PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Tesis de Maestría
en Informática

Diseño topológico de redes : casos de estudio "The generalized Steiner problem" and "The Steiner 2-Edge-Connected subgraph problem"

Franco Robledo Amoza

Febrero 2000

Orientador de Tesis: Héctor Cancela
Orientador de Maestría: Omar Viera

Diseño topológico de redes : casos de estudio "The generalized Steiner problem" and "The Steiner 2-Edge-Connected subgraph problem".
Franco Robledo Amoza

ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 00-08
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, febrero de 2000

Resumen

Dado un grafo *G=(V,E)*, una matriz *C* de costos asociados a las aristas, un subconjunto *T* de nodos denominados terminales y una matriz *R* de requerimientos de conexión entre nodos terminales, el "*Generalized Steiner Problem*" (GSP) consiste en encontrar un subgrafo G_S de *G* de costo mínimo tal que para todo par de nodos terminales existen al menos R_{ij} caminos de aristas-disjuntas en G_S. Un grafo se dice 2-arista-conexo si entre todo par de nodos existen al menos 2 caminos de aristas disjuntas que los unen. Dos casos particulares del GSP son:

- **encontrar un subgrafo G_S de *G* 2-arista-conexo de costo mínimo que cubra el conjunto de nodos terminales *T*, este problema es conocido como "*Steiner 2-edge-connected subgraph problem*" (STECSP),**
- **encontrar un subgrafo G_S de *G* de costo mínimo tal que para todo par de nodos terminales existen al menos 2 caminos de aristas disjuntas que los unen, este problema es conocido como "*Steiner 2-edge-survivable subgraph problem*" (STESNP).**

En este contexto se dice que una red esta en estado operativo si existe comunicación entre cualquier par de nodos terminales. Los modelos GSP, STECSP y STESNP son utilizados en el diseño de redes con altos niveles de confiabilidad, donde se requiere que la red se mantenga en estado operativo aún cuando se produzcan fallas en alguna de sus componentes. En especial los modelos GSP y STECSP son tomados como base para el diseño de redes de fibras ópticas metropolitanas. Este trabajo esta centralizado en el estudio de los problemas GSP, STECSP y STESNP, para cada uno de ellos se brinda un resumen del estado del arte donde se mencionan resultados teóricos y algoritmos de resolución conocidos hasta el momento. Se presentan nuevos algoritmos de resolución en forma exacta y aproximada, en particular se presenta un algoritmo de diseño paralelo distribuido para resolver en forma exacta el STECSP, los resultados experimentales muestran que la utilización del Paradigma del Desarrollo Paralelo-Distribuido es una buena alternativa si se necesita encontrar la solución óptima en tiempos de ejecución viables a instancias del STECSP de mediano y gran porte. Se presenta además un algoritmo aproximado basado en la metaheurística *Ant System* para el GSP. Hasta el momento se conoce una única heurística para el GSP aplicable a las clases más generales de grafos. Este nuevo algoritmo brinda otra forma de resolver el GSP mediante un enfoque topológico, las pruebas realizadas muestran una buena performance del algoritmo alcanzándose en la mayoría de los casos la solución óptima.

Palabras Claves: requerimientos de conexión, subgrafo 2-arista-conexo, algoritmo exacto, algoritmo paralelo distribuido, algoritmo aproximado, metaheurística *Ant System*.

1 Introducción

Cuando se diseñan ciertos tipos de redes, donde se pretende un nivel de confiabilidad elevado, uno de los objetivos es que la topología de red resultante del diseño sea capaz de mantenerse en estado operativo dado que se produce una falla en una de sus componentes; en general, se dice que una red está en estado operativo si existe comunicación entre cualquier par de nodos de la red. Algunos sistemas de este tipo son: redes de comunicación donde se requieren niveles altos de confiabilidad, redes de suministro de energía eléctrica, redes de distribución de agua, redes de uso militar, redes de ordenadores dedicados al control de centrales industriales, etc.

Un caso especial, es la tecnología de fibras ópticas aplicada a redes de telecomunicaciones. Las redes de fibras ópticas están empezando a tomar una importancia considerable en aplicaciones relacionadas con redes de área local y metropolitana. Al diseñar grandes redes de fibras ópticas, donde se tiene un ancho de banda elevado y enlaces altamente seguros, surge la necesidad de agregar "redundancia" de comunicación en la red de forma de aumentar la confiabilidad de la misma. La confiabilidad de una red está estrechamente relacionada con el grado de conexión de la red. El grado de conexión de una red, es la cantidad mínima de caminos disjuntos que existe entre cualquier par de nodos de la red; esta medida de conexión puede ser dada en términos de caminos de aristas disjuntas o bien caminos de nodos disjuntos. A pesar que los enlaces punto a punto son "seguros", debido al tipo de servicio que prestan las redes de fibras ópticas, las consecuencias de fallas en algunas de sus componentes pueden resultar catastróficas. Es así que surge la necesidad de existencia de cierto grado de conexión en el diseño de la red de fibra óptica, de manera que la red sea más confiable. Muchas veces alcanza con exigir que se cumplan requerimientos de conexión específicos para un grupo de nodos distinguidos de la red.

El problema de encontrar la topología de red de mínimo costo, que satisface ciertos requerimientos de conexión fijados de antemano, puede ser modelado como un problema de Steiner en redes con requerimientos de conexión entre nodos. Seguidamente, se definen formalmente: el Problema

de Steiner en Redes, Problema General de Steiner, Problema del subgrafo de Steiner 2-arista-conexo, y el Problema del subgrafo de Steiner 2-arista-confiable.

Dado un grafo $G = (V, E)$ conexo no dirigido, c una matriz de costos asociados a las aristas, y $T \subseteq V$ un conjunto de nodos distinguidos denominados terminales, se definen:

- Problema de Steiner en Redes (*Steiner Problem in Networks* - SPN) – consiste en encontrar el subgrafo de G de mínimo costo que cubre el conjunto de nodos terminales T,
- Problema General de Steiner (*Generalized Steiner Problem* - GSP) – dada una matriz de requerimientos de conexión entre nodos terminales, $R = \{r_{ij}, \forall i, j \in T\}$ donde los r_{ij} son enteros positivos, se pretende encontrar el subgrafo de G de mínimo costo tal que $\forall i, j \in T, i \neq j$, existen al menos r_{ij}-caminos de aristas disjuntas que los unen. De igual manera, la versión nodo-conexa del GSP exige que existan al menos r_{ij} -caminos de nodos disjuntos entre todo par de nodos terminales.
- Problema del subgrafo de Steiner 2-arista-conexo (*Steiner 2-edge-connected subgraph problem* - STECSP) – consiste en encontrar un subgrafo de G de mínimo costo que sea 2-arista-conexo y que cubra el conjunto de nodos terminales T,
- Problema del subgrafo de Steiner 2-arista-confiable (*Steiner 2-edge-survivable subgraph problem* - STESNP) – consiste en encontrar un subgrafo de G de mínimo costo tal que entre todo par de nodos de T existen al menos 2 caminos de aristas disjuntas que los unen.

Las soluciones factibles al SPN son soluciones de baja confiabilidad si se asume la posibilidad de que existan fallas en las componentes de la red, si bien en este modelo el sistema se mantiene operativo con el menor costo posible, una falla en un enlace haría que el sistema deje de estar en estado operativo. Esta es la razón por la cual en el diseño de redes de gran escala altamente confiables, no se utiliza al SPN como modelo del problema. Los

objetivos: minimizar el costo total de conexión, y maximizar la confiabilidad de la red son contrapuestos. Cuando se diseñan redes de gran escala altamente confiables, en general, se establecen requerimientos de conexión entre pares de nodos distinguidos de la red (la existencia de una cierta cantidad de caminos disjuntos), y se pretende encontrar una red que satisfaga dichos requerimientos con el menor costo posible. Los modelos de problemas del tipo GSP, STECSP y STESNP, se adaptan a este tipo de diseños.

Casi no existen algoritmos que resuelvan el GSP y los problemas STECSP-STESNP. Los algoritmos que se conocen están hechos a medida para clases particulares de grafos, como ser: grafos serie-paralelos, de Halin, y outerplanares, y la mayoría se basan en un enfoque poliédrico. Ajit Agrawal-Philip Klein-R.Ravi en [1] presentan la primera heurística conocida que resuelve en forma aproximada el problema GSP.

En este trabajo de Tesis de Maestría, el objetivo esta centralizado en el estudio de los modelos GSP y los casos particulares STECSP-STESNP. Se analizan estos problemas en base a un análisis topológico de sus soluciones y se presentan procedimientos para resolverlos. Los nuevos algoritmos propuestos en este trabajo son los siguientes:

- se presentará un nuevo algoritmo exacto para la resolución del GSP,
- un algoritmo exacto para el STECSP basado en la técnica de Backtracking,
- un algoritmo exacto para el STECSP basado en la técnica de Programación Dinámica,
- dos algoritmos paralelo-distribuidos para el STECSP,
- un algoritmo aproximado diseñado para el GSP basado en la metaheurística conocida como *Ant System*.

La organización de este trabajo es la siguiente. En el capítulo 1 se estudia el modelo GSP; se da una formulación matemática del GSP, se brinda un resumen de los algoritmos y resultados conocidos, casos especiales del GSP y se presenta un nuevo algoritmo exacto de Backtracking para el GSP. En el capítulo 2 se estudian los problemas STECSP-STESNP; se mencionan los algoritmos existentes, casos especiales, y resultados conocidos. Se presenta

además un nuevo algoritmo exacto de Backtracking para el STECSP, un nuevo algoritmo exacto de Programación Dinámica para el STECSP y dos algoritmos exactos paralelo-distribuidos para el STECSP diseñados en base a estos últimos. De los dos algoritmos paralelos presentados, uno de ellos es implementado, brindándose un análisis de los resultados experimentales obtenidos. En el capítulo 3 se desarrolla un nuevo algoritmo para la resolución en forma aproximada del GSP basado en la Metaheurística *Ant System*. Se da una descripción detallada del algoritmo, un análisis de complejidad en el peor caso, resultados experimentales obtenidos y conclusiones. En el capítulo 4 se dan las conclusiones generales del trabajo de tesis desarrollado.

El contenido del Apéndice es el siguiente. En el capítulo A1 se dan definiciones y resultados teóricos considerados como básicos para la comprensión del trabajo presentado. En el capítulo A2 se brinda un resumen de los algoritmos exactos y heurísticas más conocidas para el SPN y se mencionan los problemas de este modelo cuando se diseñan redes altamente confiables. En el capítulo A3 se da un resumen del Problema del Subgrafo de Cubrimiento k-Conexo de Mínimo Costo (*Minimum Weight k-connected Spanning Networks* – MWkCSN), se verán los enunciados de importantes teoremas que caracterizan la solución óptima de este problema, y un teorema que lo relaciona con el STECSP.

2 1. Generalized Steiner Problem (GSP)

2.1 1.1 Introducción

Ciertos tipos de redes tienen un costo de construcción elevado y además se necesita que la red sea resistente a fallas en sus componentes. Como ejemplo se tienen: redes eléctricas, redes de abastecimiento de agua, redes de comunicación. Al diseñar estas redes, surgen dos objetivos contrapuestos: la minimización del costo total de la red, y la maximización de su confiabilidad. Con frecuencia la confiabilidad de la red viene dada por el grado de conexión de la misma.

Dada una red y un conjunto de nodos distinguidos denominados terminales, un modelo del tipo SPN para dicha red cumple el objetivo primario de minimizar el costo total de la red, satisfaciendo el requisito de conexión entre todo par de nodos terminales. Este tipo de soluciones son extremadamente vulnerables en el caso que puedan ocurrir fallas en los enlaces entre los nodos y/o en los nodos, por causas como ser: falta de mantenimiento, errores, sobrecarga, influencia de agentes externos, o cualquier otro tipo de fuente destructiva. En la solución del SPN si un solo nodo o arista falla la red deja de estar en estado operativo, entendiéndose como estado operativo la posibilidad de comunicación entre todo par de nodos terminales. De esta manera, se puede ver que en el caso que sea altamente probable la ocurrencia de fallas el objetivo no debe ser solamente la minimización del costo total de conexión, sino que también se cumplan ciertos requisitos de conexión (fijados previo al diseño de la red) entre pares de nodos terminales.

El *Generalized Steiner Problem* (GSP) concierne al diseño de una subred de mínimo costo de una red dada, donde algunos nodos (no necesariamente todos) denominados terminales, deben satisfacer ciertos requerimientos de conexión entre pares de nodos. El GSP tiene aplicación en modelos de redes donde se requiere un cierto grado de confiabilidad. Dicha confiabilidad, se obtiene exigiendo la existencia de caminos alternativos entre pares de nodos terminales.

El objetivo de este capítulo es el estudio del problema GSP y la propuesta de un algoritmo que permita hallar la solución óptima para la clase

más general de grafos. En base al relevamiento de la literatura existente, primero se define formalmente el problema GSP, y se da una formulación matemática del problema como un programa de Programación Lineal Entera. Luego se brinda un resumen donde se mencionan los resultados y algoritmos publicados referentes a la resolución del GSP para clases particulares de grafos. Se presenta una formulación alternativa del GSP conocida por *Multiterminal Network Synthesis* (MNS), dado que en este marco se ha propuesto la única heurística publicada hasta el momento para la clase más general de grafos. También se presentan dos casos particulares de GSP conocidos como problemas kNCON y kECON, los cuales para ciertas clases de grafos admiten procedimientos de solución eficientes; se da además un resumen de casos particulares de GSP para los cuales existen algoritmos de orden polinomial que los resuelven. Por último se propone un nuevo algoritmo exacto para la resolución del GSP basado en la metodología de Backtracking. Si bien este algoritmo es poco eficiente (orden exponencial), permite tener un elemento de comparación con las heurísticas para instancias del GSP donde los grafos asociados son de pequeño tamaño; además su diseño hace posible realizar una versión paralela-distribuida del algoritmo, de manera de poder resolver instancias del problema con topologías de mayor escala.

2.2 1.2 Formulación del GSP

El Problema General de Steiner (GSP) formulado por Krarup [60], es el siguiente. Dado un grafo no dirigido $G = (V, E)$, una matriz de costos c asociados a las aristas, un subconjunto de nodos terminales T, tal que $2 \leq n_T \leq n$ con $n = |V|$ y $n_T = |T|$, una matriz de dimensión $n_T \times n_T$ $R = \{r_{ij}\}_{i,j \in T}$ cuyos elementos son enteros positivos que indican los requerimientos de conectividad entre todo par de nodos de T, se pretende encontrar un subgrafo G_T de G de costo mínimo tal que para todo par de nodos $i, j \in T, i \neq j$, estos sean localmente r_{ij}-arista-conexos (r_{ij}-nodo-conexos). Ha sido demostrado por Krarup que el problema GSP es NP-Completo [60] tanto para requerimientos de caminos de aristas disjuntas así como de caminos de nodos disjuntos.

2.3 1.3 Modelo Matemático del GSP

La versión arista-conexo del GSP puede ser modelada como un problema de Programación Lineal Entera. Se define un variable binaria x_{ij} para toda arista $(i, j) \in E$. Además, para toda arista $(i, j) \in E$, y $k, l \in T, k \neq l$, se define la variable y_{ij}^{kl} que denota la utilidad de la arista (i, j) en la dirección de i hacia j en un camino que une el nodo terminal k con el nodo terminal l. El siguiente problema de Programación Lineal Entera da la solución óptima al GSP.

$$Min \sum_{(i,j)\in E} c_{ij} \cdot x_{ij}$$

sujeto a:

$$x_{ij} \geq y_{ij}^{kl} + y_{ji}^{kl} \qquad \forall (i, j) \in E, \forall k, l \in T, k \neq l$$

$$\sum_{(k,j)\in E} y_{kj}^{kl} \geq r_{kl} \qquad \forall k, l \in T, k \neq l$$

$$\sum_{(i,l)\in E} y_{il}^{kl} \geq r_{kl} \qquad \forall k, l \in T, k \neq l$$

$$\sum_{(p,j)\in E} y_{pj}^{kl} - \sum_{(i,p)\in E} y_{ip}^{kl} \geq 0 \qquad \forall k, l \in T, \forall p \in V \setminus \{k, l\}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i, j) \in E$$

$$y_{ij}^{kl} \geq 0 \qquad \forall i, j : (i, j) \in E, \forall k, l \in T, k \neq l$$

[PLE_1]

El conjunto de aristas determinado por $\{(i, j) \in E | u_{ij} = 1\}$, donde $U = \{u_{ij}\}$ es la solución óptima del problema [**PLE_1**], define el subgrafo solución G_T. Se puede hacer una formulación análoga para la versión nodo-conexo del GSP, transformando el grafo G mediante la técnica de partición de nodos.

2.4 1.4 Algoritmos Conocidos para el GSP

Un caso particular de GSP, en el cual $T = V$, ha sido muy estudiado por *Christofides-Whitlock* [18], y en especial el caso en el cual los costos de las aristas del grafo son idénticas, brindándose además múltiples aplicaciones prácticas. *Winter* [59] demostró que en los casos en los cuales el grafo G es serie-paralelo o de *Halin*, existen algoritmos de tiempo lineal que lo resuelven. *Goemans-Bertsimas* [29] consideran el caso especial de *GSP* kNCON-kECON, y presentan una heurística para resolverlo.

Ajit Agrawal-Philip Klein-R.Ravi [1] presentan un algoritmo aproximado para un problema equivalente al GSP, denominado *Multiterminal Network Synthesis* (MNS) que veremos en la sección siguiente. *Grötschel, Monma y Stoer* [30,31,32] estudian el kNCON y kECON basados en un enfoque poliédrico y para clases particulares de grafos. En sección *1.6* se estudian ambos problemas.

2.5 1.5 Multiterminal Network Synthesis (MNS)

2.5.1 1.5.1 Introducción

En este punto se presenta un problema equivalente al GSP denominado *Multiterminal Network Synthesis* (MNS). Primeramente se propone un escenario real, la conexión de Centrales Hidroeléctricas entre un grupo de países limítrofes. Este escenario será tomado como base para la formalización del modelo MNS. Se darán una serie de definiciones y resultados previos a la presentación de un Teorema de Ajit Agrawal-Philip Klein-R.Ravi [1] que da la performance del algoritmo aproximado para el MNS propuesto por estos autores.

2.5.2 1.5.2 Interconección de Centrales Hidroeléctricas (Problema ICH)

Supóngase que un grupo de países limítrofes desean conectar sus Centrales Hidroeléctricas de forma de poder realizar transferencias de energía entre ellas. Una conexión entre dos Centrales no necesariamente

pasa por una Estación Intermedia. Entre las diferentes Centrales y Estaciones Intermedias de los diferentes países, existen ciertos requerimientos de conexión, los cuales indican qué pares de Centrales Hidroeléctricas hay que conectar de forma de satisfacer la demanda energética de cada país. Se conocen de antemano las disponibilidades de conexión entre las diferentes Centrales y Estaciones Intermedias, y los costos asociados. En este contexto, se pretende interconectar las Centrales Hidroeléctricas satisfaciendo los requerimientos de conexión con mínimo costo. Se asume que una conexión puede ser utilizada simultáneamente por varias Centrales para transferir energía eléctrica.

2.5.3 1.5.3 Formalización del problema

Sea $G = (V, E)$ un grafo no dirigido con costos no negativos asociados a sus aristas, y sea R un conjunto de pares de nodos $R = \{(o_i, d_i), i \in 1..k, o_i, d_i \in V\}$, donde a los nodos o_i y d_i se los denomina sitios, las parejas (o_i, d_i) pares de sitios, y k es la cantidad de pares de sitios en R. Un subgrafo H de G, se le llama *Requirement Join* si para todo par de nodos $(o_i, d_i) \in R$ el subgrafo H contiene un camino que une los sitios o_i y d_i. A los pares de nodos de R se les denomina requerimientos de conexión, pues ellos representan las restricciones de conexión que deben ser satisfechas por el subgrafo *Requirement Join*. Un *Requirement Join* asociado a un conjunto de requerimientos de conexión R, se lo denota como $R - join$.

Dado un grafo $G = (V, E)$ el problema de hallar el $R - join$ de costo mínimo tiene múltiples aplicaciones prácticas. Por ejemplo, el problema **ICH** descripto en *1.5.2* se puede modelar como un problema de encontrar el $R - join$ de mínimo costo en una red. Un caso particular de éste problema, es el *Steiner Tree Problem in Networks* (SPN). Ninguno de los algoritmos que resuelven el SPN considera el caso más general en el cual se pueden seleccionar en forma arbitraria los sitios que se desea estén conectados. En el problema de hallar el $R - join$ de mínimo costo, el subgrafo solución no necesariamente es conexo. El siguiente teorema brinda información sobre el

costo de la solución dada por un algoritmo propuesto por *Agrawal-Klein-Ravi* para encontrar un $R-join$ en un grafo $G=(V,E)$.

Teorema 1 (Agrawal-Klein-Ravi)

Existe un algoritmo de orden polinomial que encuentra un $R-join$ *de costo a lo sumo un factor* $(2-2/k)$ *veces el costo de la solución óptima, donde* k *es el número de sitios en* R [1].

El algoritmo en el cual se basa el Teorema 1, resulta útil cuando la solución construida no necesariamente tiene que ser conexa. Si la solución que se brinda tiene que ser conexa, el algoritmo del teorema se puede utilizar como subrutina de otro algoritmo que busca diseñar una red conexa que cumpla con los requerimientos de conexión R y de costo el menor "posible".

Supóngase que al problema **ICH** se le agrega la restricción en la cual las Centrales Hidroeléctricas que deben estar conectadas, lo deben estar por un cierto número de caminos múltiples disjuntos. Esto contribuye al diseño de una red de conexión más confiable, haciéndola menos vulnerable a fallas en los "links". Si se considera el grafo asociado al modelo, entre todo par de sitios $(o_i,d_i)\in R$, debe haber un cierto número de caminos de aristas-disjuntas que los una. Se asume que un grafo en este contexto puede tener múltiples aristas entre un mismo par de nodos, y todas con el mismo costo.

Cuando son necesarias conexiones múltiples entre pares de sitios (diferentes caminos de aristas-disjuntas entre pares de sitios), se redefine el conjunto de requerimientos de conexión como un conjunto de triples $R=\{(o_i,d_i,r_i), i\in 1..n_R, o_i,d_i\in V\}$, donde o_i y d_i son los sitios a ser conectados, y r_i es el valor del requerimiento, el cual indica el número de caminos de aristas-disjuntas que debe haber entre o_i y d_i en la solución.

Se define un $R-multijoin$ como un subgrafo de G cuyo conjunto de aristas determinan al menos r_i caminos de aristas disjuntas entre los sitios o_i y d_i, para todo requerimiento de conexión (o_i,d_i,r_i). El costo de un $R-multijoin$ es la suma de los costos de las aristas que lo componen, contando sus multiplicidades.

El problema de encontrar un $R-multijoin$ de costo mínimo, es llamado por *Chien-Gomory-Hu* como *Multiterminal Network Synthesis* (MNS) [1]. Un caso especial del MNS, en el cual el grafo es completo y los costos de las aristas son todos idénticos, fue estudiado por *Sridhar-Chandrasekaran* [49].
El MNS sin aristas múltiples es equivalente al GSP formulado por *Krarup*, la transformación de un problema a otro es trivial.
Existe un algoritmo aproximado propuesto por *Agrawal-Klein-Ravi*, (el cual se basa en el algoritmo aproximado del Teorema 1 para el cálculo del $R-join$ de costo mínimo) para el problema de encontrar el $R-multijoin$ de costo mínimo. El siguiente Teorema da información de la eficiencia de dicho algoritmo.

Teorema 2 (Agrawal-Klein-Ravi)

Existe un algoritmo de orden polinomial que encuentra un $R-multijoin$ *de costo a lo sumo un factor* $(2-2/k)\cdot\lceil\log_2(r_{max}+1)\rceil$ *veces el costo de la solución óptima, donde* r_{max} *es el valor de requerimiento de conexión más grande y* k *es el número de sitios en* R [1] *.*

Previo al algoritmo aproximado presentado por *Agrawal-Klein-Ravi* para resolver el problema de encontrar el $R-multijoin$ de mínimo costo de un grafo, no se conocía ningún otro algoritmo aproximado para resolver el GSP. El algoritmo que ellos presentan es aplicable aún cuando hay aristas múltiples entre pares de nodos. Una descripción completa de este algoritmo se encuentra en [1].

2.6 1.6 Problemas kNCON-kECON

2.6.1 1.6.1 Introducción

Como se mencionó en *1.3*, los problemas kNCON-kECON son casos especiales del GSP. Muchos problemas de diseño de redes de comunicaciones pueden ser modelados como problemas kNCON-kECON. Por ejemplo, en el diseño de redes de fibras ópticas donde los costos

asociados a los enlaces son elevados, se presenta el problema de diseñar una topología de red que pueda sobrevivir a fallas en sus componentes (se mantenga en un estado operativo dado que se producen fallas) y tenga el menor costo posible. Este problema es típico de las Compañías Telefónicas, y puede ser modelado como un problema kECON. Ambos problemas fueron estudiados con particular interés por Goemans, Bertsimas, M.Grötschel, C.L.Monma y M.Stoer [29,32].

2.6.2 1.6.2 Formulación del kNCON (kECON)

Sea un grafo $G=(V,E)$. Supóngase que a cada nodo $v \in V$ se le asocia un número entero positivo $r(v)$ que representa el "grado de conexión" del nodo v. Se dice que un subgrafo $H=(N,A)$ de G verifica las condiciones de fiabilidad asociadas a los nodos (resp., a las aristas), si para todo par de nodos $s,t \in V$, H contiene al menos $r(s,t)=min(r(s),r(t))$ caminos de nodos disjuntos (resp., de aristas disjuntas). Al problema de encontrar un subgrafo H que verifique las condiciones de fiabilidad asociadas a los nodos (resp., a las aristas) y que sea de costo mínimo se le denomina kNCON (resp., kECON).

La capacidad de sobrevivir o la "importancia" de los nodos, está modelada de acuerdo a tipos de nodos. En particular, cada nodo $v \in V$ tiene asociado un entero positivo $r(v)$; instanciando se denota r_v al tipo de nodo v. El grado de conexión de nodos (resp., conexión de aristas) de un grafo es el máximo k para el cual éste es k-nodo-conexo (resp., k-arista-conexo). Sin perder generalidad, se asume que existen al menos dos nodos de tipo k, donde k es el mayor tipo de nodos.

El kNCON (resp., kECON) se puede modelar como un problema de Programación Matemática, mas específicamente como un problema de Programación Lineal Entera. Se presenta a continuación el modelo, dando previamente una definición.

2.6.2.1.1 Definición 3

Se define $con(W)$ *, como* $con(W) = max\{r(s,t) | s \in W, t \in V \setminus W\}$ *, con* $W \subseteq V$ *. Se denota* $con_H(W)$ *cuando* $con(W)$ *es referido a un grafo* H *diferente de* G *.*

El problema kNCON es equivalente al siguiente problema de Programación Lineal Entera.

$$min \sum_{e \in E} (c_e \cdot x(e))$$

sujeto a:

(1) $x(\delta(W)) \geq con(W), \forall W \subseteq V, \emptyset \neq W \neq V,$

(2) $x(\delta_{G-Z}(W)) \geq con_{G-Z}(W) - |Z|, \forall Z \subseteq V, \emptyset \neq Z \neq V,$
$\forall W \subseteq (V \setminus Z), \emptyset \neq W \neq V, |W| \leq con_{G-Z}(W),$

(3) $0 \leq x(e) \leq 1, \forall e \in E$

(4) $x(e)$ *entero*, $\forall e \in E$

[PLE_2]

Donde $\delta(W) = \{(i,j) \in E | i \in W, j \in V \setminus W\}$ es el corte inducido por el conjunto W y $x(F) = \sum_{e \in F} x(e)$, con $F \subseteq E$.

Suprimiendo las restricciones (2), se obtiene un problema de Programación Lineal Entera, que modela el kECON. El problema STESNP es un caso particular del kECON cuando $r(v) \in \{0,2\}, \forall v \in V$.

2.6.3 1.6.3 Algoritmos y Resultados para kNCON-kECON

Grötschel-Monma-Stoer [32] estudian los problemas kNCON-kECON basados en un enfoque poliédrico, buscando una relajación del problema [PLE_2] para poder resolverlo en forma aproximada. Monma-Stoer [30] analizan las características básicas de los poliedros asociados a los problemas, Grötschel-Monma-Stoer [32,31] identifican además otras propiedades de los poliedros y proponen algoritmos de corte para los problemas kNCON-kECON y muestran resultados experimentales.

Stoer brinda un análisis profundo de los modelos kNCON-kECON en [50]. Goemans-Bertsimas [29], presentan una heurística en la cual atacan el problema descomponiéndolo en una serie de problemas SPN, resolviendo cada uno de los SPN usando alguna heurística estándar. Demuestran un importante teorema el cual da información sobre la performance de la heurística propuesta por ellos. El enunciado del teorema es el siguiente.

Teorema 4 (Goemans-Bertsimas)

Existe una heurística que resuelve el problema kECON (kNCON) con un factor de performance igual a $2 \cdot min(\log r_{max}, p)$ *veces el costo de la solución óptima, donde* $r_{max} = Max\{r(v), v \in V\}$*, y* p *es el número de valores* $r(v)$ *diferentes de cero, y además éste factor es rígido en el peor caso.*

2.7 1.7 Casos Particulares de GSP (Orden Polinomial)

2.7.1 1.7.1 Costos Idénticos

Cuando los costos de las aristas del grafo $G = (V, E)$ son idénticos, el GSP puede verse como el problema de encontrar un subgrafo con mínimo número de aristas que satisfaga que $\forall i, j \in V$ existan al menos r_{ij} caminos de aristas disjuntas. En algunas variantes, es permitido el uso de aristas múltiples entre pares de nodos.

Couch-Frank [50] presentan un algoritmo de orden polinomial que resuelve el siguiente problema.

Problema (Generalized Steiner Problem with Uniform Costs - GSP CU)

Dado un grafo $G = (V, E)$ *y una matriz de requerimientos de conexión* $R = \{r_{ij}; i, j \in V\}$*, se desea encontrar un subgrafo con mínimo número de*

aristas que satisfaga que $\forall i, j \in V$ *exista al menos* r_{ij} *caminos de aristas disjuntas (con o sin aristas paralelas).*

Couch-Frank demuestran el siguiente Lema.

Lema 5 (Couch-Frank)

Sea $r_i = max_{j \in V}\{r_{ij}\}$, $\forall i \in V$ *. Suponiendo que* $r_i \geq 2, \forall i \in V$ *, el mínimo número de aristas de un grafo que satisface la matriz de requerimientos de conexión* R *con caminos de aristas disjuntas entre pares de nodos, viene dado por:* $\left\lceil \frac{1}{2} \cdot \sum_{i \in V} r_i \right\rceil$ *. El uso de aristas paralelas esta permitido en la construcción.*

Couch-Frank resuelven el GSP_CU aún cuando no se permiten aristas paralelas, pero si se permite agregar nodos extra en la construcción. No se conoce en la literatura general, una solución a la versión del GSP_CU cuando se requiere la existencia de caminos de nodos disjuntos.

Problema (GSP_CU sin aristas paralelas)

Dado un grafo $G = (V, E)$ *y una matriz de requerimientos de conexión* $R = \{r_{ij}; i, j \in V\}$, *se desea encontrar un subgrafo con mínimo número de aristas que satisfaga que* $\forall i, j \in V$ *exista al menos* r_{ij} *caminos de aristas disjuntas, no permitiéndose aristas paralelas en la construcción.*

Este problema ha sido atacado solo cuando los requerimientos de conectividad entre nodos es uniforme, es decir $r_{ij} = k \quad \forall i, j \in V$ para algún

$k \geq 2$. En este caso, Harary [50] probó el siguiente Lema con la ayuda de un algoritmo polinomial.

Lema 6 (Harary)

Dado un $k \geq 2$ y $n \geq k+1$, el mínimo numero de aristas en un grafo k-nodo-conexo de n nodos sin aristas paralelas es: $\left\lceil \frac{k \cdot n}{2} \right\rceil$.

2.7.2 1.7.2 Costos 0-1

Si la matriz de costos es una matriz booleana, el problema GSP es conocido como *Augmentation Problem*. Su formulación es la siguiente.

2.7.2.1.1.1 Problema (Augmentatión Problem - AP)

Dado un grafo $G = (V, E)$, se pretende encontrar el mínimo número de aristas en $V \times V \setminus E$ (posiblemente usando aristas paralelas) tal que al agregarlas al grafo se satisface los requerimientos de conexión (de aristas o nodos) dados por la matriz $R \in \mathrm{Z}_+^{V \times V}$.

Este puede ser visto como un problema ECON o NCON sobre un grafo H y matriz de requerimientos de conexión R, donde H es un grafo completo, todas las aristas de E tienen costo 0, y todas las aristas de $(V \times V) \setminus E$ tienen costo 1.

Los casos en los cuales la solución requerida sea un grafo 2-arista-conexo o 2-nodo-conexo fue estudiada por Eswaran-Tarjan [24] y posteriormente por Rosenthal-Goldner [46]. El caso en el cual la solución deba ser un grafo k-arista-conexo fue estudiado por Watanabe-Nakamura [58], Ueno-Kajitani-Wada [55], y Cai-Sun [12]. Frank [26] resuelve el *Augmentation Problem* en el caso más general, donde se tiene una matriz de requerimientos de conexión $R \in \mathrm{Z}_+^{V \times V}$. Presenta algoritmos de resolución de orden polinomial.

El algoritmo de menor complejidad conocido hasta el momento, que incrementa la conectividad de aristas de un grafo a un nivel k en tiempo

polinomial fue propuesto por Naor [41]. Todos los demás algoritmos, excepto éste, permiten el uso de aristas paralelas.

2.7.2.1.1.1.1.1.1 Definición 7

Se define déficit de un conjunto de nodos A *como :* $def(A) = Min_{u \in A, v \notin A}\{r_{uv} - |\delta_G(A)|\}$, *donde* $\delta_G(A) = \{(i,j) \in E | i \in A, j \in V \setminus A\}$ *es el corte inducido por el conjunto de nodos* A *en el grafo* $G = (V, E)$ *y* $r_{uv} = min\{r_u, r_v\}$.

El déficit de A es la menor cantidad de aristas que hay que adicionar a $\delta_G(A)$ de forma de conectar A con todos los otros nodos. Si $\{A_i, i = 1..t\}$ es una familia de conjunto de nodos disjuntos, una cota inferior al número de aristas a agregar viene dada por: $\frac{1}{2}\sum_{i=1}^{t} def(A_i)$.

Bajo ciertas suposiciones, Frank [26] establece que esta cota inferior es suficiente para resolver el problema AP.

Teorema 8 (Frank)

Dado un grafo $G = (V, E)$ *y una matriz* $R = \{r_{ij}; i, j \in V\}$ *de requerimientos de conexión de aristas. Entonces vale lo siguiente:*

- *Si todo conjunto de nodos* $A \subseteq V$ *tiene déficit mayor que 1, entonces el mínimo número de aristas a agregar en* G *para satisfacer los requerimientos de conexión de aristas, es:* $Max_{A_1,\ldots,A_t\ disjuntos}\left\{\frac{1}{2}\sum_{i=1}^{t} def(A_i)\right\}$.
- *Si algún conjunto de nodos* A *tiene déficit* ≤ 1, *y todos los subconjuntos propios de* A *tienen déficit* ≤ 0, *entonces el mínimo número de aristas a agregar en* G *es* $def(A)$ *más el mínimo número de aristas a ser agregadas en* $G \setminus A$.

Esta permitido el uso de aristas paralelas en la aumentación [26].

2.7.3 1.7.3 Costos Generales

Cuando los costos de las aristas no son los mismos, existen algoritmos de orden polinomial para el GSP, en casos en los cuales las restricciones de conectividad de aristas (o de nodos) r_i cumplen:

- $r_i = 1$ para todos los nodos i (árbol de cubrimiento de costo mínimo),
- $r_i = k$ para exactamente dos nodos s y t, y $r_i = 0$ para todos los otros nodos i (*k-shortest path problem*),
- $r_i \in \{0,1\}$ para todos los nodos i, donde o bien el número de nodos del tipo 0 o el número de nodos del tipo 1 esta limitado. Este tipo de problema de Steiner fue resuelto por Lawler.

El SPN, puede ser resuelto en tiempo polinomial para la clase de grafos series-paralelos mediante un algoritmo recursivo. Esto fue mencionado primeramente por Takamisawa-Nishizeki-Saito [53] y probado formalmente por Wald-Colbourn [57].

Winter [50,59,60] desarrolló algoritmos lineales para los problemas 2ECON-2NCON en los casos en los cuales el grafo G es un grafo de Halin, serie-paralelo u outerplanar. Para los problemas 3ECON-3NCON, menciona algoritmos de orden lineal que los resuelven cuando G es un grafo de Halin.

Existen algoritmos exactos para costos generales aplicados a los problemas ECON y NCON, basados en *Cutting Plane* juntamente con *Branch-and-Bound*. Uno de ellos es el desarrollado por Christofides-Whitlock para el problema ECON; y otro por Chopra-Gorres para el problema 2ECON. En [50] se brinda una descripción más detallada de ambos algoritmos así como de los siguientes casos particulares.

Monma-Shallcros han desarrollado heurísticas para resolver los problemas 2ECON-2NCON. Para el 2NCON, Frederickson-JáJá propusieron una heurística con radio de performance igual a 3/2 en el peor caso, asumiendo que se cumple la desigualdad triangular en los costos de las aristas. Steiglitz-Weiner-Kleitman presentan una heurística para el caso en el cual la matriz $R \in \mathrm{Z}_+^{V \times V}$ exige la existencia de caminos de nodos-disjuntos entre pares de nodos, Ko-Monma dan heurísticas para el diseño de redes k-arista-conexas o k-nodo-conexas, Goemans-Bertsimas-Williamson brindan

una heurística cuando la matriz $R \in Z_+^{V \times V}$ exige la existencia de caminos de aristas-disjuntas entre pares de nodos.

2.8 1.8 Algoritmo Backtracking Exacto para el GSP

2.8.1 1.8.1 Introducción

Se presenta en este apartado un nuevo algoritmo para resolver el GSP en forma exacta. El algoritmo esta basado en la técnica de diseño de algoritmos conocida como Backtracking. Primeramente, se brinda una breve descripción de la metodología Backtracking, y luego una descripción detallada del algoritmo propuesto.

2.8.2 1.8.2 Backtracking

Backtracking es una de las técnicas más generales de diseño de algoritmos. Muchos problemas donde se requiere buscar dentro de un espacio de soluciones, una solución que satisfaga ciertas restricciones, puede ser usada la formulación Backtracking como técnica de diseño de algoritmo.

Cuando se aplica la metodología Backtracking en el diseño de un algoritmo para un determinado problema, las soluciones factibles deben poder expresarse como una n-tupla $(x_1, x_2, ..., x_n)$ donde los x_i son elegidos de algún conjunto finito S_i. Frecuentemente el problema a ser resuelto tiene asociado una función objetivo $P(x_1, x_2, ..., x_n)$ a ser maximizada, minimizada, o satisfecha. Supóngase que m_i es la cardinalidad del conjunto S_i. Entonces hay $m = m_1 \times m_2 ... \times m_n$ n-tuplas las cuales son candidatas a satisfacer la función P. El método de fuerza bruta evalúa las m n-tuplas y guarda la mejor como solución óptima. Los algoritmos de Backtracking tienen la virtud de poder descartar familias de soluciones durante la búsqueda de la solución óptima. La idea básica del método Backtracking, es ir construyendo el vector solución de a una componente por vez y usar funciones de cota $B_i(x_1, x_2, ..., x_i)$ para testear si el vector formado hasta el momento tiene

alguna chance de formar parte del vector solución óptima. La mayor ventaja de este método es la siguiente: si el vector parcial $(x_1, x_2, ..., x_i)$ construido hasta el momento no puede formar parte de una solución óptima, entonces los $m_{i+1} \times ... \times m_n$ posibles restos de vector, pueden ser descartados enteramente.

Muchos de los problemas que pueden ser resueltos usando Backtracking requieren que las soluciones satisfagan cierto conjunto de restricciones. Para cualquier problema, estas restricciones pueden ser divididas en dos categorías: explícitas e implícitas. Las restricciones explícitas son reglas las cuales restringen a cada x_i a tomar un valor en un determinado conjunto, además pueden depender o no de una instancia I del problema a ser resuelto. Todas las posibles n-tuplas que satisfacen las restricciones explícitas definen un espacio de soluciones factibles para I. Las restricciones implícitas determinan cuales de las n-tuplas en el espacio de soluciones de I satisfacen la función de criterio. De esto modo, las restricciones implícitas describen la forma en la cual deben estar relacionadas las x_i unas con las otras.

Los algoritmos Backtracking determinan la solución óptima, mediante búsquedas sistemáticas en el espacio de soluciones de una instancia I del problema. La búsqueda se realiza usando una organización del tipo árbol para el espacio de soluciones. En [47] se encuentra una descripción detallada de la metodología Backtracking.

2.8.3 1.8.3 Algoritmo Backtracking para el GSP

Dado un grafo $G = (V, E)$ en las condiciones del GSP, se denota al espacio de soluciones factibles de la instancia del problema como Γ_{GSP}. El mismo viene dado por el conjunto de subgrafos de G que satisfacen la matriz de requerimientos R; o sea $G_s(N, A) \in \Gamma_{GSP}$ si $T \subseteq N \subseteq V$, $A \subseteq E$ y además $\forall i, j \in T$ existen r_{ij} caminos de aristas disjuntas en $G_s(N, A)$.

El algoritmo propuesto, consta de dos procedimientos mutuamente recursivos, los cuales analizan el espacio de soluciones factibles Γ_{GSP} del

GSP en búsqueda de la solución óptima, descartando mediante la aplicación de funciones de cota, ramas del árbol de soluciones correspondiente a la instancia del problema.

Se define el conjunto de aristas $A_{sol} \subseteq E$, el conjunto de nodos $N_{sol} \subseteq V$, y la variable *min_costo* (inicializada en INF) como globales al algoritmo y es donde se almacena la mejor solución encontrada hasta el momento por el algoritmo. También se asume como globales al algoritmo las representaciones del grafo $G = (V, E)$ original, la matriz de costos c asociada a las aristas, la matriz de requerimientos de conexión R, y el conjunto de nodos terminales T.

El algoritmo consta de dos partes: un procedimiento denominado "*ALG_G_STEINER*" y un segundo procedimiento auxiliar denominado "*ALG_G_A*". Los pseudocódigos de dichos procedimientos son los siguientes.

```
Procedure ALG_G_STEINER(A,N,i,costo,num_term);
Begin
1   OK=ACT_SOL(A,N,costo,num_term);
2   if (i≤n) and (costo<min_costo) and (not(OK)) then                (c_1)
3     if (i∉ T) then                                                  (c_2)
4        ALG_G_STEINER(A,N,i+1,costo,num_term);
5        if Test_Nodo_Steiner(i,N,A) then                            (c_3)
6           ALG_G_A(A,N∪{i},i,1,costo,num_term,0,2);
7        endif;
8     else
9        r_max_c=MaxReqCon(i);
10       if Test_Nodo_Terminal(i,N,A) then                           (c_4)
11          ALG_G_A(A,N∪{i},i,1,costo,num_term+1,0,r_max_c);
12       endif;
13    endif;
14 endif;
End;
```

```
Procedure ALG_G_A(A,N,i,j,costo,num_term,cant_a,r);
Begin
1   ady_may=AdyMayores(i,j);
2   c_min=MinRestoAdy(i,j);
3   resto_r=MAX(r-cant_a,0);
4   if (cant_a+ady_may≥ r) and (costo+(resto_r· c_min))<min_costo) then   (c_1)
5     if (j< i) then                                                      (c_2)
6        if (j∈N) and ((i,j)∈ G) then                                     (c_3)
7           ALG_G_A(A∪{(i,j)},N,i,j+1,costo+c_ij,num_term,cant_a+1,r);
8           ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r);
9        else
10          ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r);
11       endif;
12    else
```

Seguidamente se brinda la descripción de las funciones y procedimientos auxiliares utilizadas por el algoritmo. Se asume que los grafos se representan mediante matrices de adyacencias, y que los nodos están numerados en el rango $1..n$, con $n = |V|$.

2.8.3.1.1.1.1 Función "*ACT_SOL*"

La función "*ACT_SOL*" recibe como parámetros de entrada, un conjunto de aristas $A \subseteq E$, un conjunto de nodos $N \subseteq V$, el costo del subgrafo $G_s(N, A)$, y el número de nodos terminales presentes en el conjunto N. Con estos parámetros, chequea utilizando la primitiva "*FACT_GSP*" si se tiene una mejor solución factible al GSP que la solución actual. En caso afirmativo, se almacena la nueva solución factible para comparar. Un posible pseudocódigo para "*ACT_SOL*" es el siguiente.

```
Function ACT_SOL(A,N,costo,num_term):boolean;
Begin
  ACT_SOL=FALSE;
  if (num_term=|T|) and (costo<min_costo) then
     if FACT_GSP(A,N) then
        A_sol=A;
        N_sol=N;
        min_costo=costo;
        ACT_SOL=TRUE;
     endif;
  endif;
End;
```

Donde $FACT\_GSP(A,N) = \begin{cases} TRUE, si\, G_s(N,A) \in \Gamma_{GSP} \\ FALSE, sino \end{cases}$.

Existe una implementación eficiente de la primitiva "*FACT_GSP*" basada en una generalización del algoritmo de Ford-Fulkerson de Flujo Máximo-Corte Mínimo, donde se tienen múltiples nodos fuente y nodos destino, dados por el conjunto de nodos terminales T, y se les asocia capacidad 1 a todas las aristas del grafo. Dado el grafo $G_s(N,A)$, el algoritmo determina $\forall i, j \in T$ el mínimo número de aristas que separan ambos nodos en el grafo; aplicando el Teorema de Menger, esto es equivalente a determinar la cantidad máxima de caminos de aristas disjuntas que unen el nodo i con el nodo j en $G_s(N,A)$ [45].

Función "*AdyMayores*"

La función "*AdyMayores*" recibe como entradas dos nodos $i, j \in V$, y devuelve la cantidad de nodos adyacentes al nodo i que son mayores que j en la representación matricial del grafo $G = (V,E)$.

Función "*MinRestoAdy*"

Esta función recibe como parámetros de entrada dos nodos $i, j \in V$, y devuelve el menor costo de una arista $(i,k) \in E$ tal que $k \geq j$.

Función "*MaxReqCon*"

La función "*MaxReqCon*" recibe como entrada un nodo $i \in T$, y devuelve el mayor requerimiento de conexión en R, que tiene como extremo al nodo terminal i. Es decir, se calcula el valor $Max\{r_{ij}, \forall j \in T\}$.

Función "*Test_Nodo_Steiner*"

Recibe como parámetros de entrada un nodo de Steiner $i \in V \setminus T$, un conjunto de aristas $A \subseteq E$, y un conjunto de nodos $N \subseteq V$. Devuelve TRUE si el nodo de Steiner i puede llegar a formar parte de una solución factible del espacio Γ_{GSP} con menor costo que la actual solución, construida a partir de $G_s(N,A)$, agregando nodos que están en el conjunto $V^{(\geq i)} = \{v \in V / v \geq i\}$ y

aristas de $(E \setminus E^{(<i)})$, donde $E^{(<i)} = \{(u,v) \in E / u,v < i\}$; devuelve FALSE en caso contrario. Los diferentes tests realizados por esta función se detallarán en Proposición 10.

Función "*Test_Nodo_Terminal*"

Recibe como parámetros de entrada un nodo terminal $i \in T$, un conjunto de aristas $A \subseteq E$, y un conjunto de nodos $N \subseteq V$. Devuelve TRUE si existe la posibilidad de encontrar una mejor solución factible en el espacio Γ_{GSP}, formada a partir del subgrafo $G_s(N \cup \{i\}, A)$ agregando nodos del conjunto $V^{(>i)} = \{v \in V / v > i\}$ y aristas de $(E \setminus E^{(<i)})$; devuelve FALSE en caso contrario. Al igual que "*Test_Nodo_Steiner*", los test realizados por la función "*Test_Nodo_Terminal*" se darán en Proposición 10.

Dado un grafo $G = (V,E)$ en las condiciones del enunciado del GSP, hay que demostrar que el algoritmo propuesto da la solución óptima del GSP analizando el espacio de soluciones factibles Γ_{GSP}. Antes de demostrar la correctitud del algoritmo, se brinda una definición y se presentan algunas proposiciones previas.

Definición 9

Dado un grafo $G = (V,E)$, dados dos conjuntos de nodos $V_1, V_2 \subseteq V$ tales que $V_1 \cap V_2 = \varnothing$, $V_1 \neq \varnothing$, $V_2 \neq \varnothing$, y un conjunto de aristas $E_1 \subseteq E$ con extremos en V_1; se define $SG(V_1,V_2,E_1) = \{G_s(X,Y) / X = V_1 \cup W, \forall W \subseteq V_2; Y = E_1 \cup E_3, \forall E_3 \subseteq E_2\}$ donde $E_2 = \{(i,j)/(i,j) \in E, i,j \in X\}$.

Proposición 10 (Test del Nodo Actual)

Dado un grafo $G = (V,E)$ en las condiciones del problema GSP. Sea $G_s(N,A)$ un subgrafo de G e $i \in V$ un nodo al cual se le denominará nodo actual. Asumiendo que:

- $T \subseteq N \subseteq V$,
- $\forall k \in (N \setminus T)$ *se cumple* $k < i$,
- $A \subseteq E^{(<i)}$ *, donde* $E^{(<i)} = \{(u,v) \in E \,/\, u,v < i\}$,
- $\cos to$ *es el costo del subgrafo* $G_s(N,A)$,
- $G_s(N,A)$ *no satisface los requerimientos de conexión de la matriz* R ,
- $min\_\cos to$ *es una variable cuyo valor es el costo de una solución factible de* Γ_{GSP} *o INF.*

Existe una serie de tests que permiten considerar (o descartar) la existencia de una solución factible de menor costo que "min_costo" (costo de la solución actual), en el espacio de subgrafos $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$ *. Cada uno de estos tests se analizan en la demostración de la proposición.*

Dem:

Sean las siguientes definiciones:

- $T^c = \{t \in T \,/\, t > i\}$,
- $c^i_{min\_N}$ es el costo del camino de menor costo que conecta el nodo i con algún nodo de N considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$, donde $V^{(\geq i)} = \{v \in V \,/\, v \geq i\}$,
- $c^i_{min\_1}$ y $c^i_{min\_2}$ los dos menores costos de aristas con extremos i ,
- $r^1_{min} = min\{r_{kj} \,/\, k \in T^c ; j \in T, j < i\}$,
- $r^2_{min} = min\{r_{kj} \,/\, k, j \in T^c\}$,
- $c^i_{min\_(N \setminus T^c)}$ el costo del camino de menor costo que une el nodo i con un nodo de $N \setminus T^c$ considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$,
- $c^i_{min\_T^c}$ el mínimo costo de un camino que une el nodo i con un nodo de T^c en el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$,
- $r^i_{Max} = Max\{r_{ik}, k \in T\}$,
- $c^i_{min(k)}$ = el k-ésimo menor costo de una arista con extremo en i ,
- $\Gamma^i_{GSP}(N,A) = SG(N \cup \{i\}, \{i..n\} \setminus T, A) \cap \Gamma_{GSP}$.

Seguidamente se presenta una serie de condiciones que se deben chequear, antes de realizar la búsqueda de soluciones factibles de menor costo que la actual en el espacio de subgrafos $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$. Se demuestra en cada caso, que las condiciones enunciadas se deben satisfacer, si se pretende buscar una solución de menor costo en el espacio de subgrafos $\Gamma^i_{GSP}(N, A)$.

Caso A (se cumple: $i \in (V \setminus T)$)

Caso A1 ($T^c = \varnothing$)

1) $g(i) \geq 2$,
2) $c^i_{min\_1} + c^i_{min\_2} + \cos to < min\_\cos to$,
3) $(2 \cdot c^i_{min\_N}) + \cos to < min\_\cos to$.

1) Como $T^c = \varnothing$ y $G_s(N, A) \notin \Gamma_{GSP}$, una solución de $\Gamma^i_{GSP}(N, A)$ de mínimo costo debe tener un *Hpath* sobre $G_s(N, A)$ que contenga al nodo de Steiner i (de no ser así, el nodo i estaría conectado a $G_s(N, A)$ en la solución, mediante un camino de nodos Steiner, lo cual implica que el costo no sería el mínimo). Luego, se tiene que cumplir que $g(i) \geq 2$.
2) Por 1), una solución de $\Gamma^i_{GSP}(N, A)$ de mínimo costo debe tener un *Hpath* sobre $G_s(N, A)$ que contenga al nodo de Steiner i. Una cota inferior para un *Hpath* sobre $G_s(N, A)$ de costo mínimo que contiene al nodo de Steiner i, viene dada por $c^i_{min\_1} + c^i_{min\_2}$. Si $c^i_{min\_1} + c^i_{min\_2} + \cos to \geq min\_\cos to$, cualquier solución de $\Gamma^i_{GSP}(N, A)$ tiene costo mayor o igual al de la mejor solución factible encontrada hasta el momento, por tanto se descarta la existencia de mejores soluciones factibles en $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$.
3) Si se verifica 1) y 2), se calcula el costo mínimo de conectar el nodo i con los nodos de N en el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$. Al igual que en 1) y 2), una solución de $\Gamma^i_{GSP}(N, A)$ de mínimo costo debe tener un *Hpath*

sobre $G_s(N,A)$ que contenga al nodo de Steiner i. Una cota inferior al costo mínimo de ese *Hpath* viene dado por $(2 \cdot c^i_{min\_N})$; entonces, si $(2 \cdot c^i_{min\_N}) + \cos to \geq min\_\cos to$ se descarta la existencia de mejores soluciones factibles en $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$.

Caso A2 $(T^c \neq \varnothing)$

1) Si $r^1_{min} > 0, r^2_{min} = 0$, se chequea $c^i_{min\_(N \setminus T^c)} + c^i_{min\_T^c} + \cos to < min\_\cos to$,
2) Si $r^2_{min} > 0, r^1_{min} = 0$, se chequea $(2 \cdot c^i_{min\_T^c}) + \cos to < min\_\cos to$.
3) Si $r^1_{min} > 0, r^2_{min} > 0$, se chequea:

$$Min\left(c^i_{min\_(N \setminus T^c)} + c^i_{min\_T^c}, 2 \cdot c^i_{min\_T^c}\right) + \cos to < min\_\cos to .$$

1) Supóngase que $r^1_{min} > 0, r^2_{min} = 0$. Si i forma parte de una solución factible de $\Gamma^i_{GSP}(N,A)$ de mínimo costo, necesariamente debe pertenecer a un camino que une un nodo de T^c con un nodo de $(N \setminus T^c)$ (de lo contrario la solución factible no seria de mínimo costo en $\Gamma^i_{GSP}(N,A)$, pues todos los caminos que unen un nodo de T^c con un nodo de $(N \setminus T^c)$ no pasarían por i, pudiéndose eliminar el nodo i, manteniéndose la factibilidad de la solución). El costo mínimo de conectar el nodo i con un nodo de $(N \setminus T^c)$ viene dado por $c^i_{min\_(N \setminus T^c)}$ y el costo mínimo de conectar el nodo i con un nodo de T^c viene dado por $c^i_{min\_T^c}$; ambos considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$. Entonces, el costo mínimo de un camino que pasa por el nodo i y une un nodo de T^c con un nodo de $(N \setminus T^c)$ utilizando aristas del subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$, viene dado por $c^i_{min\_(N \setminus T^c)} + c^i_{min\_T^c}$. De esta manera, si $c^i_{min\_(N \setminus T^c)} + c^i_{min\_T^c} + \cos to \geq min\_\cos to$, no es necesario buscar mejores soluciones factibles en $\Gamma^i_{GSP}(N,A)$; mientras que si

$c^{i}_{min\_(N\setminus T^c)} + c^{i}_{min\_T^c} + \cos to < min\_\cos to$, puede que existan soluciones factibles de menor costo que la actual en $\Gamma^{i}_{GSP}(N, A)$.

2) Supóngase que $r^{2}_{min} > 0, r^{1}_{min} = 0$. Si i forma parte de una solución factible de $\Gamma^{i}_{GSP}(N, A)$ de mínimo costo, necesariamente debe pertenecer a un camino que une un nodo de T^c con otro nodo de T^c (de lo contrario la solución factible no seria de mínimo costo en $\Gamma^{i}_{GSP}(N, A)$, pues todos los caminos que unen un nodo de T^c con otro nodo de T^c no pasarían por i, pudiéndose eliminar el nodo i, manteniéndose la factibilidad de la solución). El costo mínimo de conectar el nodo i con un nodo de T^c viene dado por $c^{i}_{min\_T^c}$, esto es considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$. Entonces, una cota inferior al costo mínimo de un camino que pasa por el nodo i y une un nodo de T^c con un otro nodo de T^c utilizando aristas del subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$, viene dada por $(2 \cdot c^{i}_{min\_T^c})$. Si se verifica la condición $(2 \cdot c^{i}_{min\_T^c}) + \cos to \geq min\_\cos to$, no es necesario buscar mejores soluciones factibles en $\Gamma^{i}_{GSP}(N, A)$; mientras que si $(2 \cdot c^{i}_{min\_T^c}) + \cos to < min\_\cos to$, puede que existan soluciones factibles de menor costo que la actual en $\Gamma^{i}_{GSP}(N, A)$.

3) Veamos ahora el caso en el cual $r^{1}_{min} > 0, r^{2}_{min} > 0$. Si i forma parte de una solución factible de $\Gamma^{i}_{GSP}(N, A)$ de mínimo costo, o bien pertenece a un camino que une un nodo de T^c con un nodo de $(N \setminus T^c)$, o bien a un camino que une dos nodos de T^c ; en ambos casos considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$ (de lo contrario, se tendría un camino de nodos de Steiner que puede ser eliminado manteniendo la factibilidad de la solución, y por tanto la solución no sería de mínimo costo en el subespacio de soluciones $\Gamma^{i}_{GSP}(N, A)$). Una cota inferior al costo mínimo de un camino que une dos nodos de T^c o un nodo de T^c con un nodo de $(N \setminus T^c)$ considerando el subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$, viene dada por $Min\left(c^{i}_{min\_(N\setminus T^c)} + c^{i}_{min\_T^c}, 2 \cdot c^{i}_{min\_T^c}\right)$. Entonces, si

$Min\left(c^i_{min\_(N\setminus T^c)} + c^i_{min\_T^c}, 2 \cdot c^i_{min\_T^c}\right) + \cos to \geq min\_\cos to$ puedo descartar las soluciones factibles del subespacio $\Gamma^i_{GSP}(N, A)$, pues éstas no serían de costo mínimo. En cambio, si se verifica $Min\left(c^i_{min\_(N\setminus T^c)} + c^i_{min\_T^c}, 2 \cdot c^i_{min\_T^c}\right) + \cos to < min\_\cos to$, no se debe descartar la existencia de mejores soluciones factibles que la actual en el subespacio de soluciones $\Gamma^i_{GSP}(N, A)$.

Caso B (se cumple: $i \in T$)

1) $g(i) \geq r^i_{Max}$,
2) $\sum_{k=1}^{r^i_{Max}} c^i_{min(k)} + \cos to < min\_\cos to$,
3) $(r^i_{Max} \cdot c^i_{min\_N}) + \cos to < min\_\cos to$.

1) Sea $g(i) = g_i$. Entre el nodo terminal i y cualquier otro nodo terminal de T, existen a lo sumo g_i caminos de aristas disjuntas. Si $g_i < r^i_{Max}$, existe al menos un nodo terminal para el cual el requerimiento de conexión con el nodo terminal i no puede ser satisfecho en el grafo G y por tanto por ningún subgrafo del espacio $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$. Si se satisface $g(i) \geq r^i_{Max}$, se chequean las condiciones 2) y 3) para analizar la existencia de mejores soluciones factibles que la actual en el subespacio de soluciones $\Gamma^i_{GSP}(N, A)$.
2) Si $r^i_{Max} > 0$, existe un nodo terminal $k \in T$ tal que sus requerimientos de conexión con el nodo terminal i no son satisfechos en $G_s(N, A)$. Una cota inferior al mínimo costo de conectar mediante r^i_{Max} caminos de aristas disjuntas los nodos terminales i y k considerando subgrafos del espacio $SG(N \cup \{i\}, \{i..n\} \setminus T, A)$, viene dada por $\sum_{k=1}^{r^i_{Max}} c^i_{min(k)}$, que es la suma de los r^i_{Max} menores costos de aristas con extremos en el nodo i. Si

$\sum_{k=1}^{r^i_{Max}} c^i_{min(k)} + \cos to \geq min\_\cos to$ se pueden descartar las soluciones factibles del subespacio $\Gamma^i_{GSP}(N, A)$, pues éstas no serían de costo mínimo; mientras que si $\sum_{k=1}^{r^i_{Max}} c^i_{min(k)} + \cos to < min\_\cos to$ se chequea 3) para analizar la existencia de mejores soluciones factibles que la actual.

3) Otra cota inferior al mínimo costo de conectar mediante r^i_{Max} caminos de aristas disjuntas el nodo terminal i con el nodo terminal para el cual se tiene el mayor requerimiento de conexión, viene dada por $(r^i_{Max} \cdot c^i_{min\_N})$. Esto es, considerando aristas del subgrafo $G_s(N \cup V^{(\geq i)}, E \setminus E^{(<i)})$ para conectar el nodo i con el subgrafo actual $G_s(N, A)$ de forma de satisfacer el requerimiento de conexión r^i_{Max} . Entonces, si $(r^i_{Max} \cdot c^i_{min\_N}) + \cos to \geq min\_\cos to$ se descartan las soluciones factibles del subespacio $\Gamma^i_{GSP}(N, A)$, pues éstas no son de costo mínimo; mientras que si se satisface $(r^i_{Max} \cdot c^i_{min\_N}) + \cos to < min\_\cos to$ existe la posibilidad de encontrar mejores soluciones factibles en el subespacio $\Gamma^i_{GSP}(N, A)$.

L.Q.Q.D.

Proposición 11

Sean $i, j \in V$ *, tales que* $1 \leq j \leq i$ *.*

Asumiendo que:

- $T \subseteq N \subseteq V$ *,*
- $A \subseteq E$ *con los extremos de las aristas en* N *,*
- *costo es el costo del subgrafo* $G_s(N, A)$ *,*
- $i \in N$ *es el nodo actual a ser analizado,*
- $j \in V$ *es el nodo actual a ser analizado como adyacente al nodo* i *(se supone que no existe* $(i, k) \in A / k \geq j$ *),*
- *num_term es el número de nodos terminales analizados hasta el momento,*

- *cant_a es la cantidad de aristas* $(i,k) \in A / k < j$ *,*
- *r es un entero que indica un cierto requerimiento de conexión para el nodo* i *,*
- *min_costo es una variable global al algoritmo y vale INF o el costo de alguna solución factible del espacio* Γ_{GSP} *,*
- *"ALG_G_STEINER(A∪B,N,i+1,costo+COSTO(B),num_term)" analiza el espacio de subgrafos* $SG(N,\{i+1..n\} \setminus T, A \cup B)$ *con* $B \subseteq C^{(j)} = \{(i,k) \in E / k \in N, j \le k \le i\}$ *en búsqueda de una mejor solución factible (con costo menor a min_costo) y actualiza la mejor solución en caso de encontrar una solución factible de menor costo.*

Al invocar "ALG_G_A(A,N,costo,i,j,num_term,cant_a,r)" se analiza el espacio de subgrafos $H = \bigcup_{\forall B \subseteq C^{(j)}} SG(N,\{i+1..n\} \setminus T, A \cup B)$ *y de existir soluciones factibles al GSP con costo menor que min_costo, el algoritmo encuentra y almacena la solución factible de menor costo del espacio* H *como mejor solución.*

Dem:

La demostración se hará por inducción hacia atrás en el valor de j.

Cuando se ejecuta (1) en "*ALG_G_A*", la función "*AdyMayores*", calcula la cantidad de nodos adyacentes al nodo i que son mayores que j (o sea la cantidad de nodos adyacentes al nodo i que todavía no han sido considerados) y se almacena el valor en la variable *ady_may*. Al ejecutarse (2), la función "*MinRestoAdy*" calcula el valor $c\_min = Min\{c_{ik} / (i,k) \in E, k \ge j\}$.

En (3) se calcula cuantas aristas hay que agregar desde el nodo i para alcanzar el valor r y se almacena este valor en la variable *resto_r*.

Luego se chequea la condición (c_1) pudiéndose dar las siguientes situaciones.

Condición (c_1) es FALSE

Si *(costo+(resto_r· c_min))≥min_costo)* significa que el costo del subgrafo $G_s(N,A)$ mas el costo mínimo que puede tener el agregar al menos la

cantidad de aristas r (requerimiento mínimo de conexión para el nodo i) desde el nodo i, es mayor o igual que el de la mejor solución factible encontrada hasta el momento para el GSP, por tanto no tiene sentido agregar más aristas al subgrafo para obtener una mejor solución factible. De esta manera el algoritmo "*ALG_G_A*" finaliza su ejecución.

Si *(cant_a+ad_may< r)*, entonces por más que se agregue al subgrafo $G_s(N,A)$ nuevos nodos del conjunto $V^{(>i)} = \{v \in V / v > i\}$ y aristas del conjunto $(E \setminus E^{(\leq i)}) \cup A^{(\geq j)}$ (donde $E^{(\leq i)} = \{(u,v) \in E / u,v \leq i\}$ y $A^{(\geq j)} = \{(i,k) \in E / k \geq j\}$), en el subgrafo resultante, el grado del nodo i será menor que el requerimiento de conexión r. Por tanto el nodo i no puede formar parte de una solución factible para de GSP, y el algoritmo "*ALG_G_A*" finaliza su ejecución.

Condición (c_1) es TRUE

Se asume ahora que *(cant_a+ady_may≥ r)* y *(costo<min_costo)*, y se analizan los posibles valores que puede tomar el nodo j.

Paso Base

Supóngase que $j = i$, entonces se tiene que $C^{(j)} = \varnothing$ por definición del conjunto $C^{(j)}$. Luego la condición (c_2) es FALSE en "*ALG_G_A*" y el algoritmo ejecuta (11) invocando a "*ALG_G_STEINER(A,N,i+1,costo,num_term)*". Por hipótesis, como resultado de ésta invocación, se analiza el espacio de subgrafos $SG(N,\{i+1..n\} \setminus T, A)$ en búsqueda de una mejor solución factible del GSP (con costo menor a *min_costo*) y se actualiza la mejor solución en caso de encontrar una solución factible de menor costo. Además como $C^{(j)} = \varnothing$ se cumple que $H = SG(N,\{i+1..n\} \setminus T, A)$.

Paso Inductivo

H.I.) La Proposición se cumple para $1 \leq k < j \leq i$.

T.I.) La Proposición se cumple para $1 \leq j = k < i$.

Supóngase que $1 \le j = k < i$, entonces la condición (c_2) es TRUE en "*ALG_G_A*" y el algoritmo chequea la condición (c_3). Se analizan las dos situaciones posibles.

Condición (c_3) es TRUE

El algoritmo ejecuta en (5) y (6) las invocaciones:

"*ALG_G_A(A∪{(i,j)},N,i,j+1,costo+c_{ij},num_term,cant_a+1,r)*" y

"*ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r)*".

Por H.I., se analizan los espacios de subgrafos:

$H_1 = \bigcup_{\forall B \subseteq C^{(j+1)}} SG(N, \{i+1..n\} \setminus T, A \cup \{(i,j)\} \cup B)$ y

$H_2 = \bigcup_{\forall B \subseteq C^{(j+1)}} SG(N, \{i+1..n\} \setminus T, A \cup B)$.

Como $C^{(j)} = C^{(j+1)} \cup \{(i,j) \in E\}$, se cumple la siguiente relación entre los espacios de subgrafos:

$$H_1 \cup H_2 = \left(\bigcup_{\forall B \subseteq C^{(j)} / (i,j) \in B} SG(N, \{j+1..n\} \setminus T, A \cup B) \right) \cup \left(\bigcup_{\forall B \subseteq C^{(j)} / (i,j) \notin B} SG(N, \{j+1..n\} \setminus T, A \cup B) \right) =$$

$$= \bigcup_{\forall B \subseteq C^{(j)}} SG(N, \{i+1..n\} \setminus T, A \cup B) = H$$

El algoritmo "*ALG_G_A*" analiza el espacio de subgrafos $H_1 \cup H_2$ (por H.I.). En caso que existan soluciones factibles al GSP con costo menor que *min_costo*, almacena la solución factible de menor costo como mejor solución. Como $H_1 \cup H_2 = H$, se tiene entonces la T.I.

Condición (c_3) es FALSE

El algoritmo ejecuta en (8) la invocación:

"*ALG_G_A(A,N,i,j+1,costo,num_term,cant_a,r)*".

Por H.I. se analiza el espacio de subgrafos:

$H_2 = \bigcup_{\forall B \subseteq C^{(j+1)}} SG(N, \{i+1..n\} \setminus T, A \cup B)$

Debido a que $j \notin N$ y/o $(i,j) \notin E$, entonces se cumple en este caso que $C^{(j)} = C^{(j+1)}$. Por tanto la se cumple la siguiente igualdad:

$$H_2 = \bigcup_{\forall B \subseteq C^{(j+1)}} SG(N, \{i+1..n\} \setminus T, A \cup B) = \bigcup_{\forall B \subseteq C^{(j)}} SG(N, \{i+1..n\} \setminus T, A \cup B) = H \ .$$

Por H.I. se sabe que de existir soluciones factibles al GSP en H_1, con costo menor que *min_costo*, el algoritmo "*ALG_G_A*" encuentra y almacena la solución factible de menor costo como mejor solución. Luego como se cumple que $H_1 = H$, entonces se tiene la T.I.

L.Q.Q.D.

Proposición 12

Sea $G = (V, E)$ un grafo en las condiciones del GSP, $T \subseteq V$ un conjunto de nodos que llamaremos terminales, $T \subseteq N \subseteq V$ un conjunto de nodos y $A \subseteq E$ un conjunto de aristas con extremos en N. Supóngase además que:

- *i es el nodo actual a ser analizado,*
- *si existe $v \in V / v > i$, entonces $v \in T$,*
- *costo es el costo del subgrafo $G_s(N, A)$ de G,*
- *num_term es el número de nodos terminales cuyas etiquetas son menores o iguales que i,*
- *r_{min} es el menor valor de la matriz R de requerimientos de conexión,*
- *inicialmente $N_{sol} = \varnothing$, $A_{sol} = \varnothing$ y min_costo=INF o el costo de alguna solución factible del espacio Γ_{GSP}.*

Entonces, al invocar "ALG_G_STEINER(A,N,i,costo,num_term,r_{min})", se analiza el espacio de subgrafos $SG(N, \{i..n\} \setminus T, A)$ y de existir soluciones factibles de menor costo que la actual, el algoritmo almacena la de menor costo como mejor solución.

Dem:

La demostración se hará por inducción hacia atrás en el valor de i.

Paso Base

Se cumple vacíamente.

Paso Inductivo

H.I.) La Proposición se cumple para $1 \leq k < i \leq n$.

T.I.) La Proposición se cumple para $1 \leq k = i < n$.

Caso A

Se analiza primero el caso $G_s(N,A) \in \Gamma_{GSP}$.

Inicialmente, al ejecutarse (1) en "*ALG_G_STEINER*", la función "*ACT_SOL*" verifica si el subgrafo $G_s(N,A)$ es solución factible al GSP. Como se supone que $G_s(N,A) \in \Gamma_{GSP}$, y además *min_costo=INF* por hipótesis, la función "*ACT_SOL*" asigna: $A_{sol} = A$, $N_{sol} = N$ y *min_costo=costo*; retornando el valor TRUE indicando que se actualizo la mejor solución. En (2), la condición (c_1) es FALSE pues el valor de la variable OK es TRUE. Luego finaliza la ejecución del algoritmo.

Se cumple entonces que $G_s(N,A)$ es la mejor solución factible al GSP considerando el espacio de subgrafos $SG(N,\{i..n\} \setminus T, A)$. Cualquier otra solución factible de este espacio tiene costo mayor o igual al de $G_s(N,A)$.

Caso B

Supóngase que $G_s(N,A) \notin \Gamma_{GSP}$.

En (1), el algoritmo "*ALG_G_STEINER*" invoca a la función "*ACT_SOL*" para verificar la factibilidad de $G_s(N,A)$. Como se asume que $G_s(N,A) \notin \Gamma_{GSP}$, el valor retornado por la función "*ACT_SOL*" es FALSE. Por hipótesis: *(i<n)* y *(costo<min_costo)*; como además OK tiene el valor TRUE, la condición (c_1) es TRUE y el algoritmo chequea en (3) la condición (c_2). Se estudia a continuación ambos casos.

Condición (c_2) es TRUE en "*ALG_G_STEINER*"

En este caso se cumple $i \notin T$.

Al ejecutarse (4) se realiza la invocación "*ALG_G_STEINER(A,N,i+1,costo,num_term,0)*". Por H.I. el algoritmo analiza el espacio de subgrafos $SG(N,\{i+1..n\} \setminus T, A)$ y si existen soluciones factibles de menor costo que la actual, el algoritmo almacena la de menor costo como

mejor solución. Luego, en (5) , el algoritmo invoca la función "*Test_Nodo_Steiner*". Por Proposición 10, si el nodo i puede llegar a formar parte de una solución factible al GSP construida a partir de $G_s(N,A)$, agregando nodos de $V^{(\geq i)} = \{v \in V / v \geq i\}$ y aristas de $(E \setminus E^{(<i)})$, donde $E^{(<i)} = \{(u,v) \in E / u,v < i\}$, con menor costo que la actual solución, entonces el valor retornado por la función "*Test_Nodo_Steiner*" es TRUE y se cumple la condición (c_3) en "*ALG_G_STEINER*", de lo contrario la función retorna FALSE y la condición (c_3) en "*ALG_G_STEINER*" no se cumple.
Seguidamente se analizan ambos casos.

Condición (c_3) es TRUE en "*ALG_G_STEINER*"

En (6) el algoritmo realiza la invocación "*ALG_G_A(A,N∪{i},i,1,costo,num_term,2)*". Por H.I. y Proposición 11, se analiza el espacio de subgrafos $H = \bigcup_{\forall B \subseteq C^{(1)}} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B)$, con $C^{(1)} = \{(i,k) \in E / k \in N, 1 \leq k \leq i\}$, y de existir soluciones factibles al GSP con costo menor que *min_costo*, el algoritmo "*ALG_G_A*" encuentra y almacena la solución factible de menor costo del espacio H como mejor solución.
Como además $SG(N,\{i+1..n\} \setminus T, A) \cup H = SG(N,\{i..n\} \setminus T, A)$ se tiene entonces la T.I.

Condición (c_3) es FALSE en "*ALG_G_STEINER*"

El algoritmo finaliza, cumpliéndose que la mejor solución factible al GSP considerando el espacio de subgrafos $SG(N,\{i..n\} \setminus T, A)$, es la mejor solución factible al GSP considerando el espacio de subgrafos $SG(N,\{i+1..n\} \setminus T, A)$.

Condición (c_2) es FALSE en "*ALG_G_STEINER*"

Se cumple $i \in T$.
Se ejecuta (9) calculándose el valor $r_{max\_c} = Max\{r_{ij}, \forall j \in T\}$. En (10), se invoca a la función "*Test_Nodo_Terminal*", la cual por Proposición 10, devuelve TRUE si existe la posibilidad de construir una solución factible para el GSP, a partir de $G_s(N,A)$, agregando nodos de $V^{(\geq i)} = \{v \in V / v \geq i\}$ y aristas de

$(E \setminus E^{(<i)})$, con un costo menor que el valor de *min_costo* (eventualmente puede valer INF); y devuelve FALSE en caso contrario. Veamos ambos casos.

Condición (c_4) es TRUE en "*ALG_G_STEINER*"

Se ejecuta (11), realizándose la invocación "*ALG_G_A(A,N∪{i},i,1,costo,num_term,r_{max_c})*". Por H.I. y Proposición 11, se analiza el espacio de subgrafos $H = \bigcup_{\forall B \subseteq C^{(1)}} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B)$ y de existir soluciones factibles al GSP con costo menor que *min_costo*, el algoritmo "*ALG_G_A*" encuentra y almacena la solución factible de menor costo del espacio H como mejor solución.

Como $i \in T$, se cumple que la mejor solución factible al GSP considerando $SG(N, \{i..n\} \setminus T, A)$, es la mejor solución factible al GSP considerando $\bigcup_{\forall B \subseteq C^{(1)}} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B)$; por tanto se verifica la T.I.

Condición (c_4) es FALSE en "*ALG_G_STEINER*"

Al haber retornado el valor FALSE la función "*Test_Nodo_Terminal*", el algoritmo finaliza, pues no es posible encontrar una mejor solución factible al GSP a partir de $G_s(N, A)$, agregando nodos de $V^{(\geq i)} = \{v \in V / v \geq i\}$ y aristas de $(E \setminus E^{(<i)})$, con un costo menor que el valor de *min_costo*.

L.Q.Q.D.

Proposición 13 (Correctitud del Algoritmo ALG_G_STEINER)

Sea $G = (V, E)$ un grafo en las condiciones del GSP. Supóngase que inicialmente: $A_{sol} = \varnothing$, $N_{sol} = \varnothing$ y min_costo=INF, entonces, como resultado de invocar "ALG_G_STEINER(A_{sol},N_{sol},1,0,0)" se obtiene (si existe) la solución óptima al GSP.

Dem:

Por proposición 12, el algoritmo analiza el espacio de subgrafos $SG(N, \{1..n\} \setminus T, A)$, en búsqueda de una solución factible al GSP de menor

costo que la actual. Como además se cumple que $\Gamma_{GSP} \subseteq SG(N, \{1..n\} \setminus T, A)$, se tiene entonces que el algoritmo encuentra y almacena la solución óptima del GSP.

L.Q.Q.D.

3 2. Problemas STECSP y STESNP

3.1 2.1 Introducción

En ciertos modelos de redes no es necesario que todos los nodos de la red tengan el mismo grado de conexión, sino que se exige que solo un subconjunto de nodos tenga un determinado grado de conexión entre pares de nodos con costo mínimo. El caso más general se conoce como GSP y fue visto en el capítulo 1. Dos casos particulares son:

- Se requiere que un subconjunto de nodos de la red sea localmente 2-arista-conexo con costo mínimo.
- Se requiere que exista un subgrafo 2-arista-conexo de costo mínimo que cubra el subconjunto de nodos especificado.

Al primero se lo denomina “Problema de la Red de Steiner 2-arista-confiable” (*Steiner 2-edge-survivable network problem,* STESNP) y al segundo “Problema del Subgrafo de Steiner 2-arista-conexo” (*Steiner 2-edge-connected subgraph problem,* STECSP).

El objetivo principal de este capítulo consiste en el estudio de los casos particulares STECSP-STESNP y la propuesta de nuevos algoritmos que permitan resolver en forma exacta el problema STECSP. En base al relevamiento de la literatura existente, primero se definen formalmente los problemas STECSP-STESNP y se brinda un resumen donde se mencionan resultados y algoritmos conocidos para clases particulares de grafos. Luego se presentan dos nuevos algoritmos exactos para el STECSP, uno basado en la metodología Backtracking y el otro en Programación Dinámica. Si bien éstos son de complejidad exponencial, sirven para comparación de performance con las heurísticas y además son tomados como base para el diseño de dos nuevos algoritmos paralelo-distribuidos que resuelven el STECSP en forma exacta explorando su espacio de soluciones factibles en forma paralela. Se elige uno de estos diseños para ser implementado utilizando el software PVM, el cual permite el desarrollo de aplicaciones paralelizables para ser ejecutadas en forma distribuida. El algoritmo implementado, es sometido a tests de performance realizados sobre un conjunto de instancias seleccionadas del STECSP para luego brindar un

análisis de los resultados obtenidos. Es importante resaltar aquí la introducción del paradigma de desarrollo paralelo-distribuido en la búsqueda de métodos alternativos para obtener soluciones óptimas a problemas NP-difíciles como el STECSP.

3.2 2.2 Formalización del STECSP y STESNP

Dado un grafo $G=(V,E)$, una matriz de costos c asociada a las aristas del grafo y un conjunto de nodos $T / T \subseteq V$ denominados nodos terminales, se definen:

- STECSP - el problema de encontrar el subgrafo de G 2-arista-conexo de costo mínimo que cubre el conjunto de nodos terminales T.
- STESNP - el problema de encontrar el subgrafo de G de costo mínimo tal que $\forall i, j \in T$ existen al menos 2 caminos de aristas disjuntas que unen i con j.

Toda solución factible del STECSP es una solución factible del STESNP, además, si los valores de la matriz de costos son estrictamente positivos, la solución óptima del STESNP es una solución óptima del STECSP. Si $T=V$ ambos problemas coinciden y resulta en un problema del tipo MW2CSN, el cual es analizado en el apéndice A.2. Los problemas STECSP y STESNP tienen aplicación en problemas de telecomunicaciones y transporte.

Cuando $T=V$, si se considera el problema TSP (*Traveling Salesman Problem*) en el cual hay que obtener un ciclo hamiltoniano de costo mínimo que cubra los nodos de V, se cumple que las soluciones factibles de este problema son soluciones factibles de los problemas STECSP-STESNP. Como el TSP es un problema NP-difícil, esto implica que en general, los problemas STECSP y STESNP son del tipo NP-difíciles.

3.3 2.3 Resultados y Algoritmos para STECSP - STESNP

Para algunas clases especiales de grafos existen algoritmos de orden polinomial que resuelven los problemas STECSP-STESNP. Wald-Colbourn

[56] demuestran que el STECSP puede ser resuelto con orden polinomial para la clase de grafos outerplanares. Monma-Sidney [40] demuestran que para la clase de grafos serie-paralelos también existe un algoritmo que resuelve el STECSP con orden polinomial. Winter demuestra que el STESNP puede ser resuelto en tiempo polinomial para la clase de grafos serie-paralelos y grafos de Halin. Monma-Munson-Pulleyblank [39] demuestran que en el caso en el cual los costos asociados a las aristas satisfacen la desigualdad triangular para todo triple (e_1, e_2, e_3) de aristas formando un triángulo en G, el mínimo costo de un subgrafo 2-arista-conexo que cubre T sin utilizar nodos de $V \setminus T$, esta acotado por $(4/3) \cdot Q$, donde Q es el mínimo costo de un subgrafo 2-arista-conexo que cubre T utilizando nodos de $V \setminus T$.

Para el caso $T = V$, Mahjoub [37] da un análisis completo del politopo asociado al STECSP cuando el grafo G es un grafo serie-paralelo. Barahona-Mahjoub [7] estudian y caracterizan el politopo del STECSP cuando G es un grafo Halin. Coullard-Rais-Radin-Wagner estudian el problema del subgrafo de Steiner 2-nodo-conexo (STNCSP) y caracterizan el politopo asociado al problema cuando G es un grafo serie-paralelo. Mourad Baïou [4,5] analiza y caracteriza los politopos asociados a los problemas STECSP-STESNP, en particular para la clase de grafos serie-paralelos.

3.4 2.4 Nuevos Algoritmos Exactos para el STECSP

3.4.1 2.4.1 Introducción

En este apartado se presentan dos algoritmos exactos para resolver el STECSP. El primero de los algoritmos se basa en la técnica de diseño de algoritmos denominada Backtracking, mientras que el segundo en la técnica de Programación Dinámica. Para cada uno de los algoritmos exactos, se dará: descripción del algoritmo, pseudocódigo, y demostración de correctitud.

3.4.2 2.4.2 Algoritmo Backtracking para el STECSP

Dado un grafo $G = (V, E)$ en las condiciones del STECSP, se denota al espacio de soluciones factibles de la instancia del problema como Γ_{STECSP}. El

mismo viene dado por el conjunto de subgrafos de G que son 2-arista-conexos y cubren el conjunto de nodos terminales T ; o sea $G_s(N,A) \in \Gamma_{STECSP}$ si $T \subseteq N \subseteq V$, $A \subseteq E$ y además $G_s(N,A)$ es 2-arista-conexo.

Se propone aquí un algoritmo basado en la técnica de Programación Backtracking, el cual halla la solución exacta al problema STECSP, explorando el espacio de soluciones factibles Γ_{STECSP} , descartando mediante la aplicación de funciones de cota y tests a nodos del tipo Steiner, ramas del árbol de soluciones correspondiente a la instancia del problema.

Se define el conjunto de aristas $A_{sol} \subseteq E$, el conjunto de nodos $N_{sol} \subseteq V$, y la variable *min_costo* (inicializada en INF) como globales al algoritmo y es donde se almacena la solución óptima del problema. Se asumen además como globales al algoritmo las representaciones del grafo $G = (V,E)$ original, la matriz de costos c asociada a las aristas, y el conjunto de nodos terminales T .

Entradas del Algoritmo

- Conjunto de aristas $A \subseteq E$,
- Conjunto de nodos $N \subseteq V$,
- Nodo actual $i \in V$,
- Nodo adyacente actual $j \in V$,
- Costo del subgrafo $G_s(N,A)$ de G ,
- Cantidad de aristas consideradas desde el nodo $i \in V$, hacia el conjunto $N \subseteq V$ (*cant_a*),
- Cantidad de nodo terminales considerados hasta el momento (*cant_t*).

El pseudocódigo del algoritmo es el siguiente.

```
Procedure ALG1_STEINER(A,N,i,j,costo,cant_a,cant_t);
Begin
1   OK=ACT_SOL(A,N,costo,cant_a,cant_t);
2   if ((i≤ n) and (costo<min_costo) and not(OK)) then                (c_1)
3         if (i∉N)  then                                               (c_2)
4            ALG1_STEINER(A,N,i+1,1,costo,0,cant_t);
5            if TEST(i,N,A) then                                       (c_3)
6               ALG1_STEINER(A,N∪{i},i,1,costo,0,cant_t);
7            endif;
8         else if (j≤ n) then                                          (c_4)
9                  if  ((j∈N) and ((i,j)∈G) and (i≠j))  then
10                   if (i∈T) then                                     (c_5)
11                      ALG1_STEINER(A∪{(i,j)},N,i,j+1,costo+c_ij,cant_a+1,cant_t+1);
12                      ALG1_STEINER(A,N,i,j+1,costo,cant_a,cant_t+1);
13                   else
14                      ALG1_STEINER(A∪{(i,j)},N,i,j+1,costo+c_ij,cant_a+1,cant_t);
15                      ALG1_STEINER(A,N,i,j+1,costo,cant_a,cant_t);
16                   endif;
17                else
18                   ALG1_STEINER(A,N,i,j+1,costo,cant_a,cant_t);
19                endif;
20             else
21                ALG1_STEINER(A,N,i+1,1,costo,0,cant_t);
22             endif;
23       endif;
24     endif;
25  endif;
End;
```

Las funciones auxiliares utilizadas en el algoritmo "*ALG1_STEINER*", se describen a continuación.

Función "*ACT_SOL*"

Esta función recibe como parámetros , un conjunto de aristas $A \subseteq E$, un conjunto de nodos $N \subseteq V$, el costo del subgrafo $G_s(N, A)$, la cantidad de aristas consideradas desde el nodo $i \in V$ hacia el conjunto de nodos N , y el número de nodos terminales presentes en el conjunto N. Chequea con la ayuda de la primitiva "*ES_2STEINER*" si se tiene una mejor solución factible

al STECSP que la solución actual; de ser así, se almacena la nueva solución factible para comparar. El pseudocódigo para "*ACT_SOL*" es el siguiente.

```
Function ACT_SOL(A,N,costo,cant_a,cant_t);
Begin
  ACT_SOL=FALSE;
 if (cant_a≥2) and (cant_t=NUM_TERM) then          (c_1)
   if  (costo<min_costo) then                        (c_2)
      if (ES_2STEINER(A,N) then    (c_3)
          A_sol=A;
          N_sol=N;
          menor_costo=costo;
          ACT_SOL=TRUE;
      endif;
   endif;
 endif;
End;
```

Donde $ES\_2STEINER(A,N) = \begin{cases} TRUE, si\, G_s(N,A) \in \Gamma_{STECSP} \\ FALSE, sino \end{cases}$.

Existen varias implementaciones eficientes de esta primitiva; una de ellas se basa en una generalización de algoritmo de Flujo Máximo-Corte Mínimo. No se pretende aquí detallar las implemetaciones, sino simplemente mencionar su existencia.

Función "*TEST*"

Tiene como entradas un nodo de Steiner $i \in V \setminus T$, un conjunto de aristas $A \subseteq E$, y un conjunto de nodos $N \subseteq V$. Devuelve TRUE si el nodo de Steiner i puede llegar a formar parte de una mejor solución factible del espacio Γ_{STECSP} , construida a partir de $G_s(N,A)$, agregando nodos que están en el conjunto $V^{(\geq i)}$ y aristas de $(E \setminus E^{(<i)})$; devuelve FALSE en caso contrario. Los tests realizados por esta función son similares a los vistos en Proposición 10 para los nodos del tipo Steiner, instanciando la matriz de requerimientos R con los valores $r_{ij} = 2, \forall i, j \in T$.

Un análisis más exhaustivo se puede realizar aplicando el algoritmo generalizado de Ford-Fulkerson de Flujo Máximo-Corte Mínimo al subgrafo

$G_s(N \cup V^{(\geq i)}, A \cup (E \setminus E^{(<i)})$ con capacidades unitarias en las aristas, y así determinar las cantidades l_{it} de aristas que separan al nodo de Steiner i con cada nodo terminal $t \in T$. Por Teorema de Menger, estas cantidades representan la cantidad máxima de caminos de aristas disjuntas que unen el nodo de Steiner i con cualquier nodo terminal en $G_s(N \cup V^{(\geq i)}, A \cup (E \setminus E^{(<i)})$. Si $\forall t \in T, l_{it} < 2$ entonces el nodo i no debe ser tomado en cuenta por el algoritmo, debiéndose retornar FALSE por la función "TEST".

Para demostrar la correctitud del algoritmo "*ALG1_STEINER*", se demuestra previamente la siguiente proposición.

Proposición 14

Sea $G = (V, E)$ un grafo en las condiciones del STECSP, $T \subseteq V$ un conjunto de nodos que llamaremos terminales, $T \subseteq N \subseteq V$ un conjunto de nodos y $A \subseteq E$ un conjunto de aristas con extremos en N . Supóngase además que:

- *i es el nodo actual a ser analizado y además cumple $i \geq k, \forall k \in (N \setminus T)$ (o sea el nodo i es el mayor nodo no terminal considerado hasta el momento, en cuyo caso suponemos que inicialmente $i \notin N$, o bien es el próximo nodo terminal a considerar, cumpliéndose $i \in T$, $i \in N$).*
- *j es el siguiente nodo a ser analizado como adyacente al nodo i , y cumple además que $j \geq Max_{(i,k) \in A}\{k\}$,*
- *costo es el costo del subgrafo $G_s(N, A)$ de G ,*
- *cant_a es el número de aristas presentes en A que van de i a algún nodo de N ,*
- *cant_t es el número de nodos terminales cuyas etiquetas son menores o iguales que i ,*
- *inicialmente $N_{sol} = \varnothing$, $A_{sol} = \varnothing$ y min_costo=INF.*

Entonces al invocar "ALG1_STEINER(A,N,i,j,costo,cant_a,cant_t)", se obtiene la mejor solución factible al problema STECSP considerando $SG(N, \{i..n\} \setminus T, A)$.

Dem:

La demostración se hará por inducción hacia atrás en el nodo i-ésimo.

Paso Base

Se cumple vacíamente.

Paso Inductivo

H.I) Bajo las condiciones de Proposición 14 al invocar "*ALG1_STEINER(A,N,k,j,costo,cant_a,cant_t)*" se obtiene como resultado la mejor solución factible al problema STECSP considerando $SG(N,\{k..n\}\setminus T,A)$, $1 \le i < k \le n$.

T.I) Bajo las condiciones de Proposición 14 al invocar "*ALG1_STEINER(A,N,i,j,costo,cant_a,cant_t)*" se obtiene como resultado la mejor solución factible al problema STECSP considerando $SG(N,\{i..n\}\setminus T,A)$, $1 \le i < k \le n$.

3.4.2.1.1.1.1 Test de Factibilidad

Al ejecutar "*ALG1_STEINER(A,N,i,j,costo,cant_a,cant_t)*", lo primero que se realiza es la invocación al algoritmo "*ACT_SOL*". Cuando se ejecuta "*ACT_SOL(A,N,costo,cant_a,cant_t)*", lo que hace esta función es ver si se cumplen las condiciones *(cant_a≥2)* y *(cant_t=NUM_TERM)* simultáneamente. Ambas deben cumplirse para luego verificar si el subgrafo $G_s(N,A)$ es una solución factible al problema STECSP (esto es así pues si *(cant_a<2)*, significa que agregué una o ninguna arista a un subgrafo de G que no era solución factible al STECSP, con lo cual tampoco éste será factible al STECSP; en caso que se cumpla *(cant_t<NUM_TERM)* significa que todavía no se obtuvo un subgrafo de G que contenga todos los nodos

terminales del conjunto T, por tanto $G_s(N,A)$ no es una solución factible al STECSP).

Si las condiciones (c_1) de "*ACT_SOL*" se cumplen, el subgrafo $G_s(N,A)$ puede que sea una solución factible al STECSP. La función "*ACT_SOL*" antes de verificar la factibilidad de $G_s(N,A)$, chequea mediante la condición (c_2) que el costo del subgrafo $G_s(N,A)$ sea menor que el costo de la mejor solución factible encontrada hasta el momento, es decir, se hace el test de factibilidad al subgrafo $G_s(N,A)$, solo si su costo es menor que el de la mejor solución encontrada hasta el momento (la condición (c_2) se cumple en nuestro caso, pues una de las hipótesis de la Proposición 1 es que inicialmente *min_costo*=INF). Luego se ejecuta la función "*(ES_2STEINER(A,N))*" que chequea si el subgrafo $G_s(N,A)$ es factible para STECSP, y en caso de serlo actualiza la mejor solución factible encontrada. La función "*ACT_SOL*" retorna un valor booleano que indica si se encontró o no una mejor solución factible para STECSP. Si el valor retornado por "*ACT_SOL*" es TRUE, el subgrafo $G_s(N,A)$ es la mejor solución factible para STECSP considerando el conjunto de subgrafos $SG(N,\{i..n\}\setminus T,A)$, pues cualquier otro subgrafo de G en este conjunto tendría al menos las mismas aristas que $G_s(N,A)$ y por tanto su costo sería mayor o igual al de $G_s(N,A)$.

Condición (c_1) es TRUE en "*ALG1_STEINER*"

Si "*ACT_SOL*" devuelve FALSE entonces se cumple la condición (c_1) en "*ALG1_STEINER*", pues por hipótesis tenemos que *(i≤n)* y además como *min_costo* tiene el valor INF, entonces se verifica (costo<*min_costo*), con lo cual la condición compuesta (c_1) es verdadera. Luego se chequea la condición (c_2) en "*ALG1_STEINER*". Se analiza a continuación cada caso.

3.4.2.1.1.1.2 Condición (c_2) es TRUE en "*ALG1_STEINER*"

Si (c_2) es verdadero, implica que el nodo i no es un nodo terminal, por tanto no se incrementa el número de nodos terminales "visitados". El algoritmo realiza la invocación recursiva:

"*ALG1_STEINER(A,N,i+1,1,costo,0,cant_t)*", lo que significa que se buscarán soluciones factibles al STECSP, no tomando en cuenta el nodo i. Por hipótesis inductiva, resulta que "*ALG1_STEINER(A,N,i+1,1,costo,0,cant_t)*", encuentra la mejor solución factible al STECSP considerando el conjunto de subgrafos $SG(N,\{i+1..n\}\setminus T,A)$.

Luego el algoritmo verifica la condición (c_3). La función TEST realiza ciertos tests al nodo i, de manera que si el nodo i los cumple, entonces puede llegar a pertenecer a alguna solución factible del STECSP de menor costo que la actual y formada a partir del subgrafo $G_s(N,A)$ agregando el nodo i y considerando el agregado de nodos del conjunto $(\{i+1..n\}\setminus T)$. Si el nodo i no pasa los tests, no puede pertenecer a ninguna solución factible del STECSP en las condiciones anteriores. Se analizarán seguidamente ambos casos.

Condición (c_3) es FALSE en "*ALG1_STEINER*"

Supóngase primeramente que (c_3) no se satisface (i no cumple tos tests), entonces el algoritmo finaliza, teniéndose que la mejor solución factible al STECSP considerando $SG(N,\{i..n\}\setminus T,A)$, es la mejor solución factible al STECSP considerando $SG(N,\{i+1..n\}\setminus T,A)$, pues se sabe que al considerar el nodo i no se obtiene una mejor solución factible del STECSP.

3.4.2.1.1.1.3 Condición (c_3) es TRUE en "*ALG1_STEINER*"

Considérese ahora el caso que (c_3) sea verdadero; el algoritmo invoca recursivamente a "*ALG1_STEINER(A,N∪{i},i,1,costo,0,cant_t)*". Esto implica que se genere la siguiente serie de invocaciones recursivas en pasos posteriores a la invocación anterior:

"*ALG1_STEINER(A∪B,N∪{i},i+1,1,costo+COSTO(B),0,cant_t)*" $\forall B \subseteq C$, donde

$C=\{(i,k)\in G / k\in N, i\neq k\}$, $COSTO(B)=\sum_{e\in B} c(e)$.

Notar que en ninguna de las invocaciones anteriores se incrementa la cantidad de nodos terminales, pues estamos suponiendo el caso en que se satisface (c_2), o sea $i \notin N$, con lo cual i no puede ser un nodo terminal.

Por hipótesis inductiva, se sabe que cada una de las invocaciones recursivas dadas por "*ALG1_STEINER(A∪B,N∪{i},i+1,1,costo+COSTO(B),0,cant_t)*" obtiene como resultado la mejor solución factible del STECSP considerando $SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B), \forall B \subseteq C$, conjuntos de subgrafos de G.

De esta manera como resultado de esta serie de invocaciones se tiene la mejor solución factible del conjunto de subgrafos de G, dado por: $H = \bigcup_{B \subseteq C} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B)$. Como además se tenía (por H.I.) que la invocación "*ALG1_STEINER(A,N,i+1,1,costo,0,cant_t)*" obtiene la mejor solución factible al STECSP considerando $SG(N, \{i+1..n\} \setminus T, A)$. Entonces si se define el conjunto:

$$H_T = SG(N, \{i+1..n\} \setminus T, A) \bigcup \left(\bigcup_{B \subseteq C} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup B) \right).$$

Se cumple en este caso que la mejor solución factible del STECSP considerando $SG(N, \{i..n\} \setminus T, A)$, coincide (al menos en el costo) con la mejor de las soluciones factibles del STECSP considerando H_T.

3.4.2.1.1.1.4 Condición (c_2) es FALSE en "*ALG1_STEINER*"

Analicemos ahora cuando la condición (c_2) es falsa, entonces el nodo i es el mayor nodo terminal analizado hasta el momento.

Si i es un nodo terminal, i puede ser o no el último nodo terminal a ser analizado. Se analiza seguidamente ambos casos.

3.4.2.1.1.1.4.1 Nodo i no es último nodo terminal

Suponiendo que i no es el último nodo terminal a ser analizado, entonces el algoritmo realiza una serie de invocaciones recursivas en las cuales la condición (c_1) es verdadera, (c_2) es falsa y (c_4) es verdadera en "*ALG1_STEINER*". Luego de un número finito de invocaciones, el algoritmo ejecuta la siguiente serie de invocaciones recursivas:

"*ALG1_STEINER(A∪B,N,i+1,1,costo+COSTO(B),0,cant_t+1)*" $\forall B \subseteq C$, donde

$C = \{(i,k) \in G / k \in N, i \neq k\}$.

Por hipótesis inductiva, se sabe que cada una de las invocaciones recursivas dadas por "*ALG1_STEINER(A∪B,N,i+1,1,costo+COSTO(B),0,cant_t+1)*" obtiene como resultado la mejor solución factible del STECSP considerando $SG(N, \{i+1..n\} \setminus T, A \cup B), \forall B \subseteq C$, conjuntos de subgrafos de G . Por tanto, como resultado de estas invocaciones, se tiene la mejor solución factible del STECSP, considerando $H = \bigcup_{B \subseteq D} SG(N, \{i+1..n\} \setminus T, A \cup B)$. Dicha solución es igual (al menos en costo) a la mejor solución factible del STECSP considerando $SG(N, \{i..n\} \setminus T, A)$.

Nodo i es último nodo terminal

Suponiendo ahora que i es el último nodo terminal a ser analizado, entonces se genera la serie de invocaciones recursivas:

"*ALG1_STEINER(A∪B$_k$,N,i,k+1,costo+COSTO(B$_k$),cant_a+$|B_k|$,cant_t+1)*"

$\forall 1 \leq k \leq n$,

$\forall B_k \subseteq C_k, C_k = \{(i,j) \in G / j \in N, i \neq j, j \leq k\}$.

Para cada uno de los subgrafos $G_s(N, A \cup B_k)$, con $|B_k| \geq 2$, generados por estas invocaciones, el algoritmo verifica mediante la función "*ACT_SOL*", si se obtuvo una mejor solución factible que la encontrada hasta el momento. Si alguno de los subgrafos $G_s(N, A \cup B_k)$ es factible para STECSP, entonces, el de menor costo es la mejor solución factible del STECSP encontrada hasta el momento. Supongamos que el de menor costo viene dado por $G_s(N, A \cup B_{\bar{k}})$, con $B_{\bar{k}} \subseteq C_{\bar{k}}$ para algún $\bar{k} \in [1..n]$. El costo de este subgrafo es almacenado en la variable $costo$ para comparar. En los siguientes pasos el algoritmo realiza la serie de invocaciones:

"*ALG1_STEINER(A∪B,N,i+1,1,costo+COSTO(B),0,cant_t+1)*"

$\forall (A \cup B)$ tal que: $B \subseteq C$ con $C = \{(i,k) \in G / k \in N, i \neq k\}$, $T \subseteq N \subset V$ y $G_s(N, A \cup B)$ no es factible para STECSP y además

$COSTO(A \cup B) < min\_costo$; entonces el algoritmo intenta encontrar una solución factible de menor costo agregando nodos del conjunto $\{i+1..n\}$.

Las invocaciones anteriores, por hipótesis inductiva obtienen como resultado la mejor solución factible del STECSP considerando el conjunto de subgrafos dado por:

$H_2 = \bigcup_{B \subseteq C \,/\, COSTO(A \cup B) < min\_costo} SG(N, \{i+1..n\} \setminus T, A \cup B)$; con lo cual la mejor solución factible encontrada, es la solución factible del STECSP de menor costo considerando los subgrafos de $G_s(N, A \cup B_{\bar{k}}) \bigcup H_2$ (los otros no son solución factible o tienen mayor costo). De esta manera es fácil ver que la mejor solución factible encontrada para el STECSP, es la mejor solución factible considerando el conjunto se subgrafos dado por $SG(N, \{i..n\} \setminus T, A)$.

En el caso que ninguno de los subgrafos $G_s(N, A \cup B_k)$, con $|B_k| \geq 2$, sea factible para STECSP, entonces el algoritmo ejecuta la siguiente serie de invocaciones recursivas:

"*ALG1_STEINER(A∪B,N,i+1,1,costo+COSTO(B),0,cant_t+1)*" $\forall B \subseteq C$, donde

$C = \{(i,k) \in G \,/\, k \in N, i \neq k\}$.

Por hipótesis inductiva "*ALG1_STEINER(A∪B,N,i+1,1,costo+COSTO(B),0,cant_t+1)*" obtiene la mejor solución factible del STECSP considerando $SG(N, \{i+1..n\} \setminus T, A \cup B)$, $\forall B \subseteq C$, conjuntos de subgrafos de G. De estas invocaciones, se tiene entonces la mejor solución factible del STECSP, considerando $H = \bigcup_{B \subseteq C} SG(N, \{i+1..n\} \setminus T, A \cup B)$, la cual coincide (al menos en costo) con la mejor solución factible del STECSP considerando $SG(N, \{i..n\} \setminus T, A)$.

L.Q.Q.D.

Proposición 15 (Correctitud del Algoritmo ALG1_STEINER)

Sea $G=(V,E)$ un grafo en las condiciones del STECSP, con $T \subseteq V$ el conjunto de nodos terminales. Entonces al invocar "ALG1_STEINER({},S,1,1,0,0,0)" se obtiene la solución óptima al problema STECSP.

Dem:

La invocación de "*ALG1_STEINER({},S,1,1,0,0,0)*", por Proposición 14, obtiene la mejor solución factible del STECSP considerando el conjunto de subgrafos $SG(S,V \setminus T,\{\})$ de G.

Por definición:

$$SG(S,V \setminus T,\{\}) = \{G_s(X,Y) / X = T \cup W, \forall W \subseteq (V \setminus T), \forall Y \subseteq E_2\}$$
$$E_2 = \{(i,j) \in E, i,j \in X\}$$

Claramente este conjunto contiene todas las soluciones factibles del STECSP, pues es el conjunto de todos los subgrafos de G, que contienen por lo menos los nodos terminales T.

L.Q.Q.D.

3.4.3 2.4.3 Algoritmo de Programación Dinámica para el STECSP

3.4.3.1 2.4.3.1 Introducción

Otra técnica de diseño de algoritmos que puede utilizarse para resolver en forma exacta el STECSP, es la Programación Dinámica. En este apartado se verá: una descripción breve de ésta técnica de diseño de algoritmos, una recurrencia de Programación Dinámica para el STECSP, y un algoritmo que implementa dicha recurrencia para resolver el STECSP en forma exacta.

3.4.3.2 2.4.3.2 Programación Dinámica

La Programación Dinámica es metodología de diseño de algoritmos que puede ser usada cuando la solución a un problema puede ser vista como el resultado de una secuencia de decisiones.

Para algunas clases de problemas, la secuencia de decisiones óptimas puede ser encontrada tomando una decisión en cada paso y no cometiendo errores de decisión. Esto es verdadero para todos los problemas que pueden ser resueltos mediante el Método Greedy. Para muchos otros problemas, no es posible construir la solución óptima mediante toma de decisiones tomadas paso a paso (en base a información local) y sin errores.

Una forma de resolver problemas en los cuales no es posible construir paso a paso una secuencia de decisiones que conduzca a la secuencia óptima de decisiones, es explorar todo el espacio de secuencia de decisiones posibles. Se podría enumerar todas las posibles secuencias de decisiones y seleccionar la mejor. La Programación Dinámica, en general, reduce drásticamente el total de enumeraciones evitando la enumeración de algunas secuencias de decisiones que no son óptimas. En Programación Dinámica una secuencia de decisiones óptima es encontrada explícitamente haciendo uso del *Principio de Optimalidad*. Este principio establece que una secuencia óptima de decisiones tiene la propiedad que, cualquiera que sea el estado y decisión inicial, las restantes decisiones deben constituir una secuencia de decisiones óptima considerando el estado resultante de la primera decisión. En [47] se puede ver una descripción detallada de esta metodología y sus aplicaciones a problemas NP-difíciles.

Seguidamente se verá la aplicación de la Programación Dinámica para la resolución en forma exacta del problema STECSP.

3.4.3.3 2.4.3.3 Programación Dinámica aplicada al STECSP

Considérese un grafo $G = (V, E)$ en las condiciones del STECSP; $V = \{1..n\}$ el conjunto de nodos, E el conjunto de aristas del grafo, a cada arista $(i, j) \in E$ se le asocia un costo no negativo c_{ij} y $T \subseteq V$ un conjunto de nodos distinguidos de G (nodos terminales). La siguiente definición, permitirá encontrar una recurrencia de Programación Dinámica que calcule el costo de la mejor solución al STECSP.

Definición 16

Se define $Cm_{2ac}(S, E_S)$ como el menor costo de un subgrafo de G 2-arista-conexo que contiene exactamente el conjunto de nodos S , y el conjunto de aristas incluido en E_S con extremos en S .

La siguiente es una expresión recursiva de $Cm_{2ac}(S, E_S)$ basada en Programación Dinámica.

$$\begin{aligned}
&Cm_{2ac}(S, E_S) = INF, si\, G_s(S, E_S)\, es\, un\, arbol\, o\, desconexo,\\
&Cm_{2ac}(S, E_S) = COSTO(E_S), si\, G_s(S, E_S)\, es\, un\, ciclo,\\
&Cm_{2ac}(S, E_S) = Min\left\{\begin{array}{l} Cm_{2ac}(W, E_W) + \cos to(p), \forall \varnothing \subset W \subset S / \\ \exists\, p \in Wpath / W \cup Nodos(p) = S \end{array}\right\}, en\, otro\, caso.
\end{aligned}$$

[Rec_1]

Proposición 17

La expresión anterior para $Cm_{2ac}(S, E_S)$ da el menor costo de un subgrafo de G 2-arista-conexo que contiene exactamente el conjunto de nodos S y conjunto de aristas incluido en E_S . Se devuelve INF en caso en que no exista ningún subgrafo en dichas condiciones.

Dem:

Caso A ($G_s(S, E_S)$ no es 2-arista-conexo)

Se demostrará por inducción en la cardinalidad de E_S , que $Cm_{2ac}(S, E_S)$ devuelve INF.

Paso Base

El paso base se da cuando $|E_S| = 0$.

Paso Inductivo

H.I) Sea $G_s(S, E_S)$ un grafo con conjunto de nodos S y de aristas E_S , $|E_S| = k < n_E$, el cual no es 2-arista-conexo. Entonces la recurrencia de Programación Dinámica para $Cm_{2ac}(S, E_S)$ da como resultado el valor INF.

T.I) Sea $G_s(S,E_S)$ un grafo con conjunto de nodos S y de aristas E_S, $|E_S| = n_E$, el cual no es 2-arista-conexo. Entonces la recurrencia de Programación Dinámica para $Cm_{2ac}(S,E_S)$ da como resultado el valor INF.

Considérese $G_s(S,E_S)$ en las condiciones de T.I.

Si $G_s(S,E_S)$ es desconexo o un árbol entonces el valor retornado para $Cm_{2ac}(S,E_S)$ es INF. Veamos ahora el caso en el cual el grafo $G_s(S,E_S)$ es conexo y no es un árbol.

Como $G_s(S,E_S)$ es conexo y no es un árbol, entonces $G_s(S,E_S)$ contiene al menos un ciclo. Por lo tanto contiene al menos 1 subgrafo 2-arista-conexo (un ciclo es un subgrafo 2-arista-conexo).

Sea G_A el mayor subgrafo 2-arista-conexo en $G_s(S,E_S)$. Como $G_s(S,E_S)$ no es 2-arista conexo, existe al menos un nodo $v \in G_s(S,E_S), v \notin G_A$, que es adyacente a un nodo $u \in G_A$ y además no existe otro camino que lo conecte con G_A (de lo contrario G_A no seria subgrafo 2-arista-conexo maximal en $G_s(S,E_S)$). Todos los conjuntos de nodos W que cumplen: $\varnothing \subset W \subset S$, $\exists p \in Wpath / W \cup Nodos(p) = S$, son considerados en la recurrencia de $Cm_{2ac}(S,E_S)$.

Pueden suceder los siguientes casos:

1. Si $u,v \in W$, G_W no es 2-arista-conexo, pues no hay dos caminos de aristas disjuntas que unan v con u en $G_s(S,E_S)$, por tanto tampoco en G_W.
2. En caso que $u,v \notin W$, entonces $u,v \in p$. Sean $w_1, w_2 \in W$ los extremos de p en G_W. Supongamos que G_W es 2-arista-conexo, entonces existe un camino p_{w_1,w_2} que une w_1 con w_2 en G_W. Los caminos p y p_{w_1,w_2} forman un ciclo que contiene a los nodos u y v (si $w_1 = w_2$, p forma un ciclo). De esta manera en $G_s(S,E_S)$, u y v están conectados por lo menos por dos caminos de aristas disjuntas, absurdo. Luego G_W no puede ser 2-arista-conexo.

3. Si $u \in W$ y $v \notin W$, sean $w_1, w_2 \in W$ los nodos extremos de p en G_W. Si G_W es 2-arista-conexo, existen dos caminos p^1_{u,w_1} y p^2_{u,w_1} de aristas disjuntas que unen u con w_1, y dos caminos p^1_{u,w_2} y p^2_{u,w_2} de aristas disjuntas que unen u con w_2. Suponiendo que $w_1 \notin p^1_{u,w_2}, w_1 \notin p^2_{u,w_2}$ y $w_2 \notin p^1_{u,w_1}, w_2 \notin p^2_{u,w_1}$, si se considera p^1_{u,w_1}, p^1_{u,w_2} y p, se tiene un ciclo que contiene a los nodos u y v, por tanto existen 2 caminos de aristas disjuntas que unen u y v en $G_s(S, E_S)$. Se llega a una contradicción, por lo cual es absurdo suponer que G_W es 2-arista-conexo. Veamos ahora el caso que w_1 o w_2 pertenezca a uno de los caminos. Supongamos para fijar ideas que $w_2 \in p^1_{u,w_1}$ y $w_2 \notin p^2_{u,w_1}$. Sea p^{1s}_{u,w_2} la parte del camino p^1_{u,w_1} que une u con w_2, los caminos p^{1s}_{u,w_2}, p^2_{u,w_1} y p forman un ciclo que contiene a u y v, entonces existen 2 caminos de aristas disjuntas que unen u y v en $G_s(S, E_S)$, absurdo. Si sucediera que $w_2 \in p^1_{u,w_1}$ y $w_2 \in p^2_{u,w_1}$, sea p^{1s}_{u,w_2} la parte del camino p^1_{u,w_1} que une u con w_2 y p^{2s}_{u,w_2} la parte del camino p^2_{u,w_1} que une u con w_2, entonces los caminos p^{1s}_{u,w_2}, p^{2s}_{u,w_1} y p forman un ciclo que contiene a u y v, lo cual implica que existen 2 caminos de aristas disjuntas que unen u y v en $G_s(S, E_S)$, absurdo.
4. El caso $u \notin W$ y $v \in W$ es igual al anterior, con un análisis similar se concluye que G_W no es 2-arista-conexo.

De los puntos anteriores surge que $\forall \varnothing \subset W \subset S / \exists p \in Wpath, W \cup Nodos(p) = S$, el grafo G_W no es 2-arista-conexo; como $|E_W| < n_E$ (E_W aristas de G_W) por H.I. el valor de $Cm_{2ac}(W, E_W)$ es INF.

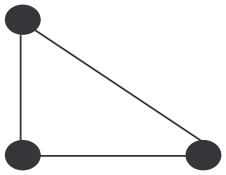
Luego se tiene que el valor de $Cm_{2ac}(S, E_S)$ dado por la recurrencia es INF.

Caso B ($G_s(S, E_S)$ es 2-arista-conexo)

Se demostrará por inducción en la cardinalidad de E_S, que $Cm_{2ac}(S, E_S)$ devuelve el menor costo del subgrafo 2-arista-conexo que contiene exactamente el conjunto de nodos S y el conjunto de aristas incluido en E_S.

Paso Base

El paso base se da cuando $|E_S| = 3$ y $G_s(S, E_S)$ tiene la forma:



Paso Inductivo

H.I) Sea $G_s(S, E_S)$ un grafo con conjunto de nodos S y conjunto de aristas E_S, $|E_S| = k < n_E$, el cual es 2-arista-conexo. Entonces la recurrencia de Programación Dinámica para $Cm_{2ac}(S, E_S)$ da como resultado el valor del subgrafo 2-arista-conexo de menor costo con exactamente el conjunto de nodos S.

T.I) Sea $G_s(S, E_S)$ un grafo con conjunto de nodos S y conjunto de aristas E_S, $|E_S| = n_E$, el cual es 2-arista-conexo. Entonces la recurrencia de Programación Dinámica para $Cm_{2ac}(S, E_S)$ da como resultado el valor del subgrafo 2-arista-conexo de menor costo con exactamente el conjunto de nodos S.

Sea $G_s(S, E_S)$ un grafo en las condiciones de T.I.

En el caso que $G_s(S, E_S)$ sea un ciclo, $Cm_{2ac}(S, E_S)$ es el costo del ciclo, pues cualquier otro subgrafo con menos aristas es desconexo o un árbol.

Se analiza ahora el caso en que $G_s(S, E_S)$ no es un ciclo. Por Proposición 14 de Apéndice 1, $G_s(S, E_S)$ es obtenido a partir de un ciclo agregando sucesivamente $Hpaths$. Sea G_A es subgrafo de $G_s(S, E_S)$ 2-arista-conexo, tal

que al agregarle el $Hpaths\ p$ se obtiene el grafo $G_s(S, E_S)$. Pueden suceder los siguientes casos.

1. $\varnothing \subset A \subset S, S = A \cup Nodos(p)$, como G_A es 2-arista-conexo y $|E_A| < n_E$ (E_A conjunto de aristas de G_A), por H.I., resulta que $Cm_{2ac}(A, E_A)$ es el menor costo de un subgrafo de G_A 2-arista-conexo cuyos nodos son exactamente los nodos de A. Además se cumple que $Cm_{2ac}(S, E_S) = Cm_{2ac}(A, E_A) + COSTO(p)$.
2. $A = S$, en este caso p es una arista. Como G_A es 2-arista-conexo y $|E_A| < n_E$, por H.I. $Cm_{2ac}(A, E_A)$ es el menor costo de un subgrafo de G_A 2-arista-conexo cuyos nodos son exactamente los nodos de A, y en este caso $Cm_{2ac}(S, E_S) = Cm_{2ac}(A, E_A)$.

De lo anteriormente expuesto, cualquiera sea la descomposición de $G_s(S, E_S)$ como un subgrafo 2-arista-conexo y un $Hpaths$, el subgrafo resultante es considerado en la recurrencia que define a $Cm_{2ac}(S, E_S)$.

L.Q.Q.D.

Proposición 18

La siguiente expresión, da el costo de la mejor solución al problema STECSP.

$Min\_Cost = Min\{Cm_{2ac}(S, E_S), \forall T \subseteq S \subseteq V\}$*, con* E_S *el conjunto de aristas del subgrafo inducido por el conjunto de nodos* S*.*

Dem:

Supongamos que $G_s(V_o, E_o)$ es el subgrafo 2-arista-conexo de menor costo que cumple $T \subseteq V_o \subseteq V$.

Sea E_{Vo} el conjunto de aristas del subgrafo inducido por el conjunto de nodos V_o.

En la expresión $Min\{C_{2ac}(S, E_S), \forall T \subseteq S \subseteq V\}$, es considerado el subgrafo $G_s(V_o, E_{Vo})$ al calcular $Cm_{2ac}(V_o, E_{Vo})$.

Por Proposición 17, $Cm_{2ac}(V_o, E_{Vo})$ da el menor costo del subgrafo 2-arista-conexo que contiene exactamente el conjunto de nodos V_o y conjunto de aristas incluido en E_{Vo}; la mejor solución es el subgrafo $G_s(V_o, E_o)$ y además $Min\_Cost = Cm_{2ac}(V_o, E_{Vo}) = COSTO(E_o)$.

L.Q.Q.D.

La recurrencia [**Rec_1**], esta definida aún para subgrafos $G_s(S, E_S)$ que no cumplen ser 2-arista-conexo. Se da una nueva definición para $Cm_{2ac}(S, E_S)$, chequeando la existencia de soluciones factibles antes de considerar la recurrencia.

$$
\begin{array}{l}
Cm_{2ac}(S, E_S) = INF, si\, G_s(S, E_S)\, no\, es\, 2AC, \\
Cm_{2ac}(S, E_S) = COSTO(E_S), si\, G_s(S, E_S)\, es\, un\, ciclo, \\
Cm_{2ac}(S, E_S) = Min\left(Min\left\{ \begin{array}{l} Cm_{2ac}(S, E_S - e) / e \in E_S, e \notin H_{E_S} \\ G_s(S, E_S - e)\, 2AC \end{array} \right\}, COSTO(E_S) \right), en\, otro\, caso. \\
donde\, H_{E_S} = \{ e \in E_S / \exists H \subset S, \exists p \in Hpaths, e \in p \}
\end{array}
$$

[**Rec_2**]

Una posible implementación recursiva de la recurrencia [**Rec_2**], es la siguiente.

```
Procedure Min_costo_2AC(N, var A, var min_costo);
Begin
   OK=Es_2AristaConexo(N,A);
   if not(OK) then
      min_costo=INF;
   else if (Es_Ciclo(N,A)) then
           min_costo=COSTO(A);
        else
           min_costo=INF;
           H=H_Paths(N,A);
           for i=1 to (n-1) do
             for j=i to n do
               if  (i,j)∈A and (i,j)∉H  then
                 A=A-{(i,j)};
                OK=Es_2AristaConexo(N,A);
                if (OK) then
                   Min_costo_2AC(N,A,costo);
                   if  costo<min_costo then
                     min_costo=costo;
                   endif;
                endif
```

Se da a continuación la descripción de las funciones auxiliares utilizadas en el algoritmo "*Min_costo_2AC*".

3.4.3.3.1.1.1 Función "*H_Paths*"

Recibe como parámetros de entrada un conjunto de nodos $N \subseteq V$ y un conjunto de aristas $A \subseteq E$. Devuelve el conjunto de $Hpaths$ disjuntos del subgrafo $G_s(N, A)$. La implementación de esta función es sencilla y se puede hacer de manera muy eficiente.

Función "*Es_Ciclo*"

Recibe como parámetros de entrada un conjunto de nodos $N \subseteq V$ y un conjunto de aristas $A \subseteq E$. Devuelve TRUE si el subgrafo $G_s(N, A)$ es un ciclo; FALSE en caso contrario. En la implentación, basta chequear que todos los nodos del subgrafo $G_s(N, A)$ tengan grado 2 para determinar si se corresponde a un ciclo.

Función "*COSTO*"

Tiene como entrada un conjunto de aristas $A \subseteq E$; devuelve el valor $\sum_{(i,j)\in A} c_{ij}$.

Función "*Es_2AristaConexo*"

Recibe como parámetros de entrada un conjunto de nodos $N \subseteq V$ y un conjunto de aristas $A \subseteq E$. Devuelve TRUE si el subgrafo $G_s(N, A)$ es 2-arista-conexo; FALSE en caso contrario. La implementación de esta primitiva es igual a la de "*ES_2STEINER*" vista en *2.4.2*.

Proposición 19

La recurrencia [**Rec_1**] *y* [**Rec_2**] *son equivalentes.*

Dem:

Si $G_s(S, E_S)$ es 2-arista-conexo, se devuelve el valor INF.

Se analiza ahora el caso en que $G_s(S, E_S)$ sea 2-arista-conexo; por inducción en la cantidad de aristas, se demostrará que la recurrencia [**Rec_2**] da el menor costo de un subgrafo de G 2-arista-conexo que contiene exactamente el conjunto de nodos S y conjunto de aristas incluido en E_S.

Paso Base

La mínima cantidad de aristas que puede tener E_S para que $G_s(S, E_S)$ sea 2-arista-conexo es $|S|$. Esto se da cuando $G_s(S, E_S)$ es un ciclo. Como al eliminar cualquier arista de un ciclo, éste deja de ser 2-arista-conexo, entonces el mínimo costo es el costo del ciclo, es decir $COSTO(E_S)$.

Paso Inductivo

H.I.) Sea $G_s(S, E_S)$ 2-arista-conexo con $|E_S| < n_E$, entonces [**Rec_2**] da el menor costo de un subgrafo 2-arista-conexo con exactemente el conjunto de nodos S y conjunto de aristas incluido en E_S.

T.I.) Sea $G_s(S, E_S)$ 2-arista-conexo con $|E_S| = n_E$, entonces [**Rec_2**] da el menor costo de un subgrafo 2-arista-conexo con exactamente el conjunto de nodos S y conjunto de aristas incluido en E_S.

El conjunto de aristas $H_E = \{e \in E_S / \exists H \subset S, \exists p \in Hpaths, e \in p\}$ tiene que estar en la solución, pues si se elimina alguna de ellas existe al menos un

nodo al cual se llega a el por caminos que tienen al menos una arista en común.

Sea $F = \{e \in E_S / e \notin H_E, G_s(S, E_S - e) es 2AC\}$.

Las aristas de F son las consideradas en la recurrencia.

Si $F = \varnothing$, entonces el grafo $G_s(S, E_S)$ es 2-arista-conexo minimal y el mínimo costo es $COSTO(E_S)$.

Si $F \neq \varnothing$, existe al menos un $e \in E_S$ tal que $G_s(S, E_S - e)$ es 2-arista-conexo. La recurrencia considera $\forall e \in F$ los subgrafos $G_s(S, E_S - e)$ que son 2-arista-conexo. Como $|E_S - e| = n_E - 1$, por H.I. [**Rec_2**] da el menor costo de un subgrafo 2-arista-conexo con exactamente el conjunto de nodos S y conjunto de aristas incluido en $E_S - e$. El menor de esos costos es el costo de la solución óptima.

L.Q.Q.D.

Resolver el STECSP, es encontrar un subgrafo $G_s(S, E_S)$ 2-arista-conexo de costo mínimo tal que el conjunto de nodos terminales esté incluido en S. Para ello, hay que considerar todos los posibles conjuntos de nodos $S / T \subseteq S$, $S \subseteq V$.

Definición 20

Se define $C_T(S)$ con $T \subseteq S \subseteq V$ como el menor costo de un subgrafo 2-arista-conexo $G_s(A, E_A)$ de G, cuyo conjunto de nodos cumple $T \subseteq A \subseteq S$ y conjunto de aristas cumple $E_A \subseteq E_S$ (siendo E_S las aristas del subgrafo inducido por el conjunto de nodos S).

En base a esta definición, se propone una recurrencia de Programación Dinámica para $C_T(S)$. La misma viene dada por:

$$\begin{array}{|l} C_T(T) = Cm_{2ac}(T, E_T) \\ C_T(S) = Min\{Min\{C_T(S-i); i \in (S \setminus T), G_{(S-i)} \in \Gamma_{STECSP}\}, Cm_{2ac}(S, E_S)\}, \forall S / T \subseteq S. \end{array}$$

[**Rec_3**]

donde Γ_{STECSP} es el espacio de soluciones factibles del STECSP para el grafo G.

Proposición 21

Sea G un grafo en las condiciones del STECSP. La recurrencia [**Rec_3**] *da la mejor solución factible al STECSP considerando solamente el conjunto de nodos $S / T \subseteq S \subseteq V$ y el conjunto de aristas E_S del subgrafo inducido por el conjunto de nodos S. Se devuelve INF en caso que no exista solución factible en dichas condiciones.*

Dem:

La demostración se hará por inducción en la cantidad de nodos del conjunto S.

Paso Base

Si $S = T$, por definición $C_T(T)$ es el menor costo de un subgrafo 2-arista-conexo con exactamente el conjunto de nodos T y conjunto de aristas incluido E_T (con E_T el conjunto de aristas del subgrafo inducido por el conjunto de nodos T). Por definición de $Cm_{2ac}(T, E_T)$, se tiene entonces que $C_T(T)$ es equivalente a $Cm_{2ac}(T, E_T)$.

Como $Cm_{2ac}(T, E_T)$ da el menor costo de un subgrafo 2-arista-conexo con exactamente el conjunto de nodos T y conjunto de aristas incluido en E_T o INF en caso que no haya solución factible, entonces $C_T(T)$ es el costo de la mejor solución factible del STECSP considerando solo el conjunto de nodos T y conjunto de aristas E_T o vale INF si no existe solución factible en dichas condiciones.

Paso Inductivo

H.I.) Sea S un conjunto de nodos de G tal que $T \subset S \subset V, |S| < n_s$, entonces $C_T(S)$ da el costo de la mejor solución factible del STECSP considerando el

conjunto de nodos S y el conjunto de aristas E_S del subgrafo inducido por S.
Se devuelve INF si no existe solución factible en dichas condiciones.
T.I.) La propiedad se cumple también para $T \subset S \subseteq V, |S| = n_s$.

Sea S un conjunto de nodos de G tal que $T \subset S \subseteq V, |S| = n_s$.
Se analizan 2 casos. Supóngase primero que $\forall i \in (S \setminus T), G_{(S-i)} \notin \Gamma_{STECSP}$, esto implica que todos los nodos se $(S \setminus T)$ deben estar presentes en cualquier solución factible del espacio Γ_{STECSP}. Entonces es necesario considerar todas las soluciones factibles con exactamente el conjunto de nodos S
y conjunto de aristas incluido en E_S y tomar como mejor solución la de mínimo costo.
Toda solución factible del STECSP cumple la propiedad de ser 2-arista-conexo. Como la recurrencia considera el valor de $Cm_{2ac}(S, E_S)$, entonces se analizan todos las soluciones factibles del STECSP con conjunto de nodos S y conjunto de aristas incluido en E_S y se toma como solución la de menor costo, además se devuelve INF en caso de no existir soluciones factibles (subgrafos 2-arista-conexo) en dichas condiciones.
Considérese ahora el caso en que $\exists i \in (S \setminus T) / G_{(S-i)} \in \Gamma_{STECSP}$, esto implica que al menos existe una solución factible al STECSP con conjunto de nodos incluido en $(S - i)$.
Sea $A = \{i \in S / G_{(s-i)} \in \Gamma_{STECSP}\}$.
Es claro que la mejor solución factible del STECSP considerando el conjunto de nodos S y conjunto de aristas incluido en E_S, es la solución factible del STECSP de menor costo teniendo en cuenta las soluciones factibles que cumplen:

- conjunto de nodos igual a S y conjunto de aristas incluido en E_S,
- conjunto de nodos incluido en $(S - i)$ para algún $i \in A$, y conjunto de aristas incluido en $E_{(S-i)}$ (conjunto de aristas del subgrafo inducido por los nodos $(S - i)$).

Por H.I., $\forall i \in A$, se cumple que $C_T(S-i)$ es el costo de la mejor solución factible del STECSP considerando el conjunto de nodos $(S-i)$ y el conjunto de aristas $E_{(S-i)}$ del subgrafo inducido por S. Como además la recurrencia considera $Cm_{2ac}(S, E_S)$, se tiene también el menor costo de las soluciones factibles con exactamente los nodos S y conjunto de aristas incluido en E_S. El menor de todos estos costos, cumple ser el costo de la mejor solución factible del STECSP considerando el conjunto de nodos S y el conjunto de aristas E_S.

L.Q.Q.D.

Proposición 22

El valor de $C_T(V)$ es el costo de la solución optima del STECSP y vale INF si $\Gamma_{STECSP} = \varnothing$.

Dem:

La demostración es directa aplicando Proposición 21.

$C_T(V)$, por Proposición 21, da el costo de la mejor solución factible al STECSP considerando el conjunto de nodos V y el conjunto de aristas E y se devuelve INF en caso que no exista solución factible en dichas condiciones.

Como se analizan todas las soluciones factibles en Γ_{STECSP}, se obtiene entonces la de menor costo.

L.Q.Q.D.

Se brinda seguidamente una posible implementación de la recurrencia de Programación Dinámica para $C_T(V)$, donde se almacena además (en variables globales), el subgrafo para el cual se da el costo óptimo.

```
Procedure ALG2_STEINER(S,T);
Begin
1   E_S=Aristas_Inducidas(S);
2   if (S=T) then                                   (c_1)
3       Min_costo_2AC(S,E_S,costo);
4       if (costo<>INF) then                        (c_2)
5           Actualizar_Solución(S,E_S,costo);
6       end;
7   else
8       costo=INF;
9       for i=1 to n do
10          if ((i∈S) and (i∉T)) then               (c_3)
11              E_(S-i)=Elimino_Aristas(E_S,i);
```

Las funciones auxiliares son los siguientes.

Función "*Aristas Inducidas*"

Recibe como parámetro un conjunto de nodos $S \subseteq V$, y devuelve el conjunto de aristas inducidas por dicho conjunto en el subgrafo $G = (V, E)$.

Función "*Elimino Aristas*"

Recibe como parámetros un conjunto de aristas $E_S \subseteq E$, un nodo $i \in V$, y devuelve el conjunto de aristas de E_S que no tiene extremos en i.

Procedimiento "*Actualizar Solución*"

Recibe como parámetros un conjunto de nodos $N \subseteq V$, un conjunto de aristas $A \subseteq E$ con extremos en N, y el costo del subgrafo $G_s(N, A)$. Realiza las asignaciones: $A_{sol} = A$, $N_{sol} = N$ y $min\_costo = costo$, donde A_{sol}, N_{sol}, y min_costo son variables globales donde se almacena la mejor solución factible encontrada hasta el momento y su costo.

Sea $G=(V,E)$ un grafo en las condiciones del STECSP. Sea además un conjunto de nodos S tal que $T \subseteq S \subseteq V$. Asumiendo que la invocación "Min_costo_2AC(S,E$_S$,costo)" retorna el subgrafo de $G_s(S,E_S)$ 2-arista-conexo de costo mínimo que tiene exactamente el conjunto de nodos S y conjunto de aristas $A_S \subseteq E_S$; entonces la invocación "ALG2_STEINER(S,T)" busca y almacena (si existe) el subgrafo de $G_s(S,E_S)$ 2-arista-conexo de costo mínimo que cubre el conjunto de nodos terminales T (E_S es el conjunto de aristas inducidas por S en $G=(V,E)$). Se supone que inicialmente las variables globales tienen asignado: $A_{sol}=\varnothing$, $N_{sol}=T$ y $min\_\cos to = INF$.

Dem:

Por inducción en $n_S=|S|$.

3.4.3.3.1.1.2 Paso Base

El paso base se da cuando $T=S$ ($n_T=|T|=n_S=|S|$).

El algoritmo ejecuta (1) hallando el conjunto E_T de aristas inducidas por T *en* $G=(V,E)$.

La condición (c_1) es TRUE y se ejecuta la invocación "*Min_costo_2AC(T,E$_T$,costo)*" en (3). Por hipótesis esta invocación retorna (en caso de existir) el subgrafo de $G_s(T,E_T)$ 2-arista-conexo de costo mínimo que tiene exactamente el conjunto de nodos terminales T y conjunto de aristas $A_T \subseteq E_T$; dicha solución es almacenada como la óptima al ejecutarse (4-6) en "*ALG2_STEINER*". De no existir tal solución, el parámetro $\cos to$ retornado por "*Min_costo_2AC*" tiene valor INF, luego la condición (c_2) es FALSE y el algoritmo finaliza sin haber encontrado una solución factible.

3.4.3.3.1.1.3 Paso Inductivo

H.I.) La proposición se cumple $\forall S / T \subseteq S, n_T \leq |S| < k \leq n$.

T.I.) La proposición se cumple $\forall S / T \subset S, n_T < |S| = k \leq n$.

Sea S un conjunto de nodos tal que $|S| = k$. Como $T \subset S$ la condición (c_1) es FALSE y el algoritmo ejecuta (7). En (8) se realiza la asignación $\cos to = INF$.

Dado $U / T \subseteq U \subseteq V$ se define a $\Gamma_{STECSP}^{\subseteq U}$, como el subespacio de soluciones factibles del espacio Γ_{STECSP} que tienen como conjunto de nodos a $N / N \subseteq U$; el mismo se denota:

$\Gamma_{STECSP}^{\subseteq U} = \{G_s(N,A) / T \subseteq N \subseteq U; A \subseteq E_N; G_s(N,A) \in \Gamma_{STECSP}\}$.

$\forall i \in (S \setminus T)$ el algoritmo "*ALG2_STEINER*" realiza lo siguiente.

En (11), calcula $E_{(S-\{i\})}$ el conjunto de aristas inducidas por $(S-\{i\})$ en G.

En (12), la primitiva "*FACTIBLE*" chequea si $G_s(S-\{i\}, E_{(S-\{i\})}) \in \Gamma_{STECSP}$ (de cumplirse, significa que eventualmente puede existir un subgrafo de éste, que sea una mejor solución factible).

Si (c_4) es TRUE en (13) se realiza la invocación "*ALG2_STEINER(S-{i},T)*". Por H.I. se tiene que el algoritmo explora y almacena la mejor solución factible del espacio $\Gamma_{STECSP}^{\subseteq (S-\{i\})}$.

Si (c_4) es FALSE el algoritmo pasa a analizar el siguiente nodo de Steiner.

Luego de ejecutar n veces (9-16) se tiene que el algoritmo "*ALG2_STEINER*" buscó y almacenó la mejor solución factible del conjunto $\bigcup_{\forall i \in (S \setminus T)} \Gamma_{STECSP}^{\subseteq (S-\{i\})}$.

Sea Γ_{STECSP}^{S} el subespacio de soluciones factibles del STECSP que tienen como conjunto de nodos a S. En (17) se invoca "*Min_costo_2AC(S,E$_S$,costo)*"; por hipótesis se encuentra la mejor solución factible del subespacio Γ_{STECSP}^{S}. De haberse obtenido una mejor solución factible que la encontrada hasta el momento, en (18-20) se actualiza la mejor solución.

Se tiene entonces que el algoritmo "*ALG2_STEINER*" explora el conjunto de soluciones factibles del STECSP, dado por: $\Gamma_{STECSP}^{S} \cup \left(\bigcup_{\forall i \in (S \setminus T)} \Gamma_{STECSP}^{\subseteq (S-\{i\})} \right)$. Además este conjunto coincide con $\Gamma_{STECSP}^{\subseteq S}$, el subespacio de soluciones factibles del STECSP que tienen conjunto de nodos incluido en S.

L.Q.Q.D.

Proposición 24 (Correctitud del Algoritmo ALG2_STEINER)

Sea $G = (V, E)$ un grafo en las condiciones del STECSP, con $T \subseteq V$ el conjunto de nodos terminales. Entonces al invocar "ALG2_STEINER(V,T)" se obtiene la solución óptima al problema STECSP.

Dem:

La invocación de "*ALG2_STEINER(V,T)*", por Proposición 23, obtiene la mejor solución factible del STECSP considerando el subespacio $\Gamma_{STECSP}^{\subseteq V}$, el conjunto de soluciones factibles que tienen conjunto de nodos incluido en V . Es claro además que $\Gamma_{STECSP}^{\subseteq V} = \Gamma_{STECSP}$, con lo cual el algoritmo "*ALG2_STEINER*" busca y encuentra la mejor solución factible de la instancia del STECSP.

L.Q.Q.D.

3.5 2.5 Algoritmos Exactos Paralelo-Distribuidos para el STECSP

3.5.1 2.5.1 Introducción

En este apartado, se presentarán dos nuevos algoritmos paralelo-distribuidos que resuelven en forma exacta el problema STECSP. Primeramente se brinda una breve descripción del paradigma de desarrollo paralelo y distribuido. Luego, para cada uno de los algoritmos propuestos, se brinda una descripción del funcionamiento del algoritmo, pseudocódigo, y la demostración de su correctitud.

3.5.2 2.5.2 Paradigma del Desarrollo Paralelo-Distribuido

Hoy en día, es posible intentar resolver modelos matemáticos “muy complejos” en términos computacionales, los cuales serían prácticamente imposible considerar utilizando las técnicas clásicas de programación secuencial. Dos hechos relevantes han posibilitado el ataque de “problemas complejos” : el avance tecnológico, que ha producido mejoras significativas en la arquitectura de los sistemas y el desarrollo de las técnicas de procesamiento paralelo y distribuido.

Las técnicas de procesamiento paralelo y distribuido han permitido la definición de un nuevo paradigma en el cual la filosofía de funcionamiento de un sistema se basa en la división del problema en subproblemas más sencillos, los cuales pueden ser ejecutados en equipos de menor potencia o en redes no dedicadas. En ambientes académicos de investigación, este hecho es importante. Permite la utilización de máquinas ociosas de la red para la ejecución de tareas resultantes del análisis de problemas complejos paralelizables.

El paradigma utilizado en los desarrollos de sistemas paralelos y distribuidos, define una única gran Máquina Virtual compuesta por un conjunto de máquinas de arquitecturas heterogéneas. En un sistema paralelo, tres puntos esenciales son el uso de la memoria compartida, el pasaje de mensajes entre las distintas tareas que se ejecutan en paralelo y la utilización de la memoria compartida como comunicación entre tareas (sistema híbrido). En general, los sistemas paralelos pueden ser ejecutados sobre una arquitectura donde se tiene una única memoria común entre los procesos que se ejecutan en paralelo (Modelo de Multiprocesamiento Simétrico) o bien en forma distribuida, en la cual se tienen diferentes procesadores con memoria local. Al procesamiento paralelo con memoria compartida se lo denota MISD (Multi-Instruction, Single-Data), mientras que al procesamiento paralelo con memoria distribuida se lo denota como MIMD (Multi-Instruction, Multi-Data). En general los sistemas de multiprocesamiento con arquitectura MIMD, son los más usados a la hora de correr aplicaciones para resolver problemas científicos costosos (del tipo NP-difíciles).

La modalidad de ejecución de las aplicaciones distribuidas que corren sobre la Máquina Virtual, viene dada por un proceso llamado Maestro, el cual se encarga de iniciar los procesos a ejecutar en forma distribuida en la Máquina Virtual y el conjunto de procesos ejecutados en forma distribuida, los cuales intercambian parámetros de información con el proceso padre. A éstos últimos se les denomina procesos esclavos. El proceso maestro podrá invocar una serie de rutinas de cálculo para que se ejecuten en paralelo, pasándole los parámetros necesarios y luego esperar que los procesos esclavos envíen los resultados obtenidos.

La política de elección de máquinas en las cuales se ejecutarán los procesos esclavos (creados por el maestro) y la elección de los momentos en que estos deben ser ejecutados, se denomina *scheduling.* Los algoritmos de *scheduling*, se basan en el modelo conceptual del sistema paralelo. En dicho modelo, se representan los diferentes procesos a ser ejecutados como nodos y las relaciones de dependencias entre dichos procesos se representan mediante aristas; de esta manera se obtiene un grafo acíclico de dependencias entre procesos, las cuales condicionan el nivel de paralelismo del sistema.

El aumento de procesadores disponibles en una arquitectura distribuida de tipo MIMD no necesariamente implica una mejora en el tiempo de ejecución del sistema. Debe existir un balance entre el grado de paralelismo (dado por el número de tareas) y la complejidad del problema. Se define *speedup* como la medida de la mejora de la performance del sistema al aumentar la cantidad de procesadores (comparado con el caso lineal en que se utiliza un solo procesador).

Otro aspecto relevante de las arquitecturas distribuidas es la *escalabilidad*, definida como la habilidad de aumentar el número de procesadores para obtener mejor performance. Si bien esta característica es muy deseable en sistemas paralelizables, no debe omitirse la *Ley de Amdhal* la cual afirma que la parte no paralelizable de un programa determina un mínimo para el tiempo de la ejecución del mismo, aunque se utilicen técnicas para paralelizar el resto del programa.

El concepto de *granularidad* da una medida de la cantidad de trabajo realizado por cada proceso a ejecutar en paralelo. Un aumento de la

granularidad de un problema implica una disminución en la cantidad de comunicaciones entre procesos. La idea fundamental consiste en lograr un equilibrio entre el número de procesos y la complejidad introducida por las comunicaciones entre ellos.

La optimización de la comunicación entre procesos se ve afectada por restricciones físicas como el ancho de banda disponible y la latencia del canal de comunicación, así como por la eficiencia del protocolo utilizado. Los conceptos de *latencia* y *granularidad* se vinculan estrechamente. En un sistema con tiempos de latencia pequeños, es posible aplicar al máximo las técnicas de paralelismo, mientras que tiempos de latencia altos implican que se deba aumentar la granularidad del problema.

El software PVM [2] (Parallel Virtual Machine) es uno de los más utilizados a la hora de desarrollar aplicaciones paralelizables para ser ejecutadas en forma distribuida. Se define una *Máquina Virtual* la cual está constituida por un conjunto de máquinas de diferentes arquitecturas. Las tareas a ejecutar en forma distribuida se despachan entre los nodos de la máquina virtual, comunicándose entre si mediante pasaje de mensajes. El manejo de mensajes, conversión de datos entre arquitecturas y despacho de tareas es transparente al usuario, permitiéndose además la introducción de directivas explícitas sobre tipos de mensaje, utilización de recursos, etc.

Cuando se inicia una aplicación, lo primero que se hace es correr la tarea denominada “maestra” en alguna de las máquinas de la máquina virtual. Esta tarea podrá eventualmente hacer que se ejecuten otras tareas en forma paralela y distribuida en la máquina virtual. El pasaje de parámetros entre estas tareas se hace por medio de directivas del tipo PVM exclusivas para este fin.

Una tarea se ejecuta en forma local en una única máquina de la máquina virtual e intercambia parámetros usando primitivas de envío de mensajes.

Existen tres modelos a seguir cuando se desarrollan aplicaciones distribuidas. El primer modelo se denomina “crowd” y consiste en una colección de procesos ejecutando el mismo código e intercambiando parámetros entre ellos; este modelo se puede dividir en 2 categorías:

- Organización del tipo Master-Slave, existe una diferenciación entre los roles de los procesos del sistema, existe una tarea denominada “master” que es la encargada del despacho de procesos, inicialización, recolección y despliegue de resultados, además de un conjunto de tareas denominadas “slave” encargadas de procesar los datos mandados por el “master” y luego regresar los resultados obtenidos.

- Organización Node-Only, en la misma múltiples instancias del mismo programa se ejecutan a la vez.

Otro modelo de desarrollo de aplicaciones distribuidas es el denominado “tree”, los procesos son despachados formando un árbol de invocaciones, estableciéndose una relación padre-hijo.

Por último se tiene el modelo “hybrid”, el cual es una combinación de los modelos anteriores.

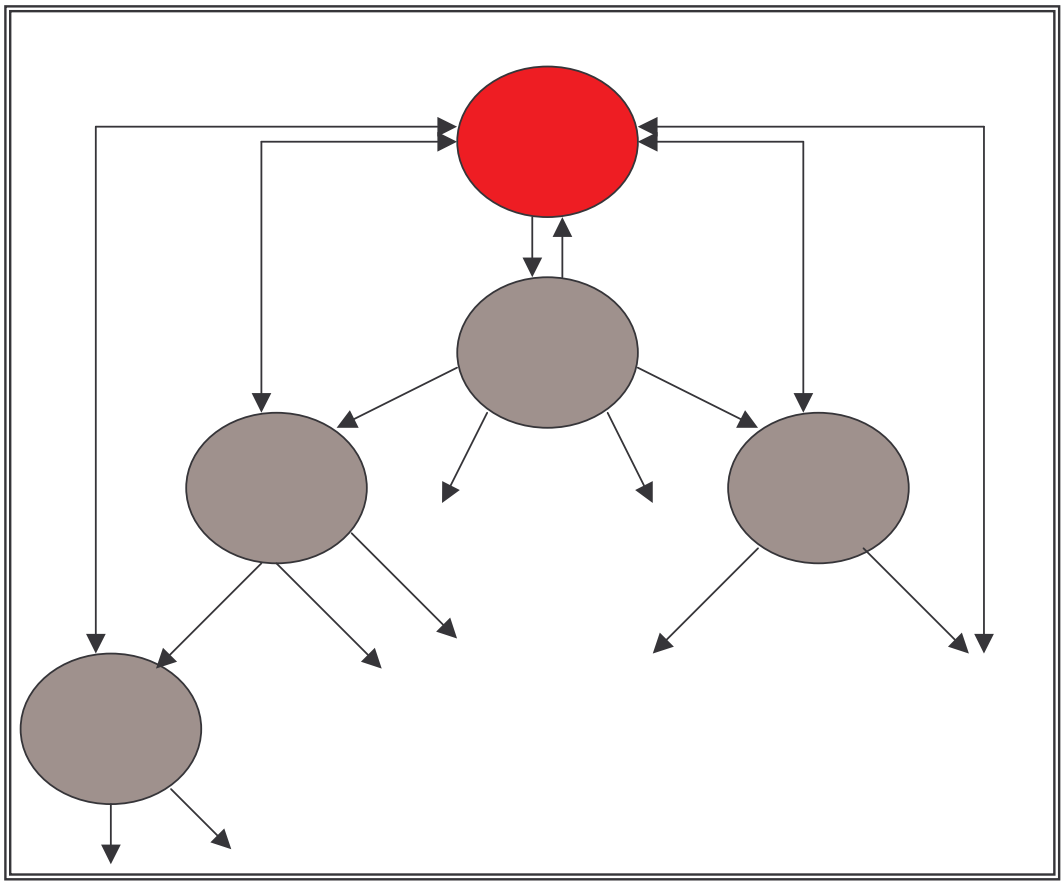
En la presentación de los algoritmos paralelo-distribuidos para la resolución del problema STECSP, se utilizarán primitivas de pasaje de mensajes del tipo PVM en la implementación de los pseudocódigos.

3.5.3 2.5.3 Algoritmo Exacto Paralelo-Distribuido-1 (AEPD1)

Se presenta ahora un algoritmo paralelo-distribuido (basado en el algoritmo de Backtracking “*ALG1_STEINER*”), el cual encuentra la solución óptima al STECSP. El algoritmo permite encontrar la solución óptima al STECSP paralelizando la búsqueda de soluciones factibles dentro del espacio de los subgrafos de G.

La arquitectura del algoritmo paralelo-distribuido es la siguiente: existe un proceso al cual se le denominará *Monitor* y procesos que se les denominará *Steiner_Slave*, los cuales se ejecutan en forma paralela y distribuida, formando una estructura del tipo “tree”, donde los procesos son despachados formando un árbol de invocaciones, estableciéndose una relación padre-hijo entre procesos.

El siguiente diagrama ilustra la arquitectura del modelo paralelo distribuido del *AEPD1*.



[Arquitectura del AEPD1]

3.5.3.1 2.5.3.1 Comunicación entre Procesos del AEPD1

Existe una comunicación bidireccional entre el proceso *Monitor* y los procesos *Steiner_Slave*. El proceso *Monitor* espera algún pedido del costo de la mejor solución hasta el momento y se lo envía al proceso *Steiner_Slave* que hizo el pedido, luego espera un segundo mensaje que indica si se enviará o no por parte del proceso *Steiner_Slave* una solución factible de menor costo que la actual. En el caso que el proceso *Steiner_Slave* haya encontrado una solución factible de menor costo, se la envía al *Monitor* y luego éste espera nuevos pedidos, en caso contrario, de no encontrarse una solución factible de menor costo, le envía al *Monitor* un mensaje indicándole que no le enviará una nueva solución.

La comunicación entre los procesos *Steiner_Slave* es unidireccional y de proceso padre a proceso hijo. Un proceso *Steiner_Slave*, luego de crear un proceso hijo del mismo tipo, le envía un mensaje pasándole los parámetros de ejecución para que el proceso *Steiner_Slave* hijo busque en forma paralela mejores soluciones dentro del espacio de soluciones factibles Γ_{STECSP} del STECSP.

En el diseño del algoritmo se utilizan una serie de primitivas, las cuales se detallan a continuación.

- *Mytid()* - devuelve un número que identifica el proceso en forma única,
- *Spawn(NOMBRE)* - crea un nuevo proceso del tipo NOMBRE, el cual se ejecutará en paralelo con el resto de la aplicación. Devuelve además el identificador del proceso creado,
- *send_Slave(id_p,PARAMETROS)* – envía los parámetros especificados en la lista PARAMETROS, al proceso *Steiner_Slave* con identificador de proceso id_p,
- *receive_Slave(*,PARAMETROS)* – recibe de cualquier proceso *Steiner_Slave*, los parámetros especificados en la lista PARAMETROS,
- *receive_Slave(id_p,PARAMETROS)* – recibe del proceso *Steiner_Slave* con identificador de proceso id_p, los parámetros especificados en la lista PARAMETROS,
- *Parent()* - devuelve el identificador del proceso padre,

- *send_Monitor(id_p,PARAMETROS)* – envía al proceso *Monitor* (identificado por id_p) los parámetros especificados en PARAMETROS,
- *receive_Monitor(id_p,PARAMETROS)* – recibe del proceso *Monitor* (identificado por id_p) los parámetros especificados en PARAMETROS.

3.5.3.2 2.5.3.2 Pseudocódigo del AEPD1

La descripción del pseudocódigo del *AEPD1* se hará detallando el funcionamiento de los procesos *Monitor* y *Steiner_Slave*, y la interacción entre ellos.

Descripción del Proceso Monitor

El proceso *Monitor* tiene como entradas un grafo G, la matriz c de costos asociados a las aristas, y el conjunto de nodos terminales T. La función del proceso *Monitor* es ir actualizando la mejor solución encontrada hasta el momento por los distintos procesos *Steiner_Slave* que se ejecutan en paralelo. Cuando se recibe de un proceso *Steiner_Slave* un pedido del tipo "*MINIMO_COSTO*", el proceso *Monitor* le envía a dicho proceso el costo de la mejor solución factible del STECSP encontrada hasta el momento. Luego espera que el proceso *Steiner_Slave* que hizo el pedido, le mande un mensaje indicando si se encontró o no una solución factible de menor costo. El proceso *Steiner_Slave* envía un mensaje que indica si es necesario actualizar o no la mejor solución. En caso que el mensaje sea del tipo "*ACTUALIZAR_SOLUCION*", el proceso *Monitor* espera que el proceso *Steiner_Slave* le envíe la nueva solución. La nueva solución encontrada y enviada por el *Steiner_Slave* consiste en un conjunto de aristas A, un conjunto de nodos $N / T \subseteq N$ y el costo del subgrafo $G_s(N, A)$. Una vez actualizada la solución, el proceso *Monitor* espera nuevos pedidos del valor de la mejor solución encontrada hasta el momento. El siguiente es el pseudocódigo del proceso *Monitor*.

```
Process Monitor(G,c,T);

1  A_sol=∅;                          /* Inicializo la mejor solución */
2  N_sol=∅;
3  min_costo=INF;
4  id_pm=Mytid();                    /* Obtengo el identificador del proceso Monitor */
5  id_ph=Spawn(Steiner_Slave);       /* Lanzo en paralelo el primer proceso Steiner_Slave */
6  send_Slave(id_ph,G,c,T,∅,T,1,1,0,id_pm);   /* Envío los parámetros al Steiner_Slave  raíz */
7  while (TRUE) do
8     receive_Slave(*,id_ps);
9     send_Slave(id_ps,min_costo);
10    receive_Slave(id_ps,str);
11    if  (str="ACTUALIZAR_SOLUCION") then
12       receive_Slave(id_ps,A,N,costo);
13       A_sol=A;
14       N_sol=N;
15       min_costo=costo;
16    endif;
17 endwhile;
EndProcess;
```

Descripción del Proceso Steiner_Slave

Cada proceso *Steiner_Slave*, recibe del proceso padre (proceso *Monitor* u otro proceso *Steiner_Slave*) los siguientes parámetros.

- Grafo G,
- Matriz c con los costos de las aristas del grafo,
- Los nodos terminales T,
- Conjunto de aristas $A \subseteq E$,
- Conjunto de nodos $N \subseteq V$,
- Nodo actual a ser considerado $i \in V$,
- Nodo adyacente actual $j \in V$,
- Costo del subgrafo $G_s(N, A)$ de G,
- Cantidad de nodo terminales considerados hasta el momento (*cant_t*),
- Identificador del proceso *Monitor* (*id_pm*).

Después de recibir estos parámetros, el proceso *Steiner_Slave* envía al proceso *Monitor* un pedido para que le sea enviado el costo de la mejor solución encontrada hasta el momento. Al recibir esta información, verifica que se hayan agregado al menos dos aristas desde el nodo actual i a un subgrafo que no era solución factible al STECSP y además que todos los nodos terminales ya hayan sido considerados. Si ambas condiciones se cumplen, compara el costo de $G_s(N, A)$ con el enviado por el proceso *Monitor*, en caso que el costo de $G_s(N, A)$ sea menor, analiza mediante la función "*ES_2STEINER*" si el subgrafo $G_s(N, A)$ es una solución factible al STECSP, si lo es, entonces el proceso *Steiner_Slave* envía al *Monitor* un mensaje indicando que le enviará la nueva solución obtenida y luego le envía la nueva solución para que el proceso *Monitor* la actualice.

Si $G_s(N, A)$ es factible para STECSP, pero su costo es mayor que el enviado por el proceso *Monitor*, entonces el *Steiner_Slave* finaliza su ejecución, no buscándose nuevas soluciones factibles agregando nodos y aristas a $G_s(N, A)$ pues aumentaría el costo de la solución. En cambio, si $G_s(N, A)$ no es factible para STECSP y su costo es menor que el costo enviado por el *Monitor*, entonces *Steiner_Slave* busca soluciones factibles agregando nodos y aristas al subgrafo $G_s(N, A)$. La forma en que se consideran los diferentes casos para la agregación de nodos y aristas al subgrafo actual, es igual que en el algoritmo "*ALG1_STEINER*", salvo que en vez de hacerse invocaciones recursivas se crean nuevos procesos *Steiner_Slave*, los cuales se ejecutan en forma paralela-distribuida y se les envía mediante mensajes los parámetros de ejecución. La ventaja de esta ejecución paralela-distribuida, está en que el espacio de soluciones factibles Γ_{STECSP} es particionado en subespacios disjuntos los cuales son analizados en forma independiente y paralela en búsqueda de una solución factible de menor costo que la actual. El siguiente es el pseudocódigo correspondiente al proceso *Steiner_Slave*.

```
Process Steiner_Slave;

1   id_pp=Parent();                                  /* Obtengo el identificador del proceso padre */
2   id_ps=Mytid();                                   /* Obtengo el identificador del proceso actual */
3   if (TypeProcess(id_pp)=Monitor) then
4     receive_Monitor(id_pm, G,c,T,A,N,i,j,costo,cant_t,id_pm);
5   else
6     receive_Slave(id_pp,G,c,T,A,N,i,j,costo,cant_t,id_pm);   /* Recibo del proceso padre
7   endif;                                                                  los parámetros de ejecución */
8   num_term=Cant_Term(T);
9   send_Monitor(id_pm,id_ps);                /* Pedido de  "min_costo"  */
10  receive_Monitor(id_pm,min_costo);   /* Recibo del Monitor el valor de "min_costo"  */
11  OK=FALSE;
12  if (cant_t=num_term) and (costo<min_costo) then
13    if   (ES_2STEINER(A,N)) then                        /* Test de factibilidad */
14       OK=TRUE;
15    endif;
16  ENDIF;
17  if (OK) then
18     send_Monitor(id_pm,"ACTUALIZAR_SOLUCION");   /* Envío al Monitor confirmación  de
                                                                        actualización */
19     send_Monitor(id_pm,A,N,costo,id_ps);                /* Envío al Monitor la nueva solución */
20  else
21       send_Monitor(id_pm,"NO_ACTUALIZAR");          /* Envío al Monitor el mensaje que
                                                                  no se actualiza la solución */
22       if  ((i≤n) and (costo<min_costo) then
23          if (i∉N) then
24             id_ph=Spawn(Steiner_Slave);
```

Proposición 25

Supóngase que en cierto instante de ejecución del "AEPD1" el valor de "min_costo" (costo de la mejor solución factible del STECSP encontrada hasta el momento) en el proceso Monitor es c_{min} *(eventualmente este valor puede ser INF), y hay además* $k > 0$ *procesos Steiner_Slave que han hecho el pedido al proceso Monitor del valor de "min_costo", entonces luego que los* k *pedidos son atendidos por el Monitor, el valor de "min_costo" es* $Min\{m_i \forall i \in 1..k, c_{min}\}$*, donde* m_i *es el costo del subgrafo* $G_s(N, A)$ *del i-ésimo proceso Steiner_Slave que hizo el pedido.*

Dem:

La demostración se hará por inducción en el valor de k.

Paso Base

Cuando $k = 1$, existe un solo proceso *Steiner_Slave* que hace el pedido del valor de "*min_costo*" al proceso *Monitor*.

Sea m_1 el costo de $G_s(N, A)$. La forma en que el pedido se atiende es la siguiente.

El *Steiner_Slave* hace el pedido ejecutando (9).

Al ejecutarse (8) en el proceso *Monitor*, se espera el pedido del valor de "*min_costo*" por parte de cualquier proceso *Steiner_Slave*. Como existe un pedido, el *Monitor* atiende el pedido enviando el valor c_{min} al *Steiner_Slave* que hizo el pedido (se ejecuta (9) en *Monitor*) y luego ejecuta (10) esperando confirmación de actualización de la solución actual. Al ejecutarse (10) en *Steiner_Slave*, se espera el valor c_{min} enviado por el *Monitor*, una vez obtenido este valor, el *Steiner_Slave* verifica en (12) las siguientes condiciones:

- grado del nodo actual i sea al menos 2 en $G_s(N, A)$ (pues como se cumple que $G_s(N - \{i\}, A - A_i)$ con $A_i = \{(x, i) \in A, x \in N\}$ no es factible para STECSP, se necesita haber agregado al menos 2 aristas desde el nodo i para que el subgrafo resultante pueda ser una solución factible al STECSP),
- el número de nodos terminales considerados sea igual a "*num_term*" (se debe cumplir que $T \subseteq N$),
- $m_1 < c_{min}$.

Si alguna de las condiciones no se cumple, entonces $G_s(N, A)$ no es una solución factible para STECSP y/o su costo es mayor que el de la mejor solución encontrada hasta el momento.

Si se satisfacen las tres condiciones, el *Steiner_Slave* verifica en (13-15) la factibilidad de $G_s(N, A)$ como solución al STECSP. En caso que $G_s(N, A) \in \Gamma_{STECSP}$, el *Steiner_Slave* en (18) envía al *Monitor* un mensaje indicándole que se debe actualizar la mejor solución y luego en (19) le envía

el conjunto de aristas A, el conjunto de nodos N, y m_1. Si $G_s(N,A) \notin \Gamma_{STECSP}$, en (21) , el *Steiner_Slave* le envía al *Monitor* un mensaje indicándole que no es necesario actualizar la mejor solución; luego a partir de (22) el *Steiner_Slave* busca soluciones factibles agregando nodos y aristas no consideradas hasta el momento.

Cuando en (10) en el *Monitor*, llega la confirmación del *Steiner_Slave*, en (11) se chequea si es necesario actualizar la solución. Si el mensaje del *Steiner_Slave* es del tipo "*ACTUALIZAR_SOLUCIÓN*", en (12) se recibe del *Steiner_Slave* la nueva solución, en (13-15) se actualiza la solución del *Monitor* (teniéndose que el valor de "*min_costo*" es ahora m_1) y luego se esperan nuevos pedidos. Si el mensaje del *Steiner_Slave* es "*NO_ACTUALIZAR*", el *Monitor* pasa a esperar nuevos pedidos y el valor de "*min_costo*" sigue siendo c_{min}.

Se tiene entonces, que luego de atender el pedido del *Steiner_Slave*, el proceso *Monitor* tiene el valor de su variable "*min_costo*" igual a $Min\{m_1, c_{min}\}$.

3.5.3.2.1.1.1 Paso Inductivo

H.I.) La Proposición se cumple para $1 \leq k < n$

T.I.) La Proposición se cumple para $k = n$.

Supóngase que hay n procesos *Steiner_Slave* que han hecho el pedido del valor de "*min_costo*" al proceso *Monitor*. Una vez que el *Monitor* recibe el pedido de un *Steiner_Slave*, no atenderá ningún otro pedido hasta que no finalice la atención del primero; esto se debe a que las primitivas utilizadas en (9), (10) y (12) del *Monitor* son para el proceso especificado con id_ps (identificador del proceso *Steiner_Slave* al cual se le atendió el pedido) y además la primitiva utilizada en (9) de *Steiner_Slave* es bloqueante. Se tiene entonces que el primer pedido se atiende sin la interrupción de los $(n-1)$ restantes, y al finalizar éste, el valor de "*min_costo*" viene dado por $Min\{m_1, c_{min}\}$, siendo m_1 el costo del subgrafo $G_s(N,A)$ correspondiente al *Steiner_Slave* que se le atendió el pedido. Como quedan $(n-1)$ pedidos de

los procesos *Steiner_Slave* para atender, por H.I., el valor de "*min_costo*" luego que los $(n-1)$ pedidos son atendidos, viene dado por :

$Min\{Min\{m_1, c_{min}\}, m_i \forall i \in 2..n\} = Min\{m_i \forall i \in 1..n, c_{min}\}$.

L.Q.Q.D.

Proposición 26

Considérese un proceso del tipo Steiner_Slave (se le llamará Steiner_Slave$^{(0)}$) que es lanzado en paralelo durante la ejecución del AEPD1, recibiendo del proceso padre los siguientes parámetros de ejecución.

Parámetros Invariantes:

- *grafo G ,*
- *matriz de costos c ,*
- *conjunto de nodos terminales T ,*
- *id_pm el identificador de proceso del Master.*

Parámetros Variables:

- *conjunto de nodos $N / T \subseteq N \subseteq V$,*
- *conjunto de aristas $A \subseteq E$ con extremos en N ,*
- *nodo $i \in V$ tal que $i \in T$ o $i \notin N$ y es analizado por vez primera por un proceso Steiner_Slave,*
- *variable costo que indica el costo del subgrafo $G_s(N, A)$,*
- *variable cant_t de nodos terminales en el conjunto N (considerados hasta el momento).*

Entonces el árbol de procesos Steiner_Slave despachados en paralelo que tiene como raíz al proceso Steiner_Slave$^{(0)}$ analizan (comparan con la mejor solución hasta el momento) las soluciones factibles del STECSP dentro del espacio de los subgrafos $SG(N, \{i..n\} \setminus T, A)$ de G ; además, si la solución óptima del Γ_{STECSP} esta en dicho subespacio, ésta es enviada al proceso Monitor y almacenada como mejor solución.

Dem:

La demostración se hará por inducción hacia atrás en el nodo i.

Una vez lanzado en paralelo el proceso *Steiner_Slave*$^{(0)}$, en (3-7) se esperan los parámetros de ejecución enviados por el proceso padre. Luego en (9) se envía al proceso *Monitor* el pedido del valor de "*min_costo*" (costo de la mejor solución factible encontrada hasta momento) y en (10) se espera dicho valor. Al ser recibido, es almacenado en la variable *min_costo*. El proceso *Monitor* queda esperando que el proceso *Steiner_Slave*$^{(0)}$ le envíe una confirmación de actualización de la solución actual.
Se analizan a continuación las diferentes situaciones de ejecución.

Caso A

Si $G_s(N,A) \in \Gamma_{STECSP}$, entonces $G_s(N,A)$ es la mejor solución al STECSP, considerando $SG(N,\{i..n\} \setminus T, A)$ (cualquier otra solución factible de éste subespacio incluye el conjunto de aristas A, por tanto su costo sería mayor).

Pueden darse la siguientes situaciones.

1. Si *(costo<min_costo)*, como *(num_term=cant_t)*, se ejecuta (13-15) verificando que $G_s(N,A)$ es una solución factible al STECSP; luego la condición en (17) es TRUE, y en (18) se le envía al proceso *Monitor* un mensaje indicándole que es necesario actualizar la mejor solución. Al recibir el proceso *Monitor* este mensaje, espera que el *Steiner_Slave*$^{(0)}$ le envíe la nueva solución, lo cual hace en (19), finalizando luego la ejecución del *Steiner_Slave*$^{(0)}$. Se cumple además que si $G_s(N,A)$ fuera la solución óptima, luego que el proceso *Monitor* actualiza la mejor solución, y suponiendo que otros procesos *Steiner_Slave* encuentren otras soluciones factibles para comparar, por Proposición 25, luego que los pedidos de estos procesos son atendidos, la solución almacenada por el *Monitor* seguirá siendo $G_s(N,A)$ hasta finalizar el AEPD1.
2. Si *(costo>min_costo)*, la condición en (17) es FALSE, ejecutandoce luego (20-21) donde se le envía al proceso *Monitor* un mensaje indicándole que no es necesario actualizar la mejor solución. Después, la condición en (22) es FALSE, y el proceso *Steiner_Slave*$^{(0)}$ finaliza.

Caso B

Veamos ahora el caso $G_s(N,A) \notin \Gamma_{STECSP}$. Luego de ejecutarse (1-16), la condición en (17) es FALSE. Se ejecuta (21) enviándole al proceso *Monitor* un mensaje que indica que no se enviará una nueva solución factible. El proceso *Monitor*, al recibir este mensaje, procede a atender otros pedidos de otros procesos *Steiner_Slave*. El *Steiner_Slave*$^{(0)}$, chequea la condición (22). Si $i > n$, se cumple que:

$\{G_s(N,A)\} = SG(N,\{i..n\} \setminus T, A)$, y el proceso finaliza no habiendo encontrado solución factible.

Si *(costo>min_costo)* en (22), entonces la solución óptima del Γ_{STECSP} no está en el subespacio $SG(N,\{i..n\} \setminus T, A)$, y el proceso finaliza su ejecución.

Si se cumple *(i≤n)* y *(costo<min_costo)* en (22), entonces se pueden dar las siguientes situaciones.

Caso B1

Si $i \notin N$, en (24) se crea un proceso *Steiner_Slave* hijo (llámesele *Steiner_Slave*$^{(1)}$), que por H.I., es raíz de un árbol de procesos *Steiner_Slave* que analiza las soluciones factibles del STECSP en el espacio de subgrafos $SG(N,\{i+1..n\} \setminus T, A)$ y además si la solución óptima de Γ_{STECSP} esta en $SG(N,\{i+1..n\} \setminus T, A)$ ésta es enviada al proceso *Monitor* y almacenada como mejor solución..

Al ejecutarse (26) en el proceso *Steiner_Slave*$^{(0)}$, si el nodo i no cumple los tests realizados por la función "*TEST*", entonces el proceso *Steiner_Slave*$^{(0)}$ finaliza, teniéndose que la mejor solución factible en $SG(N,\{i..n\} \setminus T, A)$ es la mejor solución factible de $SG(N,\{i+1..n\} \setminus T, A)$.

Si el nodo i cumple los tests realizados por la función "*TEST*", se crea otro proceso *Steiner_Slave* hijo (llámesele *Steiner_Slave*$^{(2)}$) cuyos parámetros variables son: (*A,N∪{i},i,1,costo,cant_t*). Éste proceso pretende analizar los subgrafos resultantes de incluir al nodo i como parte de la solución factible.

Se define ahora el conjunto $B = \{(i,k) \in E / k \in N, (i,k) \notin A\}$ (conjunto de aristas que van del nodo i a un nodo de N y no están en A), y se analizan las siguientes situaciones.

Caso B11

Supóngase que $\exists C \subseteq B / G_s(N \cup \{i\}, A \cup C) \in \Gamma_{STECSP}$.

Se definen los siguientes conjuntos: $F_B = \{G_s(N \cup \{i\}, A \cup C) \in \Gamma_{STECSP}, C \subseteq B\}$, el conjunto de soluciones factibles del STECSP que tienen como conjunto de nodos $N \cup \{i\}$ y conjunto de aristas $A \cup C$ para algún $C \subseteq B$, y $\overline{F_B} = \{G_s(N \cup \{i\}, A \cup C) \notin \Gamma_{STECSP}, C \subseteq B\}$, conjunto de subgrafos que no son solución factible del STECSP, y que tienen como conjunto de nodos $N \cup \{i\}$ y conjunto de aristas $A \cup C$ para algún $C \subseteq B$.

Se crea un árbol de procesos *Steiner_Slave*, que tiene como raíz al proceso *Steiner_Slave*$^{(2)}$ mediante la ejecución de (31-46) en las diferentes instancias de código de los procesos *Steiner_Slave* que van componiendo el árbol y corren en paralelo.

Los subgrafos del conjunto F_B son analizados en paralelo por procesos *Steiner_Slave* que integran el árbol. Se comparan las soluciónes factibles de F_B encontradas, con la solución factible del proceso *Monitor* en ese momento; si la solución encontrada es mejor que la del *Monitor*, se le envía le solución de F_B al proceso *Monitor* para que actualice la mejor solución encontrada hasta el momento. Los procesos *Steiner_Slave* que integran el árbol y encuentran soluciones factibles en F_B finalizan una vez que comparan la solución encontrada con la del proceso *Monitor*.

Los subgrafos del conjunto $\overline{F_B}$ también son analizados por procesos *Steiner_Slave* del árbol que tiene a *Steiner_Slave*$^{(2)}$ como raíz. Como los subgrafos de $\overline{F_B}$ no son solución factible al STECSP, hay que analizar los subgrafos resultantes de agregar nodos mayores que el nodo i a los subgrafos de $\overline{F_B}$, para ello los procesos *Steiner_Slave* que son “hojas” del árbol creado y que cumplen haber analizado un subgrafo de $\overline{F_B}$, ejecutan (48-49) en su código, creando un nuevo proceso *Steiner_Slave* al cual le

envían como parámetros variables de ejecución: *(A∪C,N∪{i},i+1,1,COSTO(A∪C),cant_t)* para algún $C \in C_{NF}$ donde $C_{NF} = \{C \subseteq B / G_s(N \cup \{i\}, A \cup C) \notin \Gamma_{STECSP}\}$.

Por H.I., si se define $H_{NF} = \bigcup_{\forall C \in C_{NF}} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup C)$, se tiene que el conjunto H_{NF} es analizado por árboles de procesos *Steiner_Slave* que tienen como raíz a un proceso *Steiner_Slave* cuyos parámetros variables de ejecución (enviados por el proceso padre) son del tipo *(A∪C,N∪{i},i+1,1,COSTO(A∪C),cant_t)* para algún $C \in C_{NF}$, y además si la solución óptima de Γ_{STECSP} esta en H_{NF}, ésta es enviada al proceso *Monitor* y almacenada como mejor solución.

Sea $H_T = SG(N, \{i+1..n\} \setminus T, A) \cup F_B \cup H_{NF}$. La mejor solución factible al STECSP considerando $SG(N, \{i..n\} \setminus T, A)$, es la mejor solución factible al STECSP considerando el conjunto H_T. Por tanto si la solución óptima de Γ_{STECSP} esta en $SG(N, \{i..n\} \setminus T, A)$, ésta es encontrada por algún proceso del árbol de procesos que tiene como raíz al proceso *Steiner_Slave*$^{(0)}$ y enviada al proceso *Monitor* para actualizar.

Caso B12

Si $\forall C \subseteq B, G_s(N \cup \{i\}, A \cup C) \notin \Gamma_{STECSP}$; se crea un árbol de procesos *Steiner_Slave* (ejecutando (31-46) en los diferentes procesos que se van creando) que tiene como raíz al proceso *Steiner_Slave*$^{(2)}$. La cantidad de procesos de éste árbol son al menos 2 y a lo sumo $2^{|B|} - 1$ (pues algunas soluciones son descartadas por comparación con soluciones factibles encontradas en paralelo por otros procesos *Steiner_Slave*) y pretenden considerar todas las agregaciones posibles de aristas del conjunto B. Los procesos "hojas" de este árbol ejecutan (48-49) en su código y se crean nuevos procesos *Steiner_Slave* hijos de éstos, a los cuales se les envía como parámetros variables:

(*A∪C,N∪{i},i+1,1,COSTO(A∪C),cant_t*) para algún *C⊆B*. Se tiene entonces que $\forall C \subseteq B$, por H.I., éstos procesos analizan las soluciones factibles al STECSP en el espacio de subgrafos $SG(N \cup \{i\}, \{i..n\} \setminus T, A \cup C)$.

Sea $H_B = \bigcup_{\forall C \subseteq B} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup C)$ y $H_T = H_B \cup SG(N, \{i+1..n\} \setminus T, A)$, entonces el árbol de procesos que tiene el proceso *Steiner_Slave*$^{(0)}$ como raíz, analiza todas las soluciones factibles del STECSP en el conjunto H_T, y como además se cumple que $H_T = SG(N, \{i..n\} \setminus T, A)$, el árbol de procesos analiza todas las soluciones factibles de $SG(N, \{i..n\} \setminus T, A)$ y en caso de encontrarse la óptima, ésta es enviada al proceso *Monitor* y almacenada por éste como mejor solución.

Caso B2

Supóngase ahora que $i \in T$.

Como $i \in T$ se ejecuta (30) en el proceso *Steiner_Slave*$^{(0)}$ (se verifica $i \in N$, pues por hipótesis $T \subseteq N$).

En pasos posteriores se crean a lo sumo $(2^{|B|} - 1)$ procesos *Stiner_Slave* hijos ejecutándoce (30-47) en cada proceso hijo creado. De éstos procesos, un número $p_F \leq |F_B|$ encuentra alguna solución factible del STECSP en F_B y la compara con la solución almacenada por el *Monitor* en ese momento, y un número $p_{NF} \leq |\overline{F_B}|$ de procesos no encuentra una solución en F_B y ejecuta (48-49) . Estos últimos, buscarán soluciones factibles al STECSP a partir de un subgrafo de $\overline{F_B}$ considerando nodos mayores al nodo i. Por H.I., el conjunto de subgrafos : $H_{NF} = \bigcup_{\forall C \in C_{NF}} SG(N \cup \{i\}, \{i+1..n\} \setminus T, A \cup C)$, es analizado por p_{NF} árboles de procesos que tienen un proceso *Steiner_Slave* como raíz, y además si la solución óptima de Γ_{STECSP} esta en H_{NF} , ésta es enviada al proceso *Monitor* y almacenada por éste como mejor solución.

Si se considera el conjunto $H_T = F_B \cup H_{NF}$, se tiene que el árbol de procesos que tiene como raíz el proceso *Steiner_Slave*$^{(0)}$ analiza las soluciones factibles de dicho conjunto, y además como la mejor solución factible de H_T es la mejor solución factible de $SG(N, \{i..n\} \setminus T, A)$, se concluye que el árbol de procesos analiza el espacio de subgrafos $SG(N, \{i..n\} \setminus T, A)$, y si la

solución óptima de Γ_{STECSP} esta en $SG(N,\{i..n\}\setminus T,A)$, ésta es enviada al proceso *Monitor* y almacenada como mejor solución.
L.Q.Q.D.

Proposición 27 (Correctitud del AEPD1)

La ejecución de "AEDP1" obtiene la mejor solución factible del espacio Γ_{STECSP} .

Dem:
Al comenzar la ejecución del *"AEDP1"*, el proceso *Monitor* al ejecutar (5) crea un proceso *Steiner_Slave* (llámesele *Steiner_Slave*$^{(Ini)}$) al cual le envía como parámetros de ejecución:
El conjunto de parámetros invariantes: grafo G , matriz de costos c , conjunto de nodos terminales T y *id_pm* el identificador de proceso del *Master*.

El conjunto de valores de los parámetros variables:

- conjunto de nodos $T = N$ de G ,
- conjunto de aristas vacío ($\varnothing$),
- nodo $1 \in V$,
- valor 0 correspondiente al costo de $G_s(T,\varnothing)$,
- valor 0 correspondiente al número de nodos terminales considerados hasta el momento.

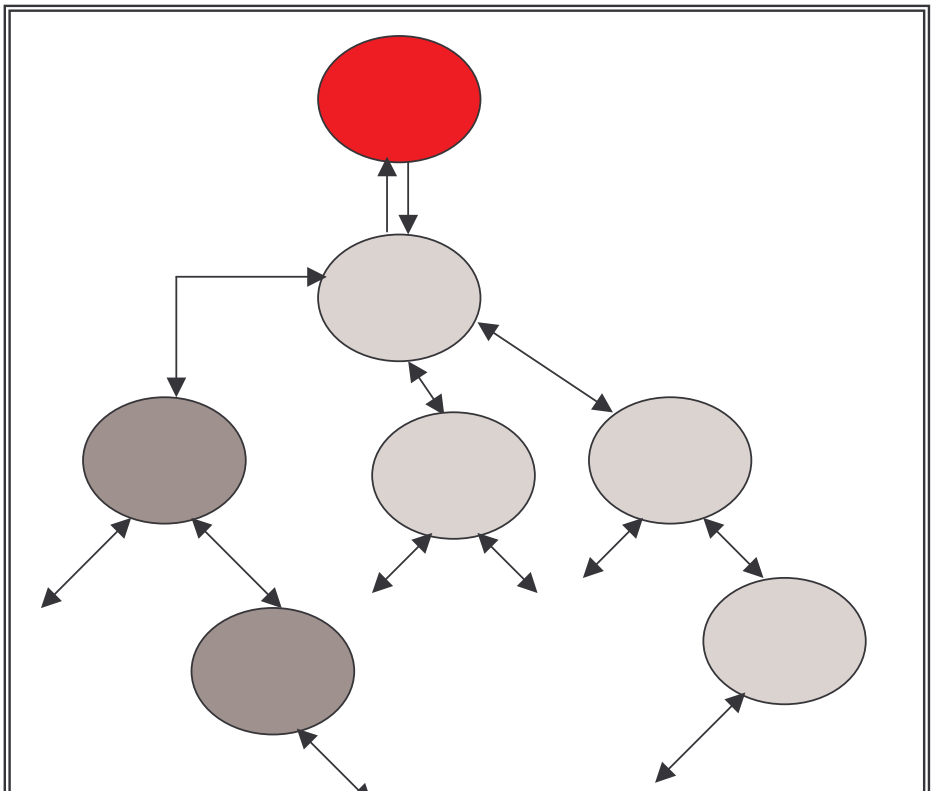
Por Proposición 26, el árbol de procesos *Steiner_Slave* que tiene como proceso raíz el proceso *Steiner_Slave*$^{(Ini)}$ analiza las soluciones factibles del STECSP dentro del espacios de subgrafos $SG(V,\{1..n\}\setminus T,\varnothing)$ de G . Como se cumple que $\Gamma_{STECSP} \subseteq SG(V,\{1..n\}\setminus T,\varnothing)$, se tiene entonces que la solución óptima es encontrada por algún proceso *Steiner_Slave* y enviada al proceso *Monitor*, el cual la almacena para comparar. Todos los procesos *Steiner_Slave* que encuentren luego nuevas soluciones factibles, por

Proposición 25, luego que todos los pedidos sean atendidos, la solución óptima será la que esté almacenada por el proceso *Monitor* como mejor solución.

L.Q.Q.D.

3.5.4 2.5.4 Algoritmo Exacto Paralelo-Distribuido-2 (AEPD2)

Se presenta ahora otro algoritmo paralelo-distribuido que obtiene la solución optima al STECSP. El mismo esta basado en el algoritmo de Programación Dinámica "*ALG2_STEINER*" visto en *2.4.3*. El algoritmo diseñado, consta de tres tipos de procesos que se ejecutan en forma paralela. Existe un proceso denominado *Master*, procesos del tipo *Steiner* y procesos del tipo *MC2AC*. La arquitectura del modelo paralelo-distribuido del *AEPD2* es la siguiente.



3.5.4.1 2.5.4.1 Comunicación entre los Procesos del AEPD2

El primer proceso a ser ejecutado es el proceso *Master*, recibiendo como parámetros de entrada un grafo G, la matriz c de costos asociados a las aristas, y el conjunto de nodos terminales T. El proceso *Master* crea un proceso del tipo *Steiner*, enviándole luego un mensaje con los parámetros de ejecución. El proceso *Steiner* creado se encarga de hallar la solución óptima al problema STECSP, particionando el espacio de soluciones factibles Γ_{STECSP} en subespacios de soluciones factibles disjuntos, los cuales son analizados en forma independiente y paralela por otros procesos *Steiner* hijos y el proceso *MC2AC*. La mejor solución factible encontrada por un proceso *Steiner* hijo, es enviada al proceso *Steiner* padre y comparada con las otras soluciones factibles enviadas por los otros procesos *Steiner* hijos y el proceso *MC2AC*. La mejor de estas soluciones será la solución óptima al STECSP y será la solución que envíe el proceso *Steiner* padre (raíz del árbol de procesos *Steiner*) al proceso *Master*.

El *AEPD2*, al igual que el *AEPD1*, utiliza una serie de primitivas de comunicación entre procesos, algunas de ellas ya fueron vistas en *AEPD1*, las otras se detallan a continuación.

- *TypeProcess(id_p)* – devuelve el tipo de proceso al cual corresponde el identificador de procesos *id_p* (tipo *Master*, tipo *Steiner* o tipo *MC2AC*),

- *receive_Master(id_p,PARAMETROS)* – recibe de un proceso del tipo *Master* con identificador de proceso id_p, los parámetros especificados en la lista *PARAMETROS*,
- *receive_Steiner(id_p,PARAMETROS)* – recibe de un proceso del tipo *Steiner* con identificador de proceso *id_p*, los parámetros especificados en la lista *PARAMETROS*,
- *receive_Steiner(*,PARAMETROS)* – recibe de cualquier proceso *Steiner*, los parámetros especificados en la lista *PARAMETROS*,
- *receive_MC2AC(id_p,PARAMETROS)* – recibe de un proceso del tipo *MC2AC*, con identificador de proceso id_p, los parámetros especificados en *PARAMETROS*,
- *receive_MC2AC(*,PARAMETROS)* – recibe de cualquier proceso *MC2AC*, los parámetros especificados en *PARAMETROS*,
- *send_Master(id_p,PARAMETROS)* – envía a un proceso del tipo *Master*, con identificador de proceso *id_p*, los parámetros especificados en la lista *PARAMETROS*,
- *send_Steiner(id_p,PARAMETROS)* – envía a un proceso del tipo *Steiner*, con identificador de proceso *id_p*, los parámetros especificados en la lista *PARAMETROS*,
- *send_MC2AC(id_p,PARAMETROS)* - envía a un proceso del tipo *MC2AC*, con identificador de proceso *id_p*, los parámetros especificados en la lista *PARAMETROS*.

3.5.4.2 2.5.4.2 Pseudocódigo del AEPD2

La arquitectura del *AEPD2* tiene como base los procesos *Master*, *Steiner* y *MC2AC*. Para cada uno de estos procesos se verá: descripción de funcionamiento, su interacción con otros procesos y pseudocódigo del mismo.

Descripción del Proceso Master

El proceso *Master*, recibe como entrada un grafo en las condiciones del STECSP (un grafo $G = (V, E)$, la matriz c de costos asociados a las aristas y el conjunto de nodos terminales T). Una vez creado el proceso

Master, éste crea un proceso del tipo *Steiner* al cual le envía un mensaje cuyo contenido es el grafo $G=(V,E)$, la matriz c , el conjunto de nodos terminales T y la cantidad n de nodos del grafo.

Luego de creado este proceso, el proceso *Master* queda esperando que éste le mande la solución óptima del STECSP. Una vez que el proceso *Steiner* encuentra la solución optima, se la envía al *Master*, finalizando así el algoritmo "*AEPD2*".

El pseudocódigo del proceso *Master* es el siguiente.

```
Process Master(V,E,c,T);

1  n=Cant_Nodos(V);
2  id_p=Spawn(Steiner);
3  send_Steiner(id_p,V,E,T,c,n);
4  receive_Steiner(id_p,N_Sol,A_Sol,min_costo);
EndProcess;
```

Descripción del Proceso Steiner

Cuando se crea un proceso *Steiner*, éste espera que el proceso padre le envíe los parámetros de ejecución. El proceso padre puede ser el proceso *Master* (solo un proceso *Steiner* tiene como padre al proceso *Master*) u otro proceso *Steiner*. Cualquiera sea el caso, los parámetros que se reciben del proceso padre son:

- un conjunto de nodos $S / T \subseteq S \subseteq V$,
- un conjunto de aristas E_S (correspondientes al subgrafo inducido por el conjunto de nodos S),
- conjunto de nodos terminales T ,
- matriz de costos c ,
- cantidad n de nodos del grafo G .

Cada proceso *Steiner*, pretende resolver el problema de encontrar la mejor solución factible al STECSP, cuyo conjunto de nodos esté incluido en S y conjunto de aristas incluido en E_S .

En caso que el conjunto de nodos S coincida con el conjunto de nodos terminales T , el proceso *Steiner* crea un proceso del tipo *MC2AC* y le envía

un mensaje con los parámetros de ejecución T, E_T y c. El proceso *MC2AC* creado, calcula el subgrafo 2-arista-conexo de menor costo que contiene como nodos al conjunto T y conjunto de aristas $E_{Sol} \subseteq E_T$ y se lo envía al proceso *Steiner* padre. Cuando el proceso *Steiner* recibe esta solución, la devuelve al proceso padre como solución óptima.

Si se cumple $T \subset S$, entonces el proceso *Steiner* crea un proceso *MC2AC* y le envía un mensaje con los parámetros de ejecución S, E_S y c. Luego $\forall i \in S / i \notin T$ y $G_s(S-\{i\}, E_{(S-\{i\})}) \in \Gamma_{STECSP}$ crea un proceso *Steiner* al cual se le envía un mensaje con los parámetros $(S-\{i\})$, $E_{(S-\{i\})}$, c y n. Una vez lanzados en paralelos todos estos procesos, como cada uno de estos procesos *Steiner* hijo, obtiene la mejor solución factible al STECSP, considerando el conjunto de nodos $(S-\{i\})$ y el conjunto de aristas $E_{(S-\{i\})}$, el proceso *Steiner* padre espera a que cada uno de sus procesos *Steiner* hijo finalice y le envíe la solución óptima encontrada. Luego que se reciben todas las soluciones enviadas por los procesos *Steiner* hijos, la mejor de éstas soluciones, es la mejor solución factible al STECSP, considerando el conjunto de nodos $(S-\{i\})$ y conjunto de aristas $E_{(S-\{i\})}$ para algún $i \in S, i \notin T$. Después, el proceso *Steiner*, recibe la solución enviada por el proceso *MC2AC* hijo, dicha solución es el subgrafo 2-arista-conexo de menor costo que contiene como nodos al conjunto S y conjunto de aristas $E_{Sol} \subseteq E_S$. La mejor de todas estas soluciones, cumple ser la mejor solución al STECSP, cuyo conjunto de nodos esté incluido en S y conjunto de aristas incluido en E_S. Esta solución, es entonces enviada como solución óptima al proceso padre.

El pseudocódigo del proceso *Steiner* es el siguiente.

```
Process Steiner;

1  min_costo=INF;          /* inicializo "min_costo" */
2  id_pp=Parent();         /* obtengo identif. del proceso padre */
3  if (TypeProcess(id_pp)=Master) then    /* recibo del padre los parámetros de ejecución */
4     receive_Master(id_pp,S,E_S,T,c,n);        /* el proceso padre es el proceso Master */
5  else
6     receive_Steiner(id_pp,S,E_S,T,c,n);       /* el proceso padre es un proceso Steiner */
7  endif;
8  if (S=T) then
```

Descripción del Proceso MC2AC

Al ser creado un proceso *MC2AC*, éste espera que el proceso padre le envíe los parámetros de ejecución. El padre de un proceso *MC2AC* puede ser un proceso del tipo *Steiner* u otro proceso *MC2AC*. En ambos casos, los parámetros que se reciben del proceso padre por medio de un mensaje son:

un conjunto de nodos $N \subseteq V$, un conjunto de aristas $A \subseteq E$ con extremos en N, y la matriz de costos c. Lo que busca un proceso *MC2AC*, es obtener un subgrafo 2-arista-conexo de costo mínimo, que contenga exactamente el conjunto de nodos N y conjunto de aristas $A_{Sol} \subseteq A$. Para ello realiza lo siguiente. Con los parámetros recibidos, chequea mediante la función "*Es_2AristaConexo*", si el subgrafo $G_s(N, A)$ es 2-arista-conexo; si no lo es, entonces no existe solución y se envía $A_{Sol} = \varnothing$ y *min_costo*=INF al proceso padre. Si $G_s(N, A)$ es un ciclo, entonces se envía al proceso padre un mensaje con $A_{Sol} = A$ y *min_costo=COSTO(A)* como solución óptima. En el caso que $G_s(N, A)$ sea 2-arista-conexo y no sea un ciclo, el proceso *MC2AC* construye el conjunto H de las aristas que están en algún $HPath$ de $G_s(N, A)$. Estas aristas no se pueden descartar, necesariamente tienen que estar en la solución pues de lo contrario habría algún nodo de grado 1. Luego, $\forall e \in A / e \notin H$, si $G_s(N, A - \{e\})$ es 2-arista-conexo, el proceso *MC2AC* crea un proceso *MC2AC* hijo, y le envía como parámetros de ejecución N y $(A - \{e\})$. El proceso *MC2AC*, espera que cada uno de los procesos *MC2AC* hijo finalice y le envíe su solución óptima. La mejor de estas soluciones, es el subgrafo 2-arista-conexo de costo mínimo que cubre N con aristas de $A \subseteq E$. Se envía dicha solución al proceso padre. Si se cumpliese que $\forall e \in A / e \notin H \quad G_s(N, A - \{e\})$ no es 2-arista-conexo, entones $G_s(N, A)$ es 2-arista-conexo minimal y no es un ciclo, en dicho caso, el proceso *MC2AC*, le envía al proceso padre $A_{Sol} = A$ y *min_costo=COSTO(A)* como solución óptima.

El siguiente es el pseudocódigo correspondiente al proceso *MC2AC*.

```
Process MC2AC;

1   id_pp=Parent();
2   A_Sol=∅;
3   if (TypeProcess(id_pp)=Steiner) then
4      receive_Steiner(id_pp,N,A,c);
5   else
6      receive_MC2AC(id_pp,N,A,c);
7   end;
8   OK=Es_2AristaConexo(N,A);
9   if not(OK) then
10     min_costo=INF;
11  else
12     if (Es_Ciclo(N,A)) then
13        min_costo=COSTO(A,c);
14        A_Sol=A;
15     else
16        min_costo=INF;
17        A_Sol=∅;
18        H=H_Paths(A);
19        cant_hijos=0;
20        for i=1 to (n-1) do
21           for j=i to n do
22              if (i,j)∈A and (i,j)∉H then
23                 A=A-{(i,j)};
24                 OK=Es_2AristaConexo(N,A);
25                 if (OK) then
26                    id_ph=Spawn(MC2AC);
27                    send_MC2AC(id_ph,N,A);
28                    cant_hijos=cant_hijos+1;
29                 endif;
30                 A=A∪{j};
31              endif;
32           endfor;
33        endfor;
34        for k=1 to cant_hijos do
35           receive_MC2AC(*,A,costo);
36           if (costo<min_costo) then
37              A_Sol=A;
```

Proposición 28

Sea un proceso del tipo MC2AC con parámetros de ejecución: un conjunto de nodos $N \subseteq V$, un conjunto de aristas $A \subseteq E$ con extremos en N, y la matriz de costos c. Entonces, al finalizar su ejecución, envía al proceso padre el subgrafo $G_s(N, A_{Sol})$ (con $A_{Sol} \subseteq A$) 2-arista-conexo de costo mínimo que cubre el conjunto de nodos N. Si $G_s(N, A)$ no es 2-arista-conexo, entonces se le envía al padre $A_{Sol} = \varnothing$ y min_costo=INF.

Dem:

La demostración se hará por inducción en la cantidad de aristas del conjunto A.

En (3-7) de *MC2AC*, se reciben los parámetros de ejecución por parte del proceso padre (un proceso *Steiner* u otro proceso *MC2AC*).

Se pueden dar los siguientes casos: $|A| < |N|$ o $|A| \geq |N|$.

Caso A

Se cumple $|A| < |N|$.

En (8) de *MC2AC*, el algoritmo chequea si $G_s(N, A)$ es 2-arista-conexo. El valor de retorno de la función "*Es_2AristaConexo*" es FALSE pues es necesario que $|A| \geq |N|$ para que $G_s(N, A)$ sea 2-arista-conexo. Luego se cumple la condición en (9), asignándose INF a *min_costo* en (10) y en (47-51) se envía al proceso padre un mensaje con $A_{Sol} = \varnothing$ y *min_costo=INF*.

Caso B

Se cumple $|A| \geq |N|$.

La función "*Es_2AristaConexo*" en (8) retorna TRUE o FALSE dependiendo si $G_s(N, A)$ es 2-arista-conexo o no. Si $G_s(N, A)$ no es 2-arista-conexo, entonces se cumple la condición en (9), en (10) se le asigna INF a *min_costo*, y en (47-51) se envía al proceso padre $A_{Sol} = \varnothing$ y *min_costo=INF* como resultado; y luego finaliza el proceso. En caso que $G_s(N, A)$ sea 2-arista-conexo, en (12) se chequea si el subgrafo $G_s(N, A)$ es un ciclo (se cumple $|A| = |N|$); si lo es, en (13-14) se asigna $A_{Sol} = A$ y a *min_costo=COSTO(A)*, y se los envía como solución al proceso padre en (47-51).

Veamos ahora el caso en que $G_s(N, A)$ es 2-arista-conexo y no es un ciclo ($|N| \leq k < |A| = n_A$). El proceso ejecuta (15), luego asigna en (16-17) $A_{Sol} = \varnothing$ y *min_costo=INF*. En (18) el proceso mediante la función "*H_Paths*" haya el conjunto de aristas $H = \{e \in p \,/\, p \; es \; un \; HPath \; de \; G_S(N, A)\}$. Estas aristas necesariamente tienen que estar en cualquier subgrafo de $G_s(N, A)$ 2-arista-conexo que cubra el conjunto de nodos N, de lo contrario habría algún nodo con grado 1 en la solución.

Hallado el conjunto H, en (19) se inicializa la variable *cant_hijos* en 0, ésta se utiliza luego para contar la cantidad de procesos *MC2AC* hijos despachados en paralelo.

Sea $B = \{e \in A \,/\, e \notin H, G_s(N, A - \{e\}) \; es \; 2AC\}$.

Si $|B| = 0$, al ejecutarse (20-33) no se despacha ningún proceso *MC2AC* hijo en paralelo, y *cant_hijos* sigue valiendo 0. Se verifica la condición en (41) y

se asigna $A_{Sol} = A$ y a *min_costo=COSTO(A)* en (42-43). Luego en (47-51) se envía el resultado al proceso padre.

El caso anterior se da cuando $G_s(N, A)$ es un grafo 2-arista-conexo minimal y no es un ciclo.

Si $|B| > 0$, al ejecutarse (20-33), $\forall e \in B$ se crea un proceso *MC2AC* hijo al cual el proceso *MC2AC* padre le envía como parámetros de ejecución el conjunto N de nodos y el conjunto $(A - \{e\})$. Como $\forall e \in B$, $G_s(N, A - \{e\})$ es 2-arista-conexo y además $|A - \{e\}| = n_A - 1 < n_A$, por H.I., cada uno de los procesos *MC2AC* hijo que corren en paralelo devuelven en (35) el subgrafo de $G_s(N, A - \{e\})$ 2-arista-conexo de costo mínimo que cubre el conjunto de nodos N. El subgrafo de menor costo, es guardado como solución en (37-38) y en (47-51) se devuelve éste como solución óptima al proceso padre.

L.Q.Q.D.

Proposición 29

Un proceso del tipo Steiner con parámetros de ejecución: un conjunto de nodos S / $T \subseteq S \subseteq V$, conjunto de aristas E_S (aristas del subgrafo inducido por el conjunto de nodos S), el conjunto de nodos terminales T, matriz de costos c, y n la cantidad de nodos de V. Luego de finalizar su ejecución, envía al proceso padre (proceso del tipo Master u otro proceso Steiner), el subgrafo $G_s(N_{Sol}, A_{Sol})$ que cumple ser la mejor solución factible al STECSP, considerando el conjunto de nodos S y el conjunto de aristas E_S.

Dem:

Al comenzar la ejecución del proceso *Steiner*, en (1) se asigna costo INF a la solución actual y en (3-7) se reciben los parámetros de ejecución del proceso padre. Se analiza a continuación el resultado de la ejecución del proceso *Steiner* con estos parámetros, demostrando la proposición por inducción en la cantidad de nodos del conjunto S.

Paso Base

Si $(T = S)$ se cumple la condición en (8). El proceso *Steiner* crea un proceso del tipo *MC2AC* en (7), y envía en (10) como parámetros el conjunto de nodos T, el conjunto de aristas E_T (inducido por T), y la matriz de costos c. Por Proposición 28, el proceso *MC2AC* creado, envía al proceso padre *Steiner*, el subgrafo $G_s(T, E_{Sol})$ 2-arista-conexo de costo mínimo que cubre los nodos de T con el conjunto de aristas $E_{Sol} \subseteq E_T$. Esta solución es enviada por el proceso *Steiner* al proceso padre en (43-47).

Paso Inductivo:

H.I.) La Proposición se cumple para $|T| \leq |S| = k < n_S$.

T.I.) La Proposición se cumple para $|S| = n_S$.

Supóngase ahora que $|T| < |S| = n_S$. El proceso *Steiner* asigna *costo=INF* y *cant_hijos=0* en (15-16). Crea un proceso *MC2AC* en (17) y le envía como parámetros de ejecución, el conjunto de nodos S, y el conjunto de aristas E_S. Sea $F_S = \{i \in S / i \notin T, G_s(S - \{i\}, E_{(S-\{i\})}) \in \Gamma_{STECSP}\}$.

Si $F_S = \varnothing$, al ejecutarse (19-27) no se crea ningún proceso *Steiner* hijo, y *cant_hijos* sigue con valor 0, luego (28-35) no se ejecuta, y en (36) el proceso *Steiner* espera que el proceso *MC2AC* que corre en paralelo, finalice y le envíe la solución. Por Proposición 27, el proceso *MC2AC* le envía un subgrafo $G_s(S, E_{Sol})$ (con $E_{Sol} \subseteq E_S$) 2-arista-conexo de costo mínimo que cubre el conjunto de nodos S o $E_{Sol} = \varnothing$ y *min_costo=INF* si $G_S(S, E_S)$ no es 2-arista-conexo. Cualquiera sea el caso, dicha solución, es la solución óptima y es enviada por el proceso *Steiner* al proceso padre en (43-47).

Si $F_S \neq \varnothing$, al ejecutarse (19-27), $\forall i \in F_S$ el proceso *Steiner* crea procesos *Steiner* hijos que corren en paralelo, y les envía como parámetros, el conjunto de nodos $(S - \{i\})$, el conjunto de aristas $E_{(S-\{i\})}$, los nodos terminales T, la matriz de costos c, y la cantidad n de nodos del grafo. Como $|(S - \{i\})| = |S| - 1 = n_S - 1 < n_S$, por H.I., cada uno de éstos procesos *Steiner*

hijo, obtiene la mejor solución factible al problema STECSP, considerando el conjunto de nodos $(S-\{i\})$ y el conjunto de aristas $E_{(S-\{i\})}$ y la envía al proceso padre. Al ejecutarse (28-35), se reciben todas las soluciones enviadas por los procesos *Steiner* hijo y se almacena la de menor costo. En (36) se espera el resultado enviado por el *MC2AC* creado en (17), por Proposición 28, éste envía al proceso *Steiner* padre el subgrafo 2-arista-conexo de costo mínimo que cubre S con aristas de E_S, el costo de este subgrafo es comparado con el costo de la solución almacenada, la solución de menor costo se almacena como solución óptima en (37-41). Luego en (43-47) se envía al proceso padre la solución óptima encontrada.
L.Q.Q.D.

Proposición 30 (Correctitud del AEPD2)

Sea $G=(V,E)$ un grafo en las condiciones del STECSP, con matriz c de costos asociados a las aristas y conjunto de nodos terminales T. Entonces, al ejecutar el AEPD2 se obtiene la solución óptima al STECSP.

Dem:

Al comenzar la ejecución del *AEPD2*, el proceso *Master* recibe como parámetros de entrada: el grafo $G=(V,E)$, la matriz c de costos y el conjunto de nodos terminales T. En (1) se calcula la cardinalidad de V (n), luego en (2) se crea un proceso del tipo *Steiner*, al cual se le envía en (3) los parámetros de ejecución V, E, T, c y n. Por Proposición 29, el proceso *Steiner* creado encuentra la mejor solución factible del STECSP considerando el conjunto de nodos V y el conjunto de aristas E, o sea, el proceso *Steiner* analiza todo el espacio Γ_{STECSP}, retornando la solución óptima. Esta solución es recibida en (4) por el proceso *Monitor*, finalizando así el algoritmo *AEPD2*.
L.Q.Q.D.

3.6 2.6 Implementación de un Algoritmo Paralelo-Distribuido para el STECSP

3.6.1 2.6.1 Introducción

Los algoritmos AEPD1 y AEPD2 vistos en *2.5.3* y *2.5.4* respectivamente, son dos posibles alternativas a la hora de implementar un algoritmo exacto paralelo-distribuido para el STECSP. Se eligió como modelo a implementar el AEPD1 debido a que éste es más sencillo de programar y además la arquitectura del modelo distribuido que modela la interacción entre los diferentes procesos es menos compleja que la del AEPD2. Seguidamente se verá: una descripción del modo de trabajo de la herramienta PVM, los aspectos de implementación del AEPD1 en PVM, los casos de prueba seleccionados y los resultados experimentales obtenidos.

3.6.2 2.6.2 PVM – Modo de Trabajo

El modo de trabajo con PVM es el siguiente. Se programan los diferentes procesos que se ejecutarán en paralelo y la forma en que éstos interactúan entre sí, además acceden a los recursos de PVM a través de una biblioteca de rutinas de interface standard. Estas rutinas permiten la iniciación y terminación de procesos a través de la red, así como también la comunicación y sincronización entre procesos. Algunas de las funcionalidades de PVM consisten en permitir que los procesos en cualquier momento de la ejecución de la aplicación en paralelo puedan comenzar o detener la ejecución de otros procesos, así como agregar o quitar máquinas que forman parte de la Máquina Virtual.

El sistema PVM esta compuesto básicamente de dos partes. Una de ellas es un proceso demonio llamado *pvmd3*, que reside en todas las máquinas que conforman la Máquina Virtual. Este demonio es el encargado de controlar y monitorear la comunicación entre las distintas máquinas que integran la Máquina Virtual. Al correr la aplicación PVM, lo primero que se debe hacer es definir (crear) una Máquina Virtual, para ello se inicia el demonio *pvmd3* en un equipo cualquiera y luego desde el mismo se inicia el

demonio en los demás equipos que integrarán la Máquina Virtual. Luego se debe ejecutar la aplicación PVM desde cualquier máquina que integre la Máquina Virtual.

La segunda componente de PVM consiste en la biblioteca que éste posee y cuyas rutinas sirven de interface con el programador. La librería incluye un conjunto de primitivas, las cuales son necesarias para poder llevar a cabo la ejecución distribuida de los procesos que componen la aplicación. Incluye también rutinas de coordinación de procesos, pasaje de parámetros, modificación de la Máquina Virtual, control de procesos, etc.

Los lenguajes de programación que actualmente soporta PVM para el desarrollo de sus aplicaciones son C, C++ y FORTRAN. Al desarrollar las aplicaciones en alguno de éstos lenguajes, el programador debe embeber en el código las invocaciones a las primitivas de biblioteca de PVM. Los distintos programas que conforman la aplicación se corresponden con los procesos que serán ejecutadas en forma distribuida en la Máquina Virtual. Estos programas se compilan por separado para cada una de las arquitecturas presentes en la Máquina Virtual, y el código objeto se deja en un lugar accesible por las máquinas.

3.6.3 2.6.3 Implementación en PVM

La implemantación del AEPD1 se realizo utilizando la herramienta de procesamiento distribuido PVM. La codificación del algoritmo se realizó en el **Lenguage de Programación C**, con las rutinas de biblioteca de PVM embebidas en el código. Se utilizó como referencia para la implemantación en PVM del AEPD1 el pseudocódigo presentado en *2.5.3.2*.

Al iniciar la aplicación AEPD1 en PVM, lo primero que se hace es correr el proceso denominado "*Monitor*" en alguna de las máquinas que conforman la Máquina Virtual. Luego este proceso lanza en paralelo el primer proceso "*Steiner_Slave*" el cual será raíz de un árbol de procesos de este tipo que correrán en forma paralela y distribuida en las diferentes máquinas que integran la Máquina Virtual, analizando en forma paralela diferentes subespacios del espacio de soluciones factibles Γ_{STECSP} . El pasaje de parámetros entre éstos procesos, se realiza mediante directivas de PVM

exclusivas para este fin. Un proceso "*Steiner_Slave*" se ejecuta localmente en una única máquina de la Máquina Virtual, e intercambia parámetros con otros procesos "*Steiner_Slave*" y el proceso "*Monitor*" usando primitivas PVM de envío de mensajes.

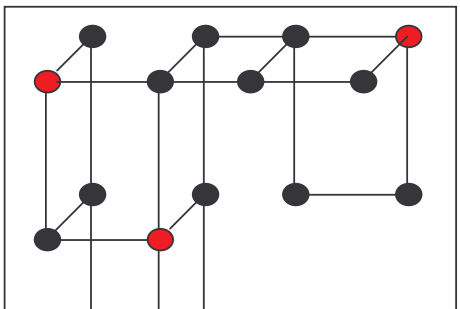
Para realizar las corridas del AEPD1 en PVM, las arquitecturas utilizadas para integrar la Máquina Virtual fueron las siguientes: 7 equipos Sun SPARCStation 5 con 64 MB de memoria RAM y 2 quipos RS/6000 SP con 128 MB de memoria RAM.

3.6.4 2.6.4 Casos de Prueba

A efectos de medir la performance de la implementación del AEPD1 en PVM, se seleccionaron tres instancias del STECSP. En la primera de ellas el grafo asociado es de dimensión "pequeña", mientras que en la otras dos, los grafos asociados a los problemas tienen un alto grado de complejidad, con una gran cantidad de nodos de nodos del tipo Steiner y nodos terminales, además de una gran densidad de aristas. Seguidamente se brinda la representación gráfica de las topologías asociadas a los casos de prueba. Los nodos negros representan los nodos del tipo Steiner, mientras que los nodos rojos representan los nodos terminales. En los tres problemas, los costos de los enlaces punto a punto entre nodos del tipo Steiner tienen costo 1, un enlace entre un nodo de Steiner y un terminal tiene costo 2 y un enlace entre dos nodos terminales tiene costo 4.

3.6.4.1 Problema 1

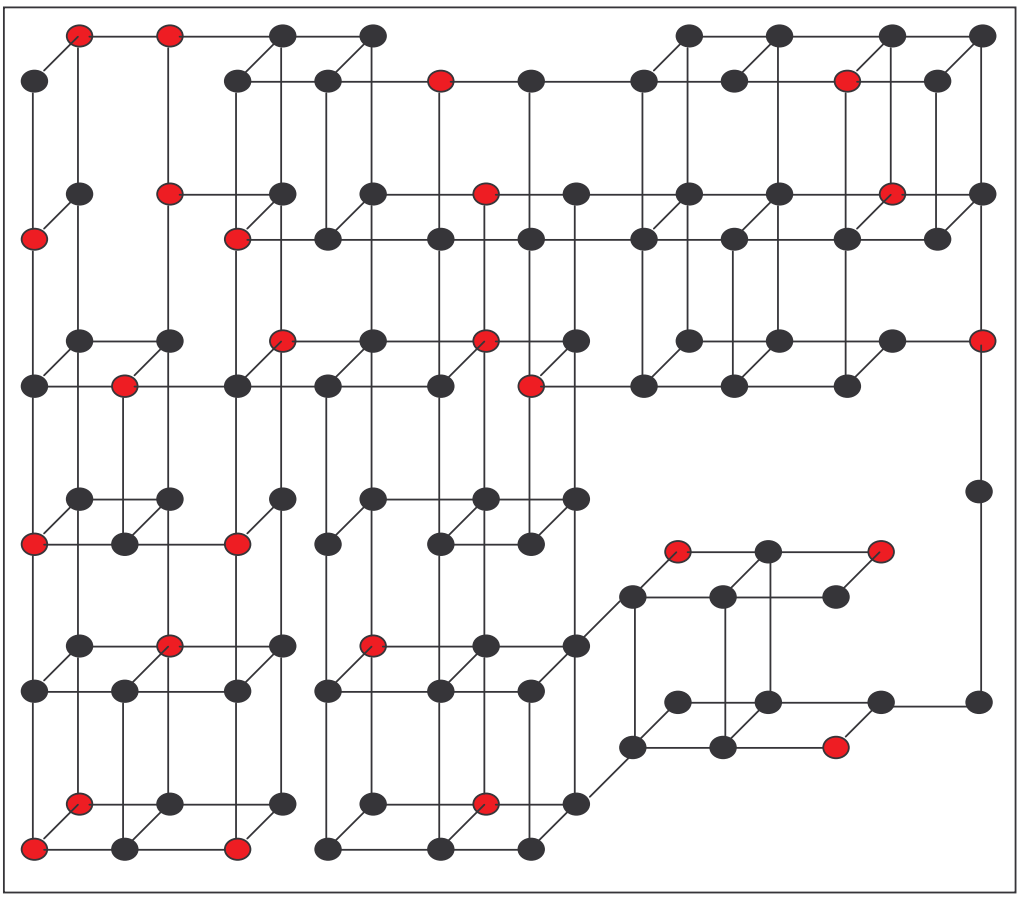
Esta topología de grafo tiene 4 nodos terminales, 14 nodos del tipo Steiner y 25 aristas.



[Grafo 1]

3.6.4.2 Problema 2

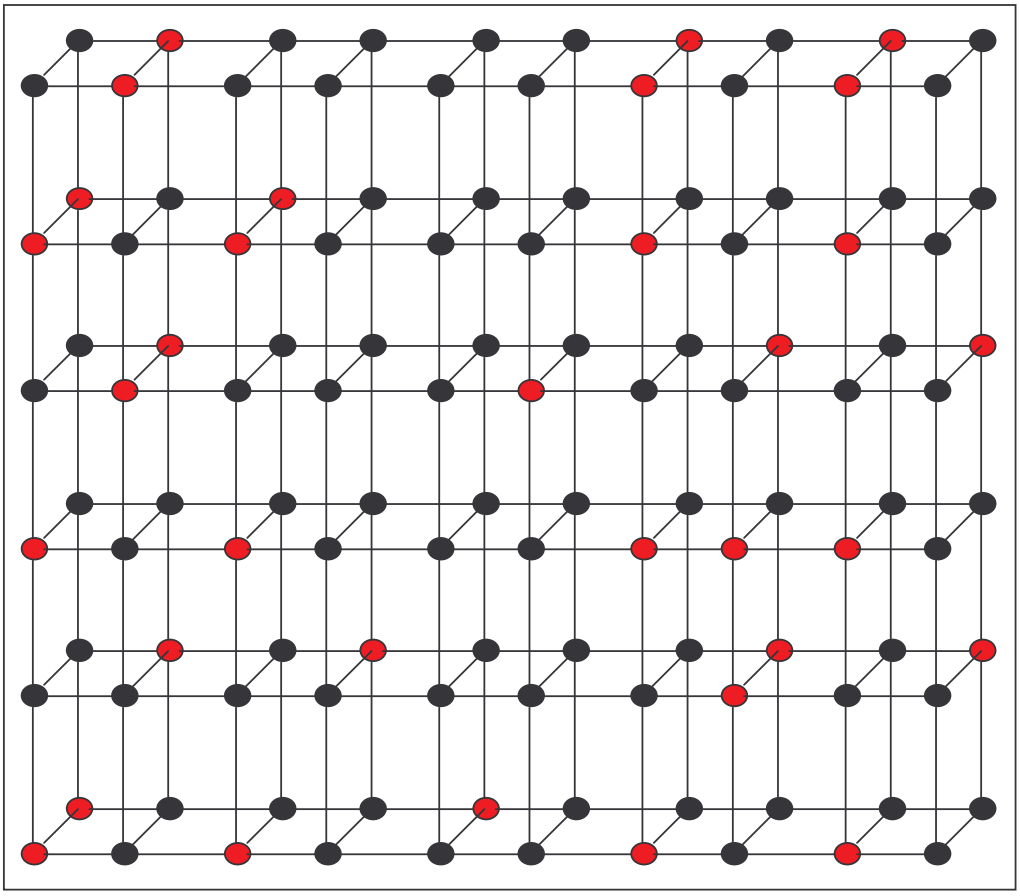
Esta topología de grafo tiene 25 nodos terminales, 79 nodos de Steiner y 183 aristas.



[Grafo 2]

3.6.4.3 Problema 3

El grafo tiene una estructura de grilla doble, con 33 nodos terminales, 87 nodos del tipo Steiner y 268 aristas.



[Grafo 3 - Grilla Doble]

3.6.5 2.6.5 Resultados Obtenidos

Las siguientes tablas muestran los tiempos requeridos para el cálculo de las soluciones óptimas en cada uno de los problemas, dependiendo del número de máquinas que integraron la Máquina Virtual.

MAQUINA VIRTUAL	TIEMPO DE EJECUCIÓN (SEG)
1 - SPARCStation 5	135
2 – SPARCStation 5	133
5 – SPARCStation 5	138
5 – SPARCStation 5 2 – RS/6000 SP	134
7 – SPARCStation 5 2 – RS/6000 SP	141

[Resultados Numéricos – Problema 1]

MAQUINA VIRTUAL	TIEMPO DE EJECUCIÓN (SEG)
1 - SPARCStation 5	1278
2 – SPARCStation 5	1152
5 – SPARCStation 5	913
5 – SPARCStation 5 2 – RS/6000 SP	497
7 – SPARCStation 5 2 – RS/6000 SP	785

[Resultados Numéricos – Problema 2]

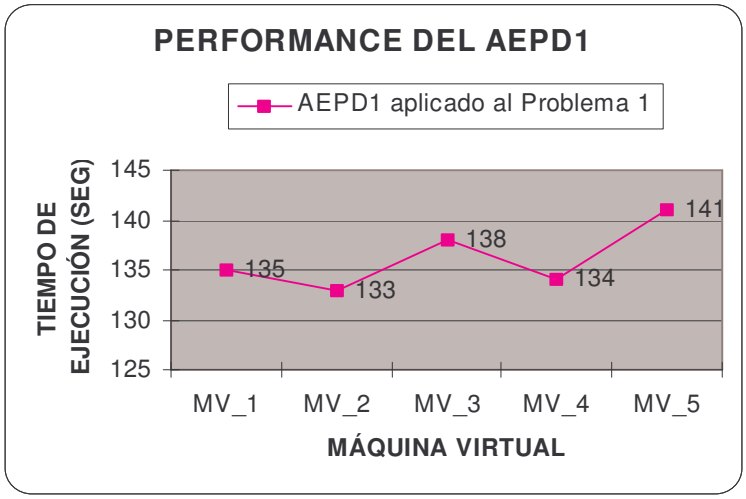
MAQUINA VIRTUAL	TIEMPO DE EJECUCIÓN (SEG)
1 - SPARCStation 5	1534
2 – SPARCStation 5	1396
5 – SPARCStation 5	1014
5 – SPARCStation 5 2 – RS/6000 SP	571
7 – SPARCStation 5 2 – RS/6000 SP	863

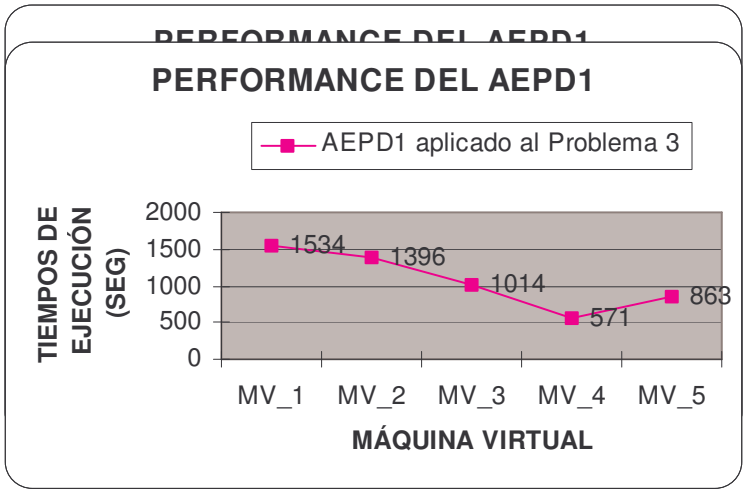
[Resultados Numéricos – Problema 3]

A continuación se brindan gráficas mostrando la evolución de los tiempos de ejecución requeridos para la obtención de la solución óptima del espacio Γ_{STECSP}, en función de los distintos tipos de arquitecturas que componen las

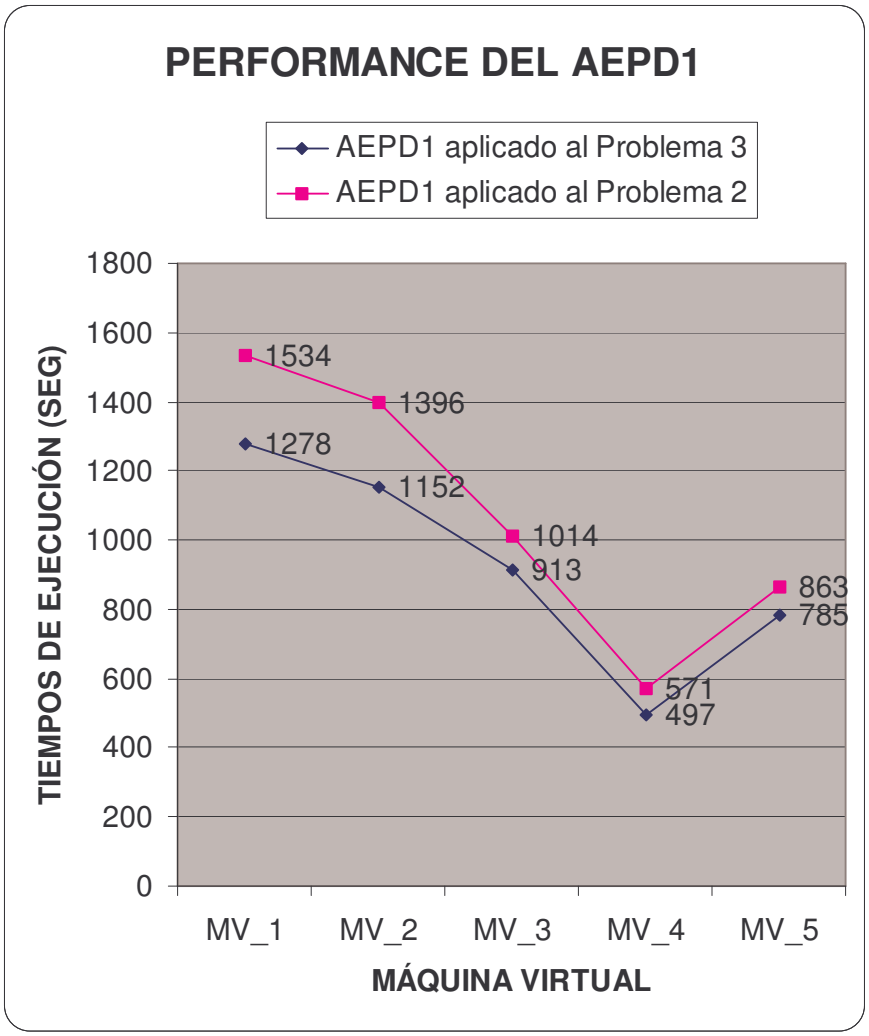
diferentes Máquinas Virtuales definidas para realizar los tests de performance. La nomenclatura utilizada para hacer referencia a las diferentes Máquinas Virtuales, viene dada por la siguiente tabla.

MÁQUINA VIRTUAL	NOTACIÓN
1 - SPARCStation 5	MV_1
2 – SPARCStation 5	MV_2
5 – SPARCStation 5	MV_3
5 – SPARCStation 5 2 – RS/6000 SP	MV_4
7 – SPARCStation 5 2 – RS/6000 SP	MV_5

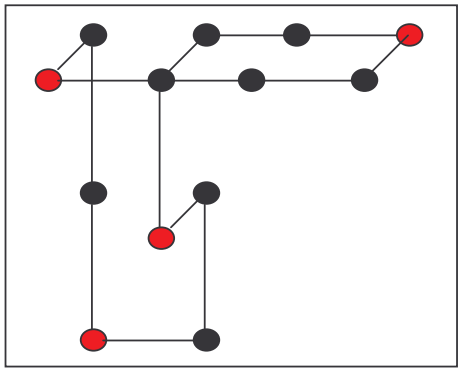




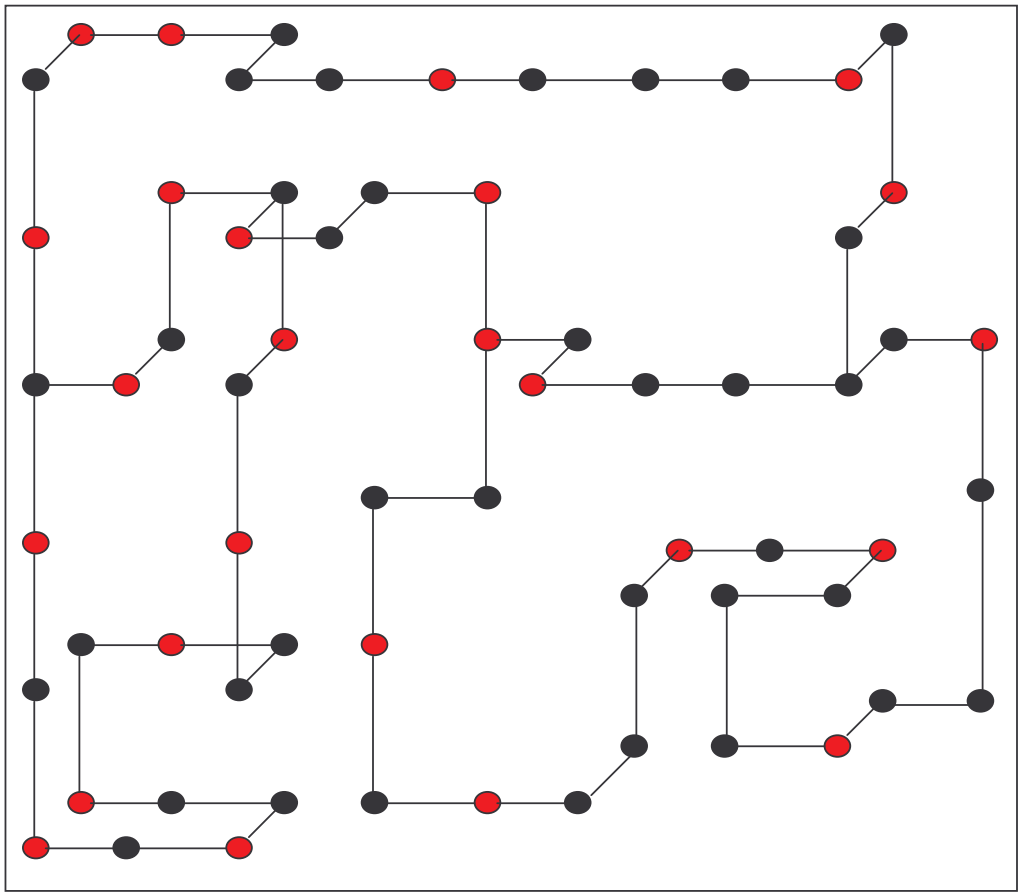
El siguiente gráfico muestra en forma mixta la evolución de los tiempos requeridos para hallar el óptimo en los problemas 2 y 3.



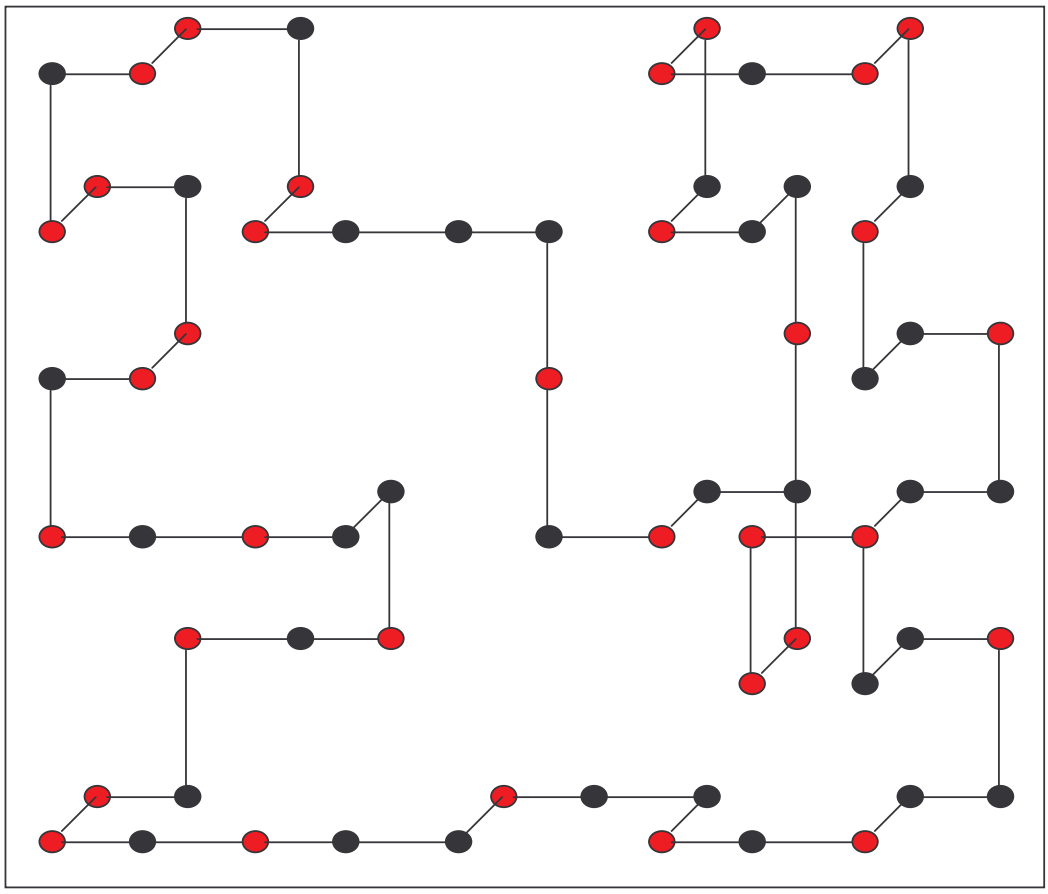
Tres de las soluciones óptimas encontradas por el AEPD1 para los problemas seleccionados vienen dadas por los siguientes gráficos.



[Solución Óptima del Problema 1]



[Solución Óptima del Problema 2]



3.6.6 2.6.6 Conclusiones

Del análisis de los resultados obtenidos de la aplicación del AEPD1 a instancias del STECSP tomadas como casos de prueba de performance, se extraen las siguientes conclusiones.

En el caso de una instancia del STECSP donde la topología del grafo asociado sea relativamente pequeña, el aumento del número de hosts que integran la Máquina Virtual no produce mejoras significativas en los tiempos de ejecución del AEPD1. Además en ciertos casos, éstos pueden crecer al aumentar el número de procesadores corriendo en paralelo.

En las instancias del STECSP de mediano y gran porte, al trabajar con una Máquina Virtual con mas de un host es posible comprobar que los tiempos de ejecución disminuyen proporcionalmente al número de procesadores disponibles.

Para problemas de mediano y gran porte, es conveniente aplicar las técnicas de paralelismo teniendo en cuenta que existe un óptimo en la cantidad de procesadores a agregar en la Máquina Virtual, luego del cual el sistema degrada su performance, o consume recursos con un costo prohibitivo. Es notoria la mejora de performance en cuanto al tiempo de ejecución, al aumentar el número de procesos *Steiner_Slave* que corren en paralelo explorando el espacio Γ_{STECSP} . El aumento de recursos para resolver un problema en forma paralela y distribuida, debe estar motivado por la resolución de instancias de problemas de gran tamaño, en lugar de mejorar los tiempos de resolución de problemas pequeños

El modelo paralelo-distribuido aplicado al STECSP ha salido bien parado en su choque con la práctica. Los resultados numéricos obtenidos avalan esta afirmación. Las técnicas de paralelismo correctamente aplicadas, surgen como una alternativa viable para la resolución de problemas del tipo NP-Difíciles donde es posible diseñar algoritmos capaces de explorar el espacio de soluciones factibles en forma paralela y distribuida; tal es el caso del AEPD1 utilizado en la resolución exacta del STECSP.

4 3. Algoritmo Aproximado para el GSP

4.1 3.1 Introducción

El objetivo de este capítulo es presentar un algoritmo para la resolución en forma aproximada del problema GSP. Dado que hasta el momento se conoce una única heurística capaz de resolver el GSP para las clases más generales de grafos y debido a las múltiples aplicaciones del modelo GSP en problemas reales de telecomunicaciones, electrónica, etc; es de gran importancia el diseño de un nuevo algoritmo aproximado alternativo basado en alguna de la metaheurísticas mas conocidas para la resolución de problemas NP-difíciles. Primeramente se brinda la elección y una breve descripción de la metodología sobre la cual esta basado el algoritmo. Luego se da una descripción detallada del algoritmo, un análisis de complejidad en el peor caso, detalles de implementación, fase de selección de parámetros del algoritmo, pruebas de validación, pruebas experimentales, resultados obtenidos y conclusiones.

4.2 3.2 Elección de la Metodología de Base

Una de las etapas previas al diseño de un algoritmo para resolver en forma aproximada el problema GSP, es la selección de una metodología de base. Algunas de las metodologías más conocidas para la resolución de problemas NP-difíciles son: *Algoritmos Genéticos*, *Simulated Annealing*, *Tabu Search*, *Redes Neuronales*, *Greedy Randomized Adaptive Search Procedure (GRASP)*, y *Ant System*. Se realizo una selección de las posibles metodologías a utilizar de acuerdo a los siguientes criterios.

4.2.1.1 Criterios de Selección

1. fácil adaptación de la metodología al problema GSP,
2. la fase de construcción de una solución del GSP no debe ser compleja,
3. las soluciones factibles construidas en cada paso deben ser "buenas" (los costos deben ser próximos al costo de la solución óptima local),

4. si la metodología presenta una fase de búsqueda local exhaustiva, debe estar claro como se define la estructura de las soluciones vecinas,
5. balance entre la complejidad de la búsqueda local y la velocidad de convergencia hacia el óptimo local,
6. balance global entre la complejidad de implementación y la eficiencia del algoritmo,
7. fácil adaptación a una eventual implementación paralela del algoritmo.

4.2.1.2 Análisis de Metodologías

4.2.1.2.1.1.1.1.1 Algoritmos Genéticos*:*

*4.2.1.2.1.1.1.1.2 No cumple el criterio 1. Si bien se puede modelar los grafos como cadenas de genes, la función de "*fitness*" (función objetivo de los algoritmos genéticos) asociada al GSP como la suma de los costos del grafo y las restricciones de factibilidad (requerimientos de conexión dados por la matriz R) como forma de depuración o selección de individuos, al representar las soluciones factibles del GSP mediante cadenas de genes (ceros y unos), no queda claro el significado del cruzamiento entre individuos de la población [16].*

Simulated Annealing, Tabu Search, Redes Neuronales:

No se cumple el criterio 1. La forma de operar de estas metodologías hace que sea extremadamente complejo adaptarlas en búsqueda de obtener "buenas" soluciones factibles a un problema de optimización del tipo GSP donde se pretende encontrar una estructura de red topológicamente óptima [16].

Greedy Randomized Adaptive Search Procedure:

GRASP cumple con los criterios 1,2 y 3; sin embargo no queda muy claro como definir el concepto de "vecindad" sobre una solución factible del GSP, con lo cual no es posible satisfacer el criterio 4. Resulta difícil definir la estructura de las soluciones vecinas a una solución factible del GSP y debido a que GRASP requiere de una búsqueda local exhaustiva en búsqueda de óptimos locales, la adaptación de la metodología GRASP al problema GSP sería incompleta.

Ant System:

La metaheurística *Ant System* cumple con los criterios 1, 2 y 3. Además, debido a que los algoritmos *Ant System* centralizan el esfuerzo en la construcción de soluciones factible de bajo costo y no se tiene una fase de búsqueda local exhaustiva, los criterios 4 y 5 también se satisfacen [16,51]. La estructura básica de los algoritmos *Ant System* aplicados a problemas de optimización combinatoria, en general es la misma a menos de algunos cambios como los requeridos en la evaluación de la factibilidad de la solución construida, los tests de factibilidad dependen del problema en cuestión. El cumplimiento de los criterios 5 y 6 dependerá de la clase de *Ant System* (existen 3 clases de *Ant System*) a utilizar como base del algoritmo aproximado y de la forma en que éste es implementado [15].

Al analizar la adaptación de las diferentes metodologías al problema GSP, utilizando los criterios anteriormente expuestos, surgió elegida como metodología de base para el diseño e implementación de un algoritmo aproximado para el GSP la meta-heurística *Ant System*.

4.2.1.3 Ant System

La metáfora sobre la cual se basa los *Ant Systems* surge de la observación de la especie de hormigas conocida como *Linepithema Humile*. Supóngase la siguiente situación: un nido de hormigas de la especie *Linephitema Humile* y cerca del mismo existe una fuente de alimentos accesible a las hormigas por medio de un puente compuesto de dos ramificaciones del mismo largo y ambas conducen a la fuente de alimentos. Aunque las hormigas son totalmente libres de elegir cualquiera de las ramificaciones que conducen a la fuente de alimentos, una vez que éstas comienzan a salir del hormiguero en busca de alimentos, rápidamente se puede observar que la mayoría de las hormigas elige la misma ramificación para llegar a la fuente de alimentos, aunque en principio no habría una razón de porque el elegir una ramificación u otra. Este comportamiento viene dado por el hecho de que las hormigas mientras caminan depositan una sustancia química conocida con el nombre de feromona; ellas son capaces de detectar

esta sustancia mediante sus antenas y la utilizan como información para determinar la dirección de avance.

El comportamiento de estas hormigas puede ser modelado mediante un proceso probabilístico: en ausencia de feromona la hormiga explora el área que la rodea eligiendo una dirección totalmente randómica. Si existe un rastro de feromona presente, entonces la hormiga elige la dirección del rastro con alta probabilidad. Si existen dos o más rastros diferentes de feromona, la hormiga elige con mayor probabilidad el rastro más “fuerte” de feromona. Al recorrer el rastro elegido, la hormiga deposita unidades adicionales de feromona en el trayecto, de esta manera, el rastro se hace aún más “fuerte” en esa dirección. El proceso es un proceso de retroalimentación positiva, pues la feromona depositada por la hormiga refuerza en cada paso el rastro de la dirección elegida. Otra característica del fenómeno, es que la feromona depositada por la hormiga se evapora, con lo cual un rastro que no es utilizado gradualmente suele desaparecer.

Al analizar el comportamiento de esta especie de hormigas, en la situación en la cual en vez de tener dos posibles ramificaciones de igual longitud, éstas tienen diferentes longitudes; vale el mismo modelo probabilístico, y además se puede observar, que las hormigas rápidamente eligen la ramificación más corta que las conduce a la fuente de alimentos. Esto se explica de la siguiente manera: al comienzo, cuando no hay rastros de feromona presentes, las hormigas eligen ir a la izquierda o derecha con la misma probabilidad. Una vez que las hormigas que eligieron la ramificación más corta arriban a la fuente de alimentos y retornan prontamente al nido, la cantidad de feromona en la ramificación más corta crece rápidamente. De esta manera las hormigas son capaces de encontrar el camino más corto entre el nido y la fuente de alimentos, aunque cada hormiga tiene una visión miope del área circundante.

Inspirados en este proceso de optimización natural, Colorini, Dorigo y Maniezzo [15] diseñaron una nueva meta-heurística que puede ser aplicada a problemas de optimización combinatoria. La idea básica está en considerar procesos de hormigas artificiales, los cuales construyen en forma repetida soluciones del problema de optimización. Cada proceso construye una solución con la ayuda de una memoria común que puede ser leída y

modificada por cualquier hormiga artificial. En esta memoria común se almacenan los "rastros" de cada una de las hormigas artificiales.

Colorini-Dorigo-Maniezzo, definen tres clases de algoritmos para los *Ant Systems*; los mismos se denominan *Ant-Density*, *Ant-Quantity* y *Ant-Cycle*. Una descripción detallada de estas clases de algoritmos se encuentra en [15,16,21].

4.3 3.3 Algoritmo Aproximado Basado en Ant System para el GSP

4.3.1 3.3.1 Introducción

Al adaptar la metaheurística *Ant System* al diseño de un algoritmo aproximado para el GSP, se tomó como base las clases *Ant-Density* y *Ant-Quantity* [15]. El algoritmo diseñado puede instanciarse con cualquiera de estas dos clases; su diseño e implementación se detallan a continuación.

4.3.2 3.3.2 Algoritmo Aproximado

Los siguientes son las descripciones y pseudocódigos del algoritmo "*GSP_Ant_System*" y sus procedimientos y funciones auxiliares.

Algoritmo "*GSP_Ant_System*"

El algoritmo recibe como entradas:

- un grafo simple $G = (V, E)$,
- el conjunto de nodos terminales $T \subseteq V$,
- matriz de costos asociadas a las aristas c,
- matriz de requerimientos de conexión entre nodos terminales R.

En base a estos parámetros, el algoritmo intenta encontrar una "buena" solución factible dentro del espacio Γ_{GSP}. La idea de su funcionamiento es la siguiente. Mientras no se cumpla un cierto número máximo de iteraciones (definido de antemano) y no se haya encontrado una cierta cantidad de

soluciones factibles (también definida de antemano), el algoritmo realiza lo siguiente. Se inicializa una serie de parámetros, entre ellos la solución actual con $N = T$, $A = \varnothing$ y $\cos to\_sol = INF$. Luego, $\forall i, j \in T$ se trata de encontrar una cantidad r_{ij} de caminos de aristas disjuntas que unan ambos nodos terminales, construyéndose de esta manera una solución factible para la instancia del GSP. La determinación de cada uno de estos caminos, se hace utilizando un algoritmo llamado "*Busco_Camino*", el cual tomando como base el algoritmo de la clase *Ant-Density* (o *Ant-Quantity*) para *Ant System*, determina caminos entre pares de nodos terminales sobre un subgrafo de G , y selecciona uno de ellos de acuerdo a un cierto criterio. A grandes rasgos, el funcionamiento del algoritmo "*Busco_Camino*" es el siguiente. Sea $G_s(V, E \setminus A)$ el subgrafo de G sobre el cual se desea determinar un camino que una los nodos terminales i y j ; para ello se ubica un cierto número de hormigas en el nodo terminal i y en el nodo j (la cantidad de hormigas a ubicar en cada nodo depende de r_{ij} y del grado del nodo en el subgrafo en cuestión), luego las hormigas se "mueven" sobre el subgrafo encontrándose una cierta cantidad de caminos que unen ambos nodos terminales. Sobre el conjunto de caminos que unen i con j encontrados por las hormigas, se selecciona aquél que satisface el denominado *"Criterio de Selección de Camino*" (dicho criterio se define luego formalmente). Una vez seleccionado un camino $\overline{w}$ por el algoritmo "*Busco_Camino*", se actualiza la solución actual mediante las asignaciones $N = N \cup Nodos(\overline{w})$, $A = A \cup Aristas(\overline{w})$ y $\cos to\_sol = \cos to\_sol + COSTO(\overline{w})$. Cada vez que se agrega un nuevo camino a la solución actual, se controla que el costo de ésta sea menor que el de la mejor solución factible encontrada hasta el momento, de no ser así, el algoritmo "*GSP_Ant_System*" intenta construir una nueva solución factible desde el comienzo. De igual manera, si para algún par de nodos terminales $i, j \in T$ el algoritmo "*GSP_Ant_System*" no puede determinar mediante múltiples aplicaciones del algoritmo "*Busco_Camino*" la cantidad r_{ij} de caminos de aristas disjuntas requerido, entonces el algoritmo resetea los parámetros e intenta construir una nueva solución factible.

Si $\forall i, j \in T$ se logran encontrar r_{ij} caminos de aristas disjuntas, construyendo una solución $G_s(N, A)$ con costo menor que el de la mejor solución factible encontrada hasta el momento, entonces, el algoritmo "*GSP_Ant_System*" actualiza la mejor solución mediante las asignaciones $N_{Sol} = N$, $A_{Sol} = A$ y $min\_\cos to = COSTO(A)$.

El algoritmo "*GSP_Ant_System*" retorna la mejor solución factible encontrada luego de un cierto número de iteraciones. En caso de no haberse hallado ninguna solución factible se indica mediante un parámetro booleano que la búsqueda no fue exitosa.

Seguidamente se brinda la definición del "*Criterio de Selección de Camino*" utilizado por el procedimiento "*Busco_Camino*".

4.3.2.1.1.1.1.1.1 Criterio de Selección de Camino

Sea $W^{i-j} = \{w^{i-j}\}$ el conjunto de caminos que unen el nodo terminal i con en el nodo terminal j, determinado por un conjunto de hormigas en una instancia de ejecución del algoritmo "*Busco_Camino*". El camino seleccionado, es aquel $\overline{w} \in W^{i-j}$ que satisface $\frac{COSTO(\overline{w})}{NUM\_TERM(\overline{w})+2} = Min\left\{\frac{COSTO(w^{i-j})}{NUM\_TERM(w^{i-j})+2}, w^{i-j} \in W^{i-j}\right\}$.

Donde $COSTO(w)$ es el costo del camino w y $NUM\_TERM(w)$ es el número de terminales distintos de i y j presentes en w y para los cuales todavía existen requerimientos de conexión no satisfechos en la solución actual. La idea es seleccionar el camino que une los nodos terminales i y j que halla recorrido la mayor cantidad de nodos terminales con el menor costo posible; los nodos terminales tomados en cuenta son aquellos para los cuales su requerimiento de conexión con algún otro nodo terminal todavía no ha sido satisfecho por la solución actual (en fase de construcción).

Aplicando este criterio, sean $w_1^{i-j}, w_2^{i-j} \in W^{i-j}$, entonces se cumple:

- si $COSTO(w_1^{i-j}) = COSTO(w_2^{i-j})$ y $NUM\_TERM(w_1^{i-j}) > NUM\_TERM(w_2^{i-j})$, entonces es preferido w_1^{i-j} antes que w_2^{i-j} (a igual costo se prefiere el camino que haya recorrido mayor número de terminales),
- si $COSTO(w_1^{i-j}) < COSTO(w_2^{i-j})$ y $NUM\_TERM(w_1^{i-j}) = NUM\_TERM(w_2^{i-j})$, entonces es preferido w_1^{i-j} antes que w_2^{i-j} (a igual número de terminales recorridos, se prefiere el camino de menor costo).

El siguiente es el pseudocódigo detallado del algoritmo "*GSP_Ant_System*".

```
Algoritmo GSP_Ant_System(in V:Nodos, E:Aristas, T:Nodos, c:Costos ,
                         R:Matriz_Req_Con; out N_Sol:Nodos, A_Sol:Aristas, exito:boolean);
cant_iter=0;
cant_sol_fact=0;
min_costo=INF;
Leer_Ant_System_Param(param);
while ((cant_iter≤MAX_ITER) and (cant_sol_fact<MAX_SOL_FACT)) do
   Inicializo_1(T,R,N,A,M,B,costo_sol);
  OK=TRUE;
   for each i, j ∈ T / i ≠ j do
      while (M_ij>0) and (OK=TRUE) do
         Grafo_Cociente(E,A,A_c);
         Busco_Camino(V,A_c,R,c,param,i,j,p,fallo);
         if (fallo=FALSE) then
            Actualizar_Solucion(c,p,N,A,costo_sol);
            if (costo_sol<min_costo) then
               Actualizar_Matrices(p,T,R,M,B);
            else OK=FALSE;
            end_if;
         else OK=FALSE;
         end_if;
      end_while;
      if (OK=FALSE) then
         exit_for_each();
      end_if;
   end_for_each;
   if (OK=TRUE) then
      Actualizar_Mejor_Solucion(N,A,costo_sol,N_Sol,A_Sol,min_costo);
      cant_sol_fact=cant_sol_fact+1;
      cant_iter=0;
   else cant_iter=cant_iter+1;
   end_if;
end_while;
if (cant_sol_fact>0) then
   exito=TRUE;
else
   exito=FALSE;
end_if;
End_Algoritmo;
```

Procedimiento "*Inicializo_1*"

Recibe como entradas el conjunto de nodos terminales T y la matriz R de requerimientos de conexión entre nodos terminales. Devuelve el subgrafo inicial $G_s(T,\varnothing)$, el costo de la solución actual inicializada en cero, la matriz auxiliar $M=\{M_{ij}\}_{n_T\times n_T}$ donde se almacena los requerimientos de conexión que faltan por satisfacer en la solución, y la matriz auxiliar booleana $B=\{B_{ij}\}_{n_T\times n_T}$ que indica que pares de nodos terminales de T están conectados en la solución actual. El pseudocódigo del procedimiento es el siguiente.

```
Procedure Inicializo_1(in T:Nodos, R:Matriz_Req_Con; out N:Nodos, A:Aristas,
                                    M:Matriz_Req_Con, B:Matriz_Bool, costo_sol:real);
  N=T;
  A=∅;
  M=R;
  B_ij=FALSE; ∀i, j ∈ T
  costo_sol=0;
End;
```

Procedimiento "*Grafo_Cociente*"

Recibe como parámetros el conjunto de aristas E del grafo original y un conjunto de aristas $A\subseteq E$. Devuelve el conjunto de aristas cociente $E\setminus A$. El pseudocódigo es el siguiente.

```
Procedure Grafo_Cociente(in E:Aristas, A:Aristas; out A_c:Aristas);
   A_c=E\A;
End:
```

Procedimiento "*Leer_Ant_System_Param*"

Lee el conjunto de parámetros *Ant System* que serán utilizados por el procedimiento "*Busco_Camino*" y los almacena en un vector denominado *param*. Los mismos son: parámetros alfa y beta usados en el cálculo de las probabilidades de transición entre nodos, coeficiente de evaporación de la feromona, y el valor de la unidad de feromona dejada por las hormigas.

Procedimiento "*Busco_Camino*"

Se basa en el algoritmo de la clase *Ant-Density* (o *Ant-Quantity*). Recibe como entradas el subgrafo $G_s(V, A_c)$ (donde $A_c = E \setminus A$), la matriz R de requerimientos de conexión entre nodos terminales, la matriz c de costos asociados a las aristas, los parámetros *Ant System*, y los nodos terminales $i, j \in T$. Retorna (si la búsqueda es exitosa) un camino p que une los nodos terminales i y j en $G_s(V, A_c)$, además de una variable booleana que indica si la búsqueda fue exitosa. El pseudocódigo del procedimiento es el siguiente.

```
Procedure Busco_Camino(in V:Nodos, Ac:Aristas, R:Matriz_Req_Con, c:Costos,
          param:Ant_System_Param,  i:Nodo, j:Nodo;  out popt:Camino, fallo:boolean);

  alfa=param[1];
  beta=param[2];
  coef_evap=param[3];
  feromona=param[4];
  cant_cam=0;
  cant_iter=0;
  min_costo_cam=INF;
  Inic_Prob_Trans(V,Ac,MPini);
  Inicializo_MP(MPini,MP);
  Distribuir_Ants(V,Ac,rij,ants_i,ants_j,cant_ants);
  Inicializo_Rastros(V,Ac,dW,W);
  while (cant_iter≤MAX_INTENTOS) and (cant_cam<MAX_CAM) do
     Inicializo_Ants(i,j,ants_i,ants_j,b);
     Inicializo_Tabu_List(i,j,ants_i,ants_j,L);
     Inicializo_Delta_Rastros(V,Ac,dW);
     for  t=1 to n do
        for k=1 to cant_ants do
            u=b[k];
            v=Proximo_Nodo(V,Ac,MP,u);
            b_aux[k]=v;
```

Procedimiento "*Inic_Prob_Trans*"

Recibe como parámetro el subgrafo $G_s(V, A_c)$, y devuelve la matriz estocástica MP_{ini} de dimensión $n \times n$, con las probabilidades iniciales de transición. La matriz se calcula de la siguiente manera:

$$MP_{ini}(i,j) = \begin{cases} 0, si\,(i,j) \notin E \\ \frac{1}{g(i)}, si\,(i,j) \in E \end{cases}.$$

Procedimiento "*Distribuir_Ants*"

Recibe como parámetros el subgrafo $G_s(V, A_c)$ y el valor r_{ij} que indica el nivel de conexión que deben tener los nodos terminales i y j en la solución. En función del grado de éstos nodos en el subgrafo y del valor r_{ij} se determina la cantidad de hormigas a ubicar en los terminales i y j respectivamente. La forma de calcular estas cantidades viene dada por:

$$ants\_i = \left\lfloor \frac{g_{G_s}(i)}{r_{ij}} \right\rfloor \times ANT\_FACTOR \qquad \text{y} \qquad ants\_j = \left\lfloor \frac{g_{G_s}(j)}{r_{ij}} \right\rfloor \times ANT\_FACTOR$$

donde $g_{G_s}(i)$ y $g_{G_s}(j)$ son los grados de los nodos i y j en el subgrafo $G_s(V, A_c)$, y ANT_FACTOR es una constante elegida previo a la corrida del algoritmo. La cantidad total de hormigas viene dada por $cant\_ants = ants\_i + ants\_j$.

Procedimiento "*Inicializo_Ants*"

Recibe como parámetros los nodos $i, j \in T$ y las cantidades $ants\_i$ y $ants\_j$ de hormigas a ubicar en los nodos i y j respectivamente. Se devuelve un vector b de dimensión $ants\_i + ants\_j$ con la ubicación inicial de las hormigas en el subgrafo actual. El pseudocódigo del procedimiento es el siguiente.

```
Procedure Inicializo_Ants(in i:Nodo, j:Nodo, ants_i:Int, ants_j:Int; out b:Pos_Ants);
   for k=1 to ants_i do      /* Ubico ants_i hormigas en el nodo i */
      b[k]=i;
   end_for;
   for k=1 to ants_j do    /* Ubico ants_j hormigas en el nodo j */
      b[ants_i+k]=j;
   end_for;
End;
```

Procedimiento "*Inicializo_Tabu_List*"

Recibe como parámetros los nodos $i, j \in T$ y las cantidades $ants\_i$ y $ants\_j$ de hormigas ubicadas en los nodos i y j respectivamente. Se inicializa la estructura tabu L donde se almacena el recorrido hecho por cada una de las hormigas. El pseudocódigo es el siguiente.

```
Procedure Inicializo_Tabu_List(in i:Nodo, j:Nodo, ants_i:int, ants_j:int; out L:Tabu_List);
   VacioEstructuraTabu(L);          /* inicialmente no tengo caminos */
   for k=1 to ants_i do
      Agrego_Nodo(L[k],i);          /* i es el primer nodo del camino de la hormiga k */
   end_for;
   for k=ants_i+1 to ants_j do
      Agrego_Nodo(L[k],j);          /* j es el primer nodo del camino de la hormiga k */
   end_for;
End;
```

Procedimiento "*Inicializo_Rastros*"

Recibe como parámetros el subgrafo $G_s(V, A_c)$ y devuelve la asignación inicial de rastros de feromona en el subgrafo en cuestión. Inicialmente estos rastros son muy "tenues", incrementándose luego con el desplazamiento de las hormigas sobre el subgrafo.

Sea W la matriz donde se almacena la intensidad de rastros de feromona sobre el subgrafo $G_s(V, A_c)$ y dW la matriz delta rastro donde se almacena el incremento de feromona sobre el subgrafo $G_s(V, A_c)$ luego de una unidad de tiempo. El procedimiento "*Inicializo_Rastros*" inicializa estas matrices de la siguiente forma:

$$W_{ij} = \begin{cases} MIN\_RASTRO, si\,(i,j) \in A_c \\ 0, sino \end{cases}, \; dW_{ij} = 0, \forall (i,j) \in A_c \; .$$

Procedimiento "*Inicializo_Delta_Rastros*"

Recibe como parámetros el subgrafo $G_s(V, A_c)$ y devuelve dW la matriz delta rastro inicializada en cero; $dW_{ij} = 0, \forall (i,j) \in A_c$.

Procedimiento "*Inicializo_MP*"

Tiene como parámetro de entrada la matriz estocástica de transición inicial MP_{ini} . Se realiza una copia de esta matriz sobre la matriz MP .

Función "*Proximo_Nodo*"

Recibe como parámetros el subgrafo $G_s(V, A_c)$, un nodo $u \in V$ y la matriz estocástica de transición MP. Se realiza un sortéo en base a la matriz MP, determinándose hacia que nodo adyacente al nodo u en $G_s(V, A_c)$, se debe de realizar la transición de la hormiga.

4.3.2.1.1.1.2 Procedimiento "*Agrego_Nodo*"

Recibe como parámetros una lista L_k de la tabu list (donde se almacena el camino recorrido hasta el momento por una cierta hormiga) y un nodo $v \in V$ a ser agregado a la lista L_k. Se retorna la lista L_k actualizada.

Procedimiento "*Calculo_Rastro_Ant*"

Recibe como parámetros la matriz c de costos asociados a las aristas, dos nodos $u, v \in V$ y la unidad de feromona depositada por cada hormiga. Se calcula la intensidad del rastro por unidad de largo dejada por la hormiga al moverse del nodo u al nodo v. El pseudocódigo es el siguiente.

Procedure *Calculo_Rastro_Ant(in c:Costos, u:Nodo, v:Nodo, feromona:real;*
out rastro_ant:real);
/ Dependiendo del tipo de algoritmo Ant-System utilizado, se calcula el rastro dejado por la hormiga de la siguiente forma */*
Utilizando el modelo Ant-quantity: $rastro\_ant = (\frac{feromona}{c_{ij}})$
Utilizando el modelo Ant-density: $rastro\_ant = feromona$
End;

Procedimiento "*Actualizo_Ants_Posicion*"

Recibe como parámetro un vector b_{aux} donde están almacenadas las nuevas posiciones donde se encuentran cada una de las hormigas (los nodos a donde se han movido las hormigas luego de una unidad de tiempo). Este vector es copiado al vector de posiciones b, el cual es retornado por el procedimiento.

Procedimiento "*Actualizo Delta Rastros*"

Recibe como parámetros la matriz delta rastro dW, dos nodos $u, v \in V$ y el rastro de feromona dejado por una hormiga en la arista $(u,v) \in E$. Se actualiza la matriz delta rastro dW mediante: $dW_{uv} = dW_{uv} + rastro\_ant$.

Procedimiento "*Actualizo Rastros*"

Recibe como parámetros la matriz delta rastro dW, la matriz de rastros W, y el coeficiente de evaporación de feromona. Se devuelve la matriz de rastros W actualizada; la actualización se realiza del siguiente modo: $W_{ij} = (coef\_evap \times W_{ij}) + dW_{ij}$, $\forall (i,j) \in A_c$.

Procedimiento "*Actualizo Prob Trans*"

Recibe como parámetros el subgrafo $G_s(V, A_c)$, la matriz de rastros W, la matriz c de costos asociados a las aristas, los alfa y beta del *Ant System*, y la matriz estocástica de transición MP. En base a estas entradas, se actualiza la matriz MP de la siguiente forma.

$$MP_{ij} = \begin{cases} \dfrac{(W_{ij})^{alfa} \times (\eta_{ij})^{beta}}{\sum_{(i,k) \in A_c} \left((W_{ik})^{alfa} \times (\eta_{ik})^{beta} \right)}, si \quad (i,j) \in A_c \\ 0, sino \end{cases} .$$

Donde η_{ij} se le denomina visibilidad de la arista $(i,j) \in A_c$, y se calcula como $\eta_{ij} = (1/c_{ij})$.

Procedimiento "*Selecciono Camino*"

Recibe como entradas la estructura tabu L, los nodos $i, j \in T$, el conjunto de nodos terminales T, y la matriz c de costos de las aristas. Devuelve el camino $p \in L$ que satisface el "*Criterio de Selección de Camino*" visto anteriormente, el costo del camino seleccionado y además una variable booleana que indica el éxito de la selección. Eventualmente, esta variable puede tener el valor FALSE si no se encontró ningún camino que une los terminales i y j en $G_s(V, A_c)$.

Procedimiento "*Act_Mejor_Cam*"

Recibe como parámetro un camino $p \in L$ que une los nodos $i, j \in T$, y copia este camino en la variable p_{opt} donde se almacena el "mejor" camino encontrado hasta el momento por "*Busco_Camino*".

Procedimiento "*Actualizar_Solución*"

Recibe como parámetros de entrada el subgrafo $G_s(N, A)$, sobre el cual el algoritmo "*GSP_Ant_System*" construye la actual solución factible, la matriz c de costos asociados a las aristas, y un camino p encontrado por el procedimiento "*Busco_Camino*" que une los nodos $i, j \in T$. Se retorna la solución actualizada $G_s(N \cup Nodos(p), A \cup Aristas(p))$, y el costo de ésta dado por $\cos to\_sol = COSTO(A) + COSTO(p)$.

4.3.2.1.1.1.3

4.3.2.1.1.1.4 Procedimiento "*Actualizar_Mejor_Solucion*"

Recibe como entrada el subgrafo $G_s(N, A)$, el cual cumple ser una solución factible del espacio Γ_{GSP} hallada por el algoritmo "*GSP_Ant_System*" y tiene costo menor que la mejor solución factible encontrada hasta el momento. El procedimiento, actualiza la mejor solución factible. Su pseudocódigo es el siguiente.

```
Procedure Actualizar_Mejor_Solucion(in N:Nodos, A:Aristas; in/out N_Sol:Nodos, A_Sol:Aristas,
                                                              min_costo:real);
   N_Sol=N;
   A_Sol=A;
  min_costo=COSTO(A);
End;
```

Procedimiento "*Actualizar_Matrices*"

Sea $G_s(N, A)$ el subgrafo solución actual sobre el cual el algoritmo "*GSP_Ant_System*" intenta construir una solución factible del espacio Γ_{GSP}. Al

tener que agregar a este subgrafo un nuevo camino (obtenido por el procedimiento "*Busco_Camino*") que une dos nodos terminales $i, j \in T$, es necesario actualizar la matriz M donde se almacena la cantidad de requerimientos aún no satisfechos para todo par de nodos terminales, además de la matriz booleana B donde se guarda la información de entre que pares de nodos terminales existe algún camino que los une en el subgrafo solución actual. La actualización de estas matrices la realiza el procedimiento "*Actualizar_Matrices*".

El procedimiento "*Actualizar_Matrices*" recibe como parámetros un camino p que une dos nodos terminales $i, j \in T$ (hallado por el procedimiento "*Busco_Camino*" sobre el subgrafo $G_s(V, E \setminus A)$), el conjunto de nodos terminales T, la matriz R de requerimientos de conexión, y como parámetros de entrada/salida la matriz M de requerimientos aún no satisfechos y la matriz booleana B. Primeramente el procedimiento analiza todos los nodos terminales presentes en p distintos de i y j. Investigando la matriz B y sabiendo que existe una nueva conexión entre los nodos terminales i y j a través del camino p encontrado, se actualizan los valores de M_{ik}, M_{jk} y M_{ij} (idem. los valores simétricos de la matriz) para todo $k \in p / k \in T, k \neq i, j$, decrementando el valor en 1 o en 2 según sea el caso. Luego para un nodo terminal $k \in p / k \in T, k \neq i, j$ fijo, se actualiza el valor de M_{uk} (idem. su simétrico) para todo $u \in p / k \in T, u \neq i, j, k$.

Una vez finalizada esta fase, se analizan todos los nodos terminales no presentes en el camino p, buscando a través de la matriz B información que pueda determinar posibles nuevas conexiones entre estos nodos y los nodos terminales de p. Durante esta fase se actualizan (en caso que corresponda) los valores M_{vj}, M_{vi} y M_{vk} (idem. sus simétricos) para todo $v \in T / v \notin p$ y para todo $k \in p / k \in T, k \neq i, j$. Si el análisis determina un nueva conexión entre dos nodos terminales y si la matriz B indica que no hay conexión entre ellos en el subgrafo $G_s(N, A)$, en una matriz auxiliar booleana A se almacena la información de esta nueva conexión. Al final del procedimiento "*Actualizar_Matrices*", se actualiza la matriz B realizando la suma booleana

con la matriz A; de esta manera se tiene en B la información entre que pares de nodos terminales hay conexión en el nuevo subgrafo solución $G_s(N \cup Nodos(p), A \cup Aristas(p))$.

El siguiente es un pseudocódigo detallado del procedimiento "*Actualizar_Matrices*".

Procedure *Actualizar_Matrices(in p:Camino, T:Terminales, R:Matriz_Req_Con;*
in/out M:Matriz_Req_con, B:Matriz_Bool);

$\forall u, v \in T, u \neq v$ *asigno* A_{uv}*=FALSE; /* A es del tipo Matriz_Bool */*
Sean $i, j \in T$ *los extremos del camino* p *;*
for each $k \in p / k \in T, k \neq i, j$ **do**
  A_{ki}*=TRUE; (Idem.* A_{ik}*=TRUE)*
  A_{kj}*=TRUE; (Idem.* A_{jk}*=TRUE)*
  if *((*B_{ij}*=TRUE)* **and** *(*B_{ik}*=FALSE)* **and** *(*B_{jk}*=FALSE))* **then**
    M_{ik}*=*M_{ik}*-2; (Idem.* M_{ki}*=*M_{ki}*-2)*
    M_{jk}*=*M_{jk}*-2; (Idem.* M_{kj}*=*M_{kj}*-2)*
    M_{ij}*=*M_{ij}*-1; (Idem.* M_{ji}*=*M_{ji}*-1)*
  else
    if *(*B_{ij}*=TRUE)* **then**
      M_{ik}*=*M_{ik}*-1; (Idem.* M_{ki}*=*M_{ki}*-1)*
      M_{jk}*=*M_{jk}*-1; (Idem.* M_{kj}*=*M_{kj}*-1)*
      M_{ij}*=*M_{ij}*-1; (Idem.* M_{ji}*=*M_{ji}*-1)*
    else *{*B_{ij}*=TRUE}*
      M_{ik}*=*M_{ik}*-1; (Idem.* M_{ki}*=*M_{ki}*-1)*
      M_{jk}*=*M_{jk}*-1; (Idem.* M_{kj}*=*M_{kj}*-1)*
      M_{ij}*=*M_{ij}*-1; (Idem.* M_{ji}*=*M_{ji}*-1)*
      A_{ij}*=TRUE; (Idem.* A_{ji}*=TRUE)*
    end_if;
  end_if;
  for each $u \in p / u \in T, u \neq i, j, k$ **do**
    if *(*B_{uk}*=TRUE)* **then**
      A_{uk}*=TRUE; (Idem.* A_{ku}*=TRUE)*
    end_if;
    M_{uk}*=*M_{uk}*-1; (Idem.* M_{ku}*=*M_{ku}*-1)*
  end_for_each;
end_for_each;
for each $v \in T / v \notin p$ **do**
  for each $k \in p / k \in T, k \neq i, j$ **do**
    if *((*B_{iv}*=TRUE)* **and** *(*B_{vj}*=FALSE))* **then**
      M_{vj}*=*M_{vj}*-1; (Idem.* M_{jv}*=*M_{jv}*-1)*
      A_{vj}*=TRUE; (Idem.* A_{jv}*=TRUE)*

```
. . . . . .
. . . . . .
if (B_iv=FALSE) and (B_jv=FALSE) then
   if (B_vk=TRUE) then
      if (B_ij=FALSE) then
         M_vi=M_vi-1; (Idem. M_iv=M_iv-1)
         M_vj=M_vj-1; (Idem. M_jv=M_jv-1)
         A_vi=TRUE; (Idem. A_iv=TRUE)
         A_vj=TRUE; (Idem. A_jv=TRUE)
      else        {B_ij=TRUE}
         if ((R_vk-M_vk)≥2) then
            M_vi=M_vi-2; (Idem. M_iv=M_iv-1)
            M_vj=M_vj-2; (Idem. M_jv=M_jv-1)
            A_vi=TRUE; (Idem. A_iv=TRUE)
            A_vj=TRUE; (Idem. A_jv=TRUE)
         else
            M_vi=M_vi-1; (Idem. M_iv=M_iv-1)
            M_vj=M_vj-1; (Idem. M_jv=M_jv-1)
            A_vi=TRUE; (Idem. A_iv=TRUE)
            A_vj=TRUE; (Idem. A_jv=TRUE)
         end_if;
      end_if;
   end_if;
end_if;
```

4.4 3.4 Análisis de Complejidad

En este punto se calculará el orden del algoritmo "*GSP_Ant_System*" en el peor caso; para ello se debe calcular previamente los órdenes de los procedimientos auxiliares utilizados por éste.

Sea $G = (V, E)$ un grafo en las condiciones del GSP. Sean además:

- $n = |V|$ la cantidad de nodos del grafo,
- $n_T = |T|$ la cantidad de nodos terminales,
- $r_{Max} = Max\{r_{ij}; i, j \in T\}$ el mayor requerimiento de conexión entre pares de nodos terminales,
- MAX_ANTS una cota máxima a la cantidad de "hormigas" posibles a ubicar en el grafo G.

En función de estos parámetros se calcularán los órdenes de los procedimientos más "costosos" y del algoritmo "*GSP_Ant_System*".

En el cálculo del orden del procedimiento "*Busco_Camino*" se toman en cuenta los ordenes de los procedimientos "*Actualizo_Rastros*" y "*Actualizo_Prob_Trans*" los cuales son $O\left(\frac{n\cdot(n-1)}{2}\right) = O(n^2)$. El orden de "*Busco_Camino*" viene dado por:

$O(MAX\_INTENTOS \times MAX\_CAM \times n \times (MAX\_ANTS + n \cdot (n-1)))$.

En el cálculo del orden del "*GSP_Ant_System*" se toman en cuenta los procedimientos "*Busco_Camino*", "*Grafo_Cociente*", "*Actualizar_Solución*" y "*Actualizar_Matrices*". El orden de "*Grafo_Cociente*" y "*Actualizar_Solución*" es $O\left(\frac{n\cdot(n-1)}{2}\right) = O(n^2)$, mientras que el de "*Actualizar_Matrices*" es $O(n_T{}^2)$.

El orden del algoritmo "*GSP_Ant_System*" viene dado entonces por: $O\left(MAX\_ITER \times MAX\_SOL\_FACT \times r_{Max} \times (T_{B\_C} + T_{G\_C} + T_{A\_S} + T_{A\_M})\right)$, donde: $T_{B\_C} = MAX\_INTENTOS \times MAX\_CAM \times n \times (MAX\_ANTS + n \cdot (n-1))$ es la cantidad de operaciones fundamentales de "*Busco_Camino*" en el peor caso, $T_{G\_C} = \frac{n \cdot (n-1)}{2}$ es la cantidad de operaciones fundamentales de "*Grafo_Cociente*", $T_{A\_S} = \frac{n \cdot (n-1)}{2}$ es la cantidad de operaciones fundamentales de "*Actualizar_Solución*" y $T_{A\_M} = n_T{}^2 + n_T$ es la cantidad de operaciones fundamentales de "*Actualizar_Matrices*" en el peor caso.

4.5 3.5 Aspectos de Implementación

La implementación del algoritmo aproximado para el GSP, se realizó en el **Lenguaje de Programación C**. Como estructuras de datos para la representación de los grafos, se utilizó matrices de adyacencias. En la búsqueda de caminos disjuntos entre un par de nodos terminales, los caminos encontrados por las hormigas, se almacenan en la "*tabu_list*"; esta estructura es una matriz en la cual la k - ésima hormiga guarda en la fila k el camino recorrido. Todas las funciones y procedimientos del código fuente, fueron implementados tomando como base el pseudocódigo presentado en *3.3*. Como resultado, se obtuvo un código fuente de sencilla comprensión y fácil corrección.

Para realizar las pruebas del algoritmo, el ejecutable fue corrido en un equipo Sun SPARCStation 5 con 64 MB de memoria RAM y sistema operativo Solaris 7.

4.6 3.6 Pruebas de Validación

4.6.1 3.6.1 Introducción

En este punto se verán las pruebas de validación del algoritmo. Éstas se realizan para verificar el funcionamiento del algoritmo en instancias particulares del GSP. La selección de estos casos se realizó pensando en

instancias del problema con grafos de dimensión “pequeña”, donde la solución óptima se conoce de antemano, buscando además topologías que creen diferentes situaciones particulares en la fase de construcción de una solución factible por parte del algoritmo.

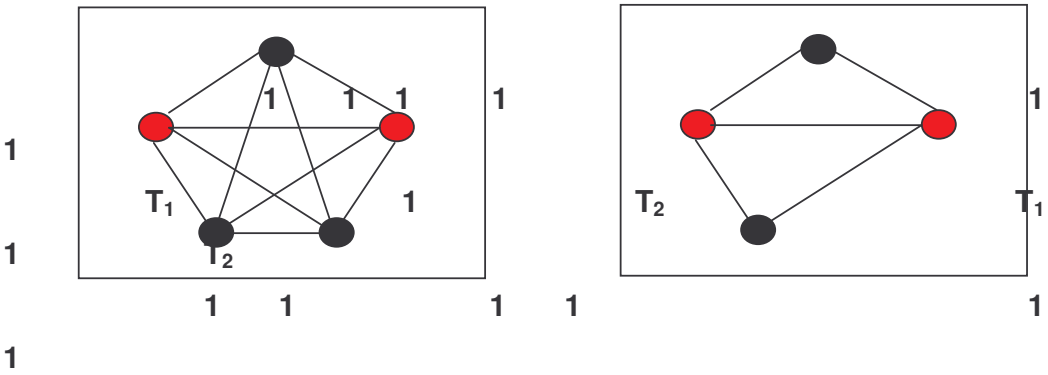
Se presentan a continuación los casos de pruebas de validación seleccionados, sus soluciones óptimas, un primer estudio de sensibilidad de sus parámetros y las soluciones dadas por el algoritmo “*GSP_Ant_System*”.

4.6.2 3.6.2 Casos de Prueba

En la representación gráfica de los grafos asociados a las instancias de GSP elegidas como casos de prueba, los nodos negros representan los nodos del tipo Steiner, mientras que los nodos rojos representan los nodos terminales. Las aristas están etiquetadas con sus costos y los requerimientos de conexión están dados por la matriz R.

4.6.2.1 Problema 1

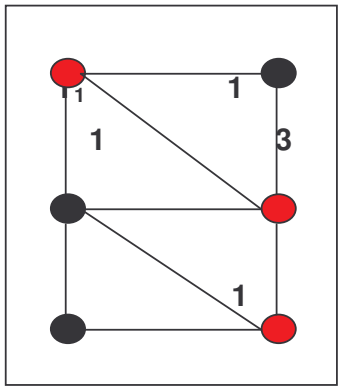
Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 3 \\ 3 & 0 \end{pmatrix}$.



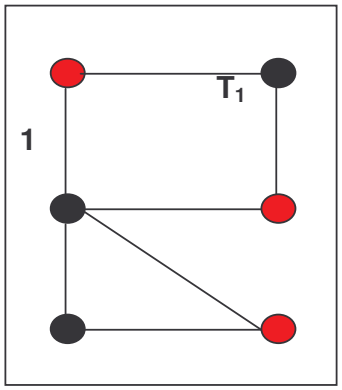
[Grafo 1 (K_5)] [Solución Óptima]

4.6.2.2 Problema 2

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}$.



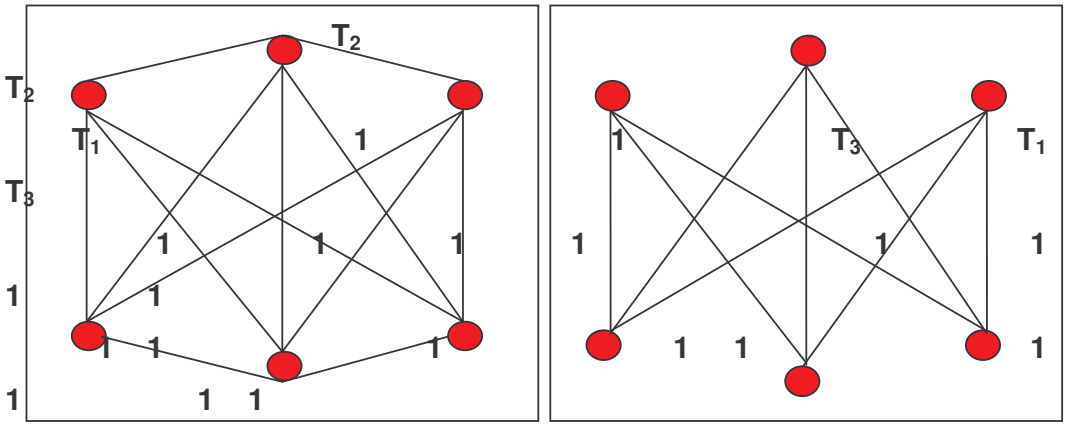
[Grafo 2]



[Solución Óptima]

4.6.2.3 Problema 3

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 3 & 3 & 3 & 3 & 3 \\ 3 & 0 & 3 & 3 & 3 & 3 \\ 3 & 3 & 0 & 3 & 3 & 3 \\ 3 & 3 & 3 & 0 & 3 & 3 \\ 3 & 3 & 3 & 3 & 0 & 3 \\ 3 & 3 & 3 & 3 & 3 & 0 \end{pmatrix}$.



T4 T6 T4

T6

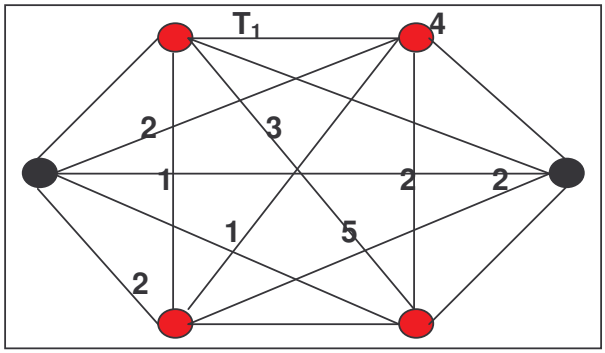
1 T5 1

T5

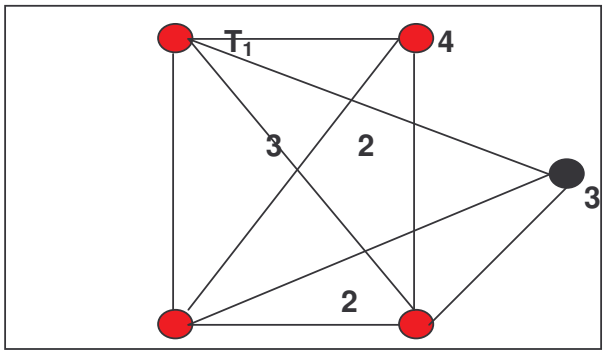
[Grafo 3] [Solución Óptima]

4.6.2.4 Problema 4

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 2 & 3 & 4 \\ 2 & 0 & 2 & 3 \\ 3 & 2 & 0 & 2 \\ 4 & 3 & 2 & 0 \end{pmatrix}$.



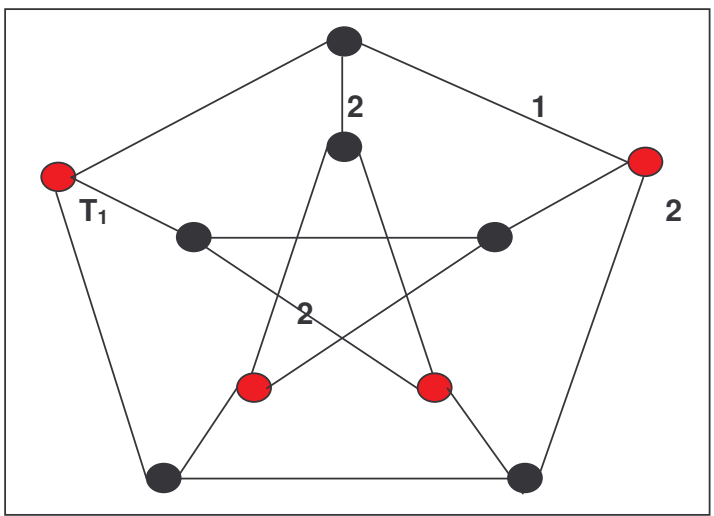
[Grafo 4 (K_6)]



[Solución Óptima]

4.6.2.5 Problema 5

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 2 & 3 \\ 2 & 2 & 0 & 3 \\ 3 & 3 & 3 & 0 \end{pmatrix}$.



1

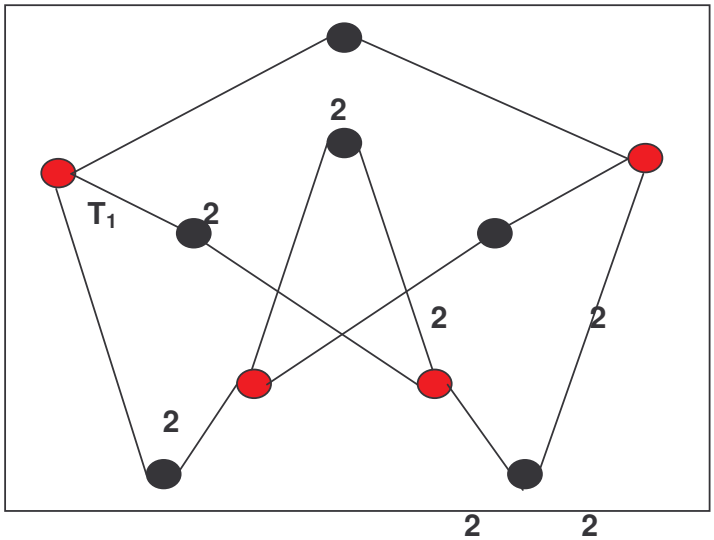
2 2 2

2

T_3 T_4

1

[Grafo 5 (Peterson)]



2 2

2

T_1 2

T_2

2 2

2

2 2

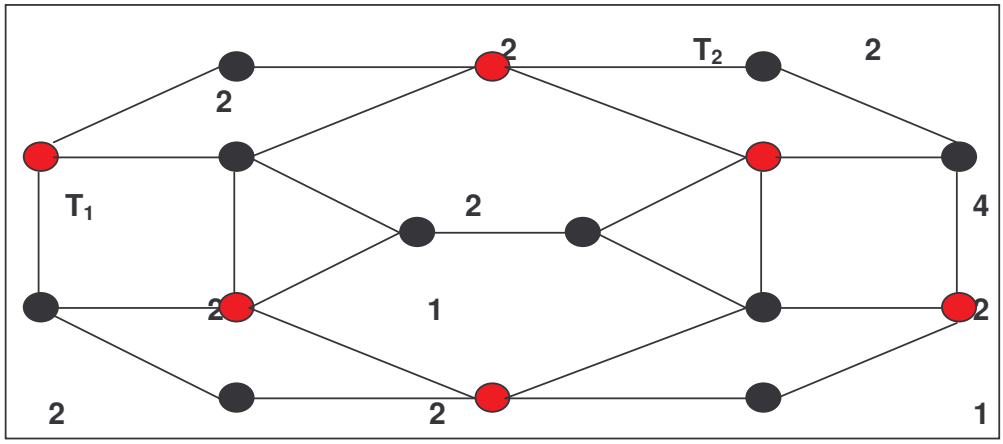
T_3 T_4

2 2

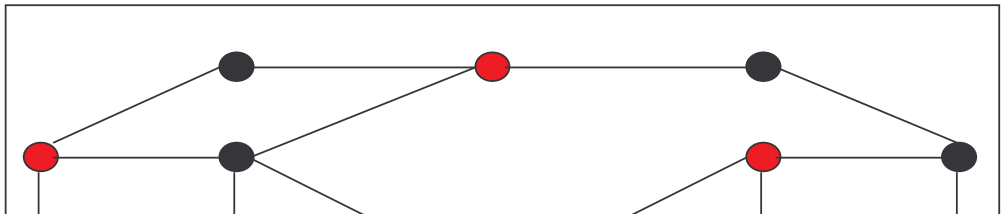
[Solución Óptima]

4.6.2.6 Problema 6

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 2 & 3 & 2 & 3 & 2 \\ 2 & 0 & 2 & 3 & 2 & 3 \\ 3 & 2 & 0 & 2 & 3 & 2 \\ 2 & 3 & 2 & 0 & 2 & 3 \\ 3 & 2 & 3 & 2 & 0 & 2 \\ 2 & 3 & 2 & 3 & 2 & 0 \end{pmatrix}$.



[Grafo 6]



2 T_2 2

2

1

T_1 2

T_3

2 1 2

2

2 2 1

2 2

2 1

2

2

T_4 2

T_6

1 2 2

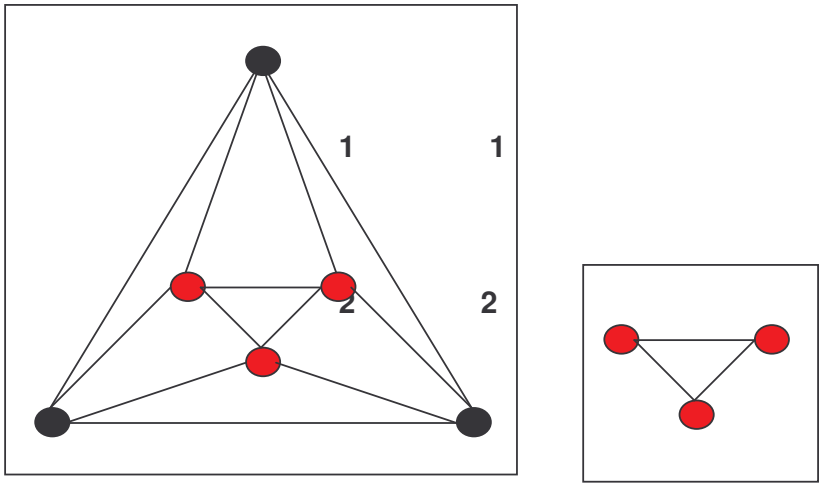
2

T_5

[Solución Óptima]

4.6.2.7 Problema 7

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix}$.



3

3

2 **3** **3** **2**

3 **3**

2 **2**

1

[Solución Óptima]

[Grafo 7]

La siguiente tabla muestra para cada problema de validación, las características estructurales del grafo asociado y de su solución óptima.

4.6.2.8 ÚMERO DE PROBLEMA	4.6.2.9 ODOS TERMINALES	4.6.2.10 ODOS DE STEINER	4.6.2.11 ÚMERO DE ARISTAS	4.6.2.12 ODOS DE STEINER EN 4.6.2.13 OL. OPTIMA	4.6.2.14 ÚMERO DE ARISTAS EN 4.6.2.15 SOL. OPTIMA
4.6.2 1)	4.6.2	4.6.2	4.6.2 0	4.6.2	4.6.2
4.6.2 2)	4.6.2	4.6.2	4.6.2	4.6.2	4.6.2
4.6.2 3)	4.6.2	4.6.2	4.6.2 3	4.6.2	4.6.2
4.6.2 4)	4.6.2	4.6.2	4.6.2 5	4.6.2	4.6.2
4.6.2 5)	4.6.2	4.6.2	4.6.2 5	4.6.2	4.6.2 2
4.6.2 6)	4.6.2	4.6.2 0	4.6.2 5	4.6.2 0	4.6.2 3
4.6.2 7)	4.6.2	4.6.2	4.6.2 2	4.6.2	4.6.2

4.6.3 3.6.3 Resultados Numéricos

En las corridas hechas sobre los diferentes casos de pruebas los valores tomados por los parámetros del *Ant System* fueron: $\alpha \in \{0.5,1,2,5,10\}$, $\beta \in \{0.5,1,2,5,10\}$, $\rho \in \{0.5,0.7,0.9\}$ y $Q \in \{10,100,10000\}$; siendo α (sensibilidad del rastro) y β (sensibilidad de la distancia) los parámetros utilizados en el cálculo de las probabilidades de transición, ρ el coeficiente de evaporación, y Q es la cantidad de feromona dejada por una hormiga en una transición. El valor del parámetro fijo ANT_FACTOR utilizado en la fórmula para el cálculo de la distribución de "hormigas" en los nodos terminales, se tomó en el conjunto $\{1,2,3,4\}$. El modelo de algoritmo *Ant System* utilizado en la implementación del "*GSP_Ant_System*" fue el "*Ant-Quantity*" [15]. La razón de esta elección, es su fácil adaptación al diseño de un algoritmo de búsqueda de caminos entre pares de nodos terminales.

Otros parámetros que utiliza la implementación del algoritmo son:

MAX_ITER - número máximo de intentos realizados por el "*GSP_Ant_System*" para encontrar una solución factible.

MAX_SOL_FACT - número máximo de soluciones factibles encontradas por el algoritmo.

$MAX\_INTENTOS$ - número máximo de intentos realizados por el procedimiento "*Busco_Camino*" para encontrar un camino entre un par de nodos terminales.

MAX_CAM - número máximo de caminos seleccionados para comparar por el procedimiento "*Busco_Camino*" entre un par de nodos terminales.

Tanto en las pruebas de validación como experimentales, los valores de estos parámetros fueron:

MAX_ITER	MAX_SOL_FACT	MAX_INTENTOS	MAX_CAM
5	**7**	**5**	**2**

Los tests de los parámetros se realizaron de la siguiente forma: se hizo variar el valor de cada parámetro tomando como fijos los valores de los otros parámetros. A continuación se brinda un resumen de los resultados para los diferentes casos de prueba. Para cada problema se seleccionó tres de los mejores resultados obtenidos de las diferentes combinaciones de valores de los parámetros del algoritmo. La siguiente tabla muestra estos resultados.

NP	α	β	ρ	Q	ANT_FACTOR	NA	NF	CS	CO	GAP
(1)	0.5	1	0.7	10	2	4	1	5	5	0
(1)	1	5	0.7	10	2	4	1	5	5	0
(1)	1	10	0.7	100	2	4	1	5	5	0
(2)	0.5	1	0.7	10	2	6	2	7	7	0
(2)	1	5	0.7	100	2	6	2	7	7	0
(2)	1	10	0.7	100	2	6	2	7	7	0
(3)	0.5	2	0.7	10	2	4	1	9	9	0
(3)	1	5	0.7	100	2	4	1	9	9	0
(3)	2	5	0.7	100	2	4	1	9	9	0
(4)	1	5	0.7	10	2	4	1	22	22	0
(4)	1	10	0.7	100	2	4	1	22	22	0
(4)	2	10	0.7	100	2	4	1	22	22	0
(5)	1	2	0.7	10	2	12	2	24	24	0
(5)	1	5	0.7	100	2	12	2	24	24	0
(5)	2	10	0.7	10	2	12	2	24	24	0

				0						
(6)	1	5	0.7	10	2	8	1	41	41	0
(6)	1	10	0.7	100	2	8	1	41	41	0
(6)	2	10	0.7	100	2	8	1	41	41	0
(7)	1	5	0.7	10	2	8	3	9	9	0
(7)	1	10	0.7	100	2	8	3	9	9	0
(7)	2	5	0.7	100	2	8	3	9	9	0

[Resultados Numéricos]

NP – es el número de problema.

ANT_FACTOR – es el parámetro fijo utilizado en el cálculo de la cantidad de "hormigas" a ubicar en cada nodo terminal.

NA – es el número máximo de "hormigas" utilizadas en la búsqueda de un camino entre un par de nodos terminales.

NF – es el número de soluciones factibles halladas en la corrida.

CS – es el costo de la mejor solución factible encontrada por el algoritmo.

CO – es el costo de la solución óptima del problema.

GAP – es el porcentaje de error y se define como: $GAP = 100 \times \left(1 - \frac{CO}{CS}\right)$.

El análisis numérico de los resultados obtenidos en las pruebas de validación muestra lo siguiente:

- La selección de los conjuntos de valores de prueba para los parámetros resulto ser "buena" ya que en la mayoría de las combinaciones de valores se alcanzo la solución óptima en los diferentes problemas. En muchos casos el algoritmo "*GSP_Ant_System*" encontró la solución óptima en la primera iteración de su fase de construcción de una solución factible y en otros casos se halló la solución óptima en iteraciones posteriores.

- En función de los resultados obtenidos algunas de las mejores combinaciones de valores de parámetros resultaron ser: $\alpha = 1$, $\beta = 5$ o $\beta = 10$, $Q = 10$ o $Q = 100$, $\rho = 0.7$ y $ANT\_FACTOR = 2$. Con estos parámetros el algoritmo "*GSP_Ant_System*" encontró la solución óptima de todos los problemas en la primera iteración de su fase de construcción de una solución factible.
- Excepto en una corrida (una instancia del Problema 4), en los casos en los cuales se alcanzó la solución óptima, ésta fue encontrada en la primera iteración de búsqueda de una solución factible por parte del algoritmo "*GSP_Ant_System*".
- El "*Criterio de Selección de Camino*" utilizado en el procedimiento "*Selecciono_Camino*" juntamente con el procedimiento auxiliar "*Actualizar_Matrices*" resultan críticos para la obtención de "buenas" soluciones factibles (de bajo costo) en la fase de construcción de una solución por parte del "*GSP_Ant_System*".

Los problemas de validación fueron seleccionados pensando en someter al algoritmo "*GSP_Ant_System*" a instancias del problema GSP donde las topologías son pequeñas y donde además surgen diferentes situaciones particulares que ponen a prueba la capacidad del algoritmo de armar una "buena" solución en su fase de construcción de una solución factible. En todos los casos, el algoritmo tuvo un muy buen desempeño obteniéndose la solución óptima en todos los problemas.

Dado que las dimensiones de las topologías de los casos de prueba son de "pequeño" porte, es necesario realizar pruebas más exhaustivas de medición de performance. Para ello se utilizaron los problemas seleccionados como pruebas experimentales. A continuación se brinda una descripción de las pruebas de performance realizadas.

4.7 3.7 Pruebas Experimentales

4.7.1 3.7.1 Introducción

Se presenta en este punto, un conjunto de casos de pruebas elegidos para testear la performance del algoritmo *"GSP_Ant_System*". La selección se realizó buscando modelos de problemas reales de optimización de redes telefónicas o eléctricas, donde los grafos asociados tienen un número considerable de nodos y aristas. Esto permite estudiar el desempeño del algoritmo en problemas con redes de mediana y gran escala. Desafortunadamente no existen muchas aplicaciones reales que requieran niveles de conexión mayores a 2 entre pares de nodos terminales. En el diseño de redes de fibras ópticas, en general, alcanza con exigir que la red sea 2-arista-conexa para lograr un nivel de confiabilidad elevado. Algunas de las pocas aplicaciones reales donde se aplica el modelo GSP, con requerimientos de conexión mayores que 2 entre nodos terminales, son sistemas críticos militares, diseño de circuitos altamente confiables, y algunos sistemas de sincronización de robots utilizados para realizar construcciones en serie; a saber maquinaria, automóviles, etc.

Seguidamente se brindan los casos de pruebas experimentales seleccionados, sus soluciones óptimas y las soluciones dadas por el algoritmo "*GSP_Ant_System*".

4.7.2 3.7.2 Casos de Prueba

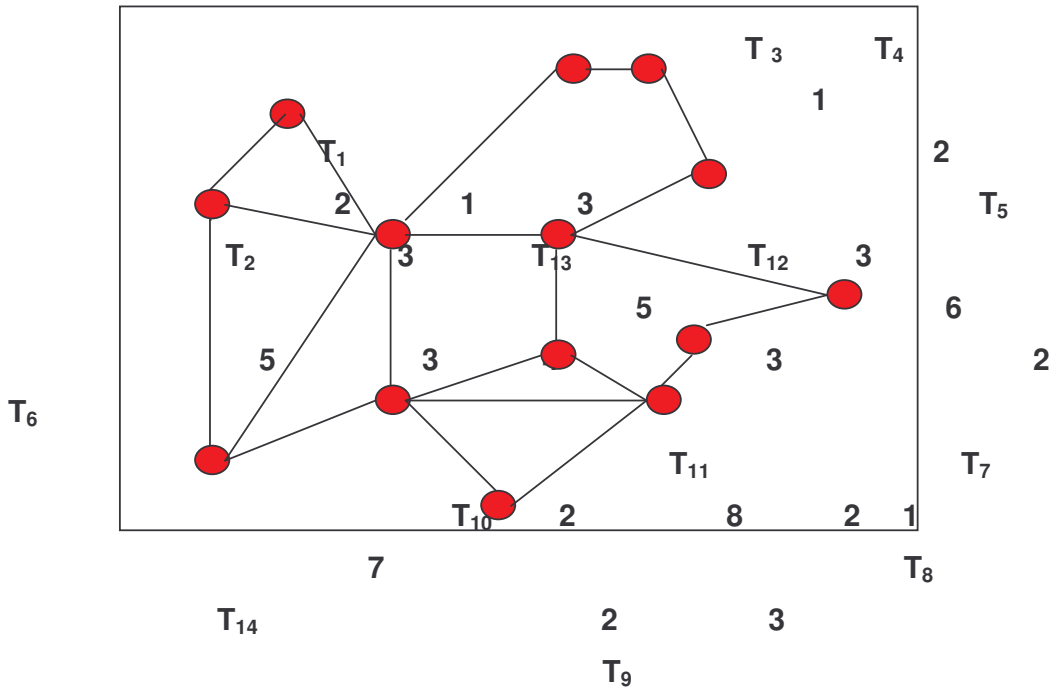
Los casos de pruebas seleccionados son los siguientes:

- El diseño de una red 2-arista-conexa tomando como base el núcleo de fibra óptica de la red telefónica de Montevideo; dicha topología de red fue extraída de [13]. Se asumen costos heterogéneos de conexión entre los diferentes nodos terminales.
- El diseño de una red 4-arista-conexa, basada en una topología de decaedro con dos nodos terminales. Los enlaces punto a punto entre nodos de Steiner tienen costo 1, entre un nodo de Steiner y un terminal tiene costo 2, y entre nodos terminales costo 4.

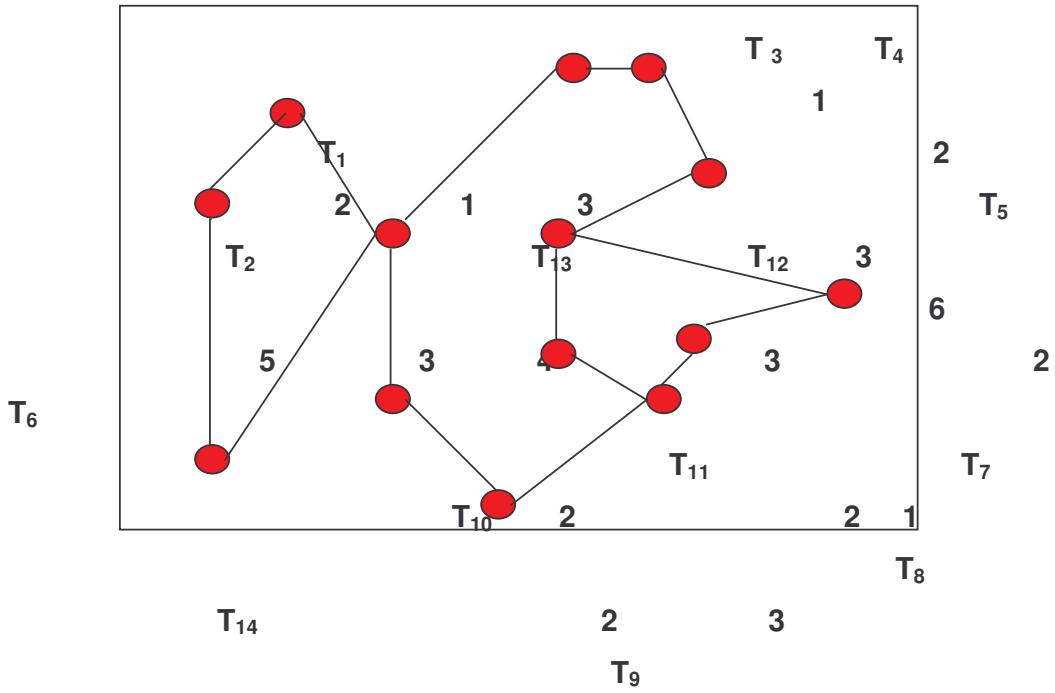
- El diseño de una red 2-arista-conexa, basada en una topología de dodecaedro con 5 nodos terminales. Los enlaces punto a punto entre nodos de Steiner tienen costo 3, entre un nodo de Steiner y un terminal tiene costo 4, y entre nodos terminales costo 6. Esta topología de red fue extraída de [13].
- Un problema real de diseño de una red de alta confiabilidad, donde se requiere niveles de conexión mayores a 2 entre nodos terminales, es el diseño de un HSODTN (*High Speed Optical Data Transmission Network*) para un portaaviones. El HSODTN es una red de comunicación de fibra óptica de alta tecnología que conecta diferentes sitios del buque permitiendo la comunicación entre los sistemas de armamento. La razón por la cual se desea un alto grado de confiabilidad en la red es obvia: ésta debe ser capaz de continuar funcionando al producirse daños por un enfrentamiento en batalla. Se presenta una topología de red (versión reducida) donde se modela las posibles líneas de comunicación de fibra óptica entre las diferentes estaciones de armamento de un portaaviones (modeladas como nodos terminales) [50]. En el problema original, se desea encontrar un subgrafo 3-nodo-conexo de costo mínimo que cubra el conjunto de nodos terminales. A efectos de poder aplicar el algoritmo "*GSP_Ant_System*", se asume en este trabajo, que una topología 3-arista-conexa satisface los requerimientos de confiabilidad exigidos en el diseño del HSODTN. Los costos de los enlaces punto a punto entre nodos de Steiner tienen costo 1, los enlaces entre un nodo de Steiner y un terminal tienen costo 2, y los enlaces entre nodos terminales tienen costo 4.
- Una instancia genérica del GSP [50], donde el grafo asociado tiene 48 nodos del tipo Steiner, 13 nodos terminales y una alta densidad de aristas. Los enlaces punto a punto entre nodos del tipo Steiner tienen costo 1, un enlace entre un nodo del tipo Steiner y un nodo terminal tiene costo 2 y un enlace entre dos nodos terminales tiene costo 4. La matriz de requerimientos de conexión tiene diferentes tipos de requerimientos para distintos pares de nodos terminales.

4.7.2.1 Problema 1

Matriz de requerimientos de conexión: $R = \{r_{ij} = 2, \forall i, j \in T\}$.



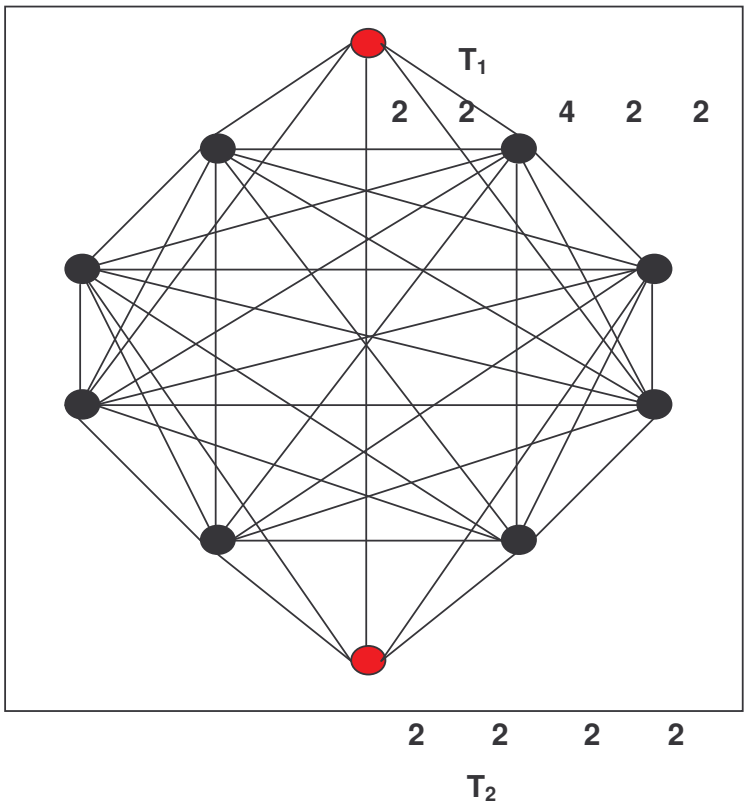
[Red Telefónica de Montevideo]



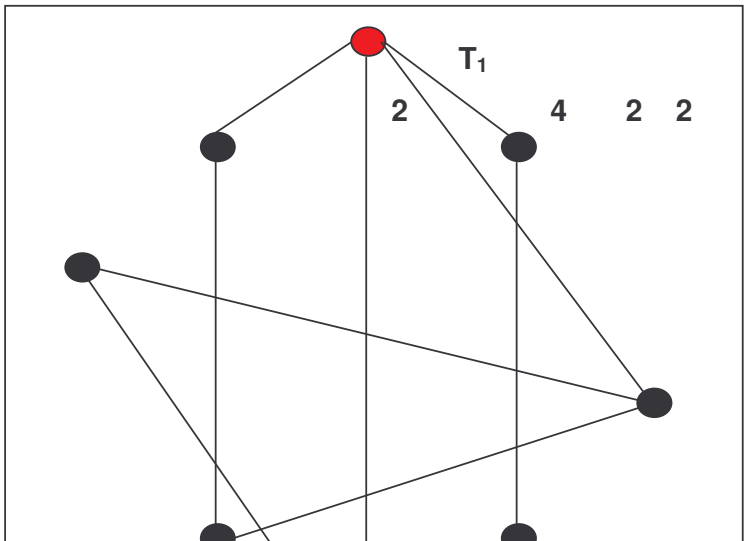
[Solución Óptima]

4.7.2.2 Problema 2

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 4 \\ 4 & 0 \end{pmatrix}$.



[Decaedro]



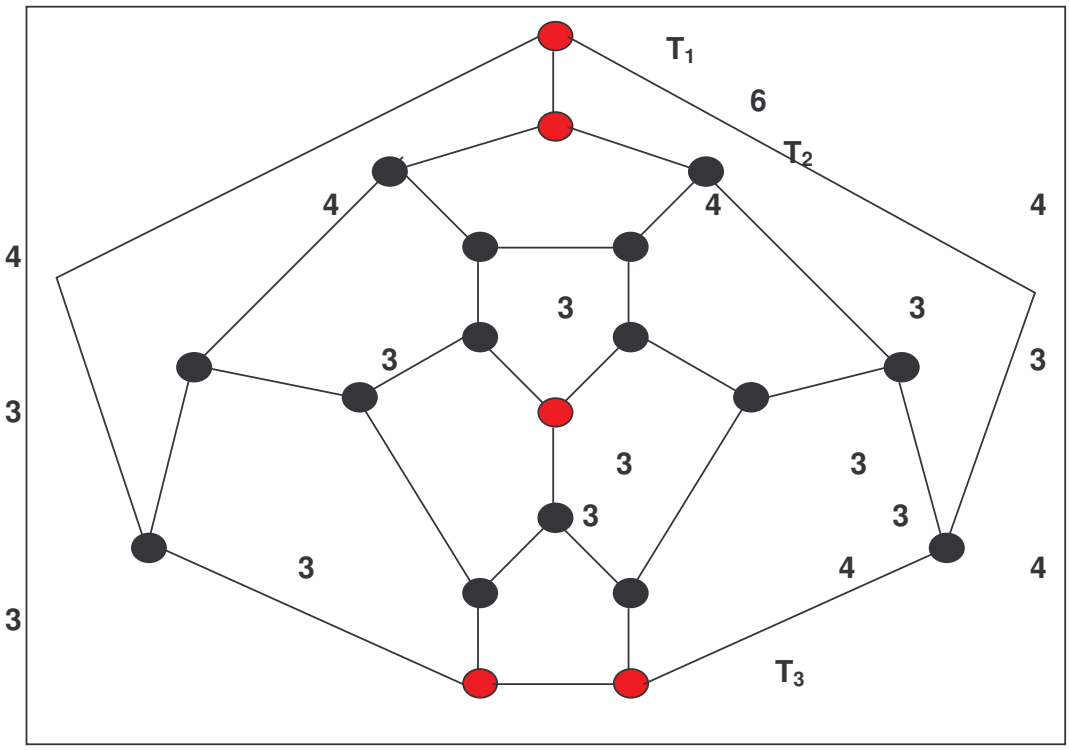
2 2 2

T_2

[Solución Óptima]

4.7.2.3 Problema 3

Matriz de requerimientos de conexión: $R = \begin{pmatrix} 0 & 2 & 2 & 2 & 2 \\ 2 & 0 & 2 & 2 & 2 \\ 2 & 2 & 0 & 2 & 2 \\ 2 & 2 & 2 & 0 & 2 \\ 2 & 2 & 2 & 2 & 0 \end{pmatrix}$.



3 3 4 3

3

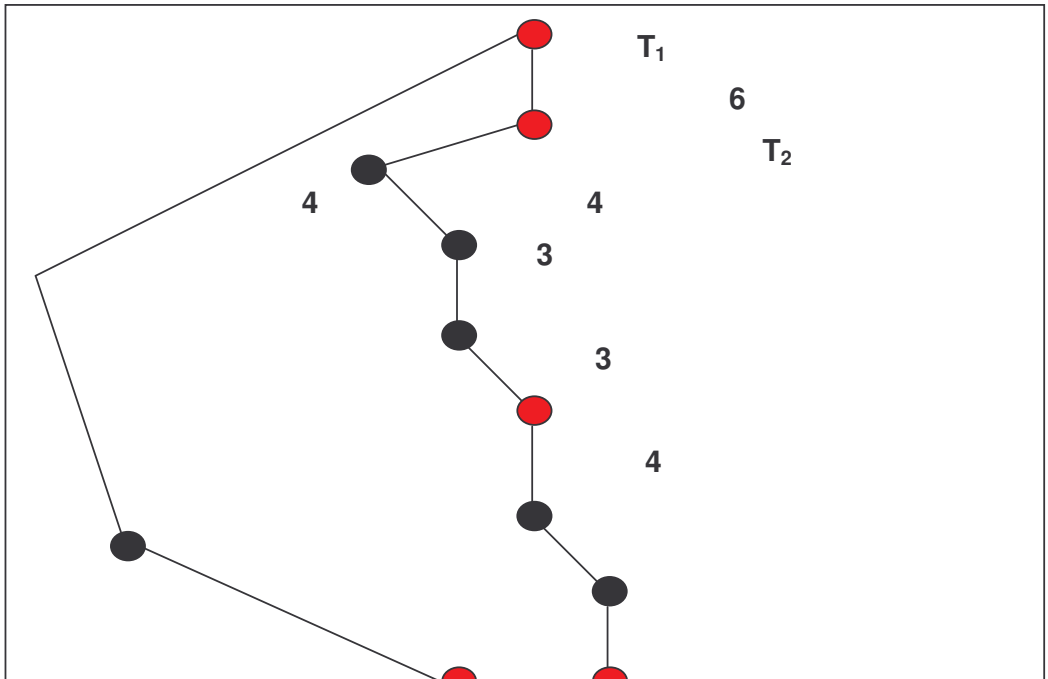
3 3

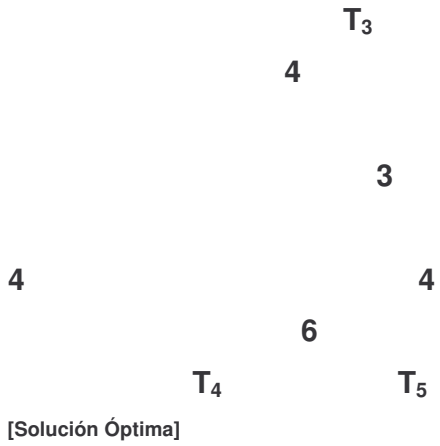
4 4 4 4

6

T_4 T_5

[Dodecaedro]

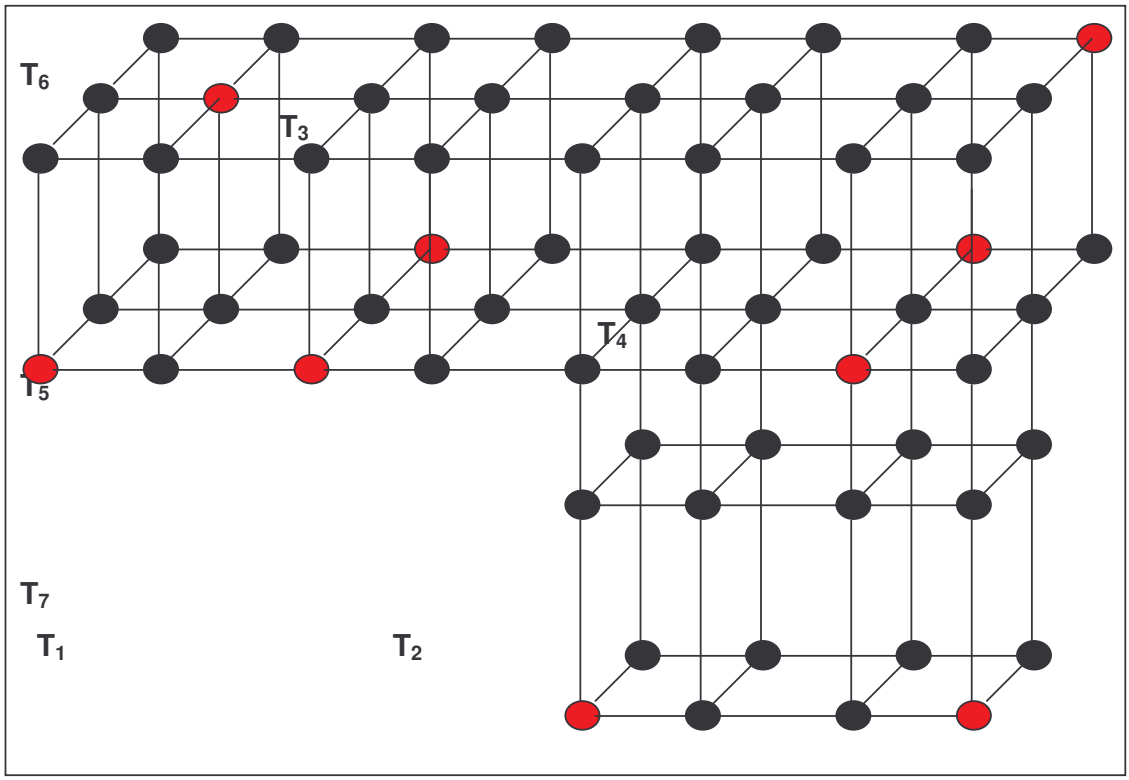




[Solución Óptima]

4.7.2.4 Problema 4

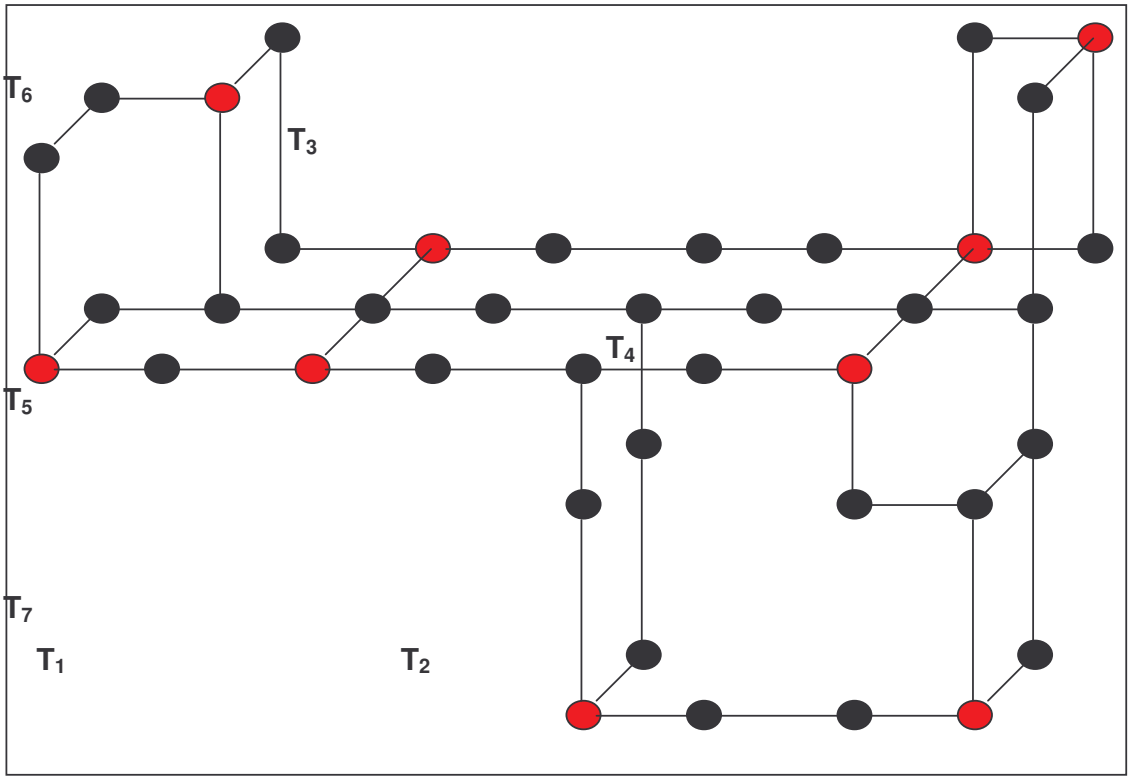
Matriz de requerimientos de conexión: $R = \{r_{ij} = 3, \forall ij \in T\}$.



T8

T9

[Problema “Ship” – Reducido]



T8

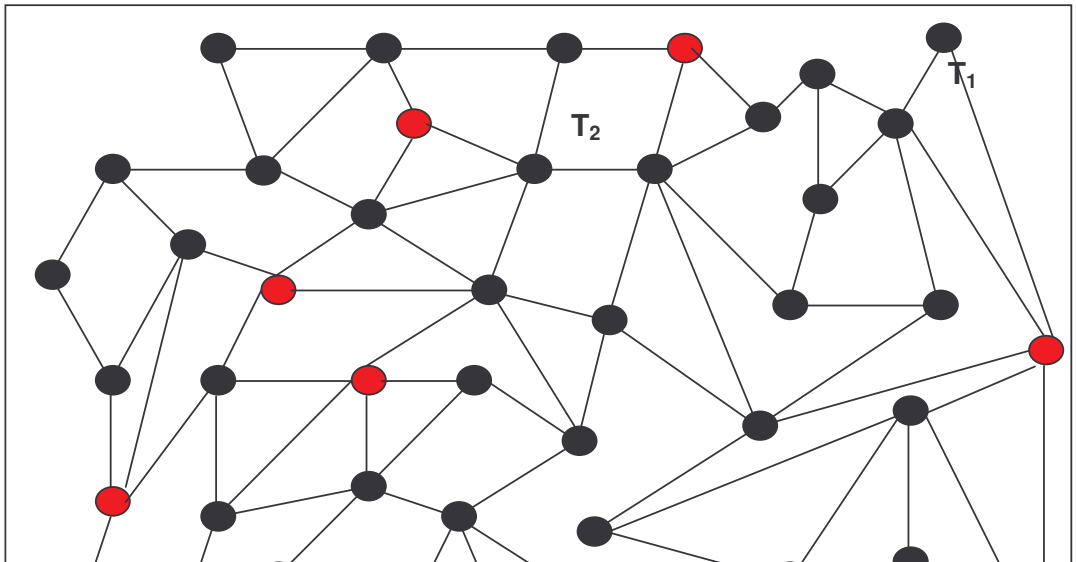
T9

[Solución Óptima]

4.7.2.5 Problema 5

Matriz de requerimientos de conexión:

$$R = \begin{pmatrix} 0 & 2 & 1 & 3 & 2 & 2 & 3 & 2 & 1 & 2 & 2 & 2 & 2 \\ 2 & 0 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 3 & 2 & 2 & 2 \\ 1 & 1 & 0 & 2 & 2 & 2 & 2 & 3 & 3 & 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 0 & 2 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 & 0 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & 2 \\ 2 & 2 & 2 & 2 & 1 & 0 & 2 & 2 & 3 & 3 & 2 & 1 & 2 \\ 3 & 2 & 2 & 2 & 1 & 2 & 0 & 1 & 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 3 & 2 & 1 & 2 & 1 & 0 & 3 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 2 & 3 & 2 & 3 & 0 & 1 & 1 & 1 & 1 \\ 2 & 3 & 1 & 1 & 2 & 3 & 2 & 2 & 1 & 0 & 2 & 2 & 2 \\ 2 & 2 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 2 & 0 & 3 & 3 \\ 2 & 2 & 2 & 2 & 2 & 1 & 2 & 2 & 1 & 2 & 3 & 0 & 2 \\ 2 & 2 & 2 & 1 & 2 & 2 & 2 & 2 & 1 & 2 & 3 & 2 & 0 \end{pmatrix}.$$



T_3

T_6

T_5

T_4

T_{10}

T_9

T_8

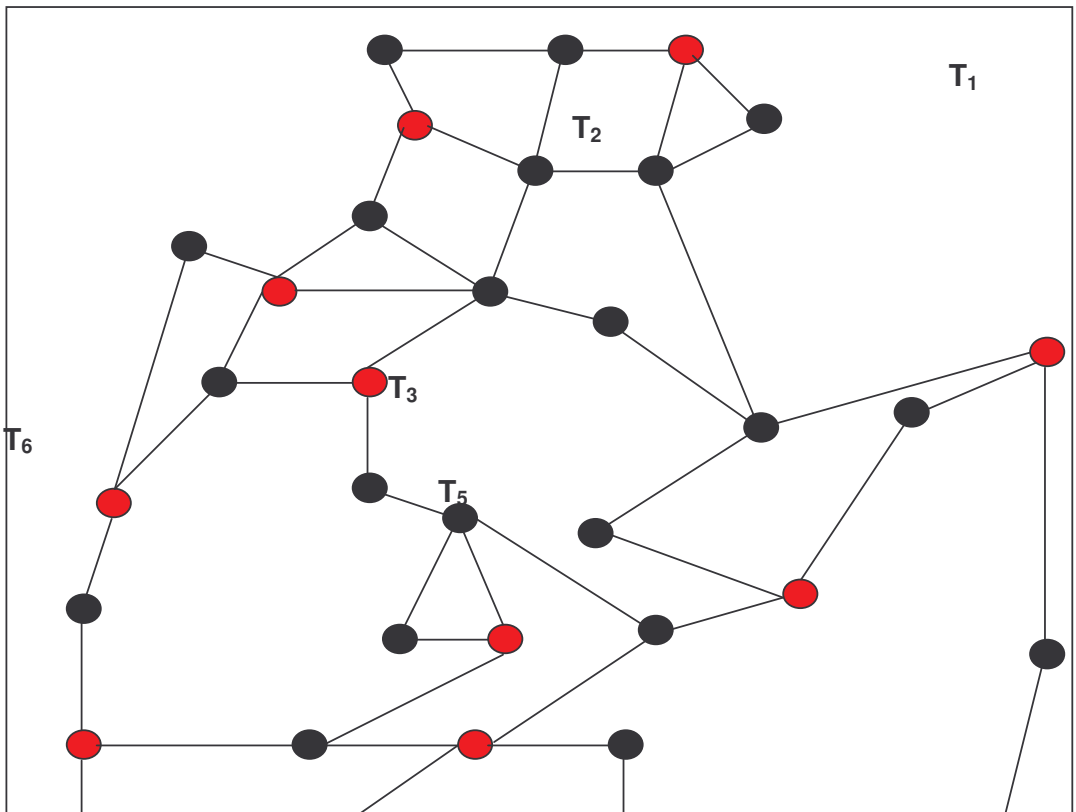
T_7

T_{12}

T_{11}

T_{13}

[Grafo Genérico Denso]



T_4

T_{10}

T_9

T_8

T_7

T_{12}

T_{11}

T_{13}

[Solución Óptima]

4.7.3 3.7.3 Resultados Numéricos

En las corridas realizadas con los casos de pruebas experimentales se tomaron los mismos conjuntos de valores para los parámetros *Ant System* que los utilizados en las pruebas de validación. Se hizo un ajuste más "fino" de los parámetros, tratando de encontrar aquellos juegos de valores que resultan en una mejor performance del algoritmo "*GSP_Ant_System*". A manera de ejemplo, la siguiente tabla contiene para cada problema, tres de los resultados obtenidos de las diferentes combinaciones de valores de los parámetros.

NP	α	β	ρ	Q	ANT_FACTOR	NA	NF	CS	CO	GAP	TM
(1)	0.5	2	0.7	10	2	8	2	50	46	8	4.35
(1)	1	5	0.7	100	2	8	2	46	46	0	4.07

(1)	2	10	0.7	100	2	8	2	46	46	0	3.57
(2)	0.5	2	0.7	10	4	8	2	20	19	5	1.47
(2)	1	5	0.7	100	4	8	3	19	19	0	1.08
(2)	2	10	0.7	100	4	8	3	19	19	0	0.54
(3)	0.5	2	0.7	10	3	6	3	48	45	6.26	7.05
(3)	1	5	0.7	100	3	6	3	45	45	0	6.58
(3)	2	10	0.7	100	3	6	3	45	45	0	6.54
(4)	0.5	1	0.7	10	4	8	2	79	74	6.32	46.17
(4)	1	5	0.7	100	4	8	2	77	74	3.89	42.02
(4)	2	10	0.7	100	4	8	2	74	74	0	38.20
(5)	1	2	0.7	10	4	16	4	109	98	10.09	53.43
(5)	2	5	0.7	100	4	16	4	103	98	4.85	48.09
(5)	2	10	0.7	100	4	16	5	102	98	3.92	43.32

[Resultados Numéricos]

TM – es el tiempo transcurrido en minutos hasta encontrar la mejor de las soluciones factibles de la "corrida".

El análisis de los resultados obtenidos de las diferentes corridas realizadas para cada problema, muestra lo siguiente:

- Los parámetros ρ y Q tienen influencia directa en la cantidad total de rastro dejado por las “hormigas” en cada iteración de movimiento sobre el grafo.
- Los parámetros α, ρ y Q afectan el tiempo requerido para que las “hormigas” elijan el mismo camino entre un par de nodos terminales (comportamiento uniforme).
- El parámetro crítico en la búsqueda de “buenos” caminos (seleccionados de acuerdo al “*Criterio de Selección de Camino*”) entre un par de nodos terminales es el parámetro β.
- De igual manera se puede decir que β es el parámetro crítico para la obtención de “buenas” soluciones factibles al GSP por parte del algoritmo “*GSP_Ant_System*” en su fase de construcción de la solución.
- En las corridas en las cuales se alcanzó el óptimo con un coeficiente de evaporación $\rho = 0.7$, también se alcanzo con $\rho = 0.9$; ambos resultan ser buenos valores para este parámetro.
- Con $\alpha = 0.5$, en todas las corridas en las cuales se encontró solución factible, ésta no era la solución óptima. Además con $\alpha = 1$ para los Problemas 4 y 5 se obtuvieron soluciones factibles subóptimas para cualquier combinación de los otros parámetros. En resumen, es recomendable que el valor del parámetro α sea mayor o igual a 2.
- Los mejores resultados se obtuvieron con el juego de parámetros: $\alpha = 2$, $\beta = 10$, $\rho = 0.7$ y $Q = 100$, alcanzándose la solución óptima en todos los problemas excepto el Problema 5.
- Para el Problema 5 en el mejor caso se obtuvo una solución factible con una diferencia porcentual de costo de un 3.92% con respecto al costo de la solución óptima.
- Al igual que lo observado en las pruebas de validación, las pruebas de performance realizadas muestran que la inteligencia del algoritmo “*GSP_Ant_System*” para encontrar “buenas” soluciones factibles está en gran parte centralizada en los procedimientos “*Selecciono_Camino*” (utilizado en “*Busco_Camino*”) y el procedimiento auxiliar “*Actualizar_Matrices*”.

Durante las corridas experimentales, es de destacar el esfuerzo realizado en el ajuste adecuado de los parámetros *Ant System* para la obtención de "buenas" soluciones factibles. Si bien en la mayoría de los casos, los juegos de valores utilizados permitían encontrar soluciones factibles en los diferentes problemas, muchas de éstas tenían una diferencia porcentual de costo con respecto al costo óptimo bastante grande (superior al 15%). En otros casos, se obtenían soluciones factibles no muy "buenas" luego de varios intentos (4 o 5 iteraciones en la fase de construcción de la solución).

Una vez realizado el refinamiento de los parámetros, las corridas hechas sobre los casos de prueba mostraron ser exitosas; obteniéndose la solución óptima en los Problemas 1, 2, 3, 4, y una solución subóptima para el Problema 5.

4.8 3.8 Conclusiones

Dado que hasta el momento se ha publicado un único algoritmo aproximado para el GSP aplicable a las clases más generales de grafos [1], es de gran importancia el diseño de una nuevo algoritmo aproximado. En este capítulo, se presentó una nueva heurística para la resolución del GSP en el caso más general. Como base para el diseño del nuevo algoritmo aproximado, se tomó la metaheurística conocida como *Ant System*. Se realizó una adaptación a medida para el GSP, principalmente se utilizó los algoritmos de la clase *Ant-Quantity* y *Ant-Density* para la búsqueda de caminos entre pares de nodos terminales.

Los resultados numéricos obtenidos en las pruebas experimentales, revelan una buena performance del algoritmo "*GSP_Ant_System*", luego de haberse realizado una minuciosa depuración de los diferentes juegos de valores de los parámetros. Se alcanzó la solución óptima en los 4 primeros problemas. En el Problema 5 se obtuvo en diferentes corridas (con parámetros distintos) dos soluciones factibles con una diferencia porcentual de costo inferior al 5% con respecto al costo óptimo. Estos resultados son muy buenos si se toma en cuenta que el cálculo de la solución óptima es un problema NP-difícil, donde los órdenes de los algoritmos que hallan la

solución óptima son exponenciales en la cantidad de nodos terminales y aristas. Además en muchas aplicaciones reales de diseño de redes no necesariamente se debe de requerir de la solución óptima, sino que basta con encontrar soluciones factibles cuyo costo esté dentro de un cierto radio de error (fijado de antemano) con respecto al costo óptimo.

Un aspecto importante referente al nuevo algoritmo aproximado, es el enfoque con el cual se ataca el problema GSP. A diferencia de los algoritmos conocidos hasta el momento (para clases particulares de grafos), donde las soluciones se hallan en base a un análisis de los politopos asociados al problema (un enfoque poliédrico), el "*GSP_Ant_System*" se basa en la estructura topológica del grafo en su fase de construcción de una solución factible.

Las aplicaciones del nuevo algoritmo son múltiples; en especial puede utilizarse para el diseño de redes de mediana y gran escala, donde los requerimientos de conexión son conocidos de antemano. Algunos ejemplos son:

- Diseño de topologías de bajo costo de redes de fibras óptica 2-arista-conexas y 3-aristas-conexas.
- Conexión múltiple entre un grupo de centrales hidroeléctricas con el menor costo posible.
- Algunos sistemas de comunicaciones militares utilizan protocolos denominados "Protocolos de Inundación" donde cada estación de transmisión retransmite paquetes de mensajes a todas las estaciones adyacentes. Este sistema de transmisión se puede mejorar analizando previamente los requerimientos reales de comunicaciones entre las diferentes estaciones militares (manteniendo cierto nivel de redundancia), y luego diseñar un sistema de comunicación confiable con un menor costo.
- Diseño de bajo costo de redes de distribución de agua potable en una ciudad, donde es necesario que existan formas alternativas de poder distribuir el agua a los diferentes sitios de la ciudad.

Existen algunos modelos de aplicaciones reales, donde es de interés encontrar solamente la solución óptima al GSP, descartándose soluciones factibles alternativas. A manera de ejemplo, se tiene el diseño de los grandes

sistemas de redes de gasoductos, donde los costos de los tendidos son extremadamente altos. Por lo general, estas redes pueden modelarse mediante topologías de grafos de mediana escala, pudiéndose encontrar en muchos casos la solución óptima mediante la aplicación de un algoritmo aproximado.

Finalmente queda como un problema abierto, el diseño de una posible versión paralela distribuida del algoritmo "*GSP_Ant_System*".

5 4. Conclusiones

En este trabajo de tesis, se estudió el problema GSP y los problemas STECSP-STESNP como casos particulares. De éstos últimos, se analizó con mayor profundidad el STECSP.

Primeramente, es importante destacar el esfuerzo realizado en la búsqueda bibliográfica para la obtención de artículos científicos relacionados a estos problemas y problemas afines. Se constató la escasez de artículos referentes al GSP y asimismo para el STECSP.

Se propuso un nuevo algoritmo exacto para el GSP diseñado en base a la técnica de programación Backtracking. A diferencia de los algoritmos exactos ya existentes, éste se aplica a cualquier clase de grafos. En su fase de búsqueda de la solución óptima, realiza análisis topológicos locales en la estructura del grafo, descartando familias de subgrafos que se sabe no pueden formar parte de una solución del espacio Γ_{GSP}. Si bien el algoritmo es de orden exponencial, puede ser utilizado para comparación de performance con heurísticas diseñadas para el GSP. La arquitectura del algoritmo permite además realizar el diseño de una versión paralela distribuida, de forma de poder explorar el espacio de soluciones factibles Γ_{GSP} en forma paralela y utilizando múltiples procesadores, los cuales forman parte de una Máquina Virtual.

Para el STECSP se diseñaron dos nuevos algoritmos exactos, uno basado en la técnica Backtracking y el otro en la técnica de Programación Dinámica. Además se presentó el diseño de versiones paralelo distribuidas de ambos algoritmos. La importancia de estos algoritmos de arquitectura paralela radica en la posibilidad de poder explorar el espacio de soluciones factibles Γ_{STECSP} en forma paralela y distribuida en un grupo de procesadores de una Máquina Virtual (arquitectura MIMD). En términos de diseño, la versión paralelo distribuida del algoritmo de Programación Dinámica es más compleja; esto se debe a que el modelo tiene una estructura del tipo "*tree*" mixta, donde procesos de un mismo tipo son despachados formando un árbol de invocaciones y a su vez cada uno de éstos es raíz de otro árbol formado por procesos de otro tipo que también son despachados formando un árbol

de invocaciones. Ambos diseños de algoritmos paralelos son una muy buena alternativa si se necesita encontrar soluciones óptimas de instancias de problemas del tipo STECSP, donde las redes son de mediano o gran porte. La versión paralela del algoritmo de Backtracking para el STECSP fue implementada utilizando el software PVM. Los resultados obtenidos muestran que instancias del STECSP relativamente grandes pueden ser resueltas en tiempos razonables (orden de minutos) utilizando como soporte de hardware una Máquina Virtual integrada por múltiples equipos ejecutando en forma paralela y distribuida los procesos que forman parte de la aplicación; estos procesos que operan en paralelo, buscan alguna solución factible del espacio Γ_{STECSP} descartando familias de subgrafos que se sabe no conducen a una solución factible óptima. Para ello, al igual que en el caso serial, se utilizan funciones de cota y se aplican diferentes tests a los nodos del tipo Steiner.

La heurística más general conocida hasta el momento para resolver el GSP en forma aproximada, viene dada por el algoritmo de Ajit Agrawal-Philp Klein-R. Ravi para la resolución de un problema equivalente al GSP: el *Multiterminal Network Synthesis* (MNS). Una nueva alternativa es el algoritmo aproximado presentado en este trabajo. La metaheurística *Ant System* tomada como base para el diseño del algoritmo, se adaptó en buena forma al problema GSP; los resultados numéricos obtenidos avalan esta afirmación. Si bien no se dispuso de muchos casos reales de prueba con altos requerimientos de conectividad, las corridas realizadas sobre los casos experimentales seleccionados, dieron buenos resultados luego de una selección adecuada del conjunto de valores a tomar por los parámetros *Ant System* y de un minucioso ajuste de los mismos.

Un trabajo futuro de interés, es el estudio del algoritmo "*GSP_Ant_System*" para el diseño de una implementación paralelo-distribuida del algoritmo. Esto permitiría construir soluciones factibles de una instancia del GSP, buscando en forma paralela y distribuida múltiples caminos que conecten pares de nodos terminales de forma de satisfacer los requerimientos de conexión especificados. Sería esperable que en problemas de gran porte con altos requerimientos de conexión, los tiempos de construcción de una solución factible del espacio Γ_{GSP} sean aún menores.

Asimismo, queda como planteo abierto el estudio y desarrollo de una heurística para la versión del GSP donde la matriz R indica requerimientos de caminos de nodos-disjuntos; asimismo el "*Steiner 2-node-connected subgraph problem*-STNCSP" de gran aplicación en diseño de redes de fibras ópticas metropolitanas.

6 Referencias Bibliográficas

[1]- Ajit Agrawal, Philip Klein, and R. Ravi, "*When trees collide: An aproximation algorithm for the Generalized Steiner Problem in Networks*" reporte CS-90-32 (1994), *Department of Computer Science, Brown University*.

[2]- Al Geist, Adam Beguelim, and Jack Dangarra, "*PVM – Parallel Virtual Machine, User Guide*", ISBN 0-262-57108-0, MIT Press (1992).

[3]- Y.P. Aneja, "*An integer linear programming approach to the Steiner problem in graphs*", Networks 10 (1980), páginas 167-178.

[4]- M.Baïou and A.R. Mahjoub, "*Steiner 2-edge-connected subgraph polytopes on series-parallel graphs*", (1993) *SIAM Journal on Discrete Mathematics*.

[5]- M. Baïou, "*Le problème du sous-graphe steiner 2-aréte-connexe: approche polyédrale*", Phd. Tesis, Universidad de Rennes 1 (1996).

[6]- A. Balakrishnan and N. Patel, "*Problem reduction methods and a tree generation algorithm for the Steiner network problem*".

[7]- F. Barahona and A.R. Mahjoub, "*On two-connected subgraph polytopes*", *Discrete Mathematics* 147 (1995), páginas 19-34.

[8]- J.E. Beasley, "*An algorithm for the Steiner problem in graphs*", *Networks* 14 (1984), páginas 147-159.

[9]- J.E. Beasley, "*An SST-based algorithm for the Steiner problem in graphs*", reporte técnico del Departamento de Ciencias del *Imperial College*, London (1985).

[10]- P. Berman and V. Ramaiyer, "*Improved approximation for the Steiner tree problem*", *ACM-SIAM Symposium on Discrete Algorithms* (1992), páginas 325-334.

[11]- D. Bienstock, E.F. Brickell and C.L. Monma, "*On the structure of minimum weight k-connected spanning networks*", *SIAM Journal on Discrete Mathematics* 3 (1990), páginas 320-329.

[12]- G. R. Cai and Y. G. Sun, "*The minimum augmentation of any graph to a k-edge-connected graph*", *Networks*, Vol. 19 (1989), páginas 151-172.

[13]- H. Cancela and María E. Urquhart, "*Elección de un método de Monte Carlo para el cálculo de la medida R_V de confiabilidad en redes de comunicaciones y su implementación en la herramienta HEIDI*", *Technical Report* INCO 95-05.

[14]- Charles J. Colbourn, "*The Combinatorics of Network Reliability*", *Monographs on Computer Science*, ISBN 0-19-504920-9, Oxford University Press, New York, 1987.

[15]- A. Colorini, M. Dorigo and V. Maniezzo, "*Distributed Optimization by Ant Colonies*", ECAL91- *European Conference on Artificial Life, Paris, France*, páginas 134-142.

[16]- A. Colorini, M. Dorigo, F. Maffioli, V. Maniezzo, G. Righini and M. Trubian, "*Heuristics from Nature for Hard Combinatorial Optimization Problems*", IFORS (1996), publicado por *Elsevier Science Ltd*.

[17]- S. Chopra, "*The k-edge-connected spanning subgraph polyhedron*", *SIAM Journal on Discrete Mathematics* 7 (1994), páginas 245-259.

[18]- N. Christofides and C. A. Whitlock, "*Network synthesis with connectivity constraints – a survey*", *Operational Research ´*81 (1981), páginas 705-723.

[19]- Cun-Quan Zhang, "*Integer Flows and Cycle Covers of Graphs*", *Pure and Applied Mathematics*, ISBN 0-8247-9790-6, Dekker, 1998.

[20]- M. Didi Biha and A.R. Mahjoub, "*k-edge-connected polyedra on series-parallel graphs*", *Op. Research Letters* 19 (1996), páginas 71-78.

[21]- M. Dorigo and Luca Maria Gambardella, "*Ant-Q, A Reinforcement Learning Approach to Combinatorial Optimization*", reporte técnico 95-01, IRIDIA, Universidad Libre de Bruselas.

[22]- S.E. Dreyfus and R.A. Wagner, "*The Steiner problem in graphs*", *Networks* 1 (1971), páginas 195-207.

[23]- C. El Arbi, "*Une heuristique pour le probleme de l'abre de Steiner*", *Operations Research* 12 (1978), páginas 207-212.

[24]- K. P. Eswaran and R. E. Tarjan, "*Augmentation problems*", *SIAM Journal on Computing*, Vol. 5 (1976), páginas 653-665.

[25]- Philippe Flajolet and Robert Sedgewick, "*An introduction to the analysis of algorithms*", ISBN 0-201-40009-X, Addison-Wesley (1996).

[26]- A. Frank, "*Augmenting graphs to meet edge-connectivity requirements*", reporte técnico del *Institute for Operations Research*, *University of Bonn* (1990).

[27]- G.N. Frederickson and .Ja'Ja', "*On the relationship between the biconnectivity augmentations and traveling salesman problem*", *Theoretical Computer Science* 13 (1982), páginas 189-201.

[28]- L. M. Gambardella, Éric Taillard and G. Agazzi, "*MACS-VRPTW: A Multiple Ant Colony System for vehicle vouting problems with time windows*", reporte técnico 06-99, IDSIA, Suiza (1999).

[29]- M. X. Goemans and D. J. Bertsimas, "*Survivable networks, linear programming relaxations and the parsimonious property*", *Mathematical Programming* 60 (1993).

[30]- M. Grötshel and C. Monma, "*Integer polyhedra arising from certain network design problems with connectivity constraints*", *SIAM Journal on Discrete Mathematics* Vol. 3, No. 4 (1990), páginas 502-523.

[31]- M. Grötshel, C.L. Monma, and M. Stoer, "*Polyhedral approaches to network survivability*", *Reliability of Computer and Communication Networks*, Vol. 5, *series Discrete Mathematics and Computer Science*, 121-141. AMS/ACM, (1991).

[32]- M. Grötshel, C.L. Monma, and M. Stoer, "*Polyhedral and computational investigations for designing communication networks with high survivability requirements*", *Operations Researh* 43 (1995).

[33]- S.L. Hakimi, "*Steiner's Problem in graphs and its implications*", *Networks* 1 (1971), páginas 113-133.

[34]- Harary, "*Graph Theory*", *Mathematics*, ISBN 0-201-02787-9, Addison-Wesley (1972).

[35]- A. Jain, "Probabilistic analysis of an LP relaxation bound for the Steiner problem in networks", Networks, Vol. 19 (1989), páginas 793-801.

[36]- N. Maculan, "*The Steiner problem in graphs*", *Ann. Discrete Math*. 31 (1987), paginas 235-257.

[37]- A.R. Mahjoub, "*Two-edge-connected spanning subgraphs and polyhedra*", *Mathematical Programming* 64 (1994), páginas 199-208.

[38]- S.L. Martins, P.M. Pardalos, M. Resende, and C. Ribeiro, "*Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs*", artículo no publicado.

[39]- C.L. Monma, B.S. Munson, and W.R. Pulleyblank, "*Minimum-weight two connected spanning networks*", *Mathematical Programming* 46 (1990), páginas 153-171.

[40]- C. L. Monma and J. B. Sidney, "*Sequencing with series-parallel precedence constraints*", *Mathematics of Operations Research* 4 (1979), páginas 215-224.

[41]- D. Naor, D. Gusfield, and Ch. Martel, "*A fast algorithm for optimally increasing the edge-connectivity*", *Proceedings of Foundation of Computer Science ´* 90, páginas 698-707.

[42]- J. Plesnik, "*A bound for the Steiner tree problem in graphs*", *Math. Slovaca* 31 (1981).

[43]- A. Prodon, T.M. Liebling, and H. Groflin, "*Steiner's problem on two-trees*", reporte técnico RO 850315, Departamento de Matemáticas de *Ecole Polytechnique Federale de Lausanne*, Suiza (1985).

[44]- V.J. Rayward and Smith, "*The computation of nearly minimal Steiner trees in graphs*", *Mathematical Programming* 14 (1983), páginas 15-23.

[45]- Reinhard Diestel, "*Graph Theory*", *Graduate Texts in Mathematics*, ISBN 0-387-98210-9, Springer (1998).

[46]- A. Rosenthal and A. Goldner, "*Smallest augmentation to biconnect a graph*", *SIAM Journal on Computing*, Vol. 46 (1977), páginas 55-66.

[47]- Sara Baase, "*Computer Algorithms, Introduction to Design and Analysis*", primera edición, ISBN 0-201-06035-3/06035.

[48]- M.L. Shore, L.R. Foulds, and P.B. Gibbons, "*An algorithm for the Steiner problem in graphs*", *Networks* 12 (1982), páginas 323-333.

[49]- R. Sridhar and R. Chandrasekaran, "*Integer solution to synthesis of communication network*", reporte técnico de la Universidad de Texas, Dallas (1989).

[50]- M. Stoer, "*Design of Survivable Networks*", *Lecture Notes in Mathematics*, ISBN 3-540-56271-0, ISBN 0-387-56271-0, Springer-Verlag (1996).

[51]- Eric. D. Taillard, "*Ant Systems*", reporte técnico 05-99, IDSIA, Suiza (1999).

[52]- H. Takahashi and A. Matsuyama, "*An aproximate solution for the Steiner problem in graphs*", *Mathematical Programming* 24 (1980), páginas 573-577.

[53]- K. Takamizawa, T. Nishizeki, and N. Saito, "*Linear-time computability of combinatorial problems on series-parallel graphs*", *Journal of the Association for Computing Machinery*, Vol. 29 (1988), páginas 623-641.

[54]- A.S. Tanenbaum, "*Computer Networks*", ISBN 0-13-162959-X, Prentice Hall (1991).

[55]- S. Ueno, Y. Kajitani, and H. Wada, "*Minimum augmentation of a tree to a k-edge-connected graph*", *Networks*, Vol. 18 (1988), páginas 19-25.

[56]- J. A. Wald and C. J. Colbourn, "*Steiner trees in outerplanar graphs*", *Congressus Numeratum* 36 (1983), páginas 159-167.

[57]- J. A. Wald and C. J. Colbourn, "*Steiner trees, partial 2-trees, and minimum IFI networks*", *Networks*, Vol. 13 (1983), páginas 159-167.

[58]- T. Watanabe and A. Nakamura, "*Edge-connectivity augmentation problems*", *Computer and System Sciences*, Vol. 35 (1987), páginas 96-144.

[59]- P. Winter, "*Generalized Steiner problem in series-parallel networks*", *Journal of Algorithms* 7 (1986), páginas 549-566.

[60]- P. Winter, "*Steiner problem in networks: A survey*", *Networks* 17 (1987), páginas 129-167.

[61]- R.T. Wong, "*A dual ascent approach for Steiner tree problems on a directed graph*", *Mathematical Programming* 28 (1984), páginas 271-287.

[62]- A. Z. Zelikovsky, "*The 11/6-aproximation algorithm for the Steiner problem on networks*", *Algorithmica* Vol. 9 (1994), páginas 463-470.

7 Apéndice 1- Definiciones Básicas y Teoremas Auxiliares

Definición 1

Dado un grafo $G=(V,E)$, dos nodos $i,j\in V$ se dicen localmente k-arista-conexos (k-nodos-conexos), si existe en G al menos k caminos de aristas disjuntas (caminos de nodos disjuntos) que los unen.

Definición 2

Un grafo $G=(V,E)$ es llamado k-arista-conexo (k-nodo-conexo), $1\le k\le n-1$ con $n=|V|$, si para todo par de nodos $i,j\in V$, hay al menos k caminos de aristas disjuntas (caminos de nodos disjuntos) que los unen.

Definición 3

Dado un grafo G, se denota como $G_s(N,A)$ al subgrafo de G inducido por el conjunto de nodos $N\subseteq V$ y el conjunto de aristas $A\subseteq E$ con extremos en N.

Definición 4

Un grafo G se dice que es 2-arista-conexo si para todos par de nodos de G, hay al menos 2 caminos de aristas disjuntas que los unen.

Definición 5

Un grafo $G=(V,T\cup C)$ es un grafo de Halin si T es un árbol que no contiene nodos de grado dos y C es un ciclo formado por nodos adyacentes a nodos de T.

Definición 6

Un grafo es outerplanar si esta formado por un ciclo con cuerdas que no se cortan (una cuerda es una arista entre dos nodos no adyacentes del ciclo).

7.1.1.1 Definición 7

Dado un grafo $G = (V, E)$ se define un camino $p = v_0 ... v_k$ sobre el grafo de manera que: todos los nodos son distintos, a los nodos v_0 y v_k se les denomina extremos, $(v_i, v_{i+1}) \in E, \forall i \in 1..k-1$, y a los nodos $v_1, .., v_{k-1}$ puntos interiores de p.

7.1.1.2

7.1.1.3 Definición 8

Sea $G = (V, E)$ un grafo y dos subconjuntos de nodos $A, B \subseteq V$; a un camino $p = v_0 ... v_k$ se dice que es un $A - B\ path$, si $Nodos(p) \cap A = \{v_0\}$ y $Nodos(p) \cap B = \{v_k\}$. A un $\{a\} - B\ path$ con $a \in V$, se lo denota como $a - B\ path$.

7.1.1.4 Definición 9

Dos o más caminos se dicen independientes, si ninguno de ellos contiene un vértice interior del otro.

<u>Definición 10</u>

Dos $a - b\ path\ s$ se dicen independientes, si tienen solamente los vértices a y b en común.

7.1.1.5 Definición 11

Dado un grafo $G = (V, E)$ y un conjunto de nodos $H \subseteq V$, se dice que un camino p es un $Hpath$ si es no trivial y esta conectado al grafo G exactamente por sus extremos, los cuales pertenecen al conjunto H. Una arista con extremos en H es un $Hpath$ de largo 1. Eventualmente, se puede admitir que los extremos de un $Hpath$ corresponden a un único nodo de H.

7.1.1.6 Definición 12

Dado un grafo $G=(V,E)$ y un conjunto de nodos $H \subseteq V$, se dice que dos $Hpaths$ son disjuntos, si sus conjuntos de nodos interiores son disjuntos.

Proposición 13

Un grafo es 2-nodo-conexo si y solo si éste puede ser construido a partir de un ciclo por sucesivas agregaciones de $Hpaths$ al grafo H ya construido [45].

Proposición 14

Un grafo es 2-arista-conexo si y solo si éste puede ser construido a partir de un ciclo por sucesivas agregaciones de $Hpaths$ al grafo H ya construido, se permiten aquí $Hpaths$ con un único extremo [45].

Definición 15

Un conjunto de $v-B$ paths es llamado un $v-B$ fan si cualquiera dos de los caminos tienen solo al nodo v en común.

Teorema 16 (Menger)

Sea $G=(V,E)$ un grafo y dos subconjuntos de nodos $A,B \subseteq V$. Entonces el mínimo número de nodos separando A y B en G es igual a el máximo número de caminos disjuntos de A hacia B en G.

Corolario 17 (Menger)

Para $B \subseteq V$ y $v \in (V \setminus B)$, el mínimo número de nodos diferentes de v separando v de B en G es igual a el máximo número de caminos formando un $v-B$ fan en G.

Corolario 18 (Menger)

Sean $u,v \in V$ dos nodos distintos de G.

1. *Si $(u,v) \in E$, entonces el mínimo número de nodos distintos de u y v separando u de v en G es igual al número máximo de caminos independientes de u a v .*
2. *El mínimo número de aristas separando u y v en G es igual a el número máximo de caminos $u-v$ de aristas disjuntas en G .*

Teorema 19 (Versión Global del Teorema de Menger)

1. *Un grafo es k-conexo si y solo si contiene k caminos independientes entre dos nodos cualesquiera.*
2. *Un grafo es k-arista-conexo si y solo si contiene k caminos de aristas disjuntas entre dos nodos cualesquiera.*

7.1.1.7

7.1.2

8 Apéndice 2 - Steiner Problem in Networks (SPN)

8.1 A.2.1 Introducción

El problema de encontrar una subred que cubra un subconjunto de nodos de una red dada, con costo mínimo, se conoce como Problema de Steiner en Redes (SPN). Las aplicaciones de este problema son heterogéneas, en especial diversos problemas de diseños de redes de comunicaciones pueden ser modelados como un SPN. Existe una gran diversidad de algoritmos exactos y heurísticas que resuelven el problema SPN.

Este capítulo pretende brindar un resumen del estado del arte del SPN, presentándose los algoritmos exactos y aproximados más conocidos que resuelven el SPN. En base a la literatura relevada, primero se presenta formalmente el SPN, se brinda un estudio de las posibles reducciones que se le puede hacer a un grafo de manera de simplificar el problema, además de los casos mas simples de instancias particulares del SPN. Luego se presentan algunos de los algoritmos exactos publicados más conocidos hasta el momento, así como también algunas de las heurísticas más conocidas para la resolución en forma aproximada del SPN. Por último, se mencionan algoritmos de resolución del SPN diseñados a medida para clases particulares de grafos, además de otras heurísticas publicados donde el radio de performance de éstas es independiente del número de nodos terminales n_T.

8.2 A.2.2 Formulación del Problema

Sea $G = (V, E)$ un grafo no dirigido conexo, con $n = |V|$, c una matriz de costos asociada a las aristas, y $T \subseteq V$, con $n_T = |T|$, un conjunto de nodos a los cuales se les denomina terminales. Se pretende encontrar un subgrafo G_T de G tal que entre todo par de nodos de T existe un camino en G_T que los une, y además el costo del subgrafo G_T es mínimo.

A los nodos de $(V \setminus T)$ se le denomina nodos de Steiner.

8.3 A.2.3 Casos Especiales de SPN y Reducciones

Las siguientes son una serie de reducciones que se le puede hacer al grafo antes de aplicarle algún algoritmo para la resolución del SPN.

1. Existen costos negativos en aristas. Sea $F = \{(i, j) \in E / c_{ij} \leq 0\}$. Considérese el grafo $\overline{G}$ resultante de contraer en G el conjunto de aristas F . Sea $\overline{G_T}$ la solución al SPN con grafo $\overline{G}$, entonces la solución al SPN con grafo G se obtiene agregando a $\overline{G_T}$ las aristas de F . Si todos los costos de las aristas son positivos, la solución es un árbol que cubre al conjunto de terminales T . A este problema se lo denomina *Steiner Minimal Tree* (SMT) para T en G .
2. Si $|T| = 1$, G_T consiste en un solo nodo.
3. Si $|T| = n$. El SPN se reduce a encontrar el árbol de cubrimiento mínimo de G (problema *Minimal Spanning Tree* - MST). Se pueden resolver aplicando los algoritmos de orden polinomial Kruskal o Prim.
4. Si $|T| = 2$. El SPN se reduce a encontrar el camino menos costoso entre dos nodos de un grafo , se pueden aplicar en este caso los algoritmos de orden polinomial de Bellman o Dijkstra.

Para simplificar el análisis de los algoritmos, se supondrá en el resto de este capítulo que el grafo G es no dirigido conexo y los costos de la matriz c son estrictamente positivos.
Bajo estas suposiciones, y en base a examinar localmente las propiedades de G , se pueden realizar las siguientes simplificaciones al grafo G .

1. Si G tiene un nodo terminal i de grado 1, entonces la arista (i, j) incidente en él es parte de la solución del SMT, y puede ser eliminada de G , tomándose el nodo j como nodo terminal en el problema reducido.

2. Si G tiene un nodo de Steiner de grado 1, entonces dicho nodo y la arista incidente en él pueden ser eliminados de G.
3. Si G tiene un nodo de Steiner k con grado 2, entonces las dos aristas (k,i) y (k,j) incidentes en k pueden ser reemplazadas por una arista (i,j) de costo $c_{ij} = c_{ik} + c_{kj}$. Si se obtiene como resultado dos aristas paralelas, la de mayor costo puede ser eliminada de G.
4. Si G tiene dos nodos terminales adyacentes $i, j \in T$, y hay un nodo terminal k, $k \neq i$, $k \neq j$, tal que $c_{ij} \geq d_{ik}$ y $c_{ij} \geq d_{jk}$ (donde se define a d_{uv} como el menor costo de un camino que une los nodos u y v en G), entonces (i,j) puede ser eliminada de G.
5. Si G tiene un nodo terminal i, j es un nodo vecino a i, k es un segundo-vecino, y $c_{ij} + min\{d_{jp} | p \in T, p \neq i\} \leq c_{ik}$ **(a)**, entonces la arista (i,j) es parte de la solución del SMT. El grafo G puede ser contraído a través de (i,j) (si j es un nodo terminal, entonces $min\{d_{jp}\} = 0$ y **(a)** siempre se cumple).
6. Si G tiene una arista (i,j) tal que $c_{ij} > d_{ij}$, entonces (i,j) puede ser eliminada de G. En caso que $c_{ij} = d_{ij}$ y haya un camino de costo d_{ij} de i a j no conteniendo a (i,j), entonces (i,j) puede ser eliminada de G.

Estas simplificaciones se basan en información local al grafo, y en general suelen tener un fuerte impacto en los tiempos de ejecución de algoritmos a medida para el SPN. Balakrishnan-Patel brindan en [6] un análisis del impacto de las reducciones en el tamaño de las instancias de un problema.

8.4 A.2.4 Algoritmos Exactos para el SPN

Existe una gran variedad de algoritmos exactos para el SPN. Seguidamente se presentan algunos de los algoritmos más conocidos.

8.4.1 A.2.4.1 Spanning Tree Enumeration Algorithm (STEA)

El algoritmo de enumeración de árboles de cubrimiento STEA fue originalmente propuesto por Hakimi. Él muestra que si se puede resolver el SPN, entonces se pueden encontrar "varios" árboles de Steiner en el grafo G que cubren el conjunto de nodos terminales T. Se brinda el siguiente algoritmo *straightforward*: la solución óptima G_T al SPN puede ser encontrada mediante la enumeración de MST's de subgrafos de G inducidos por un conjunto de nodos $W / T \subseteq W \subseteq V$.

El algoritmo STEA fue mejorado por Lawler, y luego por Balakrishnan-Patel. La complejidad del algoritmo es del orden $O(n_T^2 \cdot 2^{n-n_T} + n^3)$. En [33] se encuentra una descripción completa del algoritmo STEA de Hakimi.

8.4.2 A.2.4.2 Topology Enumeration Algorithm (TEA)

Otro algoritmo presentado por Hakimi, es una extensión del algoritmo presentado por Melzak para el problema de Steiner con métrica Euclidiana. El algoritmo se basa en las siguientes premisas.

Si G_T es la solución óptima del SPN, G_T debe tener al menos dos nodos terminales $i, j \in T$ satisfaciendo alguna de las siguientes condiciones:

1. $g(i) = g(j) = 1$ ($g(v)$ es el grado de v en G), y el camino p_{ij} que une i con j en G_T contiene solamente nodos del tipo Steiner; exactamente uno de ellos tiene grado mayor a 2 en G_T. Consecuentemente, el resto de los nodos de Steiner (si hay) tienen grado 2 en G_T.
2. $g(i) = 1$ o $g(j) = 1$, y el camino p_{ij} que une i con j en G_T contiene solamente nodos del tipo Steiner (si hay); todos ellos de grado 2.

Si i y j satisfacen (1) y (2), entonces se les llama pseudoadyacentes en G_T.

El algoritmo TEA determina recursivamente el costo mínimo de un árbol que cubre a los nodos de T en G con i y j siendo nodos pseudoadyacentes, o indica que no existe un árbol en tales condiciones. No hay forma de saber de antemano cuales nodos de T son nodos pseudoadyacentes en la solución

óptima G_T del SPN. Se deben considerar todos las posibles elecciones de i y j, construyendo el árbol de mínimo costo que cubre T con i y j siendo nodos pseudoadyacentes en G_T (o deducir que no existe tal G_T), y seleccionar G_T como el de costo mínimo.

Los detalles de diseño del algoritmo TEA se pueden ver en [33].

8.4.3 A.2.4.3 Dynamic Programming Algorithm (DPA)

Un algoritmo basado en Progamación Dinámica que resuelve el SPN fue propuesto por Dreyfus-Wagner e independientemente por Levin una versión menos eficiente.

El algoritmo se basa en las siguientes consideraciones.

Supóngase que i es un nodo de la solución óptima G_T (no necesariamente un nodo terminal) en el cual inciden más de una arista. Se descompone a G_T en dos componentes G_T^0 y G_T^1 de la siguiente manera: se reemplaza al nodo i por dos nodos i^0 e i^1 desconectados uno del otro, de las aristas incidentes en i en G_T, se elige arbitrariamente algunas de ellas haciéndolas incidentes en i^0 y el resto en i^1. Sea T^0 el conjunto de nodos de T presentes en G_T^0 y T^1 el conjunto de nodos de T presentes en G_T^1, entonces se cumple que G_T^0 es el SMT para $T^0 \cup \{i\}$ en G y G_T^1 es el SMT para $T^1 \cup \{i\}$ en G.

La propiedad de descomposición óptima en la cual se basa el DPA es: soluciones óptimas de pequeños problemas son encontradas y almacenadas para ser usadas en la resolución de subproblemas más grandes (los subproblemas mas chicos no se resuelven más de una vez). Soluciones no óptimas para subproblemas más pequeños son descartadas; éstas no se consideran cuando se resuelven subproblemas más grandes.

Aplicando esta propiedad, se construye el algoritmo de Programación Dinámica de la siguiente forma.

Sean $Y \subseteq T, Y \neq \varnothing$ e $i \in V \setminus Y$; se define $G_{Y \cup \{i\}}$ como el SMT para $Y \cup \{i\}$. Sea $c(G_i(Y))$ el mínimo costo de la unión de dos árboles, uno cubriendo el nodo i y un subconjunto no vacío $X \subset Y$, y el otro cubriendo el nodo i y el complemento $(Y \setminus X)$. $G_i(Y)$ es la unión de dos SMT's, uno cubriendo $X \cup \{i\}$ y el otro cubriendo $(Y \setminus X) \cup \{i\}$, donde $\varnothing \subset X \subset Y$. Al minimizar sobre toda los valores de X resulta: $c(G_i(Y)) = Min_{\forall \varnothing \subset X \subset Y} \{c(G_{X \cup \{i\}}) + c(G_{(Y \setminus X) \cup \{i\}})\}$.

Además $G_i(Y)$ es la unión de $G_{X \cup \{i\}}$ y $G_{(Y \setminus X) \cup \{i\}}$ para el cual $c(G_i(Y))$ fue obtenido.

Se desea obtener una relación funcional para $c(G_i(Y))$, con $|Y| > 1$; para ello se toman en cuenta las siguientes propiedades sobre la configuración de $G_{Y \cup \{i\}}$:

1. si $g(i) > 1$, entonces $G_{Y \cup \{i\}} = G_i(Y)$,
2. si $g(i) = 1$, entonces $G_{Y \cup \{i\}}$ debe ser la unión del camino más corto de i a algún nodo $k \in (V \setminus T)$ de grado al menos 3 con $G_k(Y)$, o la unión del camino más corto de i a algún nodo $k \in Y$ con G_Y. De esta forma $G_{Y \cup \{i\}}$ es un SMT y el nodo k debe ser elegido de forma tal que el costo de la unión sea mínimo. Por tanto:

$c(G_{Y \cup \{i\}}) = Min\{c(G_i(T)), Min_{k \in V}\{d_{ik} + c(G_k(Y))\}\}$.

Por definición $c(G_{\{t\} \cup \{i\}}) = d_{ti} \quad \forall t \in T$ e $i \in V$. $G_i(Y)$ puede ser determinado para cualquier subconjunto Y de dos nodos de T y cualquier $i \notin Y$. Entonces $G_{Y \cup \{i\}}$ puede ser determinado para cualquier subconjunto Y de dos nodos de T y cualquier $i \notin Y$.

En particular, se pueden obtener todos los SMT´s con exactamente tres nodos terminales; continuando de esta manera, se puede obtener el SMT para todo los nodos de T en G.

La complejidad del algoritmo DPA es del orden $O(n \cdot 3^{n_T} + n^2 \cdot 2^{n_T} + n^3)$. El orden es polinomial en el número de nodos y exponencial en la cantidad de nodos terminales.

Variantes del algoritmo DPA y un análisis más detallado, puede ser encontrado en [22].

8.4.4 **A.2.4.4 Set Covering Algorithm (SCA)**

Aneja resuelve el SPN basado en un problema de cubrimiento de conjuntos. La formulación del problema de cubrimiento tiene la desventaja que el número de restricciones crece exponencialmente con el tamaño de la entrada.

Considérese una partición $\{W, \overline{W}\}$ de V tal que $T \cap W \neq \varnothing$ y $T \cap \overline{W} \neq \varnothing$. Supóngase que los conjuntos de corte para toda partición son enumerados $C_1,...,C_q$, además supóngase que las aristas de G son también enumeradas $e_1, e_2 ..., e_m$ (con $m = |E|$). Se define una matriz $M = \{M_{ij}\}$, donde:

$$M_{ij} = \begin{cases} 1 & si\, e_j \in C_i \\ 0 & sino \end{cases}, \ \forall i \in (1..q), \forall j \in (1..m).$$

El problema de cubrimiento de conjunto (SCP) se formula de la siguiente manera:

$$Min \sum_{j=1}^{m} c_j \cdot x_j$$

sujeto a:

$$\sum_{j=1}^{m} M_{ij} \cdot x_j \geq 1, \qquad i = 1,2...,q$$

$$x_j \in \{0,1\}, \qquad j = 1,2,...,m$$

Sea $O = (o_1, o_2, ..., o_m)$ una solución óptima del SCP. Si se considera G_O el subgrafo de G inducido por el conjunto de aristas $\{e_j \in E | o_j = 1\}$, entonces G_O es conexo y cubre T; sino alguna restricción correspondiente a algún conjunto de corte $C_i, i \in (1..q)$, debe ser violada. Se cumple además que cualquier árbol que cubra el conjunto de nodos terminales T en G es una

solución factible del SCP. Entonces como $c_j > 0$ para todo j, resulta $G_O = G_T$.

El algoritmo que propone Aneja para resolver el problema SCP es una modificación del algoritmo que proponen Bellmore-Ratliff para resolver el problema general SCP.

Un análisis detallado del algoritmo SCA de Aneja se encuentra en [3].

8.4.5 A.2.4.5 Implicit Enumeration Algorithm (IEA)

Shore propone un algoritmo de enumeración implícita que resuelve el SPN. Yang-Wing también proponen un algoritmo similar pero menos eficiente que el de Shore.

El algoritmo de Shore se basa en lo siguiente.

Sea F_0 el conjunto de todas las soluciones factibles del SPN (todos los árboles que cubren T en G). El algoritmo de Shore particiona en forma sistemática el conjunto F_0 en subconjuntos mas pequeños. Cada subconjunto es analizado mediante el uso de cotas superiores e inferiores, si puede contener la solución óptima factible.

Cada subconjunto F_i de soluciones factibles es caracterizado por tener un conjunto de aristas $IN_i \subseteq E$ y excluir un conjunto de aristas $OUT_i \subseteq E$. El subconjuto F_i es analizado en profundidad, este es particionado en dos subconjuntos F_j y F_k con $IN_j = IN_i \cup \{e\}$, $OUT_j = OUT_i$, $IN_k = IN_i$, $OUT_k = OUT_i \cup \{e\}$ para toda arista $e \in E \setminus (IN_i \cup OUT_i)$. El subconjunto F_j es examinado primero por sucesivas particiones de F_j. Cuando la solución óptima factible para F_j es encontrada o se determina que la solución óptima factible para F_0 no puede estar en F_j, entonces se procede a examinar F_k de la misma manera, luego el algoritmo retorna a F_i completándose el análisis de F_i. El algoritmo utiliza una estrategia de *Backtracking* para analizar los subconjuntos de soluciones factibles; el algoritmo finaliza cuando se retorna a F_0 luego de haber analizado todas las posibles particiones. La

solución factible de menor costo encontrada en la búsqueda *Backtracking* es la solución óptima G_T.

Una descripción detallada del algoritmo *Backtracking* de Shore y sus características se encuentra en [48].

8.4.6 A.2.4.6 Lagrangean Relaxation Algortihm (LRA)

Beasley propone otro algoritmo de enumeración para resolver el SPN. En el algoritmo, cada subconjunto de soluciones factibles F_i es caracterizado por tener incluido un conjunto IN_i de nodos de Steiner, y tener excluido un conjunto OUT_i de nodos de Steiner. Sea S_i el subgrafo de G resultante de eliminar los nodos de Steiner en OUT_i, y de la aplicación de las reducciones A.2.3 al subgrafo resultante (pueden quedar nodos de Steiner en IN_i como nodos terminales). Sea T_i el conjunto de nodos terminales en S_i.

Al igual que en A.2.4.5, el determinar la solución óptima en el subconjunto F_i (si esta existe) es equivalente al problema de determinar el SMT G_i para T_i en S_i.

La cota superior $\overline{b}_i$ para F_i puede ser obtenida sumando los costos de las aristas identificadas durante el proceso de contracción como pertenecientes al SMT para $T \cup IN_i$ en $G \setminus OUT_i$, al costo de un árbol que cubre T_i en S_i obtenido por alguna de las heurísticas que se describen en A.2.5.

Para obtener la cota inferior $\underline{b}_i$ para F_i, Beasley propone tres alternativas, todas ellas basadas en relajaciones Lagrangeanas del SPN para T_i en S_i. Las tres alternativas son formuladas como varios problemas de Programación Lineal Entera Binaria (0-1LP) a resolver.

El análisis de éstas alternativas, así como la descripción del algoritmo se encuentra en [8,9].

Wong propone en [61] una forma alternativa para determinar la cota inferior $\underline{b}_i$ para F_i, reformulando los problemas 0-1LP utilizados por Breasley.

Prodon también presenta en [43] una forma alternativa de encontrar la cota inferior para F_i.

8.5 A.2.5 Heurísticas para el SPN

No existe un algoritmo de tiempo polinomial para el SPN. Karp probó que el problema es NP-Completo [60]. Además, es NP-Completo en los casos en los cuales los costos de las aristas son uniformes, el grafo G es planar, o G es bipartito con pares de vértices de T no adyacentes y pares de vértices de $(V \setminus T)$ no adyacentes.

Dada la complejidad del problema SPN, asiéndose casi intratable su resolución en forma exacta, y debido a que muchos problemas reales se modelan como un problema del tipo SPN, es que surge la necesidad de buscar soluciones alternativas, desarrollando heurísticas que permitan encontrar soluciones fiables en tiempos "razonables". Es así, que las heurísticas desarrolladas para el SPN buscan encontrar un árbol que cubra los nodos terminales T con el menor costo "posible". Seguidamente se presentan algunas de las heurísticas más eficientes conocidas hasta el momento.

8.5.1 A.2.5.1 Minimum Cost Paths Heuristic (MPH)

La heurística MPH es sugerida por Takahashi-Matsuyama para resolver el problema SPN. La misma se basa en ir construyendo un árbol G_k conteniendo un subconjunto $T_k \subset T$, y hacer una elección apropiada de un vértice $i \in T / i \notin T_k$ para agregarlo al árbol, junto con el camino más corto que conecta G_k con i.

El algoritmo opera de la siguiente manera.

Paso 1: se comienza con un subárbol G_1 de G conteniendo un nodo simple. Sea $i \in T$ un nodo terminal elegido arbitrariamente. Inicialmente se toma $k = 1$ y $T_k = \{i\}$.

Paso 2: se determina un nodo terminal $i \in T \setminus T_k$ "cercano" a G_k. Luego se construye un árbol G_{k+1} agregando el camino de menor costo que conecta el nodo i con G_k.

Paso 3: si $k < n_T$, entonces ir a paso 2. Si $k = n_T$, entonces $G_{MPH} = G_{n_T}$ es la solución.

STOP.

La heurística tiene un tiempo de ejecución de orden $O(n^2 \cdot n_T)$, ya que los caminos de costo mínimo pueden ser determinados por ejemplo aplicando el algoritmo de Dijkstra en a lo sumo un tiempo del orden $O(n^2)$. Takahashi-Matsuyama probaron que para un grafo arbitrario G y un conjunto arbitrario T de nodos terminales, el radio de error del peor caso entre el costo de G_{MPH} y el costo de la solución ópitma G_T esta acotado por $2 - 2/n_T$, entonces: $c(G_{MPH}) / c(G_T) \leq 2 - 2/n_T$ (*). La cota $2 - 2/n_T$ es rígida en el sentido que para todo n, y para todo $1 \leq n_T \leq n-1$, existe un grafo $G = (V, E)$, y un conjunto de nodos terminales $T \subseteq V$, donde $n = |V|$ y $n_T = |T|$, tal que se cumple la igualdad en (*).

Existen otras dos heurísticas propuestas por Takahashi-Matsuyama para el SPN. Las mismas se pueden ver en [52] en detalle.

8.5.2 A.2.5.2 Distance Network Heuristic (DNH)

Kou-Plesnik-El Arbi [60,42,23] proponen una heurística ligeramente diferente a la MPH. Se define a $K_T = (T, E, d)$ como el grafo de distancias inducida por los nodos de T. La heurística DNH es la siguiente.

Paso 1: Se determina el MST A para K_T.

Paso 2: Se construye un subgrafo S_T de G reemplazando cada arista en A por el correspondiente camino de costo mínimo en G (si hay varios caminos con el mismo costo, se elige uno en forma arbitraria).

Paso 3: Se determina el MST A_T para S_T.

Paso 4: Se eliminan en A_T todos los nodos de Steiner de grado 1 (uno en cada paso). El árbol resultante, denotado por A_{DNH}, es la solución.
STOP.

La complejidad del algoritmo DNH en el peor caso es del orden $O(n_T \cdot n^2)$. El radio de error del peor caso $c(A_{DNH})/c(G_T)$ esta rígidamente acotado por $2-2/n_T$.
En general el DNH y el MPH tienen las mismas características, pero para algunos casos especiales el MPH obtiene una mejor solución.
Existen algunas variantes del DNH, propuestas por Plesnik e independientemente por Sullivan, que mejoran la performance del mismo. Una descripción detallada de la heurísitica DNH se encuentra en [60].

8.5.3 A.2.5.3 Average Distance Heuristic (ADH)

Rayward-Smith, proponen una heurística diferente. En cada iteración, la heurística examina una lista $L=\{G_1,G_2,....,G_k\}$ de árboles, los cuales son subárboles del árbol final. Inicialmente L esta integrada por nodos terminales. Usando una función $f:V\to R$ dos subárboles de L son seleccionados y conectados por el camino de costo mínimo en G. En cada iteración la cardinalidad de L se reduce en uno. Luego de n_T iteraciones, L contiene un árbol que cubre todos los nodos terminales de T.
Rayward-Smith definen a la función f de la siguiente manera. Para cada nodo $i\in V$, se reetiquetan los árboles de L de manera que estén en orden no decreciente de distancia al nodo i. Se asume sin perder generalidad que $d(i,G_1)\le d(i,G_2)\le ...\le d(i,G_k)$. Entonces,

$$f(i)=\begin{cases} d(i,G_2), & si\ i\in G_1 \\ min_{2\le r\le k}\left\{\sum_{j=1}^{r} d(i,G_j)/(r-1)\right\}, & sino \end{cases}$$

Notar que si $i\in G_j, 2\le j\le k$, implica que $i\in G_1$.

Dado $f(i)$ para todo $i \in V$, se elige $\bar{i} \in V$ satisfaciendo $f(\bar{i}) \leq f(i)$ para todo $i \in V$. Nuevamente se asume (sin perder generalidad) que $d(\bar{i}, G_1) \leq d(\bar{i}, G_2) \leq ... \leq d(\bar{i}, G_k)$. Si $\bar{i} \in G_1$, entonces se unen G_1 con G_2 por el camino de mínimo costo en G. En cambio, si $\bar{i} \notin G_1$, entonces $\bar{i}$ no es un nodo de *Steiner* en cualquier $G_j, 1 \leq j \leq k$. Se une $\bar{i}$ con G_1 y G_2 respectivamente, por el camino de menor costo en G.

La complejidad del algoritmo ADH en el peor caso es $O(n^3)$.

El análisis detallado de la heurística ADH se encuentra en [44].

8.5.4 A.2.5.4 Contraction Heuristic (CH)

Plesnick propone una heurística basada en una idea más general de contracción. La descripción en detalle de la heurística se puede ver en [60].

La complejidad de la heurística CH en el peor caso es $O(n^3)$. Plesnick probó que para un grafo arbitrario G y un conjunto de nodos terminales $T \subset V$, el radio de error en el peor caso $c(G_{CH}) / c(G_T)$ tiene como cota rígida a $2 - 2/n_T$.

8.5.5 A.2.5.5 Set Covering Heuristic (SCH)

Aneja [3] propone como heurística la SCH para obtener una buena solución factible.

La heurística opera de la siguiente forma:

Paso 1: se determina la solución óptima U al problema LSCP (relajación lineal del SCP). Sea J el correspondiente cubrimiento.

Paso 2: entre todos los elementos redundantes de J, se elimina el que tiene menor valor de u. Se repite este procedimiento hasta que no haya mas elementos redundantes en J.

Paso 3: sea G_{SCH} el árbol que contiene las aristas indicadas en J. Se devuelve G_{SCH} como solución.

STOP.

8.6 A.2.6 Casos Particulares y Otros Algoritmos

Si G es un árbol, la solución es trivial y puede ser obtenida en tiempo lineal. Existen clases de grafos planares en los cuales el SPN se puede resolver con orden polinomial. Para grafos serie-paralelos, Wald-Colbourn [57] proponen un algoritmo de orden lineal que resuelve el SPN; también desarrollaron un algoritmo similar para grafos outerplanares [56]. Prodon [43] propone una versión diferente de algoritmo que también resuelve el SPN con orden lineal para grafos serie-paralelos. Winter [60] propone un algoritmo de orden lineal que resuelve el SPN para los casos en los cuales el grafo G es un grafo de Halin. Markowsky-Berman e independientemente Wong [61], proponen otras heurísticas que también poseen un radio de performance en el peor caso de $2-2/n_T$. Un algoritmo presentado por Plesnik-Sullivan mejora un poco este radio de performance. Martins-Pardalos-Resende-Ribeiro [38] aplican la metodología GRASP para encontrar "buenas soluciones" al SPN; utilizan en la fase de construcción de la solución la heurística de Melhorn, la cual construye una solución factible con un costo computacional de orden $O(|E|+|V|\cdot\log|V|)$. Zelikovsky [62] diseñó un algoritmo aproximado para el SPN con un radio de performance de $11/6$. Berman-Ramaiyer [10] mejoraron este radio con un algoritmo de radio de performance $16/9$.

> En experimentos computacionales, en general, todas las heurísticas mencionadas tienen performance considerablemente mejor que la dada por la cota. Jain [35] presentó una formulación del SPN como un problema de Programación Entera, y demostró que para dos distribuciones aleatorias de la matriz de costos asociada a las aristas, el valor dado por el modelo de Programación Entera difiere drásticamente de su relación fraccional (el cociente de los costos de las soluciones dadas por la heurística).

Para problemas de diseño de redes de alta confibilidad, las soluciones dadas por un modelo del tipo SPN, minimizan el costo total de conexión entre nodos terminales, pero tienen baja confiabilidad si se asume que pueden producirse fallas en las componentes de la red. Si ocurre una falla en un link

o nodo, el sistema deja de estar en un estado operativo, lo cual puede ser catastrófico en ciertas aplicaciones.

9 Apéndice 3 - Minimum-Weight k-Connected Spanning Networks

9.1 A.3.1 Introducción

Cuando se diseñan redes de comunicación o transporte, es frecuentemente importante que las mismas sean resistentes a fallas y que tengan el menor costo posible. Una forma de garantizar la resistencia a fallas, es exigir que la red diseñada posea caminos múltiples entre todo par de nodos, de manera que si se produce una falla en alguna componente de la red, la misma se mantiene en un estado operativo (no queda desconexa). Al agregar caminos múltiples entre pares de nodos de la red, la confiabilidad de la red aumenta, aumentando también el costo global de la red. Los objetivos: mayor confibilidad en la red y mínimo costo posible de conexión son contrapuestos; en general, se fija un grado de conexión global a la red, un entero $k \geq 2$, la red diseñada debe poseer al menos k caminos de nodos disjuntos (k caminos de aristas disjuntas) entre todo par de nodos, y tener el menor costo posible. Este capítulo se centra en el problema de encontrar el grafo k-conexo de costo mínimo que cubre un conjunto de nodos V (indistintamente se dirá k-conexo a un grafo k-nodo-conexo o k-arista-conexo). En especial, se verá el caso en el cual los costos asociados a las aristas del grafo satisfacen la "desigualdad triangular". El capítulo se divide en dos partes: en la primera se analiza el caso $k = 2$: el diseño de un grafo 2-conexo que cubre el conjunto de nodos V con costo mínimo; y en la segunda parte el caso más general: el diseño de un grafo k-conexo ($k \geq 3$) que cubre el conjunto de nodos V con costo mínimo.

9.2 A.3.2 Minimum-Weight 2-Connected Spanning Networks (MW2CSN)

9.2.1 A.3.2.1 Introducción

Se considera el problema de construir un grafo 2-conexo que cubra todos los nodos de un conjunto V con costo mínimo. Se asume que existe una función de costo $c(\cdot)$, $c : V \times V \to N$ que cumple las siguientes propiedades:

1. $c(u,u) = 0$,
2. $c(u,v) \geq 0$,
3. $c(u,v) = c(v,u)$,
4. $c(u,v) + c(v,w) \geq c(u,w)$, (se cumple la desigualdad triangular)

$\forall u, v, w \in V$.

A $c(u,v)$ se le llama costo o largo de la arista (u,v) . El costo del grafo G se define como $\sum_{(u,v)\in E} c(u,v)$. Se asume además que $|V| \geq 3$, para evitar casos particulares.

La motivación del estudio de los grafos de cubrimiento 2-conexos de costo mínimo, viene dada por sus múltiples aplicaciones en el diseño de redes de comunicación de alta confiabilidad, así como redes de transporte. En el resto de este apartado se verá: complejidad del problema, definiciones, propiedades estructurales de la solución óptima, comparación con el Traveling Salesman Problem (TSP), y problemas abiertos relacionados.

9.2.2 A.3.2.2 Complejidad del MW2CSN

En general el problema es NP-Completo, esto puede demostrarse fácilmente ya que el problema de determinar en un grafo $G = (V, E)$ un ciclo hamiltoniano (que es NP-Completo), puede reducirse al problema de encontrar el grafo de cubrimiento 2-conexo de costo mínimo, definiendo la función $c(\cdot)$ asignando costo 1 a las aristas de E y costo 2 a las aristas que

no están en E; entonces hay un ciclo hamiltoniano en G si y solo si la solución óptima al problema MW2CSN tiene costo $|V|$.

Antes de presentar las propiedades estructurales de las soluciones óptimas al MW2CSN, se brindan las siguientes definiciones.

Definición

Un punto de articulación en un grafo $G=(V,E)$ es un nodo tal que al eliminarlo el número de componentes conexas del grafo se incrementa en uno.

Definición

Una arista puente en un grafo $G=(V,E)$ es aquella que al eliminarla el número de componentes conexas del grafo se incrementa en uno.

Definición

Un grafo $G=(V,E)$ es llamado 2-nodo-conexo si G no tiene ningún punto de articulación.

Definición

Un grafo $G=(V,E)$ es llamado 2-arista-conexo si G no tiene ninguna arista puente.

Es fácil ver que si un grafo es 2-nodo-conexo, entonces es 2-arista-conexo. El recíproco no es cierto.

9.2.3 A.3.2.3 Propiedades Estructurales de la Solución Óptima del MW2CSN

Monma-Munson-PulleyBlank [39], estudiaron el problema MW2CSN, donde la función de costo $c(\cdot)$ cumple la desigualdad triangular, logrando obtener resultados que caracterizan la estructura de las soluciones óptimas.

El siguiente lema, da resultados de equivalencias estructurales que caracterizan a un grafo $G = (V, E)$ 2-nodo-conexo.

Lema 1 (Berge)

Las siguientes estados de un grafo $G = (V, E)$ son equivalentes

- *G es 2-nodo-conexo,*
- *cualquier par de nodos de G están en un ciclo común,*
- *cualquier par de aristas de G están en un ciclo común,*
- *cualquier arista y nodo de G están en un ciclo común.*

Las equivalencias para el caso 2-arista-conexo, se obtienen reemplazando "ciclo" por "circuito", permitiéndose así nodos repetidos (no aristas).

En un grafo donde la función costo cumple la desigualdad triangular, el mínimo costo de un subgrafo 2-arista-conexo es igual al mínimo costo de un subgrafo 2-nodo-conexo. Los resultados obtenidos por Monma-Munson-Pulleyblank valen para ambos casos, utilizándose la notación 2-conexo indistintamente para 2-nodo-conexo y 2-arista-conexo.

Muchas veces se pretende hacer cambios locales en una red, de forma que las características de conexión de la red no se pierdan. Por ejemplo, supóngase que se tiene un diseño optimo de una red de comunicaciones, a la cual hay que someterla localmente a ciertos cambios en su topología. Como resultado de dichos cambios, se pretende que la red resultante mantenga el mismo nivel de fiabilidad y optimalidad. El siguiente Lema muestra que para una amplia clase de cambios locales se preserva la 2-conexión en el grafo resultante.

Lema 2 (Monma-Munson-Pulleyblank)

Sea $G = (V, E)$ un grafo 2-conexo con $G' = (V', E')$ un subgrafo de G inducido por V'. Entonces reemplazando E' en G por cualquier conjunto de aristas E'' definidas en V', donde $G'' = (V', E'')$ es 2-conexo, resulta en un grafo $G^* = (V, (E \setminus E') \cup E'')$ el cual es 2-conexo.

Teorema 3 (Monma-Munson-Pulleyblank)

Para cualquier conjunto de nodos V con función de costo $c(\cdot)$ definida en $V \times V$ satisfaciendo la desigualdad triangular, existe un grafo $G = (V, E)$ 2-conexo de mínimo costo satisfaciendo las siguientes condiciones:

1. *cualquier vértice en G tiene grado 2 o 3,*
2. *eliminando cualquier arista o par de aristas en G surge un puente en una de las componentes conexas resultantes de G .*

Definición

Para cualquier grafo conexo G , se define la función costo canónica $c(u, v)$, como el número de aristas del camino más corto que une u con v en G .

La función de costo canónica cumple ser simétrica, no negativa, y satisface la desigualdad triangular.

Teorema 4 (Monma-Munson-Pulleyblank)

Sea $G = (V, E)$ un grafo el cual satisface las siguientes condiciones:

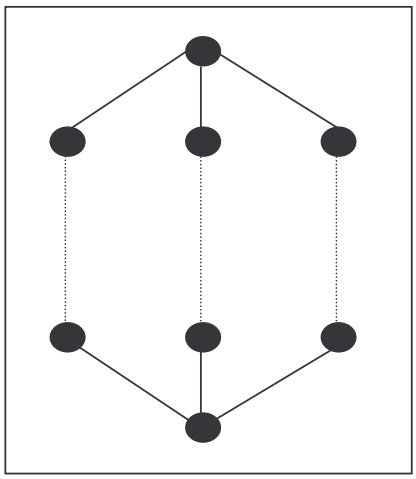
1. *G es 2-conexo,*
2. *todo nodo de G tiene grado 2 o 3,*
3. *G es arista minimal, es decir, eliminando cualquier arista surge un puente,*
4. *eliminando cualquier par de aristas de G surge un puente en una de las componentes conexas del grafo resultante.*

Entonces G es el único grafo 2-conexo de costo mínimo que cubre V para la función de costo canónica.

El Teorema 3 muestra que existe una solución óptima que cumple: todos los nodos tienen grado 2 o 3 y además al eliminar una o dos aristas, en el grafo resultante alguna de sus componente conexas no es 2-conexa . El Teorema 4 muestra que cualquier grafo 2-conexo satisfaciendo estas condiciones es la única solución óptima al problema MW2CSN cuando la función de costo es la función canónica. Estas condiciones permiten restringir la búsqueda de una solución óptima al problema MW2CSN.

Los Teoremas 3 y 4 son muy útiles para estudiar las características de las soluciones óptimas 2-conexas.

Monma-Munson-Pulleyblank, estudiaron en forma particular los grafos con ciclos, grafos cúbicos 2-conexos con aristas subdivididas, y grafos con la forma de la [**Figura 1**].



[**Figura 1**]

Corolario 5 (Monma-Munson-Pulleyblank)

Cualquier grafo $G = (V, E)$ *2-conexo satisfaciendo las condiciones (1) y (2) del Teorema 3, y el cual no es un ciclo, contiene el grafo de la* [**Figura 1**] *como un subgrafo nodo-inducido.*

9.2.4 A.3.2.4 Comparación del MW2CSN con el TSP

Seguidamente se discute la relación del problema MW2CSN con el TSP. Por teorema 4, un ciclo hamiltoniano no necesariamente es una solución óptima 2-conexa. Se utiliza el Teorema 3 para establecer una cota rígida en el peor caso, al radio de performance del costo del ciclo hamiltoniano óptimo con el costo de la solución óptima 2-conexa. Se relacionan además estos resultados con la versión de "*Steiner*" del problema MW2CSN.

Para cualquier conjunto de vértices V y función de costo $c(\cdot)$, se denota $C_{opt}(V)$ a un ciclo hamiltoniano óptimo y $TC_{opt}(V)$ a una solución óptima 2-conexa, con costos $c(C_{opt}(V))$ y $c(TC_{opt}(V))$ respectivamente.

Teorema 6 (Monma-Munson-Pulleyblank)

Para cualquier conjunto de vértices V y función de costo $c(\cdot)$ satisfaciendo la desigualdad triangular, se cumple:
$c(C_{opt}(V)) / c(TC_{opt}(V)) \leq 4/3$.

Además, esta cota puede acercarse arbitrariamente para la clase de grafos de la figura [**Figura 1**] *con $c(\cdot)$ la función de costo canónica.*

Una cota inferior para $c(TC_{opt})$ viene dada por el teorema de W.H.Cunningham. Antes de enunciar el teorema se define el poytopo de subtours para el problema TSP.
El poytopo de subtours para el TSP es el conjunto de soluciones del siguiente sistema lineal:

$$\begin{aligned} &\sum_{j\in(V\setminus\{i\})} x_{ij} = 2, \qquad \forall i \in V \\ &\sum_{i\in S,\, j\in(V\setminus S)} x_{ij} \geq 2, \qquad \forall S \subset V \,/ |S| \geq 2. \\ &0 \leq x_{ij} \leq 1 \end{aligned}$$

Todo vector de incidencia de un ciclo hamiltoniano de G es solución factible del polytopo de subtours del TSP, pero no todo vector de incidencia de un subgrafo 2-conexo que cubre V es solución factible del polytopo de subtours del TSP. Además, si C_{opt} es el ciclo hamiltoniano óptimo con costo $c(C_{opt})$ y S_{opt} es la solución factible del poytopo de subtours del TSP con mínimo costo, entonces es fácil ver que $c(S_{opt}) \leq c(C_{opt})$, siendo $c(S_{opt})$ el costo de S_{opt}.

Teorema 7 (Cunningham)

Para cualquier conjunto de nodos V y función de costo $c(\cdot)$ satisfaciendo la desigualdad triangular. Sea S_{opt} una solución del poytopo de subtours para el cual el costo total es mínimo, y sea $c(S_{opt})$ su valor. Entonces se cumple que: $c(S_{opt}) \leq c(TC_{opt})$.

El problema de encontrar un subgrafo 2-conexo de costo mínimo que cubre los nodos de V es un caso particular del problema de encontrar el subgrafo de Steiner 2-conexo de mínimo costo que cubre un conjunto de nodos $T / T \subseteq V$ llamados terminales .

El siguiente teorema brinda una cota superior, en el peor caso, al radio de performance de una solución óptima 2-conexa que cubre T y una solución de Steiner 2-conexa que cubre T, donde los nodos de $V \setminus T$ pueden ser usados para reducir el costo total de la solución.

Teorema 8 (Monma-Munson-Pulleyblank)

Para cualquier conjunto de vértices V y función de costo $c(\cdot)$ satisfaciendo la desigualdad triangular. Sea $T / T \subseteq V$ un conjunto de nodos denominados terminales. Entonces se cumple la siguiente desigualdad: $c(TC_{opt}(T)) / c(STC_{opt}(T,V)) \leq 4/3$, donde $c(STC_{opt}(T,V))$ es el costo de la solución óptima de Steiner 2-conexa $STC_{opt}(T,V)$.

Soluciones óptimas del problema MW2CSN se adaptan a problemas de diseño de redes en las cuales se necesita que existan caminos alternativos entre todo par de nodos (no existe distinción entre los nodos). De esta manera, si se produce una falla en alguna componente de la red, sigue habiendo comunicación entre cualquier par de nodos de la red. Las topologías 2-conexas son frecuentemente utilizadas en el diseño de redes de fibras ópticas y transporte. Muchas veces se requiere un diseño con un nivel más alto de confiabilidad, es decir, se necesita que existan más de 2 caminos disjuntos entre todo par de nodos. Soluciones de esta clase se utilizan en

aplicaciones de comunicaciones militares, donde resulta vital mantener el sistema en estado operativo aún cuando se producen múltiples fallas por destrucciones de componentes de la red. A continuación se analizan las diferentes propiedades de las soluciones óptimas del problema MWkCSN.

9.3 A.3.3 Minimum-Weight k-Connected Spanning Networks (MWkCSN), k≥3

9.3.1 A.3.3.1 Introducción

El caso más general de MWkCSN, con $k \geq 3$, es estudiado con particular interés por Bienstock-Brieckell-Monma [11]. Dado un grafo completo con pesos asociados a sus aristas satisfaciendo la desigualdad triangular y un entero k, se pretende encontrar el subgrafo k-arista-conexo (k-nodo-conexo) de peso mínimo que cubre todos los nodos del grafo. Bienstock-Brieckell-Monma, estudiaron este problema presentando una serie de resultados que caracterizan propiedades estructurales de las soluciones óptimas a este problema. En este apartado se verá: complejidad del MWkCSN, definiciones de k-conexidad, y los teoremas estructurales para el MWkCSN.

9.3.2 A.3.3.2 Complejidad del MWkCSN

El caso más general del MWkCSN ($k \geq 3$) con las aristas cumpliendo la desigualdad triangular es un problema NP-difícil. Este puede ser transformando a un problema MW2CSN donde las aristas cumplen la desigualdad triangular.

Definición

Un grafo $G = (V, E)$ se dice k-arista-conexo, si al eliminar cualquier conjunto de a lo sumo $k-1$ aristas, el subgrafo resultante sigue siendo conexo.

Definición

Un grafo $G=(V,E)$ se dice k-nodo-conexo, si al eliminar cualquier conjunto de a lo sumo $k-1$ nodos, el subgrafo resultante sigue siendo conexo.

Se asume que un grafo compuesto de un nodo simple cumple ser *k*-nodo-conexo y *k*-arista-conexo para todo *k*.

9.3.3 A.3.3.3 Propiedades Estructurales de la Solución Óptima del MWkCSN

Bienstock-Brieckell-Monma presentan dos teoremas que dan información sobre la existencia de soluciones al problema MWkCSN, mostrando además que la generalización del Teorema 3 para el caso $k \geq 3$ es falsa.

Proposición 9 (Monma-Bienstock-Brieckell)

Sea $k \geq 3$, no es cierto que para cualquier grafo con función de costo cumpliendo la desigualdad triangular el costo mínimo de un subgrafo 2-nodo-conexo que cubre todos los nodos del grafo sea igual al costo mínimo de un subgrafo 2-arista-conexo que cubre todos los nodos del grafo.

Nota: Como se menciono en A.3.2.3, la propiedad de la proposición anterior es válida solo en el caso $k=2$.

Teorema 10 (Bienstock-Brieckell-Monma)

Para cualquier conjunto de nodos V con función de costo $c(\cdot)$ no negativa, simétrica y satisfaciendo la desigualdad triangular, y un entero $k \geq 2$, existe un subgrafo $G=(V,E)$ k-arista-conexo de costo mínimo que cumple las siguientes condiciones:

1. *todos los nodos tienen grado k o $k+1$,*

2. *eliminando un conjunto de a lo sumo k aristas en G, las componentes conexas del subgrafo resultante no son todas k-arista-conexas (respect. k-nodo-conexas).*

Bienstock-Brieckell-Monma, presentan un ejemplo de un subgrafo 3-arista-conexo, el cual satisface las condiciones (1) y (2) del teorema 10 y no es el único subgrafo solución de mínimo costo para cualquier conjunto de costos satisfaciendo la desigualdad triangular. De esta manera, las condiciones (1) y (2) del Teorema 10 no caracterizan la clase de soluciones de mínimo costo para $k = 3$. En el caso $k = 2$, como se vio en el Teorema 3 , estas condiciones sí caracterizan a la solución óptima.

Teorema 11 (Bienstock-Brieckell-Monma)

Para cualquier conjunto de nodos V , con función de costo $c(\cdot)$ no negativa, simétrica y satisfaciendo la desigualdad triangular, y un entero $k \geq 2$, existe un subgrafo $G = (V, E)$ k-nodo-conexo de costo mínimo que cumple las siguientes condiciones:

1. *si $|V| \geq 2k$, entonces todos los nodos tienen grado k o $k+1$,*
2. *eliminando un conjunto de a lo sumo k aristas en G, las componentes conexas del subgrafo resultante no son todas k-arista-conexas (respect. k-nodo-conexas).*

Un problema abierto es el determinar las condiciones que caracterizan las soluciones óptimas al problema MWkCSN cuando $k \geq 3$ y la función de costo $c(\cdot)$ satisface la desigualdad triangular.

Heurística de Bienstock-Brieckell-Monma

La siguiente heurística permite encontrar una solución al problema MWkCSN, con costo diferente del óptimo en a lo sumo un factor constante (dependiente de $k \geq 3$). El orden de la heurística es polinomial.

Paso 1: sea C el ciclo hamiltoniano dado por la heurística de Christofides. Se cumple que el costo de C, es a lo sumo $3/2$ el costo del subgrafo 2-conexo de mínimo costo que cubre todos los nodos del grafo.
Paso 2: sea C^k el grafo obtenido de C agregando una arista que une 2 nodos cualesquiera que estén a distancia k en G. Se cumple que C^k es 2k-conexo y su costo es a lo sumo un factor $3k(k+1)/4$ diferente al costo de la solución óptima 2-conexa.
STOP.

De esta manera, cualquier subgrafo k-conexo cubriendo todos los nodos del grafo tiene costo al menos la solución óptima 2-conexa que cubre los nodos del grafo.

Las soluciones óptimas al problema MWkCSN minimizan el costo de diseño de una red con altos requerimientos de conexión (lo cual implica niveles altos de confiabilidad). Estas no hacen distinción entre los nodos de la red, exigiéndose el mismo grado de conexión (cantidad mínima de caminos disjuntos) entre cualquier par de nodos de la red. Muchas veces, no es necesario que todos los nodos de una red cumplan un mínimo grado de conexión (igual para todo par de nodos). De manera más general, se requiere que todo par de nodos de un subconjunto de nodos distinguidos satisfaga un cierto requerimiento de conexión fijado de antemano. El Problema General de Steiner se corresponde con el diseño de una red que satisfaga estos requerimientos con mínimo costo.