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

Tesis de Maestría
en Informática

Problema General de Steiner en Grafos: Resultados y Algoritmos GRASP para la versión Arista-Disjunta

Pablo Enrique Sartor Del Giudice

2011

Problema General de Steiner en Grafos : Resultados y algoritmos GRASP para la versión arista-disjunta
Sartor Del Giudice, Pablo
ISSN 0797-6410
Tesis de Maestría en Informática
Reporte Técnico RT 11-11
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.
Montevideo, Uruguay, marzo de 2011

TESIS

a ser presentada el día 11 de marzo de 2011 en la

Universidad de la República, UdelaR

para obtener el título de

MAGISTER EN INFORMÁTICA (PEDECIBA)

para

Pablo Enrique SARTOR DEL GIUDICE

Instituto de Investigación : LPE - IMERL
Componentes universitarios :
UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA

Título de la tesis :

Problema General de Steiner en Grafos: Resultados y Algoritmos GRASP para la versión Arista-Disjunta

Presentada el 11 de marzo de 2011 ante el comité de examinadores

Dr. Franco	ROBLEDO	Director de Tesis
Dr. Eduardo	CANALE	Co-Director de Tesis
Dra. Alejandra	BEGHELLI	Revisora
Msc. Ing. Omar	VIERA	
Dr. Álvaro	MARTÍN	

Agradecimientos

La realización de mi Maestría, y en particular el desarrollo de esta tesis, resultaron un proceso fluido y disfrutable gracias a mi tutor, Dr. Franco Robledo Amoza, quien supo transmitirme su entusiasmo por los temas abordados, confió plenamente en mis ideas y contribuyó a encauzarlas en el marco de un proyecto concreto. Agradezco también al Dr. Eduardo Canale con quien discutiendo café y pizarrón por delante germinaron varias de las ideas aquí plasmadas.

Este trabajo, así como cualquier logro académico que pueda alcanzar en el futuro, son directa consecuencia de la visión y la educación recibida de mis padres, así como también del apoyo, comprensión y la motivación permanente de mi esposa Vanessa. Dedico a ellos pues la presente tesis.

Índice general

Parte I

INTRODUCCIÓN

ABSTRACT

El Problema Generalizado de Steiner en Grafos (GSP) es un problema NP-hard de cobertura minimal de grafos con requerimientos de redundancia de conectividad, muy adecuado para modelar problemas reales de diseño topológico de redes de comunicación. La resolución determinística del problema sólo es factible para instancias de tamaño muy limitado, siendo en general para casos reales necesario recurrir a técnicas heurísticas para la generación de soluciones de bajo costo con un consumo razonable de recursos de cómputo y tiempo de ejecución. Los problemas se clasifican como de nodo-conectividad o arista-conectividad, según exijan o no los requerimientos de redundancia que no se compartan nodos intermedios entre caminos múltiples para conectar a las mismas parejas de nodos, modelándose así situaciones en las que tanto nodos como aristas pueden fallar, o solamente las aristas pueden hacerlo.

En este trabajo, haciendo uso de una técnica metaheurística conocida como GRASP (Greedy Randomized Adaptive Search Procedure), extendemos las ideas planteadas en un trabajo previo para la versión de nodo-conectividad del GSP, hacia la versión de arista-conectividad. Se estudia la relación entre ambas versiones del problema, así como las complejidades introducidas por el modelo de arista-conectividad y se proponen mejoras a los algoritmos existentes. Proponemos también un mecanismo alternativo de creación voraz de soluciones y un mecanismo recursivo general para la implementación de metaheurísticas que generen soluciones para el GSP, sobre lo cual esperamos basar futuros trabajos de investigación.

Capítulo 1

El Diseño de Redes Tolerantes a Fallos

El diseño topológico de redes de área extendida (wide area networks, WAN) plantea varios desafíos, a la hora de balancear el costo del tendido de líneas entre sitios de ruteo (switch sites) con los requerimientos de redundancia para tolerar cortes en dichos enlaces sin pérdida de conectividad entre sitios. Este problema ha tomado mayor relevancia en la medida que los enlaces de fibra se han tornado capaces de transmitir cantidades ingentes de información, lo cual habilita a que usualmente una topología de árbol (minimal en cuanto a la cantidad de enlaces) sea suficiente a efectos de canalizar el tráfico requerido (en términos de ancho de banda), con lo cual surge una presión lógica para reducir costos eliminando enlaces redundantes; y lo cual supone el riesgo de perder la conectividad entre partes de la red ante la falla de cualquier componente. Sobre fines de la década de 1980 este problema alcanzó importantes dimensiones; a partir de entonces se han investigado fuertemente las propiedades estructurales de redes resistentes ante cortes y ataques a fin de lograr diseños que combinen bajo costo global con niveles garantizados de redundancia de enlaces.

Usualmente las redes de comunicación se modelan como grafos no dirigidos, en las cuales los nodos representan sitios que emiten, reciben y canalizan información, y las aristas representan enlaces entre tales sitios. De este modo se cuenta con la posibilidad de formalizar y atacar estos problemas por medio del cuerpo de conocimiento de la Teoría de Grafos y el estudio de sus propiedades de conectividad ([19]). Inspirados en los problemas mencionados del diseño de redes, se han modelado y estudiado una serie de problemas de optimización combinatoria sobre grafos.

La investigación de la redundancia de conectividad entre nodos de un grafo ha sido fuertemente estudiada, en particular para los casos 2-conexos (o sea, donde se requiere que existan dos caminos independientes para comunicar cualquier par de nodos) ([40], [21]). Para grafos cuyas aristas tienen costos que cumplen con la desigualdad triangular (lo cual es una restricción que suele ser cercana a los casos reales en el diseño topológico de redes) los problemas son conocidos como “minimum-weight 2-node-connected spanning network” y “minimum-weight 2-edge-connected spanning network” (MW2NCSN y MW2ECSN en adelante). A modo de ejemplo, en el contexto de estos problemas, algunos de los resultados clásicos más notorios

son los siguientes.

Teorema (Frederickson-Jájá 1982) ([11]). Para cualquier conjunto de nodos V con "función de distancia" definida sobre VxV (grafo completo)[1], el costo (definido como sumatoria de las instancias involucradas) óptimo del MW2ECSN es igual al costo óptimo del MW2NCSN. (Para niveles de conectividad k superiores a 2, en general, el costo optimal de las soluciones k-nodo-conectadas será mayor al del costo optimal de las soluciones k-arista-conectadas).

Teorema (Monma et al. 1990) ([27]). Para cualquier conjunto de nodos V con función de distancia definida sobre VxV (grafo completo), existe un grafo de costo mínimo $G = (V, E)$ satisfaciendo las siguientes condiciones:

1. cada nodo en G tiene grado 2 o 3,
2. eliminando cualquier arista o par de aristas de G surge una arista "puente" en una de las componentes conexas resultantes.

Teorema (Monma et al 1990) ([27]). Para cualquier conjunto de nodos V y función de distancia sobre VxV (grafo completo) se cumple que el cociente entre el costo del ciclo hamiltoniano óptimo y el costo de la solución optimal al MW2NCSN (o MW2ECSN) no es mayor que $4/3$.

Si bien para el caso particular 2-conexo se cuenta con estos y otros resultados sumamente útiles, es mucho menor la disponibilidad de resultados teóricos de utilidad a la hora de enfrentar problemas más complejos (a la vez, más cercanos a los reales), que suelen presentar uno o varios de los siguientes factores:

- requisitos de conectividad mayores a 2 para ciertos sitios,
- requisitos de conectividad diferentes para distintos pares de nodos,
- que existan nodos opcionales, esto es, para los cuales no se exige conectividad con ningún otro nodo, aunque pueden usarse si ello contribuye a una mejor solución,
- que los costos de las aristas no verifiquen la desigualdad triangular.

Este tipo de problemas normalmente se ubica en la categoría de los problemas NP-hard (sabiéndose para algunos que son NP-completos), y su resolución requiere entonces el empleo de técnicas heurísticas de optimización para tamaños de problemas a partir de cierta envergadura.

En este trabajo nos enfocaremos en atacar el problema conocido como Problema Generalizado de Steiner (GSP por su sigla en inglés), el cual trata el diseño de redes de costo mínimo, que conecten entre sí a un conjunto de sitios, respetando ciertos requisitos de redundancia de

[1] Función d con codominio en los reales no negativos, tal que i) $d(u, v) = 0$ si y sólo si $u = v$; ii) es simétrica; iii) verifica la desigualdad triangular.

caminos para tolerar fallas en componentes. Este problema, que definiremos formalmente en el capítulo siguiente, supone una generalización del conocido Problema de Steiner en Grafos (SPG por sus siglas en inglés), sobre el cual existe profusa literatura (citada en el capítulo siguiente). Para resolverlo nos basaremos en la aplicación de una metaheurística conocida como GRASP (que presentamos en el Capítulo 3) y sugeriremos diversos algoritmos para implementar cada una de las fases de la misma. El enfoque seguido será similar al seguido en [33], [32] para tratar este problema, pero en la versión en la cual se desea garantizar conectividad mediante caminos arista-disjuntos, permitiéndose eventualmente compartir nodos entre caminos con iguales extremos; nosotros adaptaremos las ideas y propondremos mejoras para estos efectos. También propondremos mecanismos alternativos a investigar tanto para la construcción de soluciones en la fase de construcción de GRASP como para la optimización local de las mismas en la fase de optimización.

El resto de este documento está organizado del siguiente modo. En el Capítulo 2 enunciamos y describimos el Problema General de Steiner, definiendo a su vez la notación que emplearemos a lo largo de nuestro desarrollo; también profundizamos en la relación entre las versiones nodo-disjunta y arista-disjunta del problema. En el Capítulo 3 presentamos la metaheurística GRASP para problemas de optimización combinatoria, sobre la cual están basados los algoritmos que propondremos y estudiaremos. Luego, en los capítulos de la Sección II se presentan algoritmos para la fase de construcción de GRASP y se estudian propiedades de interés. En la Sección III se presentan diversas "vecindades" para la fase de mejora local de GRASP, se proponen algoritmos correspondientes, y se estudian diversas propiedades de los mismos así como variantes de tales algoritmos para verificar dichas propiedades; también se presenta en dicha sección una extensión del problema que llamamos Problema General de Steiner Extendido, que permite una formulación heurística recursiva para nuestro problema objeto de estudio. En la Sección IV sometemos a prueba los algoritmos propuestos empleando casos de prueba tomados de bibliotecas públicas. Finalmente, en la Sección V resumimos las principales conclusiones del trabajo junto con cuestiones de interés que surgieron durante el mismo, planteadas como trabajos futuros de investigación.

Capítulo 2

El Problema General de Steiner en Grafos (GSP)

2.1. Introducción

El Problema General de Steiner en Grafos (GSP), originalmente propuesto en [23], es una extensión del Problema de Steiner en Grafos (SPG), problema éste que ha sido estudiado fundamentalmente mediante análisis combinatorio poliedral ([3], [7], [4], [24], [27], [2], [5], [26], [25]) existiendo también heurísticas propuestas ([18], [38], [39]). Se han investigado características generales y casos puntuales del GSP, así como se ha probado para varios casos (y se cree que aplica en general) que es un problema NP-completo ([40], [35], [33], [32], [1]).

2.2. Definición del Problema

A continuación, y haciendo uso de algunas definiciones previas, presentaremos formalmente el problema GSP.

2.2.1. Definiciones Previas

En este apartado estableceremos algunas definiciones previas y convenciones de las cuales haremos uso en el resto del trabajo.

En el contexto de problemas en redes con requisitos de supervivencia, éstos suelen expresarse básicamente en dos formas distintas:

- requisitos respecto a la cantidad de aristas (links) que debe ser posible suprimir de la red sin que queden desconexos ciertos dos nodos terminales; se traducen en requisitos de caminos arista-disjuntos para tales terminales
- requisitos respecto a la cantidad de nodos (terminales y/o opcionales) que debe ser posible suprimir (obviamente junto con las aristas que les inciden), sin que queden desconex-

os dos ciertos nodos terminales; se traducen en requisitos de caminos nodo-disjuntos para tales terminales

Definición 2.2.1 *Se dice que una pareja de nodos i, j tiene **k-arista-conectividad** o está **k-arista-conectada** en un grafo dado, cuando existen al menos k caminos arista disjuntos (o sea, que no comparten ninguna arista tomados 2 a 2) que conectan i con j.*

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

Definición 2.2.2 *Se dice que un grafo es **k-arista-conexo** cuando toda pareja de nodos i, j del mismo está k-arista-conectada.*

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

Definición 2.2.3 *Se dice que una pareja de nodos i, j tiene **k-nodo-conectividad** o está **k-nodo-conectada** en un grafo dado, cuando existen al menos k caminos nodo disjuntos (o sea, que no comparten ningún nodo salvo i, j) que conectan i con j.*

Definición 2.2.4 *Se dice que un grafo es **k-nodo-conexo** cuando toda pareja de nodos i, j del mismo están k-nodo-conectada.*

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

En general trabajaremos con grafos simples no dirigidos que modelarán redes de comunicaciones. A efectos de trabajar con entidades como caminos, conjuntos de caminos, subgrafos haremos uso de las siguientes notaciones simplificadas:

- dados dos caminos p, q representaremos como $p \cap q$ al conjunto de aristas que comparten;
- dados un conjunto de caminos S y un conjunto de aristas (o un camino) p representaremos como $S \cap p$ al conjunto de aristas de p que están presentes en al menos un camino de S;
- en general representaremos como E_Γ y V_Γ a las aristas y vértices (respectivamente) de un grafo cualquiera Γ.

En definitiva, cuando sea conveniente, consideraremos a los caminos y a los conjuntos de caminos como conjuntos de aristas.

2.2.2. Enunciado Formal del GSP

El "Problema General de Steiner en Grafos" (GSP - Generalized Steiner Problem) puede ser enunciado como sigue:

Definición 2.2.5 ***Problema GSP***. *Dados un grafo simple (no dirigido) $G = (V, E, C)$ donde C es una etiqueta real no negativa para las aristas que llamaremos "costo", un subconjunto de nodos $T \subseteq V$ tal que $2 \leq \|T\|$, y una matriz de enteros no negativos simétrica $R_{\|T\|\times\|T\|} = (r_{ij})$; el objetivo es encontrar un subgrafo $G_T = (V_T, E_T, C_T)$ de G tal que $T \subseteq V_T$, cada par de nodos terminales distintos i, j tenga r_{ij}-conectividad (o sea estén unidos en G_T mediante al menos r_{ij} caminos disjuntos) y tal que el costo total de G_T sea el mínimo posible, siendo C_T la restricción de C al subconjunto T. Por disjuntos puede entenderse "nodo-disjuntos" o "arista-disjuntos" obteniéndose sendas versiones diferentes del problema.*

La versión del problema en la que R especifica requerimientos de arista-conectividad se denomina GSP-EC (GSP Edge-Connected) o también GSP-ED (GSP Edge-Disjoint). Esta es la versión del problema en la que nos enfocaremos principalmente en el presente trabajo. La versión en la que R especifica requerimientos de nodo-conectividad se denomina GSP-NC o GSP-ND (GSP Node-Connected o Node-Disjoinct). Al definir una instancia del problema, es necesario especificar el grafo ponderado G, el subconjunto T de nodos terminales, y la matriz de requerimientos R; así como también debe especificarse claramente si se trata de la versión EC o la NC.

Emplearemos la siguiente notación para tratar formalmente el problema GSP-EC:

- $G = (V, E, c)$: Grafo simple y no dirigido, con pesos en las aristas ;
- V : Nodos del grafo, representan los sitios de conmutación;
- E : Aristas del grafo; representan posibles links entre sitios que es posible construir / operar;
- $C : E \to \mathbb{R}^+$: Pesos de las aristas (reales no negativos), representan el costo de construir / operar un link; a veces nos referiremos a ella como "matriz de costos" - entendida como matriz de reales no negativos simétrica que mapea los valores de la función C ;
- $T \subseteq V$: Nodos terminales que se debe interconectar;
- R : $R \in \mathbb{Z}^{|T|\times|T|}$: Matriz simétrica de requerimientos de conectividad; $r_{ij} = r_{ji} \geq 0, \forall i, j \in T; r_{ii} = 0, \forall i \in T$.

Definimos a continuación los conceptos de solución factible y solución optimal.

Definición 2.2.6 *Una **solución factible** G_{fac} es un subgrafo de G que contiene al menos r_{ij} caminos disjuntos para cada $i \neq j \in T$. A su vez, una **solución optimal** G_T es una solución factible cuyo costo es igual al menor costo de los elementos del conjunto de soluciones factibles.*

Representaremos con $\widetilde{P}_{ij}$ un cierto conjunto de caminos **disjuntos** con extremos i, j. Entonces, podremos escribir

$$G_{fac} = \bigcup_{i \neq j \in T} \widetilde{P}_{ij}$$

para ciertos conjuntos $\widetilde{P}_{ij}$ tales que $|\widetilde{P}_{ij}| \geq r_{ij}$, $\forall i \neq j \in T$. Denotaremos como P_{ij} un conjunto $\widetilde{P}_{ij}$ con r_{ij} caminos.

Agregamos entonces los siguientes elementos de notación:

- $S = \{P_{ij}\}_{i \neq j \in T}$: Solución (conjunto cuyos elementos son conjuntos de caminos para cada pareja de terminales)
- $P_{ij} = \{p_{ijk} : 0 \leq k \leq r_{ij}\}$
- p_{ijk} : k-ésimo camino con extremos i, j, $0 \leq k \leq r_{ij}, p_{ijk} \cap p_{ijh} = \emptyset \iff k \neq h$

Dada una solución factible G_{fac} diremos que S es una solución ejemplo para G_{fac}, si los caminos de S son una evidencia de que G_{fac} es factible, o sea, si:

- $(\forall i \neq j \in T)(\forall p \in P_{ij})(aristas(p) \subseteq E)$
- $(\forall i \neq j \in T)(|P_{ij}| \geq r_{ij})$

2.3. Las versiones Arista-Disjunta y Nodo-Disjunta.

Veamos qué relación existe entre las versiones EC y NC del problema GSP.

Dados G, T, R, cualquier solución factible para la versión NC será también solución factible del problema versión EC, puesto que si existen r_{ij} caminos nodo-disjuntos entre i, j, dichos caminos son también arista-disjuntos. Ahora bien, si además es una solución optimal del NC, no necesariamente lo será para el EC; puesto que pueden existir soluciones de menor costo en éste último problema, que hagan uso de la posibilidad de compartir nodos intermedios entre caminos disjuntos para un mismo par de terminales.

En el sentido opuesto, cualquier solución factible para la versión EC puede o no ser una solución factible del problema NC; ello dependerá de que el subgrafo contenga un conjunto de r_{ij} caminos nodo-disjuntos para cada par de terminales i, j. Por último, si una solución optimal para EC resulta factible para NC, entonces es también optimal para NC; puesto que de no serlo, existiría otra solución factible NC de menor costo, que como ya hemos visto sería también factible en EC, lo cual es absurdo.

En la Figura 2.1 se esquematizan estas relaciones de inclusión entre los conjuntos de soluciones.

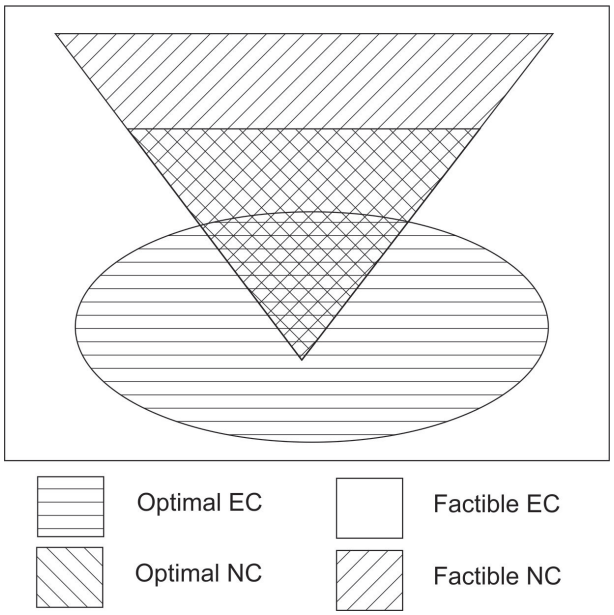


Figura 2.1: Espacios de Soluciones del GSP-EC y GSP-NC

En general existen motivos para afirmar que la resolución de un problema en su versión NC es más "fácil" que la correspondiente versión EC para los mismos grafos de conexiones factibles, terminales y requerimientos; entendiendo por fácil tanto el consumo de recursos de cómputo como también el tipo de estructuras con las que se debe lidiar al concebir "movimientos" entre soluciones tal como lo definiremos en el próximo capítulo; pues

- la relación de inclusión arriba vista hace que el espacio de soluciones factibles de la versión EC sea mayor al de la versión NC
- una vez determinado un camino para unir cierta pareja de terminales (i, j), en la versión EC se descartan sólo sus aristas antes de buscar otro camino para dicha pareja; mientras que en la versión NC se descartan todas las aristas incidentes a nodos intermedios de dicho camino, trabajándose entonces sobre un grafo menos complejo a la hora de buscar caminos subsiguientes
- las descomposiciones estructurales que puede concebirse de soluciones factibles al problema EC plantean estructuras más complejas que para el caso NC, como veremos más adelante con la descomposición que haremos en "key-paths", "key-trees" y "key-stars".

Al final de este capítulo propondremos (y demostraremos su validez) un mecanismo para transformar toda instancia del EC en una instancia NC, de forma que la solución optimal de esta última sea solución optimal de la primera. Este es uno de los resultados fundamentales del presente trabajo, ya que permite emplear algoritmos que resuelven el GSP-NC para resolver instancias del GSP-EC; punto que analizamos en detalle en capítulos posteriores.

2.4. Formulación como Problema de Programación Lineal Entera

El problema GSP-EC admite ser formulado como problema de programación lineal entera. A continuación se presenta una posible formulación, tomada de [40], que hace uso de variables x_{ij} binarias para representar la ocurrencia de cada arista (i, j) en la solución:

Problema: encontrar $x_{ij}, (i, j) \in E$ de forma de

$$minimizar \sum_{i,j \in E} c_{ij} x_{ij} \tag{2.1}$$

sujeto a las siguientes restricciones

$$x_{ij} \geq y_{ij}^{kl} + y_{ji}^{kl} \tag{2.2}$$

$$\sum_{k,j \in E} y_{kj}^{kl} \geq r_{kl} \tag{2.3}$$

$$\sum_{i,l \in E} y_{il}^{kl} \geq r_{kl} \tag{2.4}$$

$$\sum_{p,j \in E} y_{pj}^{kl} - \sum_{i,p \in E} y_{ip}^{kl} \geq 0 \tag{2.5}$$

$$x_{ij} \in \{0, 1\} \tag{2.6}$$

$$y_{ij}^{kl} \geq 0 \tag{2.7}$$

$$\forall i, j \in E, \forall k \neq l \in T, \forall p \in V \setminus \{k, l\}$$

Las variables y_{ij}^{kl} deben interpretarse como flujos desde k hacia l que pasan por la arista (i, j); puede verificarse que el conjunto de aristas (i, j) tales que $x_{ij} = 1$ supone una solución optimal para el GSP-EC.

2.5. Equivalencias entre las versiones Arista-Disjunta y Nodo-Disjunta

En este apartado demostraremos un importante resultado: cualquier problema GSP-EC puede ser transformado en un problema GSP-NC (de una forma muy sencilla) tal que cualquier solución optimal del segundo permite obtener una solución optimal para el primero. Comenzaremos definiendo algoritmos para transformar un problema GSP-EC en uno GSP-NC, y algoritmos para llevar soluciones factibles desde uno hacia el otro. Luego demostraremos propiedades que cumplen dichas transformaciones para terminar demostrando el resultado propuesto. Esto permite plantearse la posibilidad de emplear el "arsenal" de técnicas del que se disponga para resover problemas GSP-NC, también para resolver problemas GSP-EC; aunque a costa de trabajar sobre grafos más complejos en general, como veremos.
Si bien es habitual el empleo de técnicas de *node splitting* para extender algoritmos válidos en contextos arista-disjuntos a casos nodo-disjuntos y viceversa, no existen trabajos previos en los

que se relacione los problemas GSP-NC y GSP-EC haciendo uso de las mismas. Nuestros algoritmos de tranformación proponen un mecanismo inspirado en tales técnicas para transformar problemas de un tipo al otro.

2.5.1. Transformación de Problemas EC en NC

Para cada problema EC (en particular para su grafo de conexiones factibles G), existirá un grafo correspondiente en la instancia NC resultante de transformar el problema. En lo que sigue en este capítulo, a efectos de reflejar la correspondencia entre los nodos de ambos grafos, denotaremos como $t, u, v, w...$ a los nodos del grafo de conexiones factibles G del problema GSP-EC; y dado uno de tales nodos u, denotaremos mediante $[u]$ al nodo que le corresponderá en el grafo del problema correspondiente GSP-NC. En este último entonces podemos interpretar a u como una "etiqueta" del nodo $[u]$ que no puede repetirse en nodos diferentes. También existirán otros nodos en el grafo GSP-NC, que no corresponden a ningún nodo del grafo GSP-EC, con doble etiqueta $[uv]$, donde u y v son nodos del grafo GSP-EC y el orden importa (o sea el nodo $[uv]$ es distinto al $[vu]$ si u es distinto a v).

2.5.1.1. Algoritmo T1: transformación de un problema GSP-EC en otro GSP-NC

El algoritmo **T1**, presentado en la Figura 2.2, recibe como parámetros los objetos que definen una instancia de problema GSP-EC (grafo de conexiones factibles G_E, conjunto de terminales T_E, matriz de requerimientos de conectividad R_E y matriz de costos C_E), devolviendo sendos objetos que definen una instancia de problema GSP-NC. En la figura 2.3 se ilustra el funcionamiento de T1 con un ejemplo; podemos resumirlo intuitivamente del siguiente modo:

- se crea un grafo G_N sin aristas y con los mismos nodos de G_E
- los nodos de grado δ mayor que 2 en G_E son reemplazados por cliques K_δ cuyas aristas tienen costo 0
- cada nodo de cada clique se "asocia" con una arista que le incidía en G_E
- se conecta entre sí los nodos de G_N tal como estaban en G_E, y con iguales costos; cuando interviene un clique, se emplea el nodo del mismo asociado con la arista en cuestión
- se reproducen los requerimientos R_E en R_N; si intervienen cliques, se toma uno cualquiera de sus nodos; el resto de los requerimientos será 0

En las líneas 1-11 se construyen los conjuntos de nodos y terminales de la instancia GSP-NC a devolver. Los nodos de grado 0, 1 o 2 son replicados como nodos con etiqueta simple; los nodos con grado mayor que 2 son replicados como cliques cuyos nodos tienen etiqueta doble, cuyo primer elemento es el nombre del nodo original en cuestión. La línea 12 inicializa las matrices R_N y C_N. El loop de las líneas 13-23 considera todas las parejas de nodos originales, para replicar sus aristas y costos, y establecer todos los requerimientos de conectividad. Para replicar una arista original cuyos nodos tenían grados no mayores que 2, se conecta sus nodos correspondientes en G_N; pero en caso de que uno de aquéllos, o ambos, fueran de grado mayor

Algoritmo T1 (G_E, T_E, R_E, C_E)**;**
1 $V_N \leftarrow \emptyset; T_N \leftarrow \emptyset$
2 por cada $v \in V_E : \delta_E(v) \leq 2$ hacer
3 $V_N \leftarrow V_N \cup \{[v]\}$
4 si $v \in T_E, T_N \leftarrow T_N \cup \{[v]\}$
5 fin por cada
6 por cada $v \in V_E : \delta_E(v) \geq 3$ hacer
7 por cada $u \in V_E : (v,u) \in E_E$ hacer
8 $V_N \leftarrow V_N \cup \{[vu]\}$
9 si $v \in T_E, T_N \leftarrow T_N \cup \{[vu]\}$
10 fin por cada
11 fin por cada
12 sean R_N, C_N mat.cuad. de dim. $|V_N| : \forall i,j \in V_N, R_{Nij} = C_{Nij} = 0$
13 por cada $u, v \in V_E$ hacer
14 si $(u,v) \in E_E$
15 si $[u] \in V_N, x \leftarrow [u]$; si no $x \leftarrow [uv]$
16 si $[v] \in V_N, y \leftarrow [v]$; si no $y \leftarrow [vu]$
17 $E_N \leftarrow E_N \cup \{(x,y)\}$
18 $C_{Nxy} \leftarrow C_{Euv}; C_{Nyx} \leftarrow C_{Euv}$
19 fin si
20 si $[u] \in V_N, x \leftarrow [u]$; si no $x \leftarrow [uz]$ para algún z cualquiera
21 si $[v] \in V_N, y \leftarrow [v]$; si no $y \leftarrow [vw]$ para algún w cualquiera
22 $R_{Nxy} \leftarrow R_{Euv}; R_{Nyx} \leftarrow R_{Euv}$
23 fin por cada
24 por cada $[u,v] \neq [u,t] \in V_N$ hacer
25 $E_N \leftarrow E_N \cup \{([u,v],[u,t])\}$
26 fin por cada
27 return (G_N, T_N, R_N, C_N)

Fin Función

Figura 2.2: Algoritmo T1: transforma problemas GSP-EC en GSP-NC.

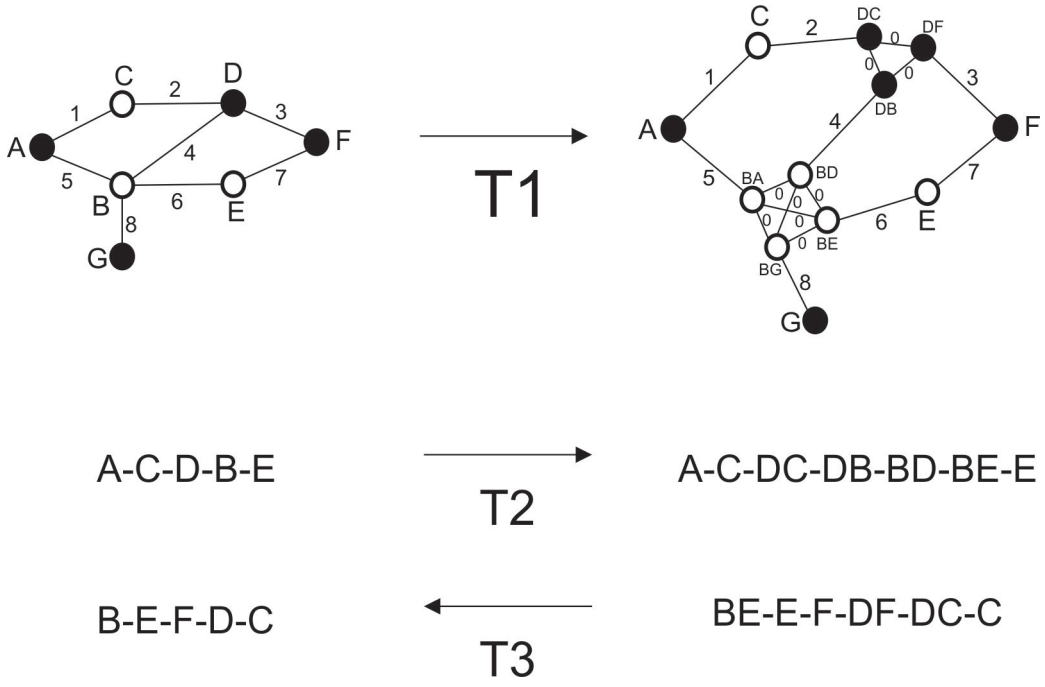


Figura 2.3: Ejemplo de Transformaciones T1, T2, T3

a 2 (y por lo tanto engendraran cliques en G_N), se elige convenientemente qué nodo de cada clique conectar, de forma que cada nodo de un clique tenga una y sólo una arista incidente hacia un nodo que no pertenece a dicho clique. En las líneas 20-22 si es necesario, se elige un nodo cualquiera de los cliques engendrados por la pareja de nodos originales bajo consideración, para generar un requisito de conectividad idéntico al que tenía dicha pareja originalmente; no nos importará la elección, puesto que al final (líneas 24-26) se interconectan todos los nodos de cada clique entre sí con aristas de costo 0. Nótese que la transformación propuesta es similar a la obtención del "grafo de línea" del grafo de conexiones factibles del GSP-EC, salvo por el tratamiento dado a los nodos de grados 0, 1 y 2.

2.5.1.2. Algoritmos T2 y T3: conversión de caminos entre instancias EC y NC en ambos sentidos

A continuación definiremos una correspondencia **T2** que pondrá en correspondencia caminos sobre el grafo de conexiones factibles G de una instancia GSP-EC (denotaremos como P_E a cualquiera de tales instancias) con sus caminos correspondientes en el grafo de conexiones factibles de la instancia correspondiente GSP-NC según T1 (denotaremos P_N a cualquiera de las instancias GSP-NC que se pueden generar a través de T1; o sea $P_N = T1(P_E)$). A su vez, definiremos también una correspondencia **T3** análoga, que pondrá en correspondencia caminos sobre una instancia de una instancia P_N con caminos sobre su instancia correspondiente P_E. Nótese que T1 transforma problemas en problemas (EC a NC); mientras que T2 y T3 transfor-

```
Algoritmo T2 (p_E);
1  p_N ← ∅
2  si largo(p_E) = 0
3     sea u el nodo origen (y extremo) de p_E
4     sea p_N camino trivial [u] o [ux] siendo x cualquiera
5  si no
6     por cada nodo u en orden de p_E hacer
7        si δ_E(u) ≥ 3
8           si u tiene anterior v