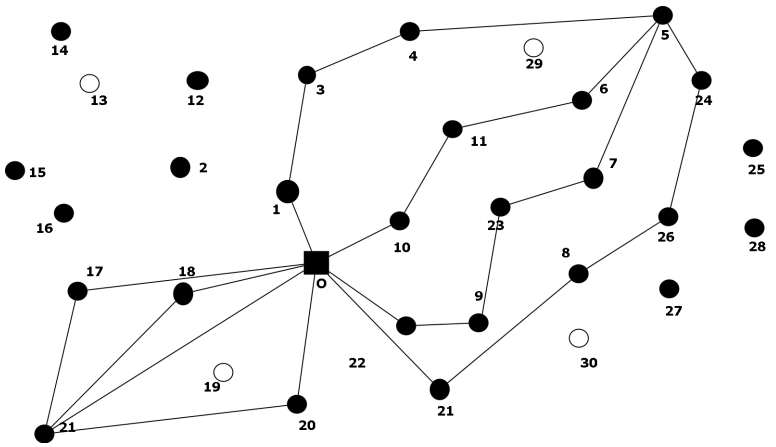

Figura 4.6: Representación de 4 caminos nodo disjuntos de suma mínima entre depot y nodos de P

- **Paso 4**. Una vez finalizada la primera etapa de construcción la solución consta de m

ciclos que comparten el nodo 0 (depot) y cada uno de ellos tiene uno de los vértices más distantes pertenecientes al conjunto P. (Figura 4.7).

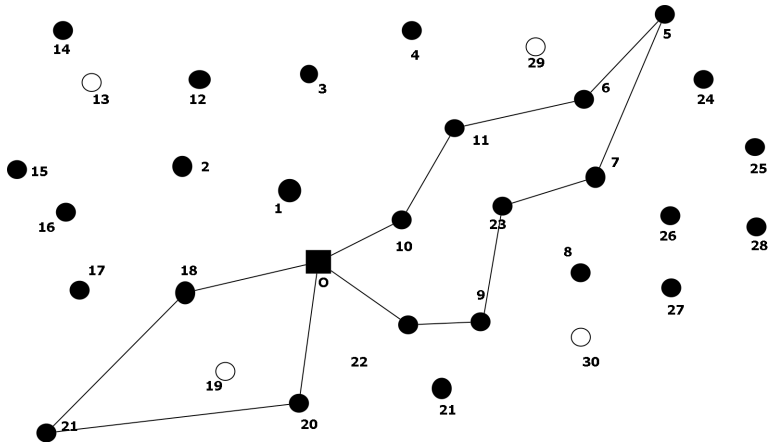

Figura 4.7: Estado del grafo luego de la primera etapa de construcción de una solución factible.

DEFINICIÓN **4.1** *Dado un grafo H decimos que el camino P es un H-path de H si P es no trivial y "toca" a H exactamente en sus extremos. Cuando el H-path es de una sola arista, dicha arista nunca es arista de H.*

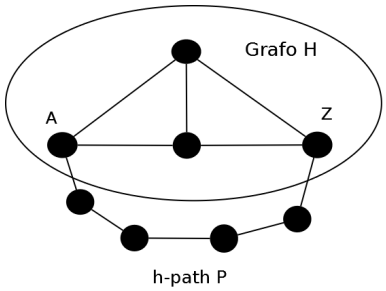

Figura 4.8: Grafo H y H-path

En la Figura 4.8 se observa un grafo H y el H-path P que comparte los extremos A y Z con el grafo H.

Una vez que tenemos el nodo y la componente en la cual lo vamos a conectar, realizamos la siguiente operación.

A continuación vamos a ir agregando los nodos terminales que no forman parte todavía de la solución en construcción. Dichos terminales se incorporan a cada una de las componentes de la siguiente forma:

Sorteamos un nodo terminal cualquiera que aun no forma parte de la solución en construcción y vamos a conectarlo formando un H-path con alguna de las componentes. Esta operación mantiene la 2-nodo-conexidad dado que agrega un camino independiente (el H-path) entre dos nodos de la componente. Para realizar esta operación procedemos de la siguente manera:

Elegimos la componente donde conectar el nodo, para esto utilizamos el criterio de menor cantidad de nodos presentes en dicha componente. Este criterio resulta particularmente útil para equilibrar la cantidad de nodos en cada una de las m componentes sin perder factibilidad por violación del parámetro Q, aunque con un trade-off de conectar el nodo a una componente "inadecuada" en lo que a costos se refiere.

- **Paso 5**. Quitamos del grafo original las aristas incidentes a nodos que forman parte de componentes diferentes a la que estamos tratando.

  Agregamos al grafo un nodo virtual v' conectado a todos los nodos de la componente seleccionada y solamente a ellos mediante aristas de costo 0, y asignamos de igual forma el valor 0 a las aristas presentes hasta el momento en la componente a tratar.

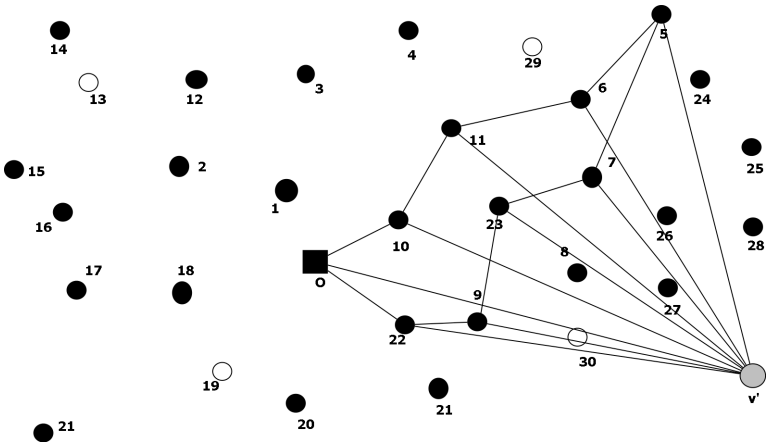

Figura 4.9: Grafo con nodo virtual agregado v'

Para esto definimos una nueva matriz $C'_{(|V|+1)\times(|V|+1)}$ con una fila y una columna más de ceros que la matriz C y valor 0 en las aristas (i, j) que al momento forman parte de la solución en construcción.

- **Paso 6.** Una vez que aplicamos la transformación anterior al grafo procedemos a obtener los 4 caminos nodo disjuntos de suma mínima (Figura 4.10) (nuevamante utilizando el

Algoritmo 3) entre el nodo terminal a incluir perteneciente al conjunto P (en nuestro ejemplo el nodo 12) y el nodo virtual v'. De estos 4 caminos sorteamos dos cualesquiera y los incorporamos a la solución en construcción.

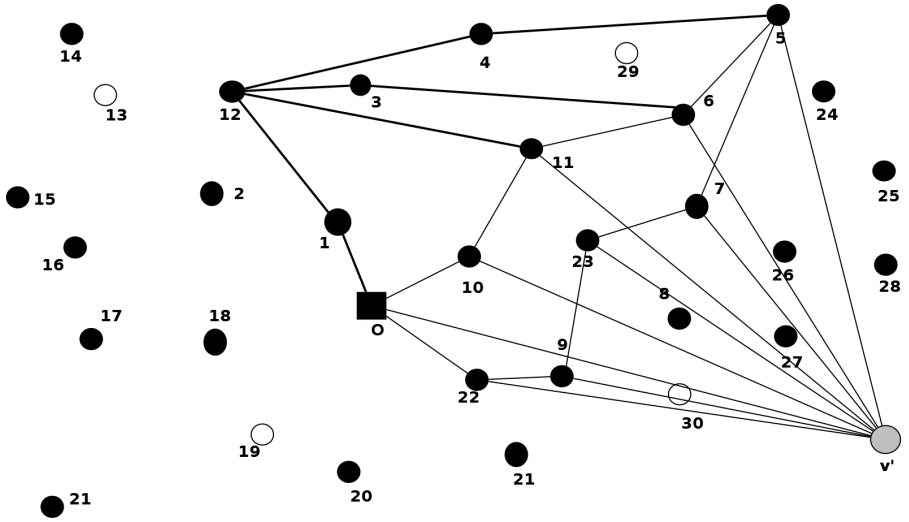

Figura 4.10: Grafo con 4 caminos nodo disjuntos para seleccionar

Como los pesos de las aristas que ya pertenecen a la solución y las aristas virtuales incidentes a v' son de costo cero, los únicos costos que tendrán incidencia serán los costos de las nuevas aristas que se incoporen.

Esto nos asegura un H-path que incluye al nodo terminal que agregamos (ver Figura 4.11) y eventualmente otros nodos que formen parte de éste. Los extremos del H-path generado son nodos que ya formaban parte de la solución en construcción y dicho H-path incluye al menos el nuevo nodo agregado.

- **Paso 7.** A continuación deshacemos la transformación, quitando el nodo virtual y sus aristas incidentes de costo 0. La componente que antes de la incorporación del H-path era 2-nodo-conexa, sigue siendo 2-nodo-conexa.

- **Paso 8.** Este procedimiento ejemplificado en las Figuras 4.9 a 4.11, donde se agrega el terminal etiquetado con 12 se repite hasta lograr la incorporación de todos los nodos terminales a las componentes 2-conexas, finalizando de esta manera la construcción de una solución factible.

No haremos en esta fase de construcción ninguna consideración acerca de los nodos

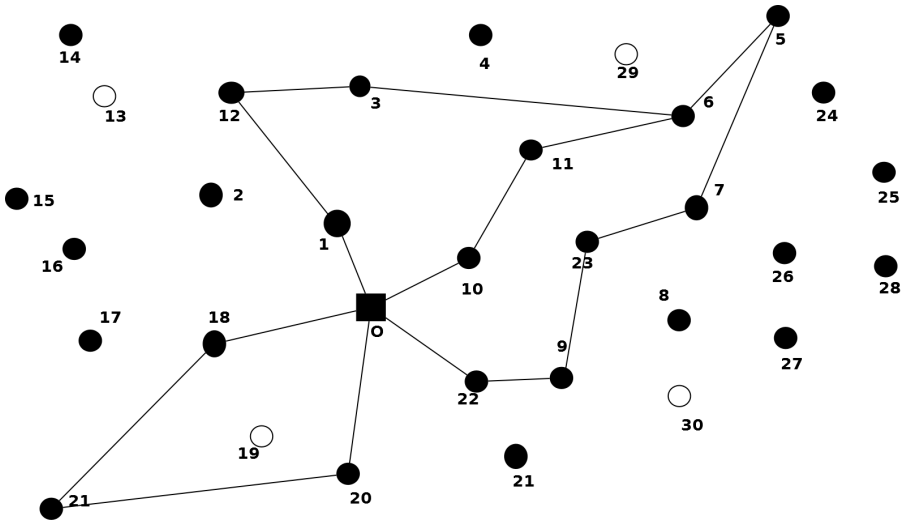

Figura 4.11: Estado final del paso de incorporación de un nodo terminal

colgantes. Los algoritmos descriptos en esta sección no generan nodo colgante alguno. Dichos nodos comenzarán a aparecer en las búsquedas locales a medida que comiencen a mejorarse las soluciones.

El procedimiento **Construct_Greedy_Randomize_Feasible_Solution** (Algoritmo 6) es el pseudocódigo detallado de la Fase de Construcción GRASP para el CmTNSSP descripta anteriormente y tiene como entrada (ver línea 2) el grafo original, que llamamos $G = (V, E)$; la matriz de costos de ruteo $C = \{c_{ij}\}_{i,j \in V}$; el tamaño de la RCL, $ListSize$; la cantidad de componentes que vamos a construir, m; el parámetro Q que corresponde a la cantidad de nodos terminales que puede tener como máximo cada una de las m componentes y $T \subseteq V$ que es el conjunto de los nodos terminales del grafo G.

De acuerdo a la metodología GRASP comenzamos con una solución vacía G_{Sol} (línea 3) que vamos a ir completando hasta obtener la solución factible deseada.

El procedimiento **Far** (línea 6) nos retorna una lista ($FarNodes$) con los m nodos terminales más lejanos entre sí luego de un cierto número de búsquedas (Ver Algoritmo 5). Estos m nodos los vamos a incorporar a las componentes 2-nodo-conexas en el primer paso de la construcción. En la línea 8 extraemos un nodo al azar perteneciente a la lista $FarNodes$ y vamos a formar un ciclo que contenga al propio nodo $node$ y al $depot$.

Para esto vamos a utilizar el procedimiento $Bhandari$, (Ver Algoritmo 3) el cual nos va a devolver una lista de caminos nodo-disjuntos entre el $depot$ y el nodo $node$ de tamaño $ListSize$ (línea 9) Esta lista contiene los $ListSize$ caminos nodo-disjuntos de suma mínima. De esta lista vamos a extraer dos caminos cualesquiera (líneas 10 y 11) y los vamos a agregar a la solución (líneas 12 y 13).

De igual forma vamos a agregar los nodos que forman estos dos caminos a una lista indexada de nodos ($component\_nodes[m]$) (líneas 14 y 15) en la cual guardaremos los nodos terminales de cada componente 2-nodo-conexa. Esta lista la usaremos más tarde para individualizar la componente a la cual pertenece cada uno de los nodos terminales.

También es necesario guardar en la lista $not\_assigned$ (líneas16 y 17) los nodos terminales que restan agregar a la solución.

Usamos para describir estas operaciones las siguientes funciones auxiliares:

- **add_path** (G**,** $path$) la cual agrega el camino $path$ al grafo G.
- **add_nodes** (a**,** b) que concatena los elementos de las listas a y b.
- **subtract_nodes** (a**,** b) que quita los nodos de la lista b que se encuentran en la lista a.

El procedimiento detallado se repite dentro del ciclo del **for**(líneas 7 a 18) para los m nodos distantes. Como resultado obtenemos m ciclos que comparten el $depot$. Cada ciclo forma parte de una componente H_i con $i \in 1...m$.

A continuación vamos a ir construyendo la solución G_{Sol} agregando H-paths desde nodos terminales aun no incorporados, a cada una de las componentes hasta agotar los nodos terminales (líneas 19 a la 32).

Para esto realizaremos las siguientes operaciones según lo describimos gráficamente en las Figuras 4.9 a 4.11:

- Seleccionaremos la componente donde vamos a agregar el nodo (línea 21). Esta selección la realizamos utilizando un criterio de optimización de la cantidad de nodos que cada componente va acumulando hasta el momento.
- Generamos un grafo de entrada $\bar{G} = (\bar{V}, \bar{E})$ mediante la función **transform** (línea 22) que tendrá los siguientes elementos:

  1. $\bar{V} = V \cup \{v'\}$, siendo v' un vértice auxiliar adicional agregado.
  2. Dado $H_i = (V_i, E_i)$ la componente a la cual vamos a agregarle un nodo terminal. El conjunto $\bar{E}$ contiene al conjunto E_i más todas las aristas incidentes a los nodos de la componente H_i a tratar que están presentes en $G = (V, E)$, y cuyo otro extremo no sean vértices de otra componente en construcción $H_j = (V_j, E_j)$ con $j \neq i$, más todas las aristas con un extremo en v' y el otro en los vértices de la componente a tratar H_i.

  Con respecto a los costos de las aristas de este nuevo grafo $\bar{G}$, la matriz de costos de ruteo $\bar{C} = \{c_{ij}\}_{i,j \in V \cup \{v'\}}$ se construye de la siguiente manera:

  1. La matriz de costos $\bar{C}$ tiene una fila y una columna más que C, matriz de costos de ruteo del grafo. G

2. Los costos de las aristas incidentes al nodo agregado v' son 0.
3. Los costos de las aristas presentes en G_{Sol} que pertenecen a la componente H_i a tratar son 0.
4. Los demás costos de $\bar{C}$ coinciden con los de C.

- Una vez construido $\bar{G}$ hallamos en este los caminos nodo-disjuntos ($minpaths$) de suma mínima entre el nodo (línea 23) seleccionado $node$ para incluir en la componente $comp$ (línea 21) y el nodo v'. Estos caminos constituyen la RCL de tamaño $ListSize$ (línea 23) sobre la cual elegiremos dos caminos al azar. Considerando el modo de construcción de estos caminos lo que estamos haciendo en realidad es agregar un H-path a cada componente H_k a tratar.

- Elegimos entonces dos caminos cualesquiera (líneas 24 y 25) y los incluimos en G_{Sol} (líneas 26 y 27). Luego incluimos los nodos de estos dos caminos $path\_1$ y $path\_2$ en la lista de nodos utilizados por componente $components\_nodes[comp]$ (líneas 28 y 29). A continuación quitamos los nodos utilizados en estos caminos, de la lista $not\_assigned$ (líneas 30 y 31) ya que dichos nodos quedaron incluidos en G_{Sol}, la solución en construcción.

- Repetimos el proceso del alcance del **repeat** (líneas 19 a 32) hasta que la lista $not\_assigned$ esté vacía, con lo cual hemos asignado todos los nodos terminales a las m componentes 2-nodo-conexas H_k con $k = 1...m$.

Para simplificar el pseudocódigo del algoritmo no estamos haciendo consideraciones de factibilidad de la solución, tales como superar la cantidad de nodos por cada componente, Q; o la imposibilidad de encontrar caminos nodo disjuntos en cada uno de los pasos (principalmente si estamos ejecutando el algoritmo con una instancia cuyo grafo original no es completo) lo cual obviamente es convenientemente tratado en la implementación.

Algoritmo 6 Construcción de una solución factible

procedure Construct_Greedy_Randomize_Feasible_Solution
input G, C, $ListSize$, m, n, Q, T
$G_{Sol} \leftarrow \phi$
$component\_nodes \leftarrow \phi$
$not\_assigned \leftarrow T$
$FarNodes \leftarrow$ **Far**(G, C, T, m, n)

for i=1 **to** m **do**
  $node$ = **ExtractRandomNode**($FarNodes$)
  $minpaths$ = **Bhandari**(G, C, $not\_assigned$, $ListSize$, $depot$, $node$)
  $path\_1 \leftarrow$ **ExtractRandomPath**($minpaths$)
  $path\_2 \leftarrow$ **ExtractRandomPath**($minpaths$)
  $G_{Sol} \leftarrow$ **add_path**(G_{Sol}, $path\_1$)
  $G_{Sol} \leftarrow$ **add_path**(G_{Sol}, $path\_2$)
  $component\_nodes[i] \leftarrow$ **add_nodes**($component\_nodes[i]$, $path\_1$)
  $component\_nodes[i] \leftarrow$ **add_nodes**($component\_nodes[i]$, $path\_2$)
  $not\_assigned \leftarrow$ **subtract_nodes**($not\_assigned$, $path\_1$)
  $not\_assigned \leftarrow$ **subtract_nodes**($not\_assigned$, $path\_2$)
end for

repeat
  $node$ = **ExtractRandomNode**($not\_assigned$)
  $comp$ = **CompSelect**(G_{Sol})
  $\bar{G}$ = **transform** (G, C, $\bar{C}$, G_{Sol}, $comp$, $components\_nodes$)
  $minpaths$ = **Bhandari**($\bar{G}$, $\bar{C}$, $not\_assigned$, $ListSize$, $node$, $virtual$)
  $path\_1 \leftarrow$ **ExtractRandomPath**($minpaths$))
  $path\_2 \leftarrow$ **ExtractRandomPath**($minpaths$))
  $G_{Sol} \leftarrow$ **add_path**(G_{Sol}, $path\_1$)
  $G_{Sol} \leftarrow$ **add_path**(G_{Sol}, $path\_2$)
  $component\_nodes[comp] \leftarrow$ **add_nodes**($component\_nodes[comp]$, $path\_1$)
  $component\_nodes[comp] \leftarrow$ **add_nodes**($component\_nodes[comp]$, $path\_2$)
  $not\_assigned \leftarrow$ **subtract_nodes**($not\_assigned$, $path\_1$)
  $not\_assigned \leftarrow$ **subtract_nodes**($not\_assigned$, $path\_2$)
until $not\_assigned = \phi$

return G_{Sol}

Capítulo 5

GRASP para el CmTNSSP: Búsquedas Locales

5.1. Introducción

Una vez que construimos una solución factible del problema CmTNSSP, dicha solución se deberá mejorar para aproximarnos a la solución óptima global. Utilizaremos para la resolución de este problema una combinación de búsquedas locales clásicas y búsquedas locales basadas en modelos exactos de programación lineal entera.

5.1.1. Definiciones

Existen varias estrategias para combinar el proceso de construcción de soluciones factibles y la secuencia de búsquedas locales a fin de recorrer de una manera eficiente el espacio de soluciones factibles.

5.1.1.1. Vecindario

Dado un problema de optimización con instancias (F, c), siendo F la región factible y c el óptimo global para cada una de éstas, definimos $vecindario$ como una función

$$N : F \rightarrow 2^F \tag{5.1}$$

definida para cada instancia[42].

Intuitivamente, dado un punto $f \in F$ definimos un vecindario $N(f)$ como el conjunto de puntos que de alguna manera son "$cercanos$" al punto f.
El concepto de vecindario es fundamental en la definición de las búsquedas locales, dado que el espacio de soluciones es explorado precisamente de acuerdo a estos criterios de vecindad.

Pongamos un ejemplo:

Dado un grafo $g \in F$ como el de la Figura 5.1, definimos un conjunto de grafos

$$N(g) = \left\{ h \in F : h \begin{array}{l} \text{se construye a partir de } g \text{ tomando dos nodos cualesquiera} \\ \text{y realizando un intercambio entre ellos} \end{array} \right\} \quad (5.2)$$

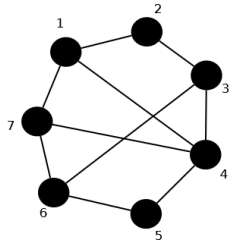

Figura 5.1: Grafo ejemplo g

Cualquier grafo construido de esta manera pertenece al mismo vecindario del grafo g que definimos en 5.2. Tomemos por ejemplo un grafo h obtenido de realizar el swapping de los nodos 2 y 5 del grafo g de la Figura 5.1 . Las aristas incidentes al nodo 2 serán luego de la transformación incidentes al nodo 5 y análogamente las incidentes al nodo 5 serán incidentes al nodo 2 (Ver Figura 5.2). Decimos entonces que los grafos g y h son "$vecinos$" en el vecindario N.

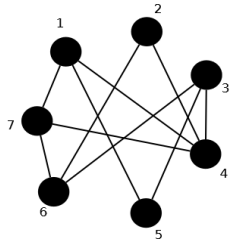

Figura 5.2: Grafo h perteneciente a $N(g)$.

En el desarrollo de las búsquedas locales que aplicamos para la resolución del CmTNSSP utilizaremos la definición de vecindarios para realizar los movimientos que nos permitan optimizar las soluciones. A partir de estas definiciones se diseñan procedimientos que combinados van a generar una mejora de dichas soluciones.

5.1.1.2. Función de mejora

Dada una instancia de un problema de optimización (F, c), siendo F la región factible y c una función de costo.

$$\text{Dado } N : F \rightarrow 2^F \tag{5.3}$$

un vecindario definido para dicha instancia y $t \in F$, definiremos una función:

$$improve(t) = \begin{cases} \text{cualquier } s \in N(t) \text{ con } c(s) < c(t) & \text{si } s \text{ existe} \\ no & \text{o.c.} \end{cases} \tag{5.4}$$

La función definida de esta manera permite mejorar soluciones factibles existentes y es el punto de partida para la construcción de un algoritmo básico de búsqueda local descripto como Algoritmo 7 [42].

Algoritmo 7 Procedimiento general de una Búsqueda Local

```
1: procedure local_search (t) // t = alguna solución factible en F
2: while improve(t) ≠ no do
3:     t = improve(t)
4: end while
5: return t
```

5.1.2. Aplicación de búsquedas locales

Existen diferentes estrategias para combinar un proceso de construcción de una solución factible y un conjunto de búsquedas locales. En nuestro trabajo, para la aplicación de las búsquedas locales utilizaremos una variante de VNS (Variable Neighborhood Search) llamada VND (Variable Neighborhood Descendant)[25], cuyo algoritmo genérico está titulado como Algoritmo 8 y explicaremos a continuación detallando previamente algunas definiciones.

5.1.2.1. Búsqueda local variable VNS.

La Búsqueda Local Variable (VNS, Variable Neighborhood Seach) ha sido propuesta por P. Hansen y N. Mladenovic [35]. La idea básica de VNS es explorar sucesivamente un conjunto de vecindarios predefinidos a fin de proporcionar una mejor solución. Este conjunto de vecindarios es explorado ya sea al azar o de forma sistemática para conseguir diferentes óptimos locales y también para escapar de ellos. VNS explota el hecho de que el uso de diversos vecindarios de la búsqueda local puede generar diferentes óptimos locales y que el óptimo global es un óptimo local para un determinado vecindario [25].

5.1.2.2. Búsqueda local descendente VND.

El algoritmo de búsqueda local descendente (Algoritmo 8) se basa en el descenso por vecindarios variables. VND utiliza sucesivos vecindarios en el descenso a un óptimo local.

Definamos en primer lugar un conjunto de estructuras de vecindario N_i (= 1, ..., Imax). Sea N_1 el primer vecindario a usar y x la solución inicial. Si no es posible una mejora de la solución de x en la vecindad corriente $N_i(x)$, la estructura de vecindad se cambia de N_i a N_{i+1}. Si se encuentra una mejora de la solución actual x, se vuelve a utilizar la primera estructura de vecindad $N_1(x)$ para reiniciar la búsqueda. Esta estrategia será eficaz si los diferentes vecindarios utilizados son complementarios en el sentido de que un óptimo local para una vecindad de N_i no será un óptimo local en la vecindad N_j.

Algoritmo 8 Secuencia de ejecución de búsquedas locales, según un esquema de VND.

```
1: input Un conjunto de estructuras de Vecindarios N_i,  con i = 1 ··· max
2: x = x_0  Solucion factible inicial
3: i = 1
4: while (i ≤ max) do
5:    Encontrar x' mejor vecino de x en el vecindario i,  x' = N_i(x)
6:    if f(x') < f(x) then
7:       x = x'
8:       i = 1
9:    else
10:      i = i + 1
11:   end if
12: end while
13: return x
```

Hemos diseñado para el problema CmTNSSP cinco vecindades correspondientes a las cinco búsquedas locales que desarrollaremos a continuación. Dichas búsquedas locales serán denominadas como Swapping de nodos **(Swapping)**, Extracción e inserción de nodos **(Extract Insert)**, Crossing de componentes **(Crossing)**, Mejor camino con nodos colgantes **(Best PWR)** y Mejor componente 2-nodo-conexa **(Best 2NC)**.

El diagrama de flujo que representamos en la Figura 5.3 corresponde al algoritmo completo del método aproximado para resolver el problema CmTNSSP, con la fase de construcción y el orden de cada una de la búsquedas locales que serán aplicadas. Conjuga el Algoritmo 1 como metodología de la metaheurística GRASP y el Algoritmo 8 como metodología de una búsqueda local VND, agregando además un algoritmo de perturbación **(Shaking)** como mecanismo adicional para escapar de óptimos locales.

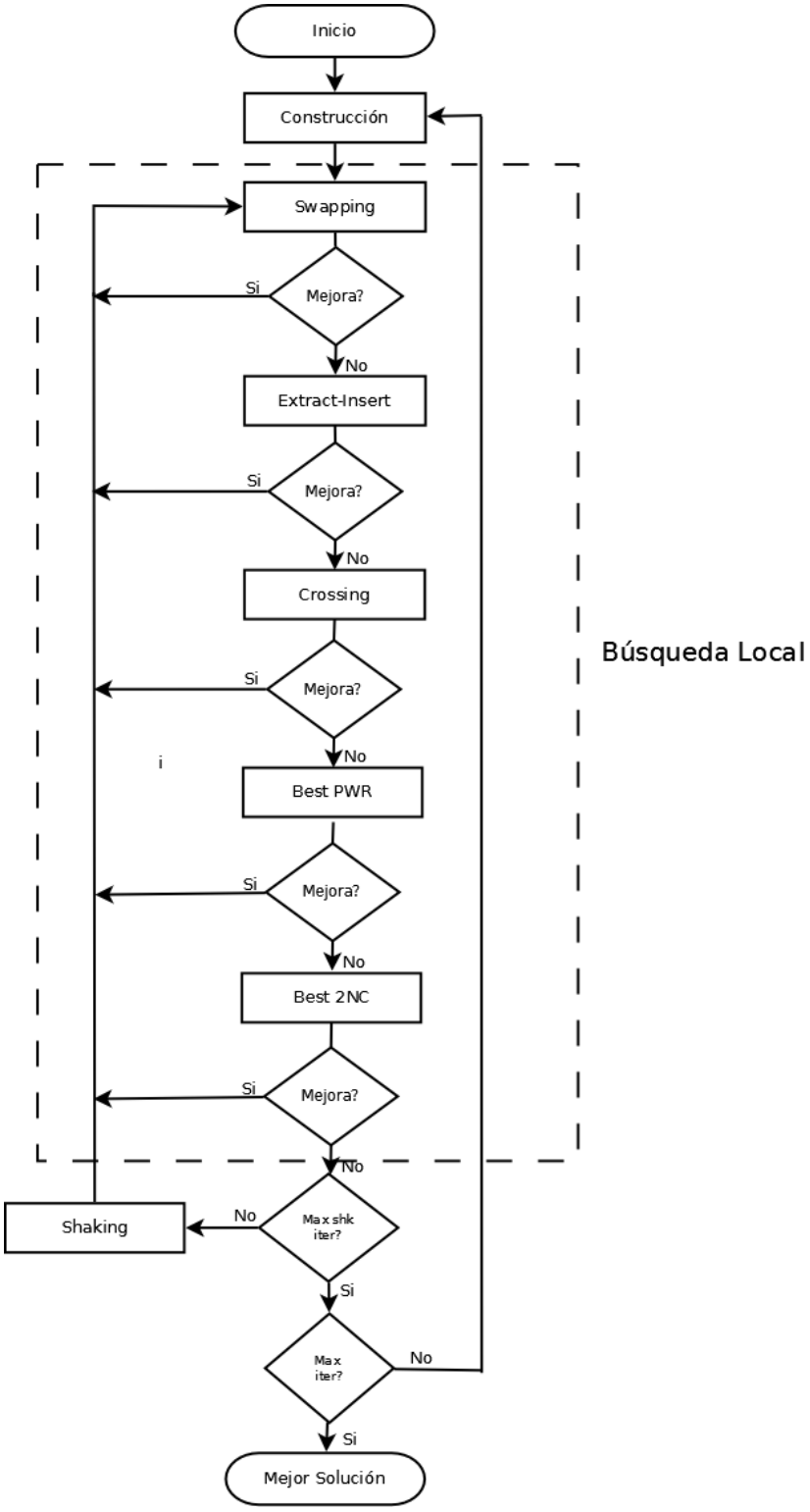

Figura 5.3: Diagrama de flujo de la metaheurística GRASP-VND para el CmTNSSP.

5.2. Extracción e Inserción de Nodos

5.2.1. Descripción

Esta búsqueda local consiste en realizar la extracción de todos los nodos terminales en un orden aleatorio desde su posición actual en la solución, y reubicarlos en otra posición (ya sea en la misma componente o en otra) que mejore el costo total sin perder la factibilidad.

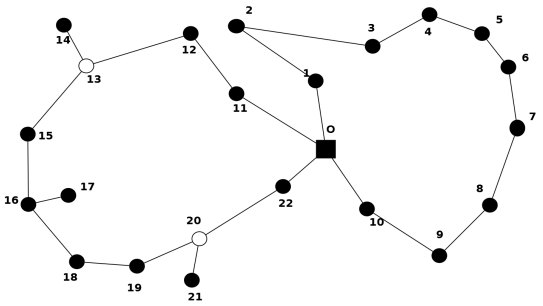

Figura 5.4: Estado de la solución previo a la extracción del nodo $i = 2$

El procedimiento de extracción es sencillo, consiste en extraer un nodo terminal cualquiera que llamaremos i y conectar entre sí los nodos adyacentes al nodo extraído. En la Figura 5.4 se observa una solución antes de extraer el nodo i (utilizaremos para el ejemplo el nodo $i = 2$) y en la Figura 5.5 el estado luego de la extracción de dicho nodo.

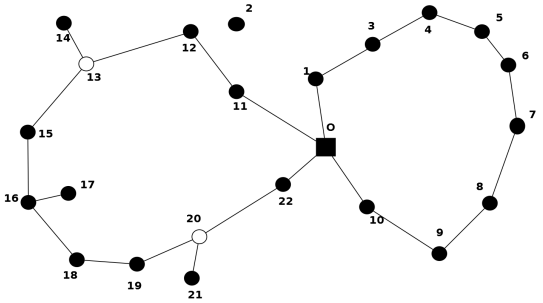

Figura 5.5: Estado post extracción del nodo $i = 2$

Para realizar la inserción del nodo extraído consideremos lo siguiente:
Sea $i \in T$ un nodo extraido con T el conjunto de nodos terminales del grafo y un vecindario N definido de la siguiente forma:

$$N(i) = \left\{ j \in T : j \quad \begin{array}{l} \text{son los } l \text{ nodos más cercanos al nodo } i \text{ considerando las distancias} \\ \text{de ruteo } c_{ij} \text{ definidas en el grafo original } G \end{array} \right\} \tag{5.5}$$

Consideremos j (en nuestro ejemplo $j = 12$) como el nodo más cercano al nodo $i = 2$. Si el nodo j fuera un nodo colgante no realizamos ningún movimiento y seguimos con el siguiente nodo, es decir tomamos un nuevo j que va a ser el nodo más cercano a i después del nodo j tomado anteriormente.
Si el nodo j no es un nodo colgante asignamos el nodo i antes o después del nodo j, dependiendo la posición en la cual obtengamos el mínimo costo de inserción (o eventualmente solo factibilidad). En el ejemplo el nodo extraido $i = 2$ puede ser insertado entre el nodo $j = 12$ y alguno de los nodos adyacentes a él (nodo 11 o nodo 13). Elegimos el nodo 11, que resulta ser la posición en la cual obtenemos el mejor costo. (Ver Figura 5.6)

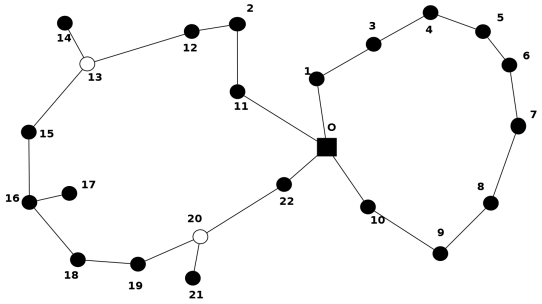

Figura 5.6: Estado luego del movimiento de inserción- extracción

El ciclo para cada nodo terminal i finaliza después de haber considerado todas las posibles inserciones entre los l nodos más cercanos y haber seleccionado el movimiento que produce el menor costo total.

El algoritmo sigue este mismo procedimiento para todos los $i \in T$ no considerados aun, examinando $N(i)$ hasta que finalmente seleccionamos, el movimiento que produce el menor costo total.

5.2.2. Consideraciones de factibilidad

Al aplicar los movimentos especificados anteriormente la solución podría dejar de ser factible si se dan alguna de estas condiciones:

Algoritmo 9 En este algoritmo se extrae aleatoriamente un nodo y se inserta en una posición que mejora el costo total.

```
1: input G_inic, T, k
2: G_best ← G_inic
3: for (i = 1 to |T|) do
4:    if (i no es un nodo colgante) then
5:       Extraer el nodo i de su posición
6:       sea K el conjunto ordenado de los k nodos más cercanos al nodo i
7:       for (u = 1 to k) do
8:          Sea j = u − esimo nodo más cercano al nodo i
9:          if (i no era base de un nodo colgante) then
10:            estado_1 = cuelgo el nodo i del Nodo j
11:         end if
12:         estado_2 = inserto el nodo i entre el nodo j y su adyacente
13:         se elige el adyacente que genere solución factible con mejor costo resultante
14:         seleccionar el mejor estado factible entre estado_1 y estado_2
15:         improve = actualizar(G_best)
16:         if (improve) then
17:            breakfor
               // salgo del for, no considero los siguientes nodos cercanos
18:         end if
19:      end for
20:   end if
21: end for
22: return G_best
```

- Si no existen las aristas correspondientes en el grafo original que sean incidentes al nodo i en la nueva posición (es decir incidentes a i y a los adyacentes de j en la solución).

- Que los nodos adyacentes al nodo quitado i no sean adyacentes entre sí (esta condición y la anterior podrían ocurrir si el grafo original no fuera completo)

- Que sea violado el parámetro Q (candidad de nodos por cada componente) al mover el nodo i hacia otra componente.

Cualquier solución que altere la factibilidad será descartada y no se aplica ninguna estrategia de reparación para volver a refactibilizar la misma. En general dentro del vecindario considerado en esta búsqueda local no hay demasiadas soluciones no factibles si el grafo es completo o lo suficientemente denso, tal cual son los casos de prueba con los que trabajamos en esta tesis.

5.2.3. Observaciones

Dado que el problema consiste en obtener m componentes 2-nodos conexas con colgantes que no superan los Q nodos en cada una de ellas, esta búsqueda local es particularmente útil para equilibrar la cantidad de nodos en cada una de dichas componentes. Es la única búsqueda local utilizada que en todos sus movimientos aumenta la cantidad de nodos en una componente y disminuye esta cantidad en otra (siempre que los nodos involucrados pertenezcan a distintas componentes). Esta clase de movimiento nos permite explorar algunas regiones del espacio de soluciones que no alcanzan otras búsquedas locales, lo cual es deseable para la performance general de la metaheurística.

5.3. Swapping de Nodos

5.3.1. Descripción

El principio básico de esta búsqueda local consiste en seleccionar dos nodos y hacer un intercambio (swapping) entre ellos. Este procedimiento comienza con una selección aleatoria de un nodo terminal que no sea colgante y prueba todas las formas posibles para intercambiar este nodo con otro nodo perteneciente a una componente 2-nodo conexa que se encuentre "$cerca$" del nodo seleccionado. No consideramos en estos movimientos nodos colgantes.

Para aclarar el concepto "$cerca$" mencionado en el párrafo anterior, debemos definir un vecindario con respecto al nodo considerado.

Nuevamente apelaremos a la misma definición de vecindario que utilizamos en la búsqueda local de extracción e inserción, detallada en 5.2, es decir el vecindario N de los l nodos $j \in T$ más cercanos al nodo i.
El algoritmo comienza tomando un nodo i al azar y procede de la siguiente manera.
Consideremos nuevamente a j como el nodo más cercano al nodo i. Si el nodo j es un nodo colgante no realizamos ningún movimiento y seguimos con el siguiente nodo, es decir tomamos un nuevo j que va a ser el nodo más cercano a i después del nodo j tomado anteriormente. Si el nodo j pertenece a la misma o a otra componente 2-nodo conexa entonces los intecambiamos.

Cada vez que un movimiento de swapping conduce a una mejora y mantiene la factibilidad, la solución corriente se actualiza y el posible swapping con los otros nodos j en orden descendente de distancia es desechado y se continua con el siguiente nodo i a analizar.

Dado dos nodos a intercambiar que llamaremos primer nodo y segundo nodo, el movimiento de swapping consiste en eliminar las aristas incidentes al primer nodo, eliminar las aristas incidentes al segundo nodo, y generar nuevas aristas incidentes entre los antiguos adyacentes al primer nodo y el segundo nodo y los antiguos adyacentes al segundo nodo y el primero. En las

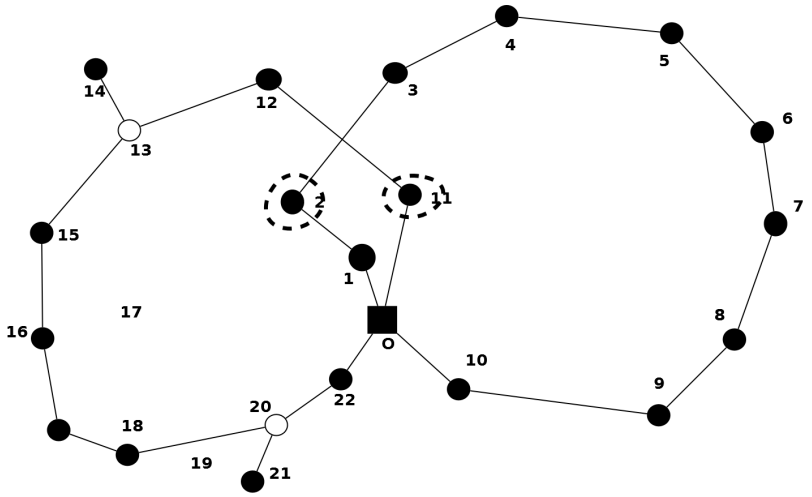

Figura 5.7: Solución en estado previo al movimiento de swapping

Figuras 5.7 y 5.8 se puede ver la solución antes y después de la ejecución de un movimiento de intercambio entre los nodos 2 y 11.

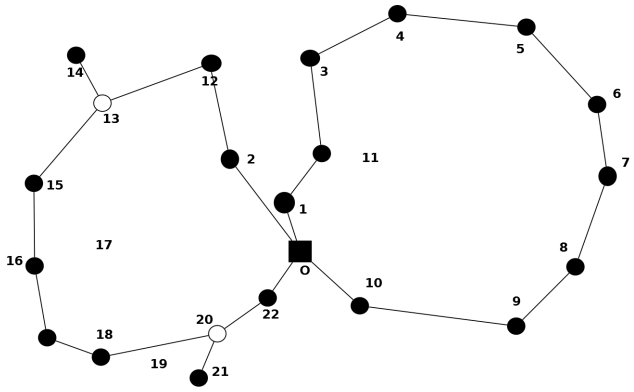

Figura 5.8: Solución en estado posterior al movimiento de swapping

El procedimiento continúa con el siguiente nodo terminal no considerado y repite el movimiento de swapping bajo las premisas explicadas en los párrafos anteriores y se detiene cuando todos los nodos terminales de la solución corriente hayan sido considerados.

5.3.2. Consideraciones de factibilidad

Al aplicar los movimentos de intercambio la solución para esta búsqueda local podría dejar de ser factible si se dan alguna de estas condiciones:

Algoritmo 10 Este algoritmo realiza el swap de un nodo terminal con otro nodo cercano, en la misma o en otra componente 2-nodo-conexa.

```
1: input G_inic, T, k
2: G_best ← G_inic
3: for (i = 1 to |T|) do
4:    if (i no es un nodo colgante) then
5:       sea K el conjunto ordenado de los k nodos más cercanos al nodo i
6:       for (u = 1 to k) do
7:          Sea j = u − esimo nodo más cercano al nodo i
8:          intercambiar el nodo i con el nodo j
9:          improve = actualizar(G_best)
10:         if (improve) then
11:            breakfor // salgo del for, no considero los siguientes nodos cercanos
12:         end if
13:      end for
14:   end if
15: end for
16: return G_best
```

- Si no existen las aristas correspondientes en el grafo original que sean incidentes a los nodos i o j en las nuevas posiciones respectivas.

- Que sea violado el parámetro Q (candidad de nodos por cada componente) al mover el nodo i hacia otra componente. Esta condición puede darse si alguno de los nodos involucrados en el swapping (nodos i y j) tiene una cantidad de nodos colgantes adyacentes diferente al otro nodo a intercambiar.

Cualquier solución que altere la factibilidad también será descartada en esta búsqueda local y de igual forma que en la extracción-inserción de nodos, no se aplica ninguna estrategia de reparación para volver a refactibilizar dicha solución.

5.3.3. Observaciones

Esta búsqueda local mueve los nodos hacia mejores posiciones dentro de una misma componente y entre dos componentes cualesquiera. Estos movimientos cambian los nodos que integran cada componente, lo que además de mejorar las soluciones, estos movimientos serán aprovechados por otra de las búsquedas locales implementadas **-mejor camino con colgantes-**, que se introduce en la sección 5.5 la cual optimiza sobre el conjunto de nodos que cada componente tiene establecido en la solución hasta ese momento.

5.4. Crossing de Componentes

5.4.1. Descripción

Esta búsqueda local toma dos nodos "$cercanos$", cada uno de ellos en diferente componente, elimina una de las aristas adyacentes (para cada uno) y conecta mediante la arista que genere mejor costo, cada par de nodos de distinta componente.

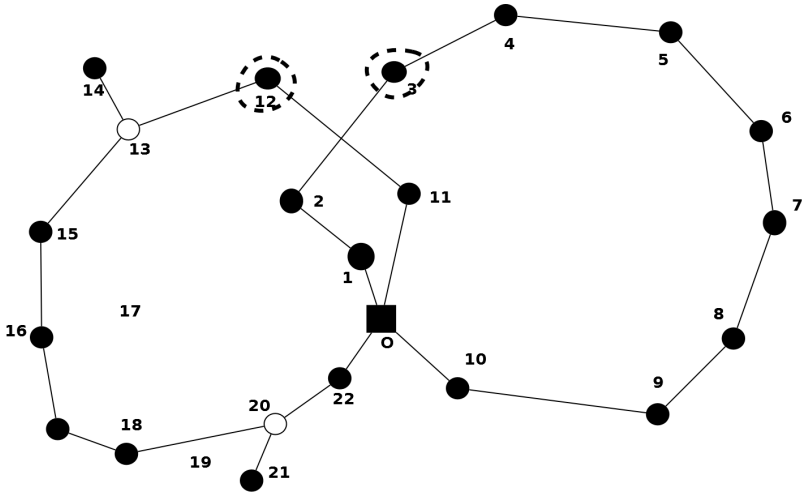

Figura 5.9: Estado del grafo antes de realizar el crossing de componentes

Pongamos un ejemplo gráfico. Tomemos el grafo inicial de la Figura 5.9 y consideremos un terminal no colgante cualesquiera, en nuestro ejemplo el nodo 12. Consideremos ahora el nodo más cercano en otra componente (nodo 3). Quitamos ahora una de las aristas incidentes a cada uno de los nodos (aristas $(12, 11)$ y $(3, 2)$) obteniendo el grafo de la Figura 5.10

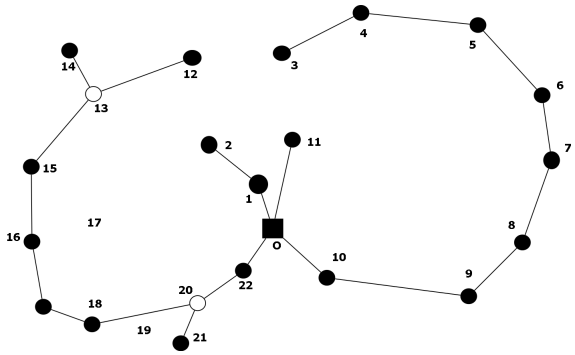

Figura 5.10: Estado del grafo con aristas eliminadas

Una vez realizada esta operación nos queda una cuaterna de nodos $(12, 3, 2, 11)$ que debemos conectar dos a dos, de manera que cada uno de los nodos del par pertenezca a distinta componente y genere la solución de mejor costo posible (o al menos factibilidad).

En nuestro ejemplo podemos generar las aristas $(2, 11)$ y $(12, 3)$ o bien $(12, 2)$ y $(3, 11)$. En el primer caso se puede observar que la cantidad de nodos resultantes en cada una de las componentes generadas después de realizar el movimiento es dispar (incluso para algún caso podrían llegar a desaparecer componentes, quedando reducidas al nodo depot y otras podrían violar el parámetro Q).

Supongamos que este primer movimento no es factible y tomemos el segundo resultando la solución de la Figura 5.11.

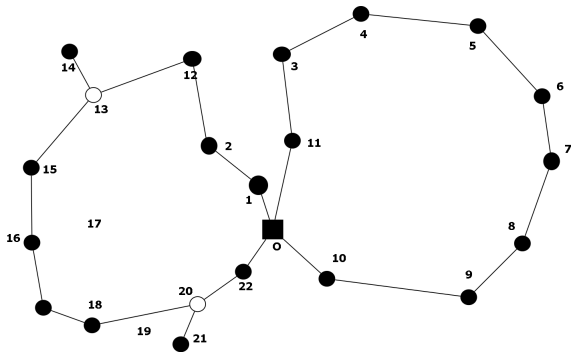

Figura 5.11: Estado del grafo luego de realizar el crossing de componentes

Si con el nodo más cercano (en nuestro ejemplo el nodo 3) no obtuvieramos mejores soluciones factibles consideraríamos entonces el siguiente nodo más cercano y así sucesivamente hasta una cierta cantidad especificada por un parámetro N.

Si con algunos de los nodos más cercanos obtuvimos una solución factible mejorada, los siguientes más cercanos no son considerados y seleccionamos otro nodo terminal al azar para aplicar el mismo proceso.

El procedimiento se detiene cuando todos los nodos terminales no colgantes de la solución corriente hayan sido considerados.

5.4.2. Consideraciones de factibilidad

Al aplicar los movimentos de crossing la solución para esta búsqueda local podría dejar de ser factible si se dan alguna de estas condiciones:

Algoritmo 11 En este algoritmo se elige un nodo terminal y uno de los más cercanos en otra componente, se elimina una arista adyacente de cada uno y se cruzan dichas componentes.

input G_{inic}, T, k
$G_{best} \leftarrow G_{inic}$
for ($i = 1$ **to** $|T|$) **do**
  if (i no es un nodo colgante) **then**
    sea K el conjunto ordenado de los k nodos más cercanos al nodo i
    for ($u = 1$ **to** k) **do**
      Sea $j = u - esimo$ nodo más cercano al nodo i
      elimino una arista adyacente al nodo i
      elimino una arista adyacente al nodo j
      Sea i' el otro extremo de la arista incidente a i
      Sea j' el otro extremo de la arista incidente a j

      estado_1=genero aristas (i, j') y (i', j)
      estado_2=genero aristas (i, j) y (i', j')
      selecciono el estado que genere solución factible con mejor costo resultante
      $improve = actualizar(G_{best})$
      if $(improve)$ **then**
        breakfor
        // salgo del for, no considero los siguientes nodos cercanos
      end if
    end for
  end if
end for
return G_{best}

- Si no existen las aristas correspondientes en el grafo original que sean incidentes a los pares de nodos considerados para el crossing en las nuevas posiciones respectivas.

- Que sea violado el parámetro Q (candidad de nodos por cada componente) al realizar el crossing.

- Que desaparezca alguna componente quedando reducida a una trivial (componetne compuesta solamente por el nodo depot).

Cualquier solución que altere la factibilidad también será descartada en esta búsqueda local, y de igual forma que en otras búsquedas locales no se aplica ninguna estrategia de reparación para volver a refactibilizar dicha solución.

5.4.3. Observaciones

Esta búsqueda local genera dos componentes nuevas a partir de dos componentes anteriores. Produce una diversidad de soluciones y de acuerdo a las pruebas realizadas permite escapar de óptimos locales un número considerable de veces.

5.5. Mejor camino con nodos colgantes

5.5.1. Descripción

Esta búsqueda local esta basada en un modelo de programación lineal entera. Daremos previamente la definición de las estructuras utilizadas por esta búsqueda local, que llamaremos **camino con nodos colgantes** o **camino con colgantes**.

DEFINICIÓN **5.1** ***Camino con nodos colgantes***. *Dado un grafo $G = (V, E)$ decimos que G es un camino con nodos colgantes con extremos a y $z \in V$, si se cumplen las siguientes condiciones:*

- *G es conexo y sin ciclos.*
- *Definimos camino principal $p(a, z) \subseteq G$ al camino que conecta los nodos a y z.*
- *Todos los nodos que no pertenecen a p están conectados a algún nodo de p a través de una arista simple.*

Dada una solución factible del problema el Algoritmo 12 consiste en identificar todos los ciclos que existen en cada componente y descomponerlos en caminos agregando los nodos colgantes. A cada camino con colgantes se le aplica una búsqueda local exacta obteniendo para cada uno la mejor solución con esa topología. Veamos un ejemplo gráfico. En la Figura 5.12 tenemos una solución del problema, factible pero no óptima. Consideremos ahora los ciclos presentes en dicha solución. Dichos ciclos son:

- (0, 12, 11, 10, 9, 8, 5, 7, 4, 2, 1, 0)
- (0, 13, 14, 16, 17, 18, 26, 27, 28, 0)
- (14, 16, 17, 18, 20, 21, 23, 25, 14)
- (0, 13, 14, 25, 23, 21, 20, 18, 26, 27, 28, 0)
- (0, 30, 31, 32, 34, 35, 38, 39, 40, 43, 46, 47, 0)

Cada uno de estos ciclos vamos a descomponerlos en caminos agregando los nodos colgantes (si los hubiese). Dichos caminos podrán comenzar en cualquier vértice del ciclo, tendrán una longitud mínima de por los menos 2 aristas (3 nodos) y como máximo un parámetro dado que llamaremos MAX_PATH_LENGTH que especifica largo máximo del camino a considerar. Esto es particularmente útil para grafos grandes donde los ciclos puenden tener una cantidad considerable de nodos y por lo tanto en los algoritmos que están basados en programación lineal entera se pueden disparar los tiempos de ejecución.

Descompongamos el primer ciclo (0,12,11,10,9,8,5,7,4,2,1,0). Una descomposición válida podría ser:

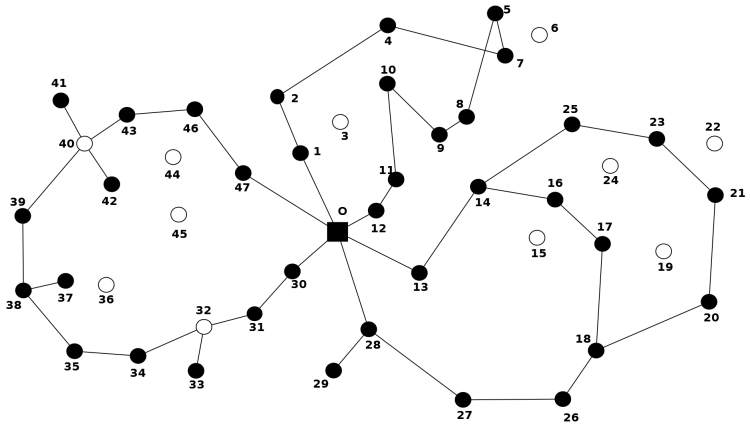

Figura 5.12: Solución factible inicial

- (11, 10, 9, 8, 5, 7, 4)

- (4, 2, 1, 0, 12, 11)

Tomamos como ejemplo el primero de los caminos y le agregamos los nodos colgantes (si los hubiera) tal cual están presentes en la solución. Consideremos ahora el subgrafo inducido del grafo original, que contiene los vértices del camino seleccionado y los nodos colgantes. Dicho subgrafo contiene las aristas que forman parte del camino, las aristas cuyo extremo es un colgante y las que están presentes en el grafo original y no forman parte de la solución factible inicial.

Consideremos ahora los nodos de Steiner. Para su inclusión podemos utilizar varios criterios. El que utilizamos nosotros es elegir los nodos de Steiner adyacentes al subgrafo y sortear dos de ellos. Las aristas a incluir serán todas las incidentes a los nodos de Steiner sorteados y a algún nodo del subgrafo considerado. El subgrafo resultante se detalla en la Figura 5.13. Las aristas que no formaban parte del camino original se dibujan mediante lineas más claras.

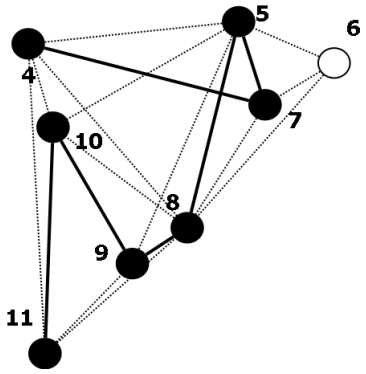

Figura 5.13: Subgrafo inducido por el camino con nodos colgantes

El grafo así definido es la entrada del algoritmo **path_with_rays** declarado en la línea 11 del algoritmo 12 que calcula el mejor camino con nodos colgantes entre los vértices 11 y 4 (en nuestro ejemplo, ver Figura 5.14). El comienzo (nodo 11) y el fin (nodo 4) del camino deberán mantenerse como inicio y fin respectivamente del nuevo camino con colgantes.

El Algoritmo **path_with_rays** transforma el camino con nodos colgantes en un problema de programación lineal entera cuya solución óptima es un nuevo camino con nodos colgantes con los mismos extremos, de menor o igual valor al proporcionado como entrada.

Una vez obtenida la solución, sustituimos el camino con nodos colgantes original por el camino óptimo, obteniendo así una nueva solución factible con mejor costo que la anterior (Ver Figura 5.15). Estos movimientos deben ser realizados para todos los caminos obtenidos de la descomposición de los ciclos de cada una de las componentes como es especificado en el Algoritmo 12.

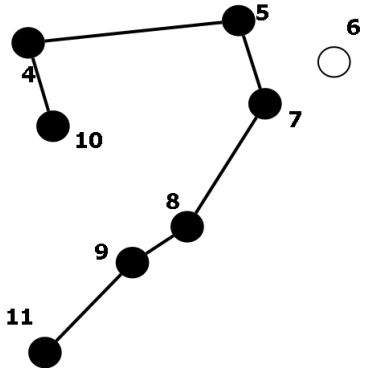

Figura 5.14: Mejor camino con nodos colgantes del subgrafo de la Figura 5.13

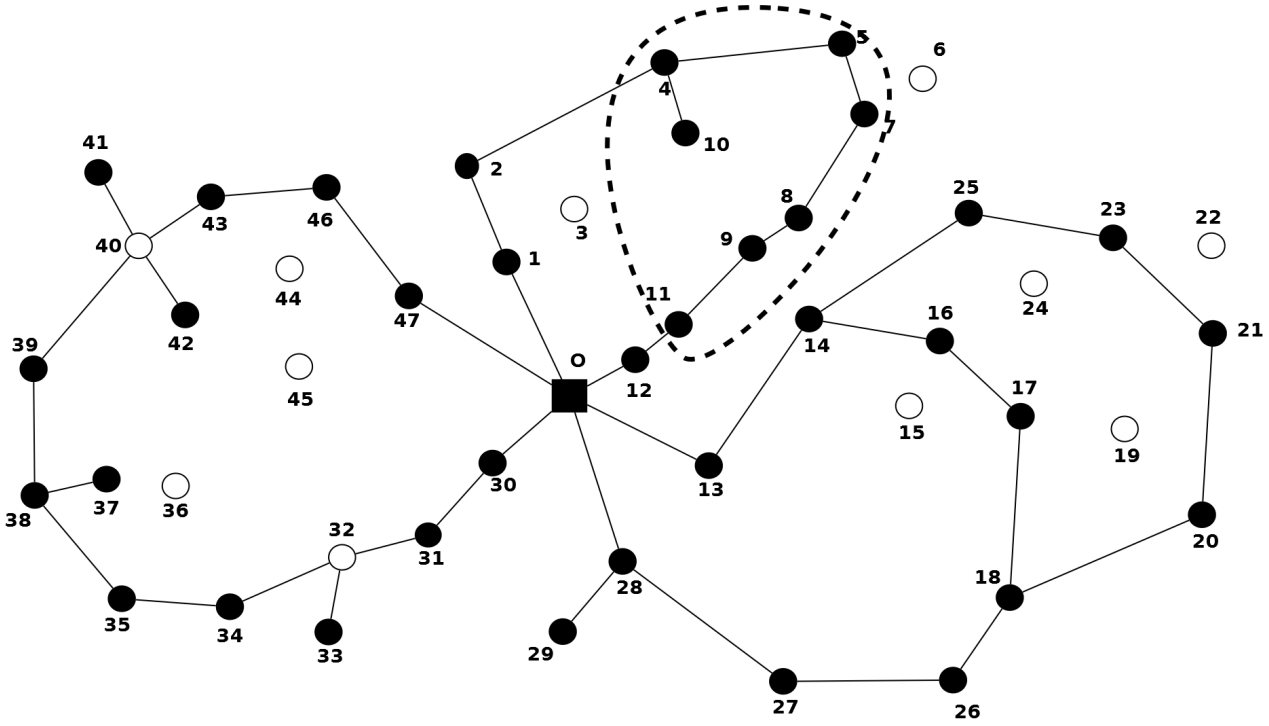

Figura 5.15: Solución mejorada luego de la sustitución de caminos

5.5.2. Algoritmo del mejor camino con nodos colgantes

Este algoritmo se basa en un modelo de programación lineal entera que toma por entrada un grafo con dos nodos distinguidos a y z y devuelve como solución el camino con nodos colgantes óptimo de extremos a y z.

5.5.2.1. Modelo de MIP para el problema de mejor camino con colgantes

Definiciones previas

Sea $G = (V, E)$ un grafo donde V es el conjunto de vértices del grafo y E es el conjunto de aristas.

Sea $\hat{\mathrm{T}}$ el conjunto de los nodos terminales del grafo G.

Sean a y z 2 nodos distinguidos del conjunto de nodos terminales tal que $a \in \hat{\mathrm{T}}$ y $z \in \hat{\mathrm{T}}$.

Sea $T = \hat{\mathrm{T}} \setminus (\{a\} \cup \{z\})$ el conjunto de los nodos terminales que no incluye los extremos.

Definimos $C = \{c_{ij}\}_{i,j \in V}$ como la matriz de costos de ruteo del grafo, es decir los costos de la arista (i, j) cuando dicha arista pertenece al camino principal $p(a, z)$ de extremos a y z.

Definamos ahora $D = \{d_{ij}\}_{i,j \in V}$ como la matriz de costos de conexión del grafo, es decir los costos de la arista (i, j) cuando dicha arista tiene por uno de los extremos a un nodo colgante y el otro extremo es un nodo del camino principal.

Sea $W = V \setminus \hat{\mathrm{T}}$ el conjunto de los nodos de Steiner.

Definamos ahora las variables del modelo.

$$X_i = \begin{cases} 1 & \text{si el nodo } i \in \hat{\mathrm{T}} \text{ forma parte del camino principal} \\ 0 & \text{o.c.} \end{cases} \tag{5.6}$$

$$Y_i = \begin{cases} 1 & \text{si el nodo } i \in T \text{ es un nodo colgante del camino principal} \\ 0 & \text{o.c.} \end{cases} \tag{5.7}$$

$$Yc_{i,j} = \begin{cases} 1 & \text{si } i \in \hat{\mathrm{T}} \text{ y } j \in V \text{ están conectados, siendo } i \text{ un nodo colgante y } j \text{ un nodo del camino principal} \\ 0 & \text{o.c.} \end{cases} \tag{5.8}$$

$$x_{i,j} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es usada en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{5.9}$$

$$w_{i,j} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es arista colgante y es usada en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{5.10}$$

$$Y_{(i,j)}^{(u,v)} = \begin{cases} 1 & \text{si la arista } (i, j) \text{ es usada en el camino que va del nodo } u \text{ a } v \\ 0 & \text{o.c.} \end{cases} \tag{5.11}$$

Modelo de Programación Matemática

Una vez especificadas las variables del modelo de programación lineal entera, éste quedará entonces definido de la siguiente forma:

$$min(\sum_{i,j \in V} c_{ij}(x_{ij} - w_{ij}) + \sum_{i,j \in V} d_{ij} w_{ij}) \tag{5.12}$$

sujeto a:

$$X_i + Y_i = 1 \qquad \forall i \in T \tag{5.13}$$

$$X_i = 1 \qquad \forall i \in (\{a\} \cup \{z\}) \tag{5.14}$$

La restricción 5.13 nos asegura que cualquier nodo terminal que no sea extremo del camino, o bien pertenece al camino principal, o bien está colgado del camino principal mediante una arista colgante, mientras que la restricción 5.14 asegura que los extremos pertenecen exclusivamente al camino principal.

$$Yc_{ij} \leq X_j \qquad \forall i \in T \quad \forall j \in Ady[i] \tag{5.15}$$

La restricción 5.15 implica que si i y j están conectados y el nodo i es un nodo colgante, entonces el nodo j pertenece al camino principal.

$$Y_i = \sum_{j \in Adj(i)} Yc_{ij} \qquad \forall i \in T \tag{5.16}$$

$$\sum_{j \in V} w_{i,j} \leq Y_i \qquad \forall i \in T \tag{5.17}$$

La restricción 5.16 implica que si el nodo i cuelga del camino principal entonces cuelga a través de una sola arista y 5.17 asegura que no haya más de una arista incidente a un nodo colgante en la matriz que representa a dichas aristas.

$$Yc_{i,j} = w_{i,j} \qquad \forall i \in T \quad j \in Ady[i] \tag{5.18}$$

La restricción 5.18 implica que si el nodo i cuelga del nodo j entonces la arista (i, j) es colgante y pertenece a la solución.

$$\sum_{j \in Ady[i]} x_{i,j} \leq M(1 - Y_i) + 1 \quad \forall i \in T \quad M \in \mathbb{Z}^+, M \geq max(\delta_i) \;\; i = 1 \cdots |V| \tag{5.19}$$

La desigualdad 5.19 restringe el grado de los nodos colgantes a 1 y deja en libertad el grado de cualquier otro nodo del camino principal. El parámetro M debe ser al menos el grado mayor de los nodos del grafo. Una cota superior es |V| -1 en caso que el grafo sea completo.

$$w_{i,j} \leq x_{i,j} \qquad \forall i \in T \quad j \in Ady[i] \tag{5.20}$$

La ecuación 5.20 implica que si una arista es colgante entonces pertenece a la solución.

$$\sum_{j \in Ady[u]} y_{(u,j)}^{(u,v)} = 1 \qquad \forall u, v \in \hat{\mathrm{T}}, u \neq v, \tag{5.21}$$

$$\sum_{i \in Ady[v]} y_{(i,v)}^{(u,v)} = 1 \qquad \forall u, v \in \hat{\mathrm{T}}, v \neq u, \tag{5.22}$$

Las restricciones 5.21y 5.22 son las restricciones de conectividad simple entre los nodos del camino con colgantes.

$$\sum_{i \in Ady[p]} y_{(i,p)}^{(u,v)} - \sum_{i \in Ady[p]} y_{(p,i)}^{(u,v)} \geq 0 \quad \forall u, v \in \hat{\mathrm{T}}, \quad \forall p \in V \setminus u, v \tag{5.23}$$

La restricción 5.23 es la ecuación de balance de los nodos internos.

$$y_{(i,j)}^{(u,v)} + y_{(j,i)}^{(u,v)} \leq x_{i,j} \quad \forall u, v \in \hat{\mathrm{T}}, u \neq v, \quad \forall (i,j) \in E \tag{5.24}$$

La restricción 5.24 garantiza que el camino con colgantes es arista-disjunto.

$$Y_i = 0 \qquad \forall i \in W \tag{5.25}$$

$$\sum_{j \in Ady(i)} Yc_{i,j} = 0 \qquad \forall i \in W \tag{5.26}$$

Las igualdades 5.25 y 5.26 restringen la pertenencia de los nodos de Steiner al camino principal, dichos nodos no pueden ser colgantes.

$$\Big(\sum_{i \in Ady[j]} Yc_{i,j} + 2X_j - \sum_{i \in Ady[j]} x_{j,i} = 0 \Big) \qquad \forall j \in W \tag{5.27}$$

La igualdad 5.27 garantiza que los nodos de Steiner si son utilizados, deben tener grado 2 considerando las aristas que forman parte del camino principal, no se considera en el conteo del grado las aristas cuyo extremo es un nodo colgante.

$$\sum_{i \in Ady[j]} (Yc_{i,j} + Yc_{j,i}) + 2X_j - \sum_{i \in Ady[j]} x_{j,i} = 0 \qquad \forall j \in T \tag{5.28}$$

La ecuación 5.28 fija en 2 el grado de los nodos internos del camino principal sin considerar en dicho grado las aristas colgantes.

$$\sum_{i \in Ady[j]} (Yc_{i,j}) + X_j - \sum_{i \in Ady[j]} x_{j,i} = 0 \qquad \forall j \in (\{a\} \cup \{z\}) \tag{5.29}$$

La ecuación 5.29 fija en 1 el grado de los nodos extremos del camino principal, sin considerar en dicho grado las aristas colgantes.

Hemos definido así el modelo de programación lineal entera para el mejor camino con nodos colgantes. Describiremos a continuación un pseudocódigo de dicho algoritmo.

5.5.2.2. Pseudocódigo del algoritmo

El Algoritmo 12 describe la búsqueda local que consiste en la sustitución de un camino con colgantes, por otro con los mismos nodos e idénticos extremos cuyo costo total sea menor.

El algoritmo comienza tomando como entrada el grafo G_{Sol}, solución factible del CmTNSSP. Para cada una de las m componentes de G_{Sol} cuenta los ciclos, identifica cada uno de ellos y los almacena en la lista indexada all_cycles (Líneas 3 y 4).

A continuación se tratan cada uno de los ciclos identificados en los pasos anteriores, ejecutando las operaciones definidas en al alcance del **for** (Líneas 5 al 17) hasta agotar todos los ciclos.

Cada ciclo se divide en una cierta cantidad de caminos de largo variable que dependen del parámetro MAX_PATH_LENGTH y del largo del ciclo. Se fija un nodo de comienzo y un nodo de fin del primer camino dentro del ciclo (Líneas 7 y 8) que fijamos para esta primera inicialización en 1 y largo del ciclo respectivamente.
Una vez inicializado el camino a procesar, se ingresa en un ciclo repetitivo determinado por el alcance del (**while**) (Líneas 9 a 16) donde reajustamos el largo del camino de forma aleatoria (Línea 10).

A cada camino obtenido en el paso anterior se le agregan los nodos colgantes presentes en G_{Sol} (Línea 11) obteniendo un camino con nodos colgantes de extremos $begin\_path$, end_path, tal

Algoritmo 12 Mejora de la solución descomponiendo todos los ciclos del grafo en caminos con sus respectivos nodos colgantes, obteniendo el mejor sustituto para cada uno de ellos.

1: input G_{sol}, G, C, D, T, MAX_PATH_LENGTH
2: $G_{best} \leftarrow G_{sol}$
3: $q\_cycles = cycles\_count(G_{sol})$ // Cantidad de ciclos de G_{sol}
4: $all\_cycles \leftarrow cycles(G_{sol})$ // Arreglo con los ciclos de G_{sol}
5: **for** ($i = 1$ **to** q_cycles) **do**
6: $path\_long$ = min (length($all\_cycles(i)$), MAX_PATH_LENGTH))
7: $begin\_path = 1$
8: end_path =length($all\_cycles(i)$)
9: **while** ($end\_path \leq$ length($all\_cycles(i)$)) **do**
10: end_path = $begin\_path$ + 3 + (rand() MOD ($path\_long$ - 2))
11: P = **path_with_rays**(G_{sol}, $all\_cycles(i)$, $begin\_path$, (end_path MOD length($all\_cycles(i)$))
12: $H \leftarrow$ **induced_graph_path**(P, G, T)
13: P_{best} = **best_pwr**(G_{sol}, $G\ P\ C$, D, H)
14: $G_{best} \leftarrow G_{best}$ - P + P_{best}
15: $begin\_path = end\_path$
16: **end while**
17: **end for**
18: **return** G_{best}

cual lo especificamos en la Definición 5.1.

En el siguiente paso generamos el grafo inducido H por los nodos del camino con colgantes P con respecto al grafo original G. (Línea 12).

El grafo H generado en el paso anterior es la entrada del proceso **best_pwr** que nos devuelve el mejor camino con colgantes de extremos $begin\_path$, end_path (Línea 13).

En la Línea 14 realizamos la sustitución en G_{best} del camino con colgantes P por el camino con colgantes P_{best}, obteniendo un solución de mejor costo.

A continuación se reajusta el nodo de comienzo y nodo de fin en el ciclo que estamos tratando, (Línea 15) para generar un nuevo camino.

Una vez procesados todos los caminos dentro de cada uno de los ciclos, devolvemos la solución de mejor costo G_{best} (Línea 18).

5.6. Mejor componente 2-nodo-conexa

5.6.1. Descripción

Esta búsqueda local también esta basada en un modelo de programación lineal entera. De igual forma que en la búsqueda local anterior dada una solución factible del problema, el Algoritmo 13 identificará todos los ciclos que existen en cada componente. A cada ciclo le aplicamos ahora un algoritmo exacto obteniendo para cada uno la mejor solución sustituto dedicho ciclo con topología 2-nodo-conexa.

Como vimos en el Capítulo 2 la mejor solución dos nodo-conexa que cubre un cierto conjunto de nodos no necesariamente es un ciclo, por lo tanto esta búsqueda local podrá incluir dichas topologías en nuestra solución.

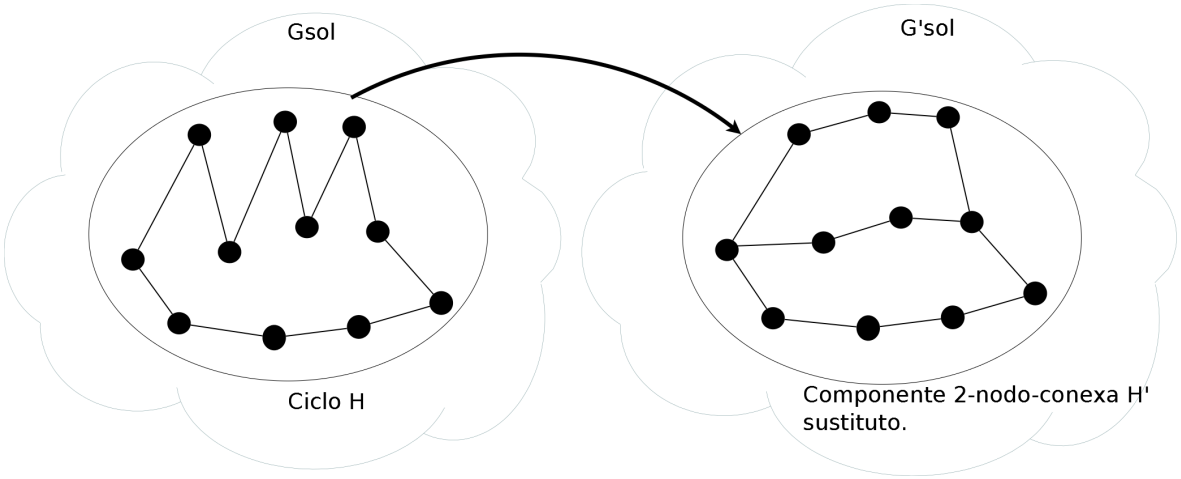

Figura 5.16: G'_{sol} es resultante de la sustitución del ciclo H por la componente 2-nodo-conexa H' de mejor costo.

5.6.2. Algoritmo de la mejor componente 2-nodo-conexa

Este algoritmo toma por entrada el subgrafo inducido del grafo original, con los nodos del ciclo y algún nodo de Steiner sorteado entre los más próximos a dichos nodos.

La salida del algoritmo es un grafo de cubrimiento 2-nodo conexo óptimo que sustituiremos por el ciclo tomado como entrada.

5.6.2.1. Modelo de programación matemática

Uilizaremos para modelar esta búsqueda local un caso particular del **GSP** (Problema general de Steiner) donde la conectividad de todos sus nodos terminales es 2.

Definiciones previas

Sea $G = (V, E)$ un grafo donde V es el conjunto de vértices del grafo y E es el conjunto de aristas.

Sea $\hat{\mathrm{T}}$ el conjunto de los nodos terminales del grafo G.
Definimos $C = \{c_{ij}\}_{i,j \in V}$ como la matriz de costos de ruteo del grafo, es decir los costos de la arista (i, j) cuando dicha arista pertenece a la estructura 2-nodo- conexa de la componente. Solo utilizaremos esta matriz de costos dado que en esta búsqueda local no se consideran los nodos colgantes que se hayan generado hasta el momento.

Pasemos a definir a continuación las variables del modelo.

$$x_{i,j} = \begin{cases} 1 & \text{si la arista } (i,j) \text{ es usada en la solución} \\ 0 & \text{o.c.} \end{cases} \tag{5.30}$$

$$y_{(i,j)}^{(u,v)} = \begin{cases} 1 & \text{si la arista } (i,j) \text{ es usada en el camino que va del nodo } u \text{ a } v \\ 0 & \text{o.c.} \end{cases} \tag{5.31}$$

Modelo de Programación Matemática

Una vez especificadas las variables del modelo de programación lineal entera (binaria) éste quedará entonces definido de la siguiente forma:

$$min(\sum_{i,j \in V} c_{ij} x_{ij}) \tag{5.32}$$

sujeto a:

$$\sum_{j \in Ady[u]} y_{(u,j)}^{(u,v)} = 2 \qquad \forall u, v \in \hat{\mathrm{T}}, u \neq v, \tag{5.33}$$

$$\sum_{i \in Ady[v]} y_{(i,v)}^{(u,v)} = 2 \qquad \forall u, v \in \hat{\mathrm{T}}, v \neq u, \tag{5.34}$$

$$\sum_{i \in Ady[p]} y_{i,p}^{(u,v)} - \sum_{i \in Ady[p]} y_{(p,i)}^{(u,v)} \geq 0 \quad \forall u, v \in \hat{\mathrm{T}}, \quad \forall p \in V \setminus u, v \tag{5.35}$$

$$y_{(i,j)}^{(u,v)} + y_{(j,i)}^{(u,v)} \leq x_{i,j} \quad \forall u, v \in \hat{\mathrm{T}}, u \neq v, \quad \forall (i,j) \in E \tag{5.36}$$

El modelo corresponde al GSP (Problema de Steiner Generalizado) donde el requerimientos de conectividad entre cada par de nodos terminales es 2.

5.6.2.2. Pseudocódigo del algoritmo

Algoritmo 13 Mejora de la solución cambiando los ciclos del grafo por la mejor componente 2-nodo conexa.

1: input $G,\ G_{sol},\ C,\ T$
2: $G_{best} \leftarrow G_{sol}$
3: $q\_cycles = cycles\_count(G_{sol})$ // Cantidad de ciclos de G_{sol}
4: $all\_cycles \leftarrow cycles(G_{sol})$ // Arreglo con los ciclos de G_{sol}
5: **for** ($i = 1$ **to** q_cycles) **do**
6: $\quad best =$ **best_2nc**($G_{sol},\ G_{orig},\ all\_cycles(i)$)
7: $\quad G_{best} \leftarrow G_{best}$ - $all\_cycles(i)$ + $best\_2nc$
8: **end for**
9: **return** G_{best}

De manera análoga al Algoritmo 12 el Algoritmo 13 cuenta e identifica los ciclos presentes en G_{sol} (líneas 3 y 4).

Para cada uno de estos ciclos el proceso **best_2nc** (línea 6) nos devuelve la mejor componente 2-nodo-conexa y en la línea 7 realiza la sustitución del ciclo tratado por dicha componente 2-nodo-conexa.

5.7. Implementación de los algoritmos con programación entera.

Explicaremos brevemente en esta subsección como llevamos los modelos de programación lineal entera a una implementación computacional, representando las variables en forma matricial para ser interpretadas como entrada del solver CPLEX utilizado.

Nuestro problema de programación lineal entera es de la forma:

$$\textbf{min } z = \sum_{j=1}^{n} c_j x_j \tag{5.37}$$

sujeto a:

$$\sum_{j=1}^{n} a_j x_j \left| \begin{matrix} \leq \\ = \\ \geq \end{matrix} \right| b_i \quad x_j \in \mathbb{Z}^+ \tag{5.38}$$

En 5.38 utilizamos como notación los corchetes para indicar que las desigualdades (o igualdad) pueden tener cualquiera de los símbolos contenidos en dichos corchetes. Algunas de las

variables (la amplia mayoría) deberán ser binarias y otras enteras.

Cada variable de nuestro modelo, sean de una dimensión tal como X_i, dos dimensiones como Y_{ij} o de cuatro dimensiones tales como $y_{(i,j)}^{(u,v)}$ se corresponden biunívocamente con una variable x_j.

El valor n especificado en 5.37 y en 5.38 depende de la cantidad de vértices del subgrafo inducido, la cantidad de nodos de Steiner que agreguemos para ser considerados, y la cantidad de aristas entre sus nodos.

Una vez desarrolladas todas las restricciones del modelo y presentadas en una matriz de m restricciones $\times$ n variables y en un vector de n variables (en realidad los coeficientes de dichas variables) para la función objetivo, se procede a realizar las llamadas al solver CPLEX. El solver nos devolverá la solución óptima en un vector de n variables, entre las cuales debemos obtener las variables binarias x_{ij} del modelo utilizado, las cuales nos indicará las aristas, tanto del nuevo camino con nodos colgantes como la mejor componente 2-nodo-conexa, dependiendo del modelo que estemos utilizando.

Esta relación biunívoca entre variables del modelo y variables de la implementación de su presentación en forma matricial, nos exige estructuras de datos optimizadas. Para poder implementar estos algoritmos debemos utilizar matrices dispersas, con pocos valores no nulos en relación a su cantidad de elementos.

Estas matrices dispersas, así como también las estructuras de almacenamiento de los procesos de codificación y decodificación de las variables del modelo en variables del programa, se realizaron en arreglos unidimensionales utilizando técnicas de hashing [43].

A modo de ejemplo podemos decir que el orden de filas y columnas de la matriz dispersa están en pocas decenas de miles de filas (restricciones) y columnas (variables) con cientos de miles de valores no nulos para una instancia de camino con nodos colgantes o de componentes 2-nodo-conexas de una docena de nodos.

5.8. Escapando de óptimos locales

5.8.1. Shaking

A medida que las búsquedas locales evolucionan, éstas comienzan a dar los mismos resultados, es decir que al aplicar dichas búsquedas las soluciones obtenidas no mejoran. Esto significa que los resultados óptimos locales se repiten. Para evitar que las soluciones obtenidas caigan en dichos óptimos locales se perturba de algún modo la solución y se vuelve a ejecutar las búsquedas locales para esta solución perturbada. La perturbación de las soluciones desde un punto de vista intuitivo, es un proceso que mantiene en cierto aspecto la solución óptima local y a la vez modifica ésta mínimamente. Para nuestro problema, donde construimos una red con ciertas características, la solución perturbada tendrá gran parte de las aristas de la solución que resultó ser un óptimo local pero algunas otras serán distintas, como serán distintos alguno de los nodos que integran cada componente.

El algoritmo de $shaking$ toma al azar una cierta cantidad p de nodos terminales los desconecta y guarda el orden en que se fueron desconectando dichos nodos en un lista.

A continuación vuelve a conectar los nodos de la siguiente manera. Tomando cada uno de los nodos desconectados de la lista ordenada (el orden es el mismo en que fueron desconectados) intenta conectarlos; en primer término con el nodo más cercano que no pertenezca a la lista, si la solución no es factible, consideramos el siguiente nodo más cercano y repetimos el proceso, si es factible seguimos con el siguiente nodo desconectado. El procedimiento finaliza cuando se hayan reconectado todos los nodos de la lista. El Algoritmo 14 muestra en detalle el procedimiento de shaking utilizado.

5.8.2. Observaciones

Si en el proceso de reconexión de nodos quedara algún nodo sin conectar, es decir que no pudiera ser conectado con los u nodos más cercanos en ninguna posición que genere una solución factible, repetimos el proceso de shaking.

Esta situación se ve en grafos no completos y poco densos, donde la reconexión puede ser difícil en algunos casos. Normalmente aumentando el parámetro k (cantidad de nodos más cercanos al nodo que vamos a reconectar) se minimiza la probabilidad de ocurrencia de tal situación.

Dada la estrategia seguida en la metaheurística, la cual corresponde a un algoritmo GRASP, si no fuera posible escapar de óptimos locales a través del proceso de $shaking$, el flujo del GRASP vuelve a construir una solución factible y a aplicar nuevamente las búsquedas locales. (Ver Figura 5.3).

Algoritmo 14 Este algoritmo perturba una solución factible desconectando una cierta cantidad de nodos al azar y volviéndolos a conectar con los nodos más cercanos.

input G_{inic}, T, p, k
$G_{best} \leftarrow G_{inic}$
$discon \leftarrow \phi$
for ($i = 1$ **to** p) **do**
  desconecto el nodo terminal i
  guardo el nodo i en la lista ordenada $discon$
end for
for $i = 1$ **to** p **do**
  obtengo el nodo i de la lista ordenada $discon$
  sea K el conjunto ordenado de los k nodos más cercanos al nodo i
  for ($u = 1$ **to** k) **do**
    if ($u \notin discon$) **then**
      inserto el nodo i entre el nodo u y alguno de sus adyacentes
      (se elige el adyacente que genere solución factible con mejor costo resultante)
      if (es $factible$) **then**
        $actualizar\ G_{best}$
        quito el nodo i de la lista $discon$
        break // salgo del for, no considero los siguientes nodos cercanos
      end if
    end if
  end for
end for
if ($discon = \phi$) **then**
  return G_{best}
else
  return G_{inic}
end if

Capítulo 6

Estudio experimental

6.1. Introducción

Hasta donde sabemos, no existe en la literatura una resolución exacta del problema CmTNSSP, por lo tanto no contamos con los datos suficientes para comparar la efectividad de la metaheurística desarrollada en esta tesis. Una alternativa hubiera sido generar casos de prueba aleatorios y estudiar los resultados obtenidos, pero creemos que esta metodología no aporta los elementos de evaluación necesarios.

Considerando que el CmTNSSP es una relajación del CmRSP y que toda solución del CmRSP es solución del CmTNSSP nos remitimos al trabajo acerca del CmRSP [4] donde la resolución exacta de la mayoría de las instancias allí definidas es un elemento de gran valor a la hora de evaluar el desempeño de nuestra implementación del CmTNSSP. En el trabajo de Baldacci et al. se resuelven la gran mayoría de dichas instancias hasta la optimalidad y las que no son resueltas tienen definidas cotas inferiores que nos servirán de guía para medir los resultados generados por nuestra aplicación.

6.1.1. Configuración del ambiente

El hardware donde se ejecutaron los algoritmos está compuesto por un computador con procesador Intel I7 con 8 Gb. de memoria RAM y Sistema Operativo Fedora Core 20.

Se utilizó como lenguaje de modelado para validar los modelos aplicados que forman parte de la metaheurística, **AMPL** y el solver fue **CPLEX** 12.5.

El modelo de resolución exacta del CmTNSSP solamente se escribió en **AMPL** para su validación y para resolver pequeñas instancias de prueba.

Los modelos programación lineal entera asociados a las búsquedas locales del mejor camino con colgantes y la sustitución de ciclos por la mejor componente 2-nodo conexa fueron validados en **AMPL** antes de su implementación integrada a la solución.

El software utilizado para la implementación de la solución está escrito en lenguaje **C** con llamadas al solver **CPLEX** mediante ***CPLEX** Callable Library* [44].

Para implementar la representación de los grafos se utilizaron estructuras y rutinas propias, tratándose de una implementación "from scratch". Trataremos entonces la resolución exacta del problema utilizando un lenguaje de modelado y resolviendo una instancia sencilla del problema.

6.2. Resolución exacta del problema CmTNSSP

El modelo de programación matemática presentado en el Capítulo 3 fue transcripto en el lenguaje de modelado AMPL. Esto nos permitió validarlo, y resolver instancias pequeñas del problema tratado.

No es el objetivo primario de este trabajo el desarrollo de un modelo optimizado y la resolución del problema en forma exacta. Aun así consideramos pertinente modelar el problema en lenguaje algebraico y resolverlo. Dada la naturaleza del problema ($\mathcal{NP}$-Difícil) nos hemos limitado a instancias reducidas para contar con una referencia más a la hora de su resolución mediante métodos aproximados.

Hemos revisado algunos lenguajes de modelado algebraico tales como GLPK, GAMS y AMPL seleccionando éste último, dado que el modelo es extenso (aún para instancias pequeñas) y fundamentalmente debido a la posibilidad de uso conjunto con el solver CPLEX [45].

"AMPL es un lenguaje de modelado algebraico amplio y de gran alcance para los problemas de optimización no lineal y lineal, en variables discretas o continuas. Desarrollado en los laboratorios Bell, AMPL permite utilizar la notación común y conceptos familiares para formular modelos de optimización y examinar las soluciones, mientras que el computador gestiona la comunicación con un solver apropiado"[46].

El solver que utilizamos para la resolución exacta del problema CmTNSSP fue IBM ILOG CPLEX Versión 12.5 [45]. CPLEX es un paquete de software para resolver problemas de optimización. IBM lo define como un motor de programación matemática de alto rendimiento. CPLEX toma su nombre de la combinación de método simplex y el lenguaje C, ya que fue diseñado originalmente para resolver el método simplex y escrito en lenguaje C. Hoy en día, contiene interfaces para diferentes lenguajes.

El modelo se ha implementado y ejecutado sobre varias instancias pequeñas y hemos seleccionado una de ellas para describirla.

A modo de ejemplo hemos definido un grafo que denominamos $nut30$ y que denotaremos

$N = (V, E)$ donde $V = T \cup W \cup \{d\}$, siendo:

$T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$ el conjunto de nodos terminales del grafo N, $W = \{20, 21, 22, 23, 24, 25, 26, 27, 28, 29\}$ el conjunto de los nodos de Steiner y $d = \{0\}$ el nodo depot.

Las matrices de costos C y D coinciden y los valores son las distancias euclídeas entre los vértices del grafo N.

Con el fin de acortar el tiempo de procesamiento computacional insumido en la ejecución del solver CPLEX, no hemos considerado el grafo completo y hemos generado solo algunas aristas del grafo N, por lo tanto el conjunto E contiene solamente las aristas que pueden observarse en la Figura 6.1. Aun así, dada la complejidad del modelo, la transformación a un problema de programación lineal entera para esta instancia contó con 721.244 filas, 618.913 columnas y 629.149 valores no nulos.

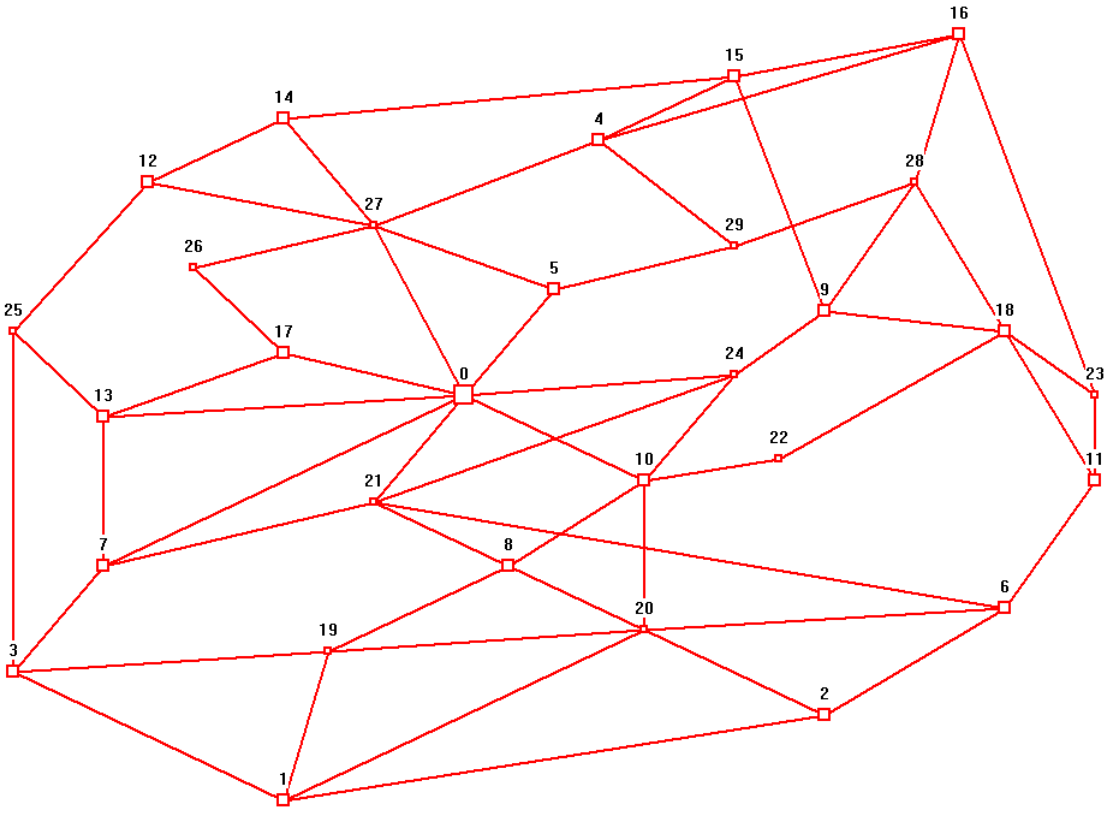

Figura 6.1: Grafo inicial ($nut30$) para test del modelo de programación matemática del CmTNSSP.

Fijamos para el problema CmTNSSP la cantidad de componentes en $m = 2$, y la restricción de capacidad en $Q = 12$. Una vez ejecutado el modelo obtenemos la solución exacta del CmTNSSP para la instancia definida anteriormente. La representación gráfica podemos observarla en la Figura 6.2. El tiempo de ejecución fue de 31 días 14 horas y 29 minutos. Este orden de tiempos de ejecución en la resolución del problema completo en forma exacta, nos motiva al diseño de algoritmos aproximados que sean mas eficientes en el uso de recursos computacionales.

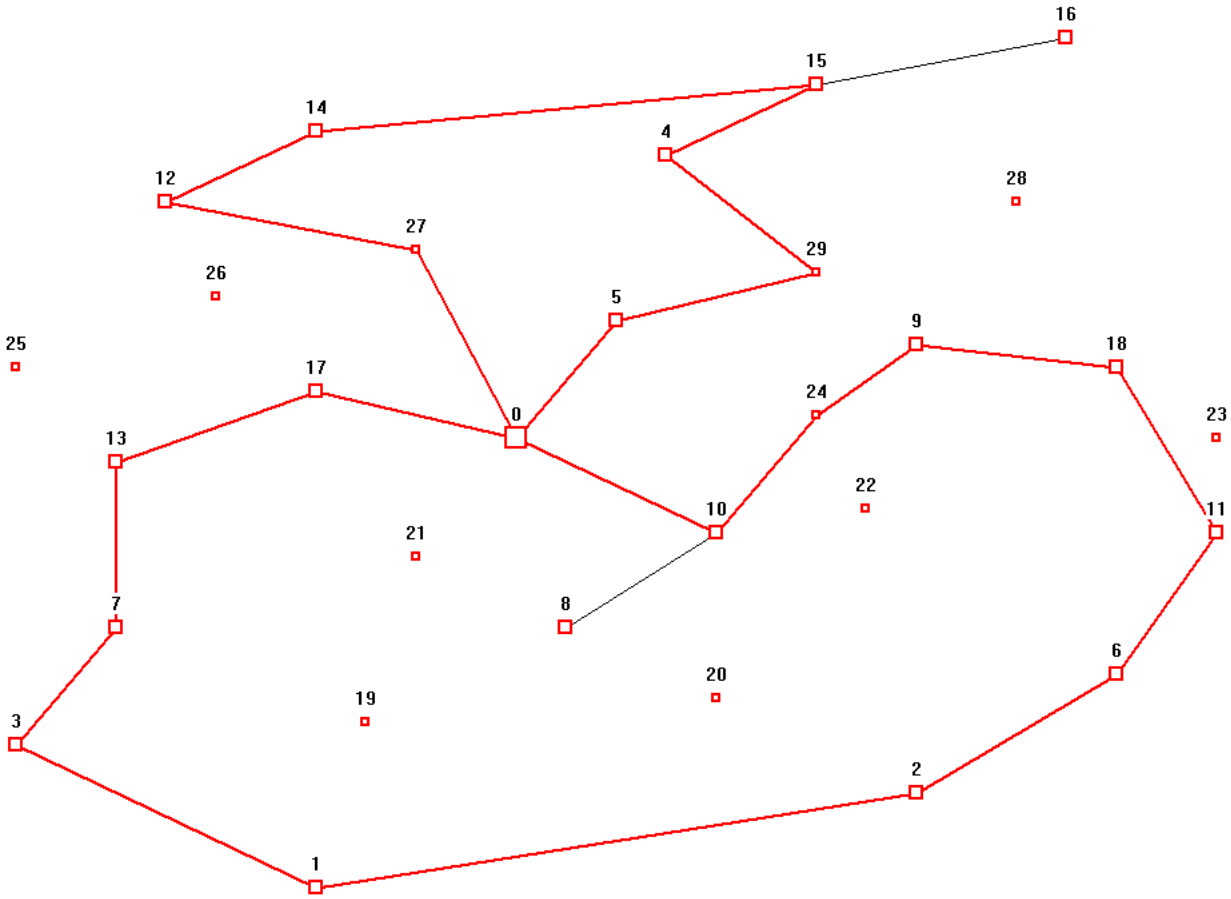

Figura 6.2: Solución óptima del CmTNSSP para el $nut30$, hallada mediante AMPL y CPLEX.

6.3. Casos de Prueba

Para ejecutar nuestros algoritmos con casos de prueba del trabajo de Baldacci et al. [4] procederemos a explicar la definición que el autor hace de las instancias utilizadas.

Los casos de prueba se dividen en dos clases, A y B. En la clase A los costos de ruteo y los costos de conexión coinciden. En la clase B los costos de ruteo son mayores a los costos de conexión. Para ambas clases de instancias los grafos utilizados son los mismos, la única diferencia se encuentra en los costos de sus aristas según sean o no incidentes a un nodo colgante.

Dichos grafos son $eil51$, $eil76$ y $eil101$ obtenidos de la TSPLIB (Travelling Salesman Problem Library) [47]. Adicionalmente se agrega un grafo más que denominamos $eil26$, construido con los primeros 26 vértices del $eil51$.

Definiremos entonces n=$\{26, 51, 76, 101\}$ como el total de vértices para cada uno de los grafos definidos en el párrafo anterior.

El primer nodo de cada uno de estos grafos será rotulado como nodo central o depot. Los restantes 25, 50, 75 y 100 respectivamente se dividirán en nodos terminales y nodos de Steiner de acuerdo a un parámetro $\alpha \in \{0{,}25, 0{,}5, 0{,}75, 1\}$, siendo U (conjunto de nodos terminales) los primeros $\alpha(n-1)$ nodos y W (conjunto de los nodos de Steiner) los restantes.

Para cada una de estas combinaciones se generarán instancias con $m \in \{3, 4, 5\}$ y el Q correspondiente a cada una de ellas será calculado para obtener una utilización de las componentes igual o superior al 90 % mediante la siguiente fórmula:

$$Q = \left\lceil \frac{|U|}{0{,}9m} \right\rceil \tag{6.1}$$

Esta definición de Q se corresponde con los casos reales donde la utilización de cada una las componentes deben estar en cierta forma equilibrada.

Para cada terna (n, α, Q) consideramos una instancia salvo para $Q \leq 3$. Quedan entonces 9 instancias para $n = 26$ y 12 instancias para los restantes $n \geq 51$.
Con respecto a los costos para las clases de instancias A y B los mismos se definen de la siguente manera:

- **Clase A**. Los costos de ruteo y de conexión son iguales y se corresponden con la distancia euclídea $e_{i,j}$ entre los nodos (i,j) considerados. $c_{i,j} = d_{i,j} = e_{i,j}$
- **Clase B**. Los costos de ruteo $c_{i,j} = \lceil \beta\ e_{i,j} \rceil$, siendo β un entero definido en el intervalo [6,9]. Los costos de conexión que utilizaremos serán $d_{i,j} = \lceil (10-\beta) e_{i,j} \rceil$. Para nuestras instancias de Clase B utilizaremos $\beta = 7$.

Además de las definiciones especificadas en los párrafos anteriores, existen algunas restricciones con respecto a los costos de conexión. Cada arista que conecta un nodo de la componente 2-nodo conexa con un nodo colgante no puede tener un costo mayor a un cierto costo d_{max}.
Sea el grafo original $G = (V, E)$ sobre el cual vamos a aplicar los algoritmos, con E el conjunto de aristas de dicho grafo, matriz de costos de ruteo C y matriz de costos de conexión D , d_{max} se calcula de la siguiente manera:

$$d_{max} = 0{,}2 \times \frac{\sum_{(i,j)\in E} d_{ij}}{|E|} \tag{6.2}$$

Por lo tanto las aristas del grafo solución que no pertenezcan a las componentes 2-nodo-conexas, no podrán tener un costo mayor a d_{max}.

En la implementación se realizó la lectura de los archivos con los casos de prueba tal cual fueron proporcionados por el Prof. Roberto Baldacci y la topología de las soluciones óptimas del CmRSP (en las instancias donde se hallaron óptimos globales) a sugerencia de él, se obtuvieron del sitio de los autores A. Hoshino y C. De Souza [48].

6.3.1. Compatibilidad de los problemas

La restricción mencionada anteriormente no forma parte de la definición del CmTNSSP que especificamos en 1.2.2. En realidad esta proviene del problema de Localización del Ciclo Medio en Redes tratado por Labbé et al. [7] donde se construye una red que consta de un ciclo principal y nodos colgantes cuyo costo total, compuesto de los costos de las aristas que pertenecen al ciclo (costos de ruteo) más los costos de conexión de las aristas con incidencia en nodos colgantes, es mínimo. En este problema se acota el costo total de conexión a un valor dado. En [8] el mismo autor resuelve el RSP (Ring Star Problem), un problema muy semejante al del ciclo medio en redes, sin hacer restricciones de costos en las aristas que no pertenecen al ciclo.

Baldacci et al. [4] resuelve el problema generalizado CmRSP en el cual existe más de un ciclo (m ciclos en realidad) que comparten un nodo especial denominado depot y capacitando el problema de modo que en cada componente pueda haber a lo sumo un número de nodos Q. Para este último problema se vuelve a introducir la acotación de costos de conexión, pero no se hace referencia a la suma de dichos costos, sino a cada uno de ellos individualmente.

A fin de mantener las mismas restricciones del problema CmRSP resuelto por Baldacci [4] y dado que el autor nos ha cedido gentilmente estas instancias para que las utilicemos como casos de prueba de nuestro problema CmTNSSP agregamos en la implementación dicha restricción, no permitiendo "colgar" nodos cuya distancia al nodo más cercano de la componente, exceda la cota especificada d_{max} (6.2).

Existe además otra sutil diferencia entre nuestro problema y la resolución del CmRSP propuesta en [4]. Dada la forma en que este se resuelve haciendo un split del nodo depot (resultando nodos 0 y $n+1$) y buscando a partir del nodo 0 llegar al nodo $n+1$ a través de un camino, puede haber caminos de una sola arista entre 0 y $n+1$ lo que en nuestro problema estaríamos ante una arista doble. Nuestra definición de CmTNSSP acepta como expresión mínima de componente 2-nodo-conexa al menos un ciclo de tres vértices, por lo que las aristas múltiples no están permitidas.

Para corregir esta diferencia, en el algoritmo que controla la factibilidad permitimos componentes 2-conexas de dos nodos y contamos dos veces la arista en los costos. En el algoritmo de construcción no llegaríamos nunca a estructuras de este tipo, pero sí en las búsquedas locales donde al ir suprimiendo nodos de la componente, podemos encontrar un escenario de estas características que debemos aceptar como válido.

6.3.2. Parámetros de inicialización

Los parámetros de inicialización de la metaheurística son los siguientes:

$ListSize = 4$, tamaño de la RCL del algoritmo de construcción de Soluciones Factibles para el CmTNSSP (Algoritmo 6).

$k = 0{,}2 \times |V|$ siendo k el parámetro correspondiente a la cantidad de nodos candidatos a intercambiar en el Algoritmo 10 (Swapping de nodos), la cantidad de nodos a extraer e insertar en el Algoritmo 9 (Extracción e inserción de nodos) y la cantidad de nodos a considerar para realizar el Crossing de componentes en el Algoritmo 11 y $|V|$ la cantidad de nodos de la instancia a tratar.

$p = 0{,}3 \times |T|$, siendo p la cantidad de nodos a retirar de la solución para volverlos a reconectar en la mejor posición en el algoritmo de Shaking (Algoritmo 14), y $|T|$ la cantidad de nodos terminales de la instancia a tratar.

El largo máximo del camino principal en los caminos con colgantes $MAX\_PATH\_LENGTH = \{4, 5, 6, 7\}$ para el Algoritmo 12 con instancias de $|V| = \{26, 51, 76, 101\}$ respectivamente.

Estos parámetros fueron inicializados de este modo luego de algunas pruebas con las instancias Clase A y Clase B más pequeñas, tomando como referencia los mejores valores obtenidos para los óptimos y los tiempos de ejecución insumidos.

6.4. Resultados

6.4.1. Comparación contra resultados del CmRSP

En el Cuadro 6.1 se describen los resultados de las soluciones para las instancias de Baldacci de Clase A. Las notaciones correspondientes a cada una de las columnas son las siguientes:

|T| corresponde a la cantidad de nodos terminales en la instancia especificada, Q es la restricción de capacidad utilizada, CN es la cantidad de nodos presentes en las estructuras 2-nodo-conexas de las componentes, PN es la cantidad de nodos colgantes en la solución, STN es la cantidad de nodos de Steiner que se utilizaron en dicha solución, Z_{best} es el valor del objetivo encontrado, $\bar{Z}$ es el valor de objetivo obtenido en el trabajo de Baldacci, y gap es la

diferencia porcentual con nuestra solución calculada de la siguiente forma:

$$gap = \frac{Z_{best} - \bar{Z}}{\bar{Z}}$$

Finalmente la columna t expresa el tiempo máximo de proceso de la instancia en segundos. Para las instancias en las cuales el óptimo global había a sido alcanzado en el problema original, se puso una cota de 7200 segundos de tiemp máximo de ejecución.

Dicho cuadro 6.1 expresa las mejores soluciones Z_{best} encontradas para el problema CmTNSSP. En negrita aparecen destacados los valores donde se mejora la solución encontrada.

La Figura 6.3 muestra el aspecto de la solución encontrada para las instancia **A26-m04-n076**.

Análogamente en el Cuadro 6.2 se observan los mejores valores obtenidos por nuestro algoritmo para las instancias de Baldacci Clase B.

INSTANCIA	*|T|*	*Q*	*CN*	*PN*	*STN*	Z_{best}	$\bar{Z}$	*gap %*	*t(s)*
A01-n026-m03	12	5	12	0	1	242	242	0,000	1.61
A02-n026-m04	12	4	12	0	1	261	261	0,000	0.97
A03-n026-m05	12	3	12	0	1	292	292	0,000	13.77
A03-n026-m05	12	3	12	0	0	292	292	0,000	4.54
A04-n026-m03	18	7	18	0	0	301	301	0,000	34.29
A05-n026-m04	18	5	18	0	0	339	339	0,000	62.58
A05-n026-m04	18	5	18	0	1	339	339	0,000	9.34
A06-n026-m05	18	4	18	0	0	375	375	0,000	2.67
A07-n026-m03	25	10	24	1	0	325	325	0,000	14.06
A08-n026-m04	25	7	25	0	0	362	362	0,000	3.99
A10-n051-m03	12	5	12	0	0	242	242	0,000	20.09
A11-n051-m04	12	4	12	0	3	261	261	0,000	6.42
A12-n051-m05	12	3	11	1	2	286	286	0,000	37.69
A13-n051-m03	25	10	22	3	3	322	322	0,000	130.85
A14-n051-m04	25	7	24	1	1	360	360	0,000	49.75
A15-n051-m05	25	6	23	2	2	379	379	0,000	117.67
A16-n051-m03	37	14	33	4	1	373	373	0,000	296.60
A17-n051-m04	37	11	33	4	1	405	405	0,000	80.49
A18-n051-m05	37	9	33	4	1	432	432	0,000	2720.60
A19-n051-m03	50	19	45	5	0	458	458	0,000	1674.86
A20-n051-m04	50	14	48	2	0	490	490	0,000	3429.11
A21-n051-m05	50	12	43	7	0	520	520	0,000	6338.64
A22-n076-m03	18	7	17	1	5	330	330	0,000	36.13
A23-n076-m04	18	5	15	3	7	385	385	0,000	112.97
A24-n076-m05	18	4	17	1	4	448	448	0,000	109.91
A25-n076-m03	37	14	35	2	2	403	402	0,249	3624.35
A26-n076-m04	**37**	**11**	**36**	**1**	**3**	**456**	**460**	**-0,870**	**7200.00**
A27-n076-m05	37	9	36	1	4	483	479	0,835	7200.00
A28-n076-m03	56	21	48	8	1	474	471	0,637	7200.00
A29-n076-m04	**56**	**16**	**49**	**7**	**1**	**519**	**523**	**-0,765**	**7200,00**
A30-n076-m05	56	13	50	6	2	547	545	0,367	7200.00
A31-n076-m03	75	28	71	4	0	571	564	1,241	7200.00
A32-n076-m04	75	21	73	2	0	617	606	1,815	7200.00
A33-n076-m05	**75**	**17**	**68**	**7**	**0**	**651**	**654**	**-0,459**	**7200.00**
A34-n101-m03	25	10	21	4	7	363	363	0,000	199.27
A35-n101-m04	25	7	21	4	9	415	415	0,000	1023.84
A36-n101-m05	25	6	22	3	9	448	448	0,000	1264.62
A37-n101-m03	50	19	46	4	8	500	500	0,000	4020.65
A38-n101-m04	50	14	47	3	6	538	532	1,128	7200.00
A39-n101-m05	50	12	46	4	5	573	568	0,880	7200.00
A40-n101-m03	75	28	69	6	5	613	595	3,025	7200.00
A41-n101-m04	75	21	73	2	1	651	625	4,160	7200.00
A42-n101-m04	75	17	70	5	2	677	662	2,266	7200.00
A43-n101-m03	100	38	84	16	0	662	646	2,477	7200.00
A44-n101-m04	100	28	87	13	0	680	680	0,000	7200.00
A45-n101-m05	100	23	84	16	0	713	700	1,857	7200.00

Cuadro 6.1: Mejores valores encontrados para instancias de Baldacci Clase A

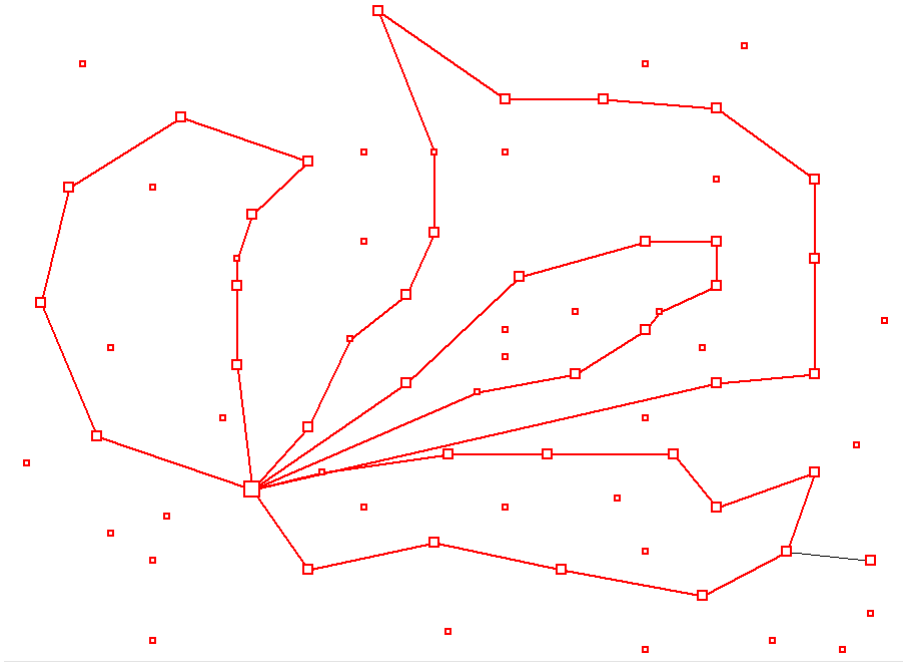

Figura 6.3: Topología de la solución obtenida para la instancia **A26-n076-m04.**

En la Figura 6.4 y 6.5 podemos observar la topología para **A29-n076-m04** y **A33-n076-m05** respectivamente, otras de las instancias en las cuales se obtuvieron mejores resultados.

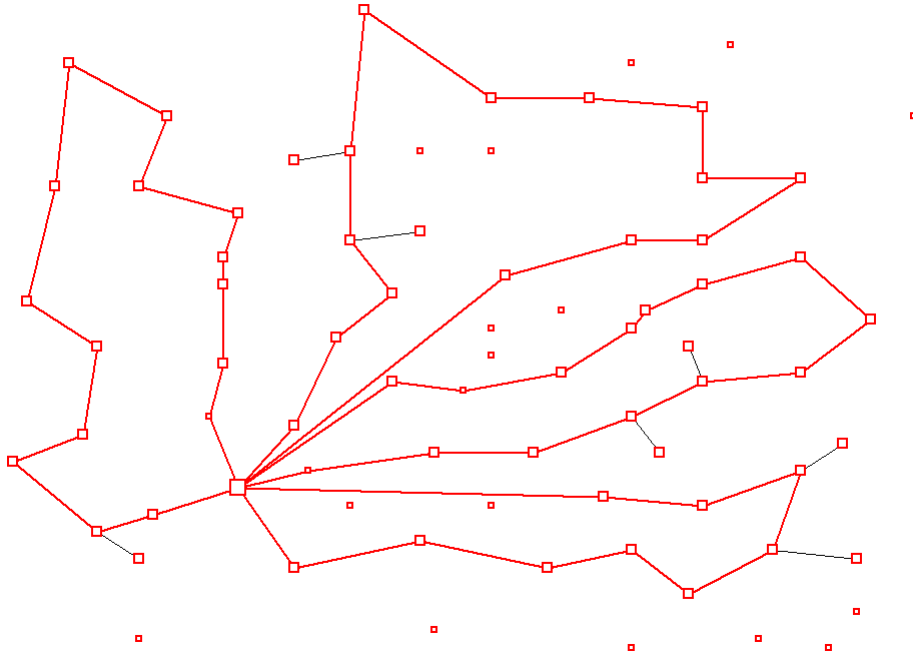

Figura 6.4: Topología de la solución obtenida para la instancia **A29-n076-m04.**

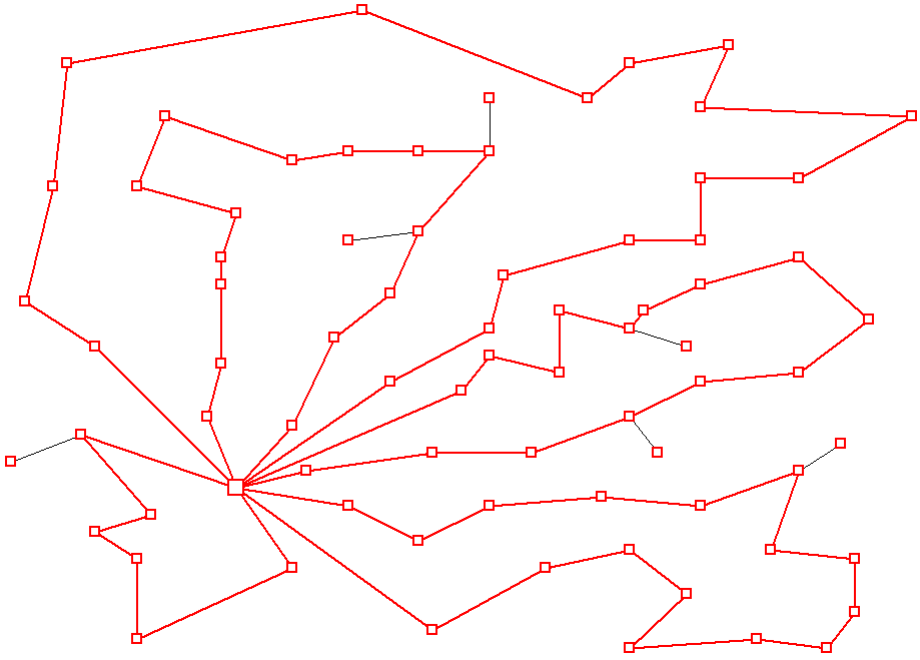

Figura 6.5: Topología de la solución obtenida para la instancia **A33-n076-m05.**

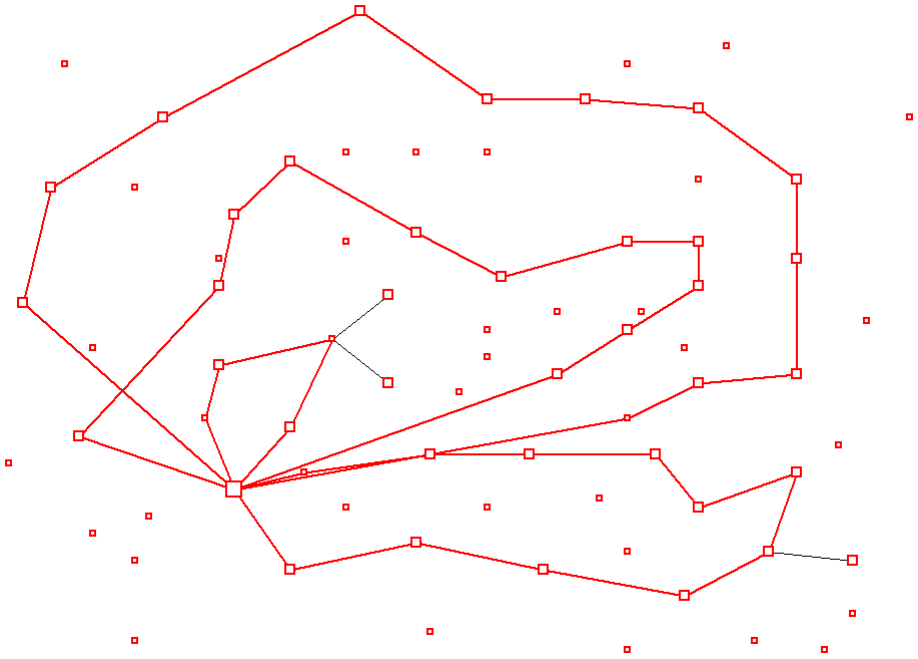

Figura 6.6: Topología de la solución obtenida para la instancia **B26-n076-m04**.

INSTANCIA	*|T|*	*Q*	*CN*	*PN*	*STN*	Z_{best}	$\bar{Z}$	*gap %*	*t(s)*
B01-n026-m03	12	5	11	1	1	1684	1684	0,000	3.09
B02-n026-m04	12	4	12	0	1	1827	1827	0,000	1.09
B03-n026-m05	12	3	11	1	2	2041	2041	0,000	10.68
B04-n026-m03	18	7	17	1	1	2104	2104	0,000	24.90
B05-n026-m04	18	5	17	1	1	2370	2370	0,000	78.21
B06-n026-m05	18	4	17	1	2	2615	2615	0,000	47.01
B07-n026-m03	25	10	24	1	0	2251	2251	0,000	35.13
B08-n026-m04	25	7	24	1	0	2510	2510	0,000	51.65
B09-n026-m05	25	6	25	0	0	2674	2674	0,000	150.31
B10-n051-m03	12	5	10	2	2	1681	1681	0,000	2035.19
B11-n051-m04	12	4	10	2	3	1821	1821	0,000	49.26
B12-n051-m05	12	3	10	2	2	1975	1972	0,152	930.42
B13-n051-m03	25	10	21	4	3	2176	2176	0,000	1724.28
B14-n051-m04	25	7	22	3	3	2470	2470	0,000	626.97
B15-n051-m05	25	6	21	4	4	2579	2579	0,000	92.66
B16-n051-m03	37	14	29	8	2	2490	2490	0,000	3699.45
B17-n051-m04	37	11	29	8	2	2735	2721	0,515	3605.47
B18-n051-m05	37	9	32	5	2	2908	2908	0,000	197.51
B19-n051-m03	50	19	39	11	0	3015	3015	0,000	871.33
B20-n051-m04	50	14	39	11	0	3267	3260	0,215	7200,00
B21-n051-m05	50	12	38	12	0	3404	3404	0,000	3773.22
B22-n076-m03	18	7	15	3	4	2253	2253	0,000	186.10
B23-n076-m04	18	5	13	5	8	2620	2620	0,000	90.78
B24-n076-m05	18	4	15	3	9	3155	3059	3,138	7200,00
B25-n076-m03	37	14	32	5	6	2731	2720	0,404	7200,00
B26-n076-m04	**37**	**11**	**34**	**3**	**4**	**3134**	**3138**	**-0,127**	**28825.80**
B27-n076-m05	37	9	36	1	3	3329	3311	0,544	7217.19
B28-n076-m03	**56**	**21**	**40**	**16**	**4**	**3044**	**3088**	**-1,425**	**28815.84**
B29-n076-m04	**56**	**16**	**44**	**12**	**2**	**3439**	**3447**	**-0,232**	**14418.05**
B30-n076-m05	**56**	**13**	**44**	**12**	**2**	**3635**	**3648**	**-0,356**	**3797.03**
B31-n076-m03	**75**	**28**	**55**	**20**	**0**	**3724**	**3740**	**-0,428**	**2112.23**
B32-n076-m04	75	21	57	18	0	4096	4026	1,739	7200,00
B33-n076-m05	75	17	58	17	0	4489	4288	4,688	7200,00
B35-n101-m04	25	7	19	6	9	2795	2782	0,467	7200,00
B35-n101-m04	25	7	19	6	6	2795	2782	0,467	7200,00
B36-n101-m05	25	6	18	7	4	3009	3009	0,000	597.71
B37-n101-m03	**50**	**19**	**40**	**10**	**8**	**3331**	**3332**	**-0,030**	**7200,00**
B38-n101-m04	50	14	38	12	8	3560	3.533	0,764	7200,00
B39-n101-m05	50	12	41	9	8	3873	3872	0,026	7200,00
B40-n101-m03	75	28	68	7	5	3931	3.923	0,204	7200,00
B41-n101-m04	75	21	68	7	6	4332	4.125	5,018	7200,00
B42-n101-m05	75	17	69	6	6	4494	4.458	0,808	7200,00
B43-n101-m03	100	38	96	4	0	4403	4110	7,129	7200,00
B44-n101-m04	100	28	95	5	0	4526	4506	0,444	7200,00
B45-n101-m05	100	23	96	4	0	4639	4632	0,151	7200,00

Cuadro 6.2: Mejores valores encontrados para las instancias de Baldacci Clase B

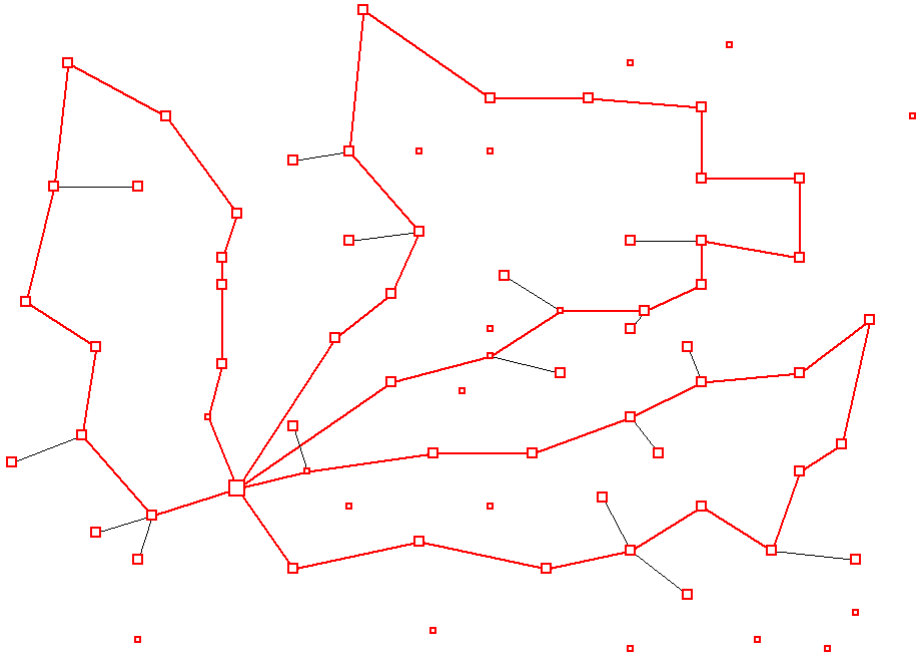

Figura 6.7: Topología de la solución obtenida para la instancia **B28-n076-m03.**

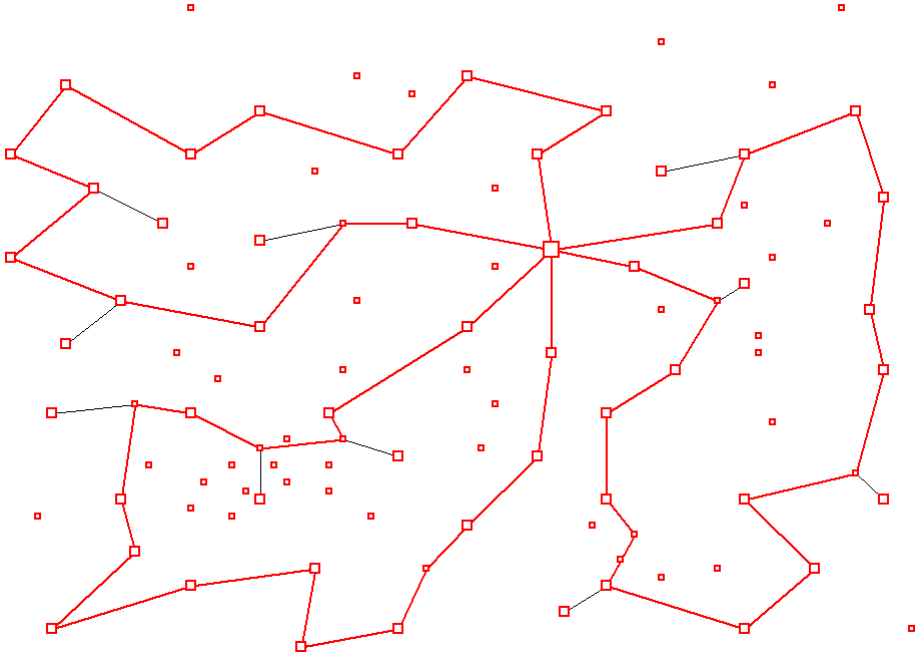

Figura 6.11: Topología de la solución obtenida para la instancia **B37-n101-m03.**

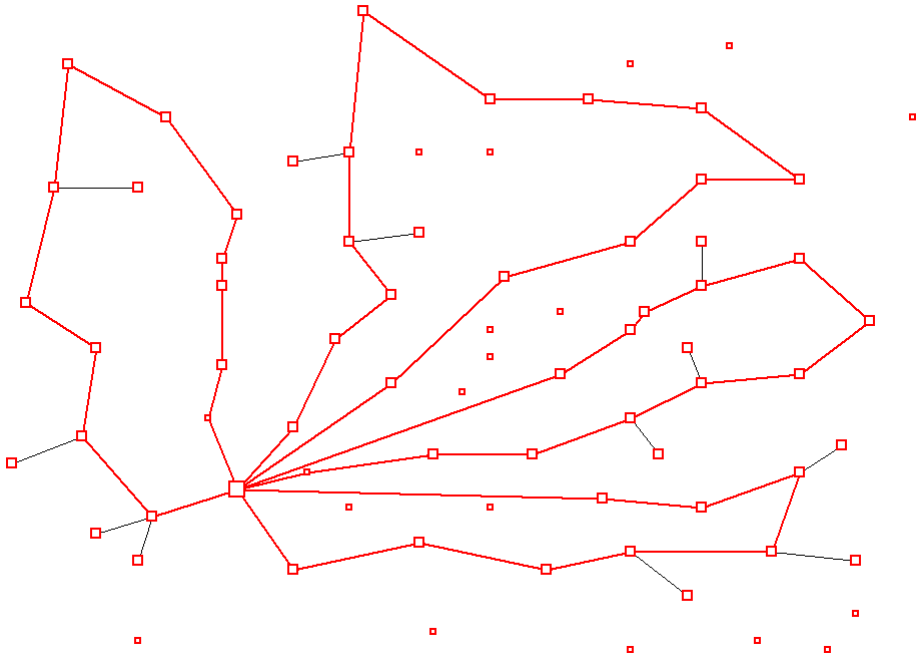

Figura 6.8: Topología de la solución obtenida para la instancia **B29-n076-m04.**

Hemos detallado hasta aquí los Cuadros con los mejores valores hallados para las instancias Clase A y Clase B, y representamos gráficamente las topologías donde se obtuvieron mejoras con respecto a los valores obtenidos para la solución del CmRSP con las mismas instancias.

6.4.2. Validación

Para realizar una validación del algoritmo GRASP-VND diseñado y verificar la robustez del mismo hemos ejecutado las siguientes pruebas.

Realizamos la ejecución de cada una de las 90 instancias de Baldacci, 10 veces de manera independiente, cambiando solamente la semilla del generador de números aleatorios y sin modificar los parámetros de inicialización.

Para cada una de dichas instancias obtuvimos 10 valores del objetivo sobre los cuales calculamos el gap y promediamos esos 10 valores de gaps obtenidos. En el Cuadro 6.3 se muestra el mejor valor del objetivo encontrado, el peor valor encontrado y el gap promedio. El valor final del Cuadro (Gap promedio de las instancias Clase A) se calcula haciendo el promedio de cada uno de los gaps según lo especificado en el párrafo anterior. Idem para las instancias Clase B en el Cuadro 6.4.

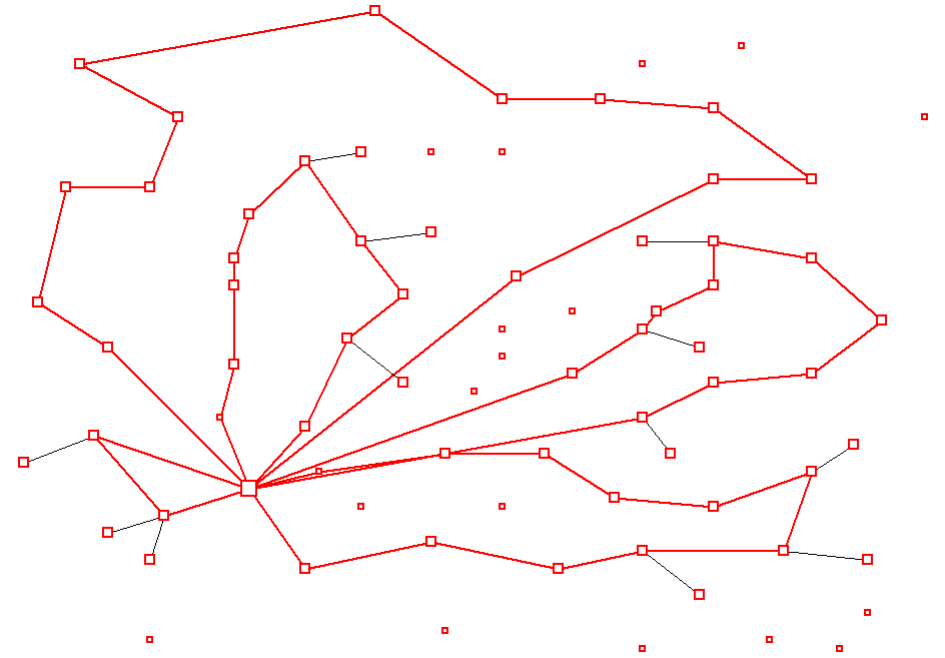

Figura 6.9: Topología de la solución obtenida para la instancia **B30-n076-m05.**

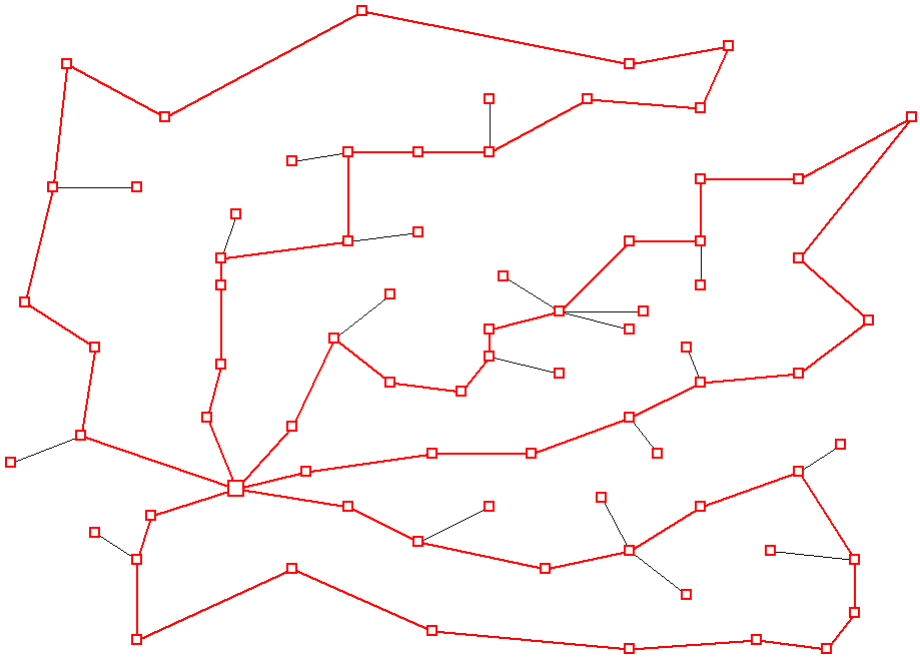

Figura 6.10: Topología de la solución obtenida para la instancia **B31-n076-m03.**

INSTANCIA	*|T|*	*Q*	Z_{best}	Z_{worst}	$\bar{Z}$	*gap %*
A01-n026-m03	12	5	242	242	242	0,000
A02-n026-m04	12	4	261	261	261	0,000
A03-n026-m05	12	3	292	292	292	0,000
A03-n026-m05	12	3	292	292	292	0,000
A04-n026-m03	18	7	301	301	301	0,000
A05-n026-m04	18	5	339	339	339	0,000
A05-n026-m04	18	5	339	339	339	0,000
A06-n026-m05	18	4	375	375	375	0,000
A07-n026-m03	25	10	325	325	325	0,000
A08-n026-m04	25	7	362	363	362	0,055
A09-n026-m05	25	6	382	385	382	0,157
A10-n051-m03	12	5	242	242	242	0,000
A11-n051-m04	12	4	261	261	261	0,000
A12-n051-m05	12	3	286	286	286	0,000
A13-n051-m03	25	10	322	322	322	0,000
A14-n051-m04	25	7	360	360	360	0,000
A15-n051-m05	25	6	379	379	379	0,000
A16-n051-m03	37	14	373	373	373	0,000
A17-n051-m04	37	11	405	405	405	0,000
A18-n051-m05	37	9	432	438	432	0,556
A19-n051-m03	50	19	458	465	458	0,611
A20-n051-m04	50	14	490	497	490	0,367
A21-n051-m05	50	12	520	522	520	0,192
A22-n076-m03	18	7	330	336	330	0,545
A23-n076-m04	18	5	385	385	385	0,000
A24-n076-m05	18	4	448	448	448	0,000
A25-n076-m03	37	14	403	403	402	0,249
A26-n076-m04	**37**	**11**	456	460	**460**	-0,391
A27-n076-m05	37	9	483	483	479	0,835
A28-n076-m03	56	21	474	474	471	0,637
A29-n076-m04	**56**	**16**	520	523	**523**	-0,268
A30-n076-m05	56	13	547	563	545	1,835
A31-n076-m03	75	28	571	574	564	1,489
A32-n076-m04	75	21	611	619	606	1,287
A33-n076-m05	75	17	651	665	654	0,459
A34-n101-m03	25	10	363	384	363	1,983
A35-n101-m04	25	7	415	420	415	0,386
A36-n101-m05	25	6	448	455	448	1,161
A37-n101-m03	50	19	500	505	500	0,520
A38-n101-m04	50	14	538	553	532	2,444
A39-n101-m05	50	12	573	582	568	1,690
A40-n101-m03	75	28	613	623	595	3,597
A41-n101-m04	75	21	651	660	625	4,896
A42-n101-m05	75	17	677	683	662	2,779
A43-n101-m03	100	38	662	670	646	2,910
A44-n101-m04	100	28	680	694	680	0,647
A45-n101-m05	100	23	713	735	700	3,200
GAP PROMEDIO						***0,74104***

Cuadro 6.3: Muestra de los gaps promedio para las instancias Clase A.

En los cuadros 6.3 y 6.4 podemos ver los valores correspondientes a los gaps promedio y el gap general promedio para las instancias Clase A y Clase B respectivamente.

INSTANCIA	***|T|***	***Q***	Z_{best}	Z_{worst}	$\bar{Z}$	***gap %***
B01-n026-m03	12	5	1.684	1.684	1.684	0,000
B02-n026-m04	12	4	1.827	1.827	1.827	0,000
B03-n026-m05	12	3	2.041	2.041	2.041	0,000
B04-n026-m03	18	7	2.104	2.104	2.104	0,000
B05-n026-m04	18	5	2.370	2.370	2.370	0,000
B06-n026-m05	18	4	2.615	2.615	2.615	0,000
B07-n026-m03	25	10	2.251	2.251	2.251	0,000
B08-n026-m04	25	7	2.510	2.519	2.510	0,072
B09-n026-m05	25	6	2.674	2.690	2.674	0,120
B10-n051-m03	12	5	1.681	1.681	1.681	0,000
B11-n051-m04	12	4	1.821	1.821	1.821	0,000
B12-n051-m05	12	3	1.975	1.978	1.972	0,213
B13-n051-m03	25	10	2.176	2.176	2.176	0,000
B14-n051-m04	25	7	2.470	2.473	2.470	0,024
B15-n051-m05	25	6	2.579	2.610	2.579	0,644
B16-n051-m03	37	14	2.490	2.510	2.490	0,321
B17-n051-m04	37	11	2.735	2.768	2.721	0,941
B18-n051-m05	37	9	2.908	3.004	2.908	0,791
B19-n051-m03	50	19	3.015	3.039	3.015	0,438
B20-n051-m04	50	14	3.267	3.309	3.260	0,748
B21-n051-m05	50	12	3.404	3.466	3.404	0,635
B22-n076-m03	18	7	2.253	2.276	2.253	0,231
B23-n076-m04	18	5	2.620	2.722	2.620	1,557
B24-n076-m05	18	4	3.134	3.155	3.059	2,726
B25-n076-m03	37	14	2.731	2.782	2.720	0,978
B26-n076-m04	37	11	3.134	3.162	3.138	0,127
B27-n076-m05	37	9	3.315	3.329	3.311	0,375
B28-n076-m03	**56**	**21**	**3.044**	**3.088**	**3.088**	**-0,907**
B29-n076-m04	56	16	3.439	3.466	3.447	0,035
B30-n076-m05	56	13	3.635	3.711	3.648	0,773
B31-n076-m03	75	28	3.724	3.768	3.740	0,043
B32-n076-m04	75	21	4.096	4.107	4.026	1,878
B33-n076-m05	75	17	4.489	4.602	4.288	5,215
B34-n101-m03	25	7	2.435	2.460	2.434	0,723
B35-n101-m04	25	7	2.795	2.820	2.782	0,798
B36-n101-m05	25	6	3.009	3.024	3.009	0,199
B37-n101-m03	50	19	3.331	3.350	3.332	0,180
B38-n101-m04	50	14	3.560	3.660	3.533	2,338
B39-n101-m05	50	12	3.873	3.912	3872	0,589
B40-n101-m03	75	21	3.931	3.998	3.923	0,892
B41-n101-m04	75	21	4.332	4.374	4.125	5,343
B42-n101-m05	75	17	4.494	4.613	4.458	2,194
B43-n101-m03	100	38	4.403	4.435	4110	7,372
B44-n101-m04	100	28	4.526	4.606	4506	1,190
B45-n101-m05	100	23	4.639	4.658	4632	0,289
GAP PROMEDIO						***0,89076***

Cuadro 6.4: Muestra de los gaps promedio para las instancias Clase B.

6.4.3. Verificación para la instancia resuelta de forma exacta

Dentro de las pruebas realizadas hemos ejecutado el algoritmo GRASP-VND para la instancia $nut30$ que resolvimos de forma exacta en la sección 6.2.

El algoritmo se ejecutó con la siguiente configuración de parámetros:

$ListSize = 4$, tamaño de la RCL del Algoritmo de construcción de Soluciones Factibles de GRASP para el CmTNSSP (Algoritmo 6).

$k = 7$, siendo k el parámetro del Algoritmo 10 (Swapping de nodos), Algoritmo 9 (Extracción e inserción de nodos) y Algoritmo 11 (Crossing de Componentes).

$p = 9$, siendo p la cantidad de nodos a retirar en el Algoritmo de Shaking (Algoritmo 14.

El largo máximo del camino principal en los caminos con colgantes MAX_PATH_LENGTH=5 (Algoritmo 12).

Para el caso de prueba así definido obtuvimos el óptimo global del CmTNSSP, el cual coincide con el calculado de forma exacta en la sección 6.2 coincidiendo también las topologías de redes encontradas para ambos casos (ver Figura 6.2). El tiempo insumido en encontrar el óptimo global para esta instancia fue de 3,5 segundos, el cual contrasta sin duda con el tiempo utilizado en su resolución exacta.

6.4.4. Relajaciones

Otra de las pruebas realizadas para este trabajo consiste en relajar el problema quitando la cota d_{max} definida en la igualdad 6.2. En realidad y tal como fue explicado en la subsección 6.3.1 esta restricción la adoptamos para hacer posible la comparación de nuestros resultados con los correspondientes al trabajo de Bladacci et al. y obtener un valor comparativo de buena calidad, pero no forma parte de la definición que hemos hecho del problema CmTNSSP. Esta restricción desde el punto de vista técnico y aplicado a casos reales puede ser levantable.

Hemos probado también los algoritmos sin esta restricción, obteniendo muy buenos resultados y que superan cualitativamente en casi todos los casos (topologías de menor costo) a los óptimos globales del problema CmRSP para las instancias anteriormente mencionadas.

En los Cuadros 6.5 y 6.6 podemos observar los resultados y en las Figuras 6.12 a la 6.16 observamos las topologías solución de las instancias, las cuales obviamente presentan más nodos colgantes, dado que es posible conectar cualquiera de ellos porque no existe una restricción de distancia máxima que impida que el nodo sea colgante.

INSTANCIA	*|T|*	*Q*	*CN*	*PN*	*STN*	Z_{best}	$\bar{Z}$	*gap %*	*t(s)*
A01-n026-m03	12	5	7	5	0	214	242	-11,570	4.28
A02-n026-m04	12	4	6	6	0	234	261	-10,345	2.40
A03-n026-m05	12	3	5	7	0	261	292	-10,616	0.79
A04-n026-m03	18	7	12	6	0	286	301	-4,983	5.68
A05-n026-m04	18	5	14	4	0	326	339	-3,835	3.17
A06-n026-m05	18	4	8	10	0	358	375	-4,533	2.48
A07-n026-m03	25	10	20	5	0	313	325	-3,692	1.69
A08-n026-m04	25	7	15	10	0	347	362	-4,144	5.23
A09-n026-m05	25	6	16	9	0	380	382	-0,524	28.19
A10-n051-m03	12	5	8	4	1	219	242	-9,504	3.82
A11-n051-m04	12	4	7	5	2	233	261	-10,728	3.88
A12-n051-m05	12	3	5	7	2	259	286	-9,441	4.25
A13-n051-m03	25	10	18	7	4	322	322	0,000	36.45
A14-n051-m04	25	7	15	10	3	344	360	-4,444	47.55
A15-n051-m05	25	6	17	8	1	369	379	-2,639	24.41
A16-n051-m03	37	14	29	8	1	369	373	-1,072	443.31
A17-n051-m04	37	11	30	7	0	405	405	0,000	289.21
A18-n051-m05	37	9	30	7	0	418	432	-3,241	63.27
A19-n051-m03	50	19	41	9	0	452	458	-1,310	132.08
A20-n051-m04	50	14	40	10	0	484	490	-1,224	120.98
A21-n051-m05	50	12	39	11	0	520	520	0,000	52.48
A22-n076-m03	18	7	15	3	3	321	330	-2,727	34.03
A23-n076-m04	18	5	14	4	1	372	385	-3,377	18.36
A24-n076-m05	18	4	11	7	4	439	448	-2,009	26.64
A25-n076-m03	37	14	32	5	1	398	402	-0,995	152.89
A26-n076-m04	37	11	27	10	4	441	460	-4,130	28801.73
A27-n076-m05	37	9	32	5	2	478	479	-0,209	213.35
A28-n076-m03	56	21	45	11	0	470	471	-0,212	194.82
A29-n076-m04	56	16	47	9	1	510	523	-2,486	1215.30
A30-n076-m05	56	13	45	11	1	541	545	-0,734	4202.70
A31-n076-m03	75	28	60	15	0	545	564	-3,369	448.44
A32-n076-m04	75	21	66	9	0	603	606	-0,495	1026.81
A33-n076-m05	75	17	62	13	0	627	627	0,000	19978.43
A34-n101-m03	25	10	20	5	3	363	363	0,000	397.04
A35-n101-m04	25	7	19	6	1	415	415	0,000	40.46
A36-n101-m05	25	6	19	6	5	436	448	-2,679	96.27
A37-n101-m03	50	19	38	12	1	499	500	-0,200	341.71
A38-n101-m04	50	14	39	11	2	524	532	-1,504	2933.34
A39-n101-m05	50	12	37	13	2	560	568	-1,408	200.75
A40-n101-m03	75	28	58	17	0	589	595	-1,008	929.00
A41-n101-m04	75	21	61	14	1	621	625	-0,640	1545.53
A42-n101-m05	75	17	57	18	0	649	662	-1,964	1420.18
A43-n101-m03	100	38	77	23	0	631	646	-2,322	7389.56
A44-n101-m04	100	28	76	24	0	683	680	0,441	1420.18
A45-n101-m05	100	23	73	27	0	701	700	0,143	1250.15

Cuadro 6.5: Mejores valores encontrados para instancias de Baldacci Clase A relajadas

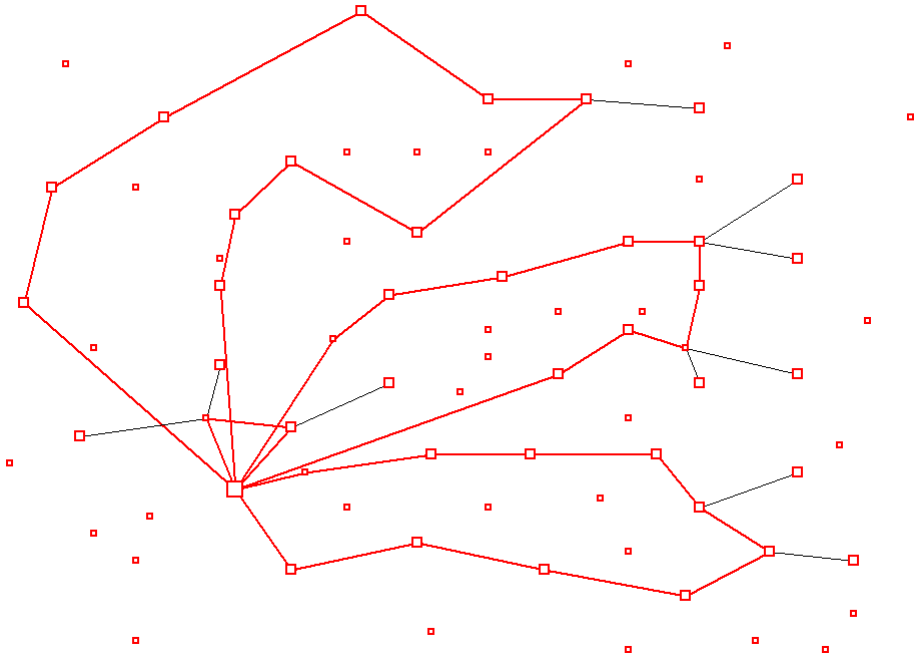

Figura 6.12: Topología de la solución obtenida para la instancia **A26-n076-m04** relajada.

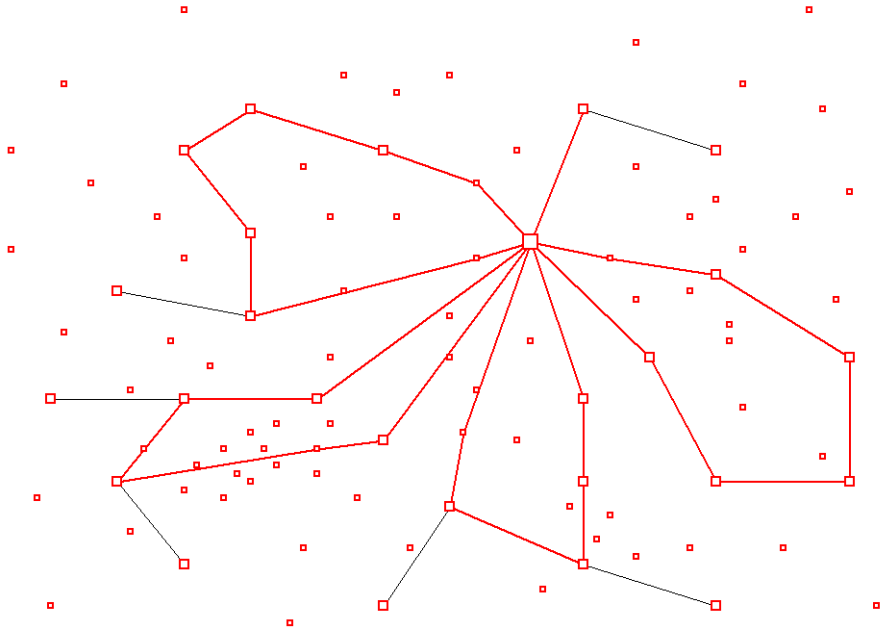

Figura 6.13: Topología de la solución obtenida para la instancia **A36-n076-m05** relajada.

En la Figura 6.13 podemos apreciar que una de las 5 componentes se encuentra entre los casos de componentes que cuentan como estructura 2-nodo-conexa una arista doble que une el *depot* con un nodo terminal.

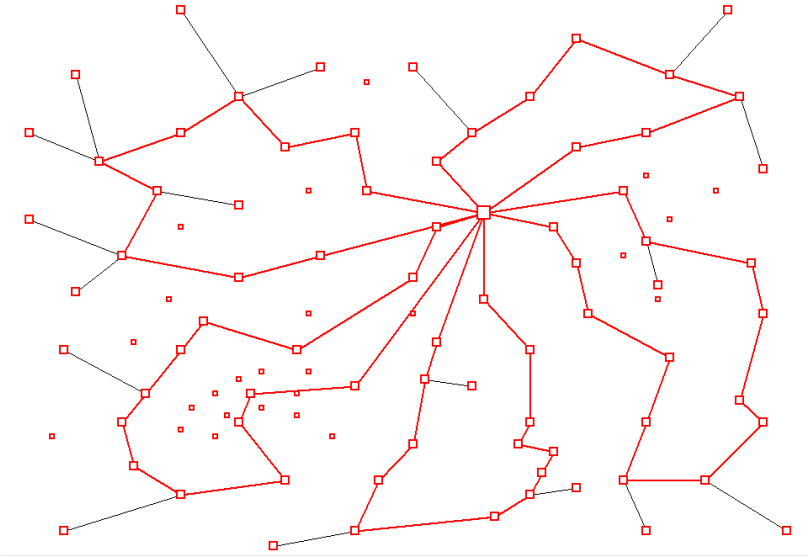

Figura 6.14: Topología de la solución obtenida para la instancia **A42-n101-m05** relajada.

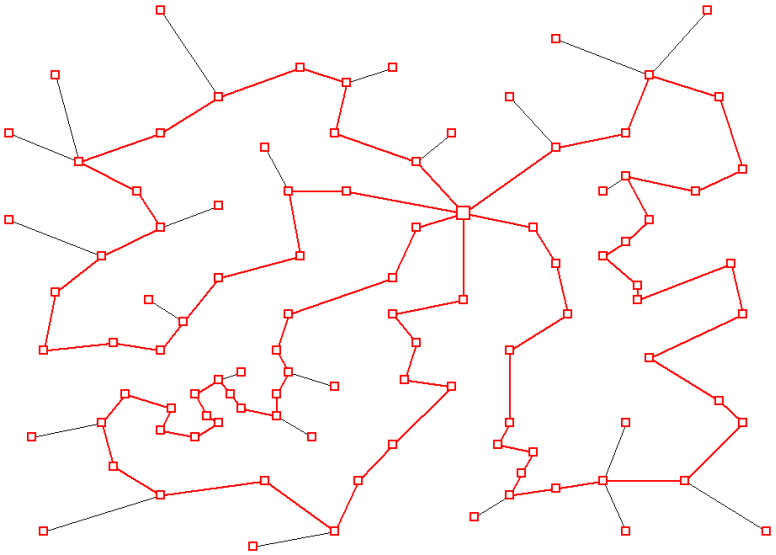

Figura 6.15: Topología de la solución obtenida para la instancia **A43-n101-m03** relajada.

A continuación en el Cuadro 6.6 podemos ver los valores obtenidos de las ejecuciones realizadas para las instancias de Baldacci Clase B sin la restricción especificada en 6.2.

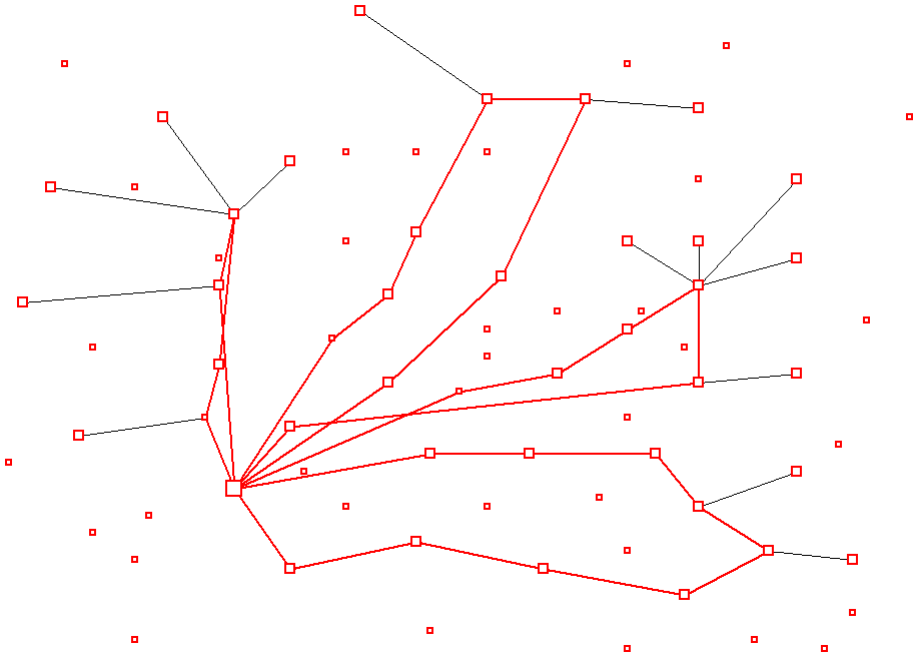

Figura 6.16: Topología de la solución obtenida para la instancia **B26-n076-m04** relajada.

INSTANCIA	*|T|*	*Q*	*CN*	*PN*	*STN*	Z_{best}	$\bar{Z}$	*gap %*	*t(s)*
B01-n026-m03	12	5	5	7	0	1.148	1.684	-31,829	5.63
B02-n026-m04	12	4	5	7	0	1.252	1.827	-31,472	1.83
B03-n026-m05	12	3	5	7	0	1.494	2.041	-26,801	1.10
B04-n026-m03	18	7	7	11	1	1.507	2.104	-28,375	14.79
B05-n026-m04	18	5	9	9	0	1.685	2.370	-28,903	9.65
B06-n026-m05	18	4	8	10	0	1.947	2.615	-25,545	2.12
B07-n026-m03	25	10	13	12	0	1.733	2.251	-23,012	6.71
B08-n026-m04	25	7	14	11	0	1.957	2.510	-22,032	6.10
B09-n026-m05	25	6	12	13	0	2.234	2.674	-16,455	4.33
B10-n051-m03	12	5	2	10	3	1.149	1.681	-31,648	3.58
B11-n051-m04	12	4	4	8	4	1.586	1.821	-12,905	1.16
B12-n051-m05	12	3	4	8	3	1.457	1.972	-26,116	2.25
B13-n051-m03	25	10	12	13	3	1.779	2.176	-18,244	41.46
B14-n051-m04	25	7	14	11	0	2.175	2.470	-11,943	14.11
B15-n051-m05	25	6	13	12	2	2.237	2.579	-13,261	23.16
B16-n051-m03	37	14	24	13	0	2.165	2.490	-13,052	409.04
B17-n051-m04	37	11	23	14	1	2.647	2.721	-2,720	128.05
B18-n051-m05	37	9	21	16	1	2.463	2.908	-15,303	480.54
B19-n051-m03	50	19	29	21	0	2.731	3.015	-9,420	422.97
B20-n051-m04	50	14	31	19	0	3.062	3.260	-6,074	110.79
B21-n051-m05	50	12	32	18	0	3.128	3.404	-8,108	359.32
B22-n076-m03	18	7	9	9	0	1.901	2.253	-15,624	20.08
B23-n076-m04	18	5	5	13	3	1.993	2.620	-23,931	19.76
B24-n076-m05	18	4	5	13	2	2.383	3.059	-22,099	17.21
B25-n076-m03	37	14	23	14	5	2.591	2.720	-4,743	128.32
B26-n076-m04	37	11	23	14	3	2.815	3.138	-10,293	619.62
B27-n076-m05	37	9	20	17	4	3.118	3.311	-5,829	90.27
B29-n076-m04	56	16	38	18	1	3.367	3.447	-2,321	507.28
B30-n076-m05	56	13	37	19	2	3.576	3.648	-1,974	181.80
B32-n076-m04	75	21	45	30	0	3.594	4.026	-10,730	352.53
B33-n076-m05	75	17	48	27	0	4.134	4.288	-3,591	526.72
B34-n101-m03	25	10	14	11	4	2.197	2.434	-9,737	41.96
B35-n101-m04	25	7	14	11	2	2.530	2.782	-9,058	68.09
B36-n101-m05	25	6	15	10	2	2.564	3.009	-14,789	24.76
B37-n101-m03	50	19	27	23	4	2.968	3.332	-10,924	2887.51
B38-n101-m04	50	14	33	17	4	3.270	3.533	-7,444	1385.59
B39-n101-m05	50	12	26	24	4	3.277	3.872	-15,367	7384.00
B40-n101-m03	75	28	47	28	1	3.682	3.923	-6,143	1512.83
B41-n101-m04	75	21	47	28	1	3.873	4.125	-6,109	2104.75
B42-n101-m05	75	17	45	30	2	4.454	4.458	-0,090	1386.32
B43-n101-m03	100	38	59	41	0	3.804	4.110	-7,445	1376.05
B44-n101-m04	100	28	66	34	0	3.992	4.506	-11,407	1215.84
B45-n101-m05	100	23	60	40	0	4.041	4.632	-12,759	3068.60

Cuadro 6.6: Mejores valores encontrados para instancias de Baldacci Clase B relajadas

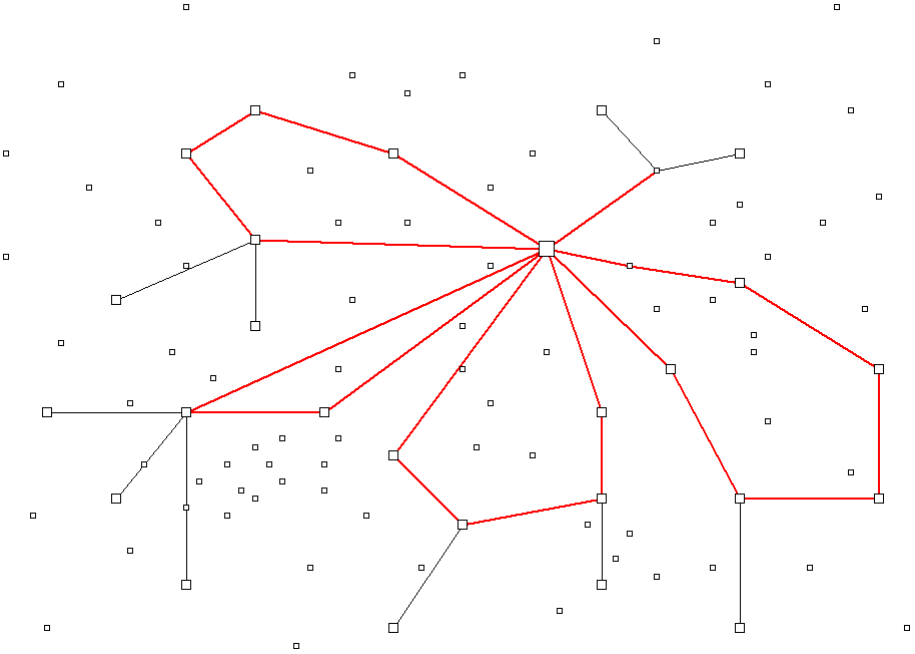

Figura 6.17: Topología de la solución obtenida para la instancia **B36-n101-m05** relajada.

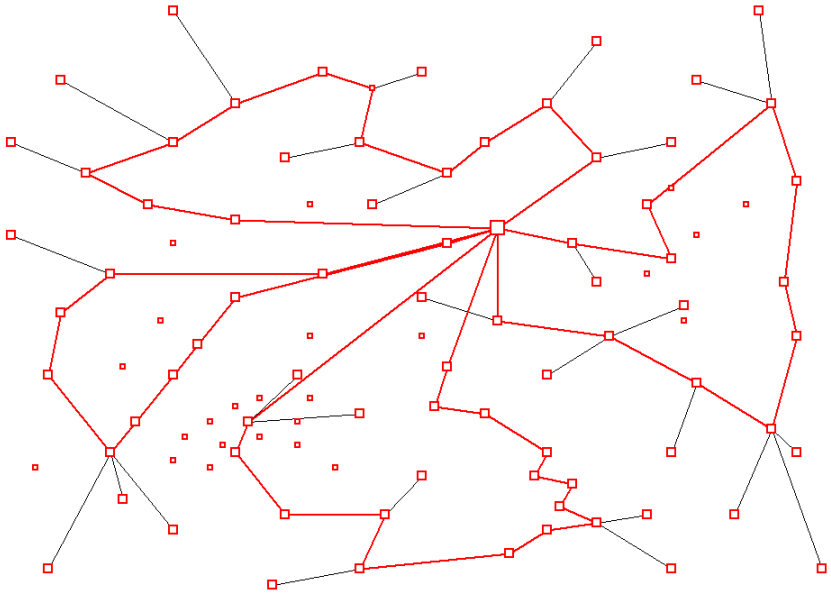

Figura 6.18: Topología de la solución obtenida para la instancia **B41-n101-m04** relajada.

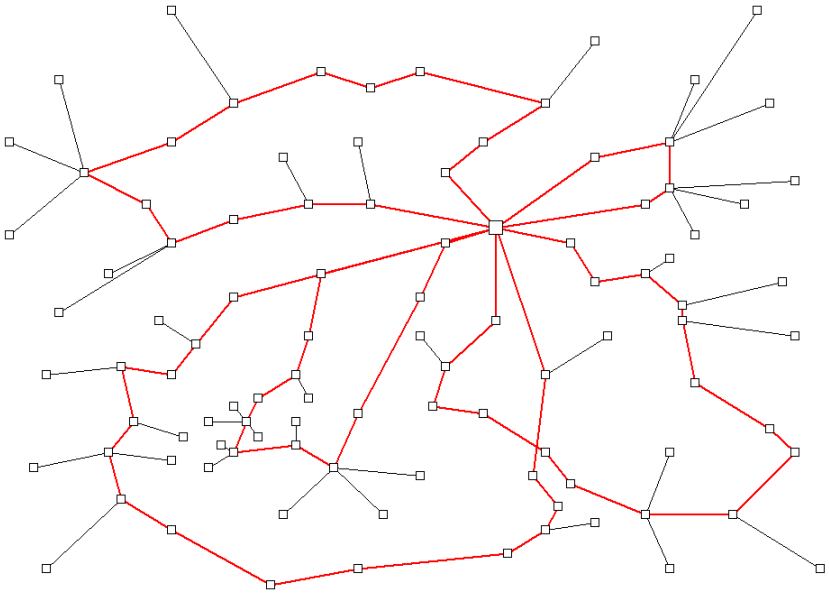

Figura 6.19: Topología de la solución obtenida para la instancia **B45-n101-m05** relajada.

Se han descripto aquí gráficamente algunas de las topologías halladas por la metaheurística GRASP-VND que hemos implementado, obteniéndose buenos resultados en el problema relajado para las instancias con las cuales hemos trabajado.

CLASE INSTANCIA	*ORIGINAL*	*RELAJADA*
Clase A	0,74 %	-3,50 %
Clase B	0,89 %	-8,13 %

Cuadro 6.7: Comparación de los gaps promedio entre instancias normales y relajadas.

La metaheurística GRASP-VND utilizada logró un buen desempeño en el calculo de soluciones relajadas. En el Cuadro 6.7 pueden observarse los gaps promedio obtenidos para cada grupo de instancias, trabajando el problema en su forma original, y en su forma relajada.

6.4.5. Componentes 2-nodo conexas no cíclicas

En los resultados expresados en los Cuadros 6.1 y 6.2 no se han encontrado soluciones 2-nodo-conexas no cíclicas que formen parte de las componentes. Para verificar que el algoritmo encuentra dichas soluciones generamos un caso de prueba adicional.

Sea $G = (V, E)$ el grafo seleccionado. Dicho grafo es completo con $|V| = 36$ y sus nodos, distribuidos de la siguiente forma:

$$d = \{0\}$$

$$T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27\}$$

$$W = \{28, 29, 30, 31, 32, 33, 34, 35\}$$

Los vértices se situan en un sistema de coordenadas planas (x, y) con los siguientes valores:

$0(11, 9)$	$13(14, 6)$	$26(28, 6)$
$1(9, 13)$	$14(16, 9)$	$27(30, 9)$
$2(7, 11)$	$15(18, 12)$	$28(7, 9)$
$3(6, 13)$	$16(19, 9)$	$29(9, 4)$
$4(3, 12)$	$17(17, 6)$	$30(3, 10)$
$5(1, 9)$	$18(20, 6)$	$31(16, 5)$
$6(5, 9)$	$19(21, 17)$	$32(21, 10)$
$7(3, 7)$	$20(21, 12)$	$33(22, 14)$
$8(8, 8)$	$21(22, 9)$	$34(25, 5)$
$9(7, 6)$	$22(24, 6)$	$35(28, 9)$
$10(4, 4)$	$23(25, 12)$	
$11(13, 15)$	$24(25, 9)$	
$12(14, 12)$	$25(28, 12)$	

Los costos de las matrices $C = \{c_{ij}\}_{i,j \in V}$ y $D = \{d_{ij}\}_{i,j \in V}$ están ambos definidos por las distancias euclídeas entre los vértices i, j multiplicadas por un factor 10, salvo un conjunto de aristas $E' \subseteq E$ a las cuales se le asignan los siguientes costos:

$c_{0,11} = c_{11,0} = d_{0,11} = d_{11,0} = 1$
$c_{12,15} = c_{15,12} = d_{12,15} = d_{15,12} = 5$
$c_{0,14} = c_{14,0} = d_{0,14} = d_{14,0} = 1$
$c_{16,14} = c_{14,16} = d_{16,14} = d_{14,16} = 1$
$c_{0,13} = c_{13,0} = d_{0,12} = d_{12,0} = 1$
$c_{0,13} = c_{13,0} = d_{0,13} = d_{13,0} = 1$
$c_{0=15,20} = c_{20,15} = d_{15,20} = d_{20,15} = 1$
$c_{22,26} = c_{26,22} = d_{22,26} = d_{26,22} = 1$
$c_{20,23} = c_{23,20} = d_{20,23} = d_{23,20} = 1$
$c_{18,22} = c_{22,18} = d_{18,22} = d_{22,18} = 1$
$c_{14,17} = c_{17,14} = d_{14,17} = d_{17,14} = 80$
$c_{18,17} = c_{17,18} = d_{18,17} = d_{17,18} = 1$
$c_{24,27} = c_{27,24} = d_{24,27} = d_{27,24} = 5$

Los parámetros de construcción son los siguientes:

$$m = 2; \quad Q = 18; \quad ListSize = 4$$

$$k = 7; \quad p = 11; \quad MAX\_PATH\_LENGTH = 4$$

Para los valores especificados anteriormente el algoritmo GRASP-VND encontró una solución factible óptima (local hasta donde sabemos) con una estructura no cíclica en una de sus componentes (Figura 6.20)

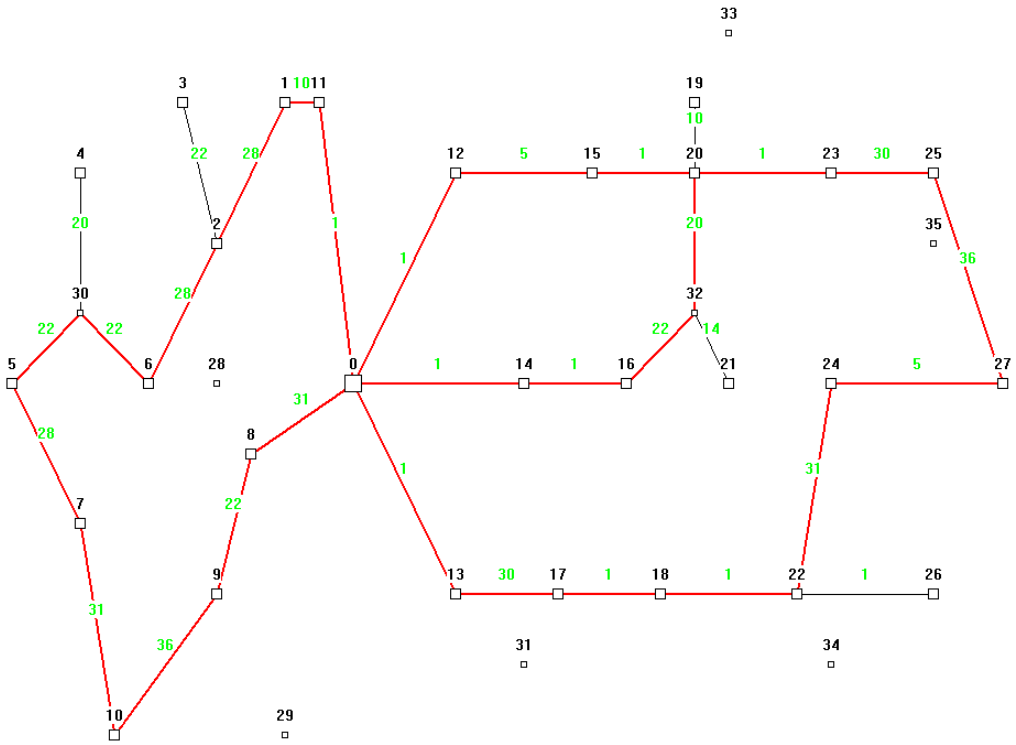

Figura 6.20: Topología de la solución 2-nodo-conexa no cíclica encontrada.

Estos resultados muestran que la metaheurística GRASP-VND diseñada consigue soluciones donde las estructuras 2-nodo-conexas son no cíclicas siempre y cuando los costos sean mejores a otras soluciones con ciclos.

Considerando que para la solución encontrada (Figura 6.20) no contamos con el óptimo global, plantearemos otro caso más sencillo para validar la obtención de dicho óptimo global con

topología de estructuras 2-nodo-conexas no cíclicas en las componentes.

Sea el grafo $K = (V, E)$ ilustrado en la Figura 6.21. con las siguientes definiciones:

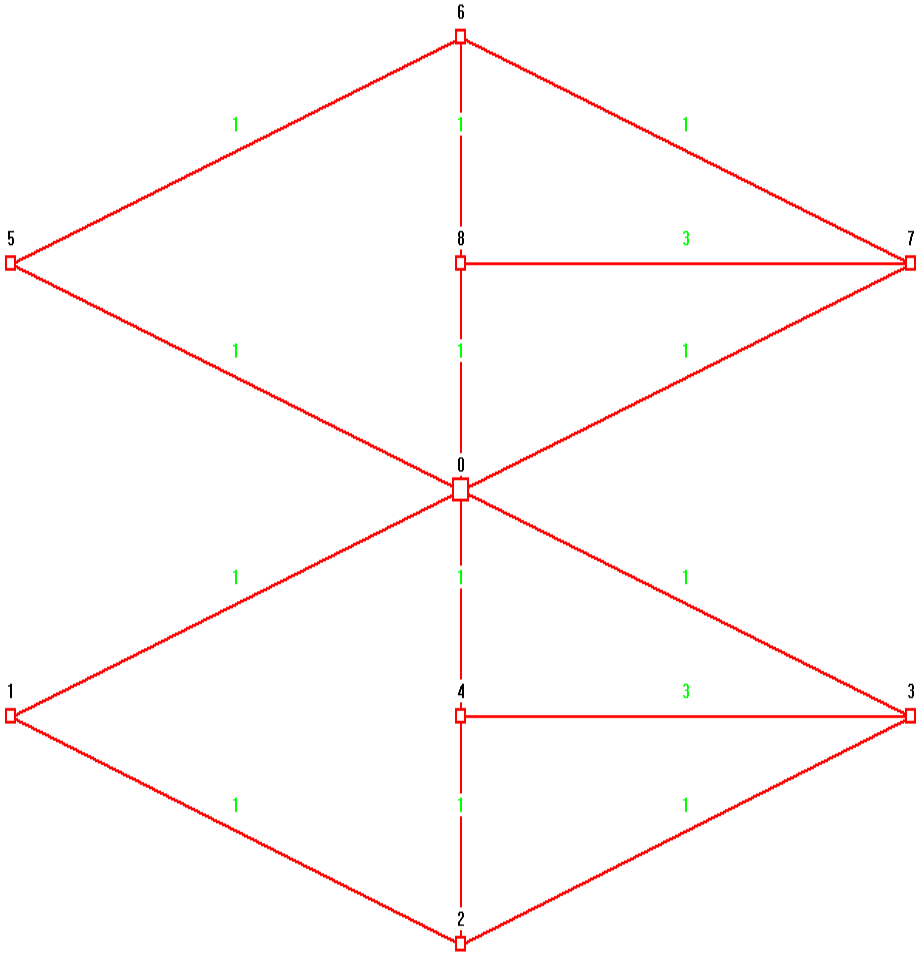

Figura 6.21: Instancia de prueba K utilizada para validación de soluciones con estructuras 2-nodo-conexas no cíclicas.

$$T = \{1, 2, 3, 4, 5, 6, 7, 8\}$$
$$W = \phi$$
$$d = \{0\}$$

Siendo T el conjunto de los nodos terminales del grafo K, W el conjunto de los nodos opcionales o de Steiner y d el nodo $depot$.

La matrices C y D que expresan los costos de ruteo y conexión del grafo K son las siguientes:

$$C = \begin{pmatrix} 0 & 1 & - & 1 & 1 & 1 & - & 1 & 1 \\ 1 & 0 & 1 & - & - & - & - & - & - \\ - & 1 & 0 & 1 & 1 & - & - & - & - \\ 1 & - & 1 & 0 & 1 & - & - & - & - \\ 1 & - & 1 & 3 & 0 & - & - & - & - \\ 1 & - & - & - & - & 0 & 1 & - & - \\ - & - & - & - & - & 1 & 0 & 1 & 1 \\ 1 & - & - & - & - & - & 1 & 0 & 3 \\ 1 & - & - & - & - & - & 1 & 3 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 0 & 5 & - & 5 & 5 & 5 & - & 5 & 5 \\ 5 & 0 & 5 & - & - & - & - & - & - \\ - & 5 & 0 & 5 & 5 & - & - & - & - \\ 5 & - & 5 & 0 & 5 & - & - & - & - \\ 5 & - & 5 & 5 & 0 & - & - & - & - \\ 5 & - & - & - & - & 0 & 5 & - & - \\ - & - & - & - & - & 5 & 0 & 5 & 5 \\ 5 & - & - & - & - & - & 5 & 0 & 5 \\ 5 & - & - & - & - & - & 5 & 5 & 0 \end{pmatrix}$$

Los costos de ruteo y conexión especificados con "-" en las matrices C y D no están definidos, dado que no existen aristas entre los vértices considerados.

Los parámetros de construcción de soluciones para cantidad de componentes a construir y capacidad de las mismas, son los siguientes:

$$m = 2, \quad Q = 5.$$

Por construcción es fácil ver que una solución óptima global del CmRSP tiene la topología mostrada en la Figura 6.22 y un costo total de 14.

Nuestro primer paso en las pruebas consistió en la ejecución del modelo en AMPL utilizando solver CPLEX para la obtención de la solución exacta para la instancia K definida anteriormente. Dicha solución tiene la topología mostrada en 6.23 y un costo total de 12. Observando dicha topología podemos ver que la misma contiene soluciones no cíclicas en la componente y mejora el costo de la solución del CmRSP detallado en la Figura 6.22. Hay entonces una solución óptima global del CmTNSSP de mejor costo que la solución óptima global del CmRSP.

Nuestro segundo paso fue ejecutar la metaheurística GRASP-VND con los siguientes parámetros:

$$LIST\_MAX\_SIZE = 4, \quad k = 2, \quad p = 3, \quad MAX\_PATH\_LENGTH = 4.$$

donde $LIST\_MAX\_SIZE$ es el tamaño de la RCL de la fase de construcción de soluciones factibles del GRASP (Algoritmo 6), k es el parámetro de los Algoritmos de Extracción

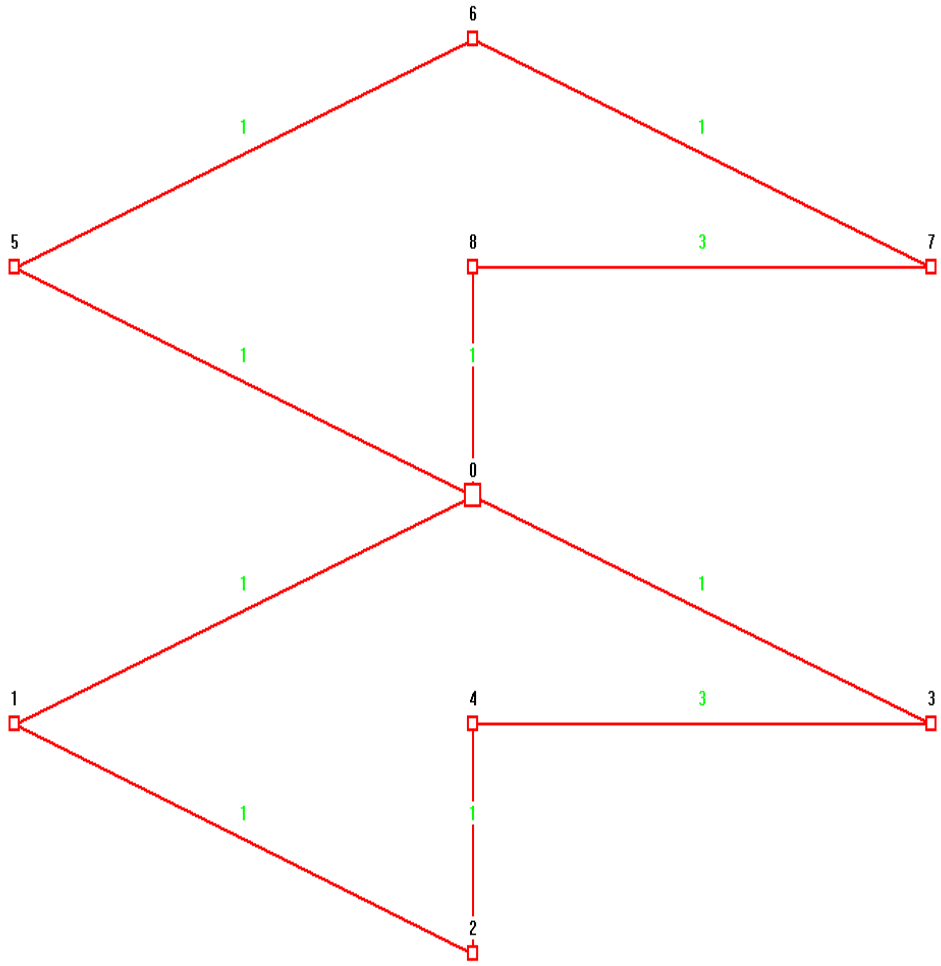

Figura 6.22: Solución óptima global del CmRSP para la instancia correspondiente al grafo K.

e Inserción de nodos (9), Swaping de Nodos (10) y Crossing de Componentes (11); p es la cantidad de nodos a extraer en el Algoritmo de Shaking (14) y MAX_PATH_LENGTH el largo máximo del Algoritmo de Mejor Camino con Nodos Colgantes (12).

La ejecución del GRASP-VND para la instancia K definida con los parámetros especificados en el párrafo anterior, obtuvo como resultado la misma solución óptima global con la topología detallada en la Figura 6.23 y un costo de 12.

Podemos afirmar entonces la validez del algoritmo GRASP-VND en la búsqueda de soluciones al CmTNSSP que mejoren los resultados del CmRSP para las mismas instancias.

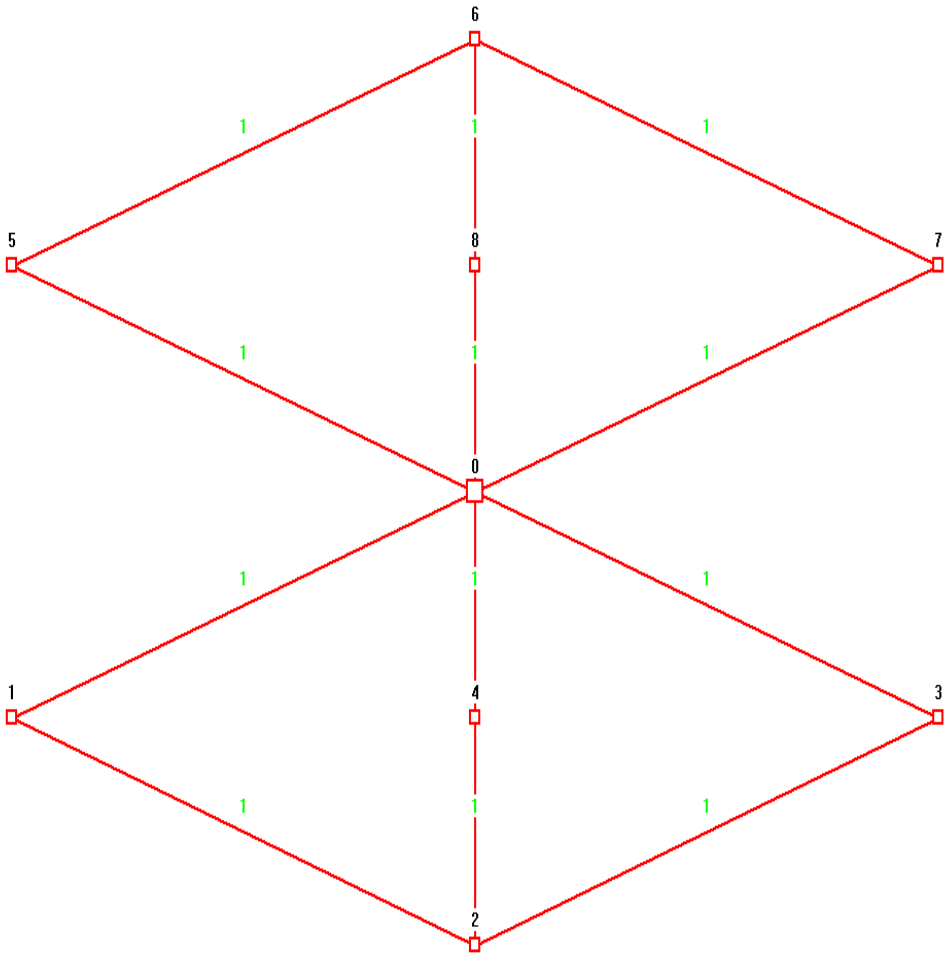

Figura 6.23: Solución óptima global de CmTNSSP para instancia correspondiente al grafo K.

Capítulo 7

Conclusiones

Hemos estudiado en este trabajo un problema hasta donde sabemos no tratado en la literatura, el CmTNSSP, problema similar al CmRSP donde además de ciclos consideramos componentes 2-nodo-conexas no cíclicas.

Hicimos una referencia documentada de la importancia de mantener redundancia al momento de construir redes de telecomunicaciones, como forma de paliar accidentes con enlaces o nodos a fin de mantener la red operativa.

Se describieron propiedades de los grafos 2-conexos, con la obtención de cotas que indican una posible mejora sobre las soluciones 2-conexas cíclicas, fundamentando debidamente el estudio del problema tratado en esta tesis y su perspectiva de mejora en los resultados del CmRSP.

La presentación de estas propiedades estructurales de los grafos dos-conexos nos permitieron construir el modelo y los algoritmos necesarios para su resolución aproximada.

Se definió un modelo de programación lineal entera para el problema CmTNSSP y resolvimos de forma exacta una instancia pequeña utilizando este modelo mediante su transcripción a lenguaje algebraico AMPL y aplicación del solver CPLEX.

Se diseñó un algoritmo de optimización para el CmTNSSP basado en las metodologías GRASP y VND. Se codificaron y se realizaron las pruebas sobre instancias generadas en el trabajo de Baldacci et al. [4] obteniéndose muy buenos resultados, alguno de los cuales mejoran las soluciones encontradas en dicho trabajo.

Para las 45 instancias Clase A donde se aplicó la metaheurística GRASP-VND diseñada, se alcanzó el óptimo global del CmRSP en 29 de ellas, en 3 instancias se mejoraron los valores encontrados por Baldacci et al. [4] que no eran óptimos globales, y en las 13 restantes se obtuvieron gaps menores a 2,5 % salvo en un par de casos.

En relación a las 45 instancias Clase B donde se aplicó la metaheurística GRASP-VND, se

alcanzó el óptimo global del CmRSP en 19 de ellas, en 6 instancias se mejoraron los valores encontrados por Baldacci et al. [4] donde los autores no obtuvieron optimalidad, y en las 20 restantes se obtuvieron gaps menores a 3,1 % salvo en tres casos.

Los gaps promedio alcanzados para las instancias Clase A y Clase B fueron de 0,74 % y 0,89 % respectivamente, lo cual muestra un algoritmo robusto y performante.

También se realizó una prueba sobre nuestra propia instancia $nut30$ resuelta en forma exacta en la sección 6.2 (Figura 6.2), donde obtuvimos mediante GRASP-VND el óptimo global.

Adicionalmente trabajamos con las mismas instancias Clase A y Clase B [4] relajando la restricción de colgar nodos si y solo si el costo de conexión no superaba cierta cota máxima, obteniendo soluciones significativamente superiores en calidad, con gaps promedio inferiores a -3,50 % para las instancias Clase A y -8,13 % para las instancias Clase B.

Se observa además que en las instancias Clase A relajadas, sobre un total de 45 casos, se obtuvo en 39 casos gaps inferiores a cero, en 4 casos se obtuvieron gaps de cero (i.e. el mismo valor que el óptimo reportado por Baldacci et. al), y en 2 casos gaps inferiores al 0.5 %. Respecto a las instancias Clase B relajadas, sobre un total de 45 casos, se obtuvieron en todos los casos gaps inferiores a cero; en muchos de estos casos alcanzando soluciones con mejoras en costo de más de un 25 % respecto al valor de referencia (los óptimos del CmRSP)."

Estos resultados son muy prometedores. La relajación de la restricción de eliminar la cota d_{max} para los costos de conexión de los nodos colgantes (ver ecuación 6.2) en el CmTNSSP, es razonable para un planificador de una red de telecomunicaciones de área extendida de alto porte cuando las centrales (nodos switch) son similares en capacidad/tamaño y costo de puesta en funcionamiento en estado operacional.

Los costos preponderantes de construcción de una red de fibra óptica robusta vienen dados por la canalización y dragado a realizar para la instalación del cableado de fibra óptica [2].

Dado que en las pruebas con las instancias elegidas no llegamos a obtener en ningún caso soluciones que contuvieran estructuras 2-nodo-conexas no cíclicas en sus componentes, generamos dos instancias de prueba donde encontramos soluciones óptimas con esa topología. Para una de esas instancias calculamos el óptimo global del CmRSP (por construcción) y calculamos el óptimo global del CmTNSSP de forma exacta mediante la aplicación de nuestro modelo en AMPL utilizando el solver CPLEX. La solución del CmTNSSP fue mejor que la del CmRSP y su topología evidenció estructuras 2-nodo-conexas no cíclicas.

Para esta misma instancia se ejecutó la metaheurística GRASP-VND y se obtuvo el mismo óptimo global con idéntica topología a la encontrada por el algoritmo de resolución exacta, validando de esta forma el GRASP-VND diseñado.

Podemos afirmar que el problema estudiado en la tesis tiene por un lado relevancia prácti-

ca, dado que las soluciones obtenidas son competitivas (en términos de valor objetivo) con las soluciones obtenidas por trabajos anteriores para un problema similar. Por otro lado, la resolución del CmTNSSP plantea nuevos desafíos académicos, que fueron atacados en esta tesis mediante conceptos de optimización, teoría de grafos y metaheurísticas híbridas.

Trabajos Futuros

En esta tesis no establecimos un modelo optimizado, simplemente nos limitamos a definir un modelo y validarlo. En ese sentido puede existir un trabajo interesante de mejora del modelo para resolver de forma exacta instancias con mayor cantidad de nodos y con mejores tiempos de ejecución que las trabajadas en esta tesis.

Dada la naturaleza de este problema y la pertenencia a la clase de problemas $\mathcal{NP}$-Difíciles, se pueden establecer líneas de trabajo con otras metaheurísticas que mejoren los resultados en donde no se llega a la optimalidad. La ejecución de instancias que involucren casos reales donde el CmRSP obtuvo soluciones óptimas y la posible obtención de mejoras con respecto a estas mediante la resolución del CmTNSSP para los mismos casos, podría significar una confirmación de la validez de esta variante.

Otra línea de trabajo podría definirse estudiando un problema de aumentación del CmRSP con la variante estudiada en esta tesis con el fin de obtener posibles mejores soluciones a problemas de conexión de redes en escenarios reales actualmente en producción.

Bibliografía

[1] G. Zorpette. Keeping the phone lines open. *Spectrum, IEEE*, 26(6):32–36, 1989. ISSN 0018-9235.

[2] M. Stoer. *Design of Survivable Networks (Lecture Notes in Mathematics)*. Springer, 1992. ISBN 3540562710.

[3] C. L. Monma, B. Spellman Munson, and W.R. Pulleyblank. Minimum-weight two-connected spanning networks. *Mathematical Programming*, 46(1-3):153–171, 1990. ISSN 0025-5610.

[4] R. Baldacci, M. Dell'Amico, and J. J. Salazar González. The capacitated m-ring-star problem. *Operations Research*, 55(6):pp. 1147–1162, 2007. ISSN 0030364X.

[5] E. A. Hoshino and C. C. De Souza. A branch-and-cut-and-price approach for the capacitated m-ring-star problem. *Discrete Appl. Math.*, 160(18):2728–2741, December 2012. ISSN 0166-218X.

[6] Z. Naji-Azimi, M. Salari, and P. Toth. A heuristic procedure for the capacitated m-ring-star problem. *European Journal of Operational Research*, 207(3):1227 – 1234, 2010. ISSN 0377-2217.

[7] M. Labbé, G. Laporte, I. Rodríguez Martín, and J. J. Salazar González. Locating median cycles in networks. *European Journal of Operational Research*, 160(2):457 – 470, 2005. ISSN 0377-2217. Decision Support Systems in the Internet Age.

[8] M. Labbé, G. Laporte, I. Rodríguez Martín, and J. J. Salazar González. The ring star problem: Polyhedral analysis and exact algorithm. *Networks*, 43(3):177–189, 2004.

[9] M. B. Richey. Optimal location of a path or tree on a network with cycles. *Networks*, 20 (4):391–407, 1990. ISSN 1097-0037.

[10] C. H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, DTIC Document, 1976.

[12] D. P. Williamson. Analysis of the held-karp heuristic for the travelling salesman problem. Technical report, Cambridge, Cambridge, MA, USA, 1990.

[13] Y Marinakis, A. Migdalas, and P. M. Pardalos. Expanding neighborhood grasp for the traveling salesman problem. *Computational Optimization and Applications*, 32(3):231–257, 2005. ISSN 0926-6003.

[14] E. F. Gouvea Goldbarg, M. C. Goldbarg, and J. Farias. Grasp with path-relinking for the tsp. In KarlF. Doerner, Michel Gendreau, Peter Greistorfer, Walter Gutjahr, RichardF. Hartl, and Marc Reimann, editors, *Metaheuristics*, volume 39 of *Operations Research/Computer Science Interfaces Series*, pages 137–152. Springer US, 2007. ISBN 978-0-387-71919-1.

[15] G. Jager, C. Dong, B. Goldengorin, P. Molitor, and D. Richter. A backbone based tsp heuristic for large instances. *Journal of Heuristics*, 20(1):107–124, 2014.

[16] S. E. Gelenbe, V. Koubi, and F. Pekergin. Dynamical random neural network approach to the traveling salesman problem. In *Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on*, pages 630–635 vol.2, Oct 1993. doi: 10.1109/ICSMC.1993.384945.

[17] M. Mestria, L. Satoru Ochi, and S. de Lima Martins. {GRASP} with path relinking for the symmetric euclidean clustered traveling salesman problem. *Computers & Operations Research*, 40(12):3218 – 3229, 2013. ISSN 0305-0548.

[18] L. Lovász. On some connectivity properties of eulerian graphs. *Acta Mathematica Hungarica*, 28(1):129–138, 1976.

[19] P. Winter. Steiner problem in networks: a survey. *Networks*, 17(2):129–167, 1987.

[20] A. Agrawal, P. Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks, 1994.

[21] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. In *Proceedings of the American Mathematical Society, 7*, 1956.

[22] F. K Hwang, D. S Richards, and P. Winter. *The Steiner tree problem*. Elsevier, 1992.

[23] D. Paik, S. Reddy, and S. Sahni. Vertex splitting in dags and applications to partial scan designs and lossy circuits. 1990.

[24] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986. ISSN 0305-0548. Applications of Integer Programming.

[25] E. Talbi. *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, 2009. ISBN 0470278587.

[26] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, September 2003. ISSN 0360-0300.

[27] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2-3):95–99, 1988. ISSN 0885-6125.

[28] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, Feb 1996. ISSN 1083-4419. doi: 10.1109/3477.484436.

[29] X. S. Yang. *Cuckoo Search and Firefly Algorithm: Theory and Applications (Studies in Computational Intelligence)*. Springer, 2013. ISBN 3319021400.

[30] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence (The Morgan Kaufmann Series in Evolutionary Computation)*. Morgan Kaufmann, 2001. ISBN 1558605959.

[31] M. G.C. Resende. Greedy randomized adaptive search procedures. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Encyclopedia of Optimization*, pages 1460–1469. Springer US, 2009. ISBN 978-0-387-74758-3.

[32] F. Glover. Tabu search part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[33] F. Glover. Tabu search part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[34] H. R. Lourenço, O. C. Martin, and T. Stutzle. Iterated local search. *arXiv preprint math/0102188*, 2001.

[35] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997. ISSN 0305-0548.

[36] C. Voudouris, E. P. K. Tsang, and A. Alsheddy. Guided local search. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 321–361. Springer US, 2010. ISBN 978-1-4419-1663-1.

[37] E. Talbi, editor. *Hybrid Metaheuristics (Studies in Computational Intelligence)*. Springer, 2012. ISBN 3642306705.

[38] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[39] R. Bhandari. Optimal physical diversity algorithms and survivable networks. In *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pages 433–441, 1997. doi: 10.1109/ISCC.1997.616037.

[40] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984. ISSN 1097-0037.

[41] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[42] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982. ISBN 0-13-152462-3.

[43] M. Aspnäs, A. Signell, and J. Westerhölm. Efficient assembly of sparse matrices using hashing. In Bo Kagstrom, Erik Elmroth, Jack Dongarra, and Jerzy Wasniewski, editors, *Applied Parallel Computing. State of the Art in Scientific Computing*, volume 4699 of *Lecture Notes in Computer Science*, pages 900–907. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-75754-2.

[44] IBM. Cplex callable library reference manual, 2014. URL http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r2/topic/ilog.odms.cplex.help/html/refcallablelibrary/html/overview.html.

[45] ILOG, Inc. ILOG CPLEX: High-performance software for mathematical programming and optimization, 2006. See http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html.

[46] R. Fourer, D.M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Cengage Learning, 2002. ISBN 0534388094.

[47] G. Reinelt. TSPLIB - A t.s.p. library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.

[48] E. A. Hoshino and C. C. De Souza. Instances for the capacitated m-ring-star problem. www.ic.unicamp.br/~cid/Problem-instances/CmRSP/.

[49] H. Barbalho, I. Rosseti, S. L. Martins, and A. Plastino. A hybrid data mining grasp with path-relinking. *Comput. Oper. Res.*, 40(12):3159–3173, December 2013. ISSN 0305-0548.

[50] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1995. ISBN 0133350681.

[51] T. A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. ISSN 0925-5001.

[52] P. Festa. Greedy randomized adaptative search procedure. *AIROnews*, 2003. URL www.dti.unimi.it.

[53] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[54] J. W. George and C. S. Revelle. Bi-objective median subtree location problems. *Annals of Operations Research*, 122(1-4):219–232, 2003. ISSN 0254-5330.

[55] F. Harary. The maximum connectivity of a graph. *Proc Natl Acad Sci USA*, 48(7):1142–1146., 1962.

[56] A. Mauttone, S. Nesmachnow, A. Olivera, and F. Robledo. Solving a ring star problem generalization. In *CIMCA/IAWTIC/ISE*, pages 981–986, 2008.

[57] E. Minieka. The optimal location of a path or tree in a tree network. *Networks*, 15(3): 309–321, 1985. ISSN 1097-0037.

[58] C. L. Monma and D. F. Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):pp. 531–541, 1989. ISSN 0030364X.

[59] M. Resende and C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 283–319. Springer US, 2010. ISBN 978-1-4419-1663-1.

[60] F. Robledo. *GRASP heuristics for Wide Area Network design*. PhD thesis, INRIA, February 2005. URL http://www.fing.edu.uy/inco/pedeciba/bibliote/tesis/tesisd-robledo.pdf.

[61] F. Robledo and E. Canale. Designing backbone networks using the generalized steiner problem. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, pages 327–334, Oct 2009. doi: 10.1109/DRCN.2009.5339990.

[62] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974. ISSN 1097-0037.

[63] Zizhen Zhang, Hu Qin, and Andrew Lim. A memetic algorithm for the capacitated m-ring-star problem. *Appl. Intell.*, 40(2):305–321, 2014. doi: 10.1007/s10489-013-0460-6. URL http://dx.doi.org/10.1007/s10489-013-0460-6.

Índice de figuras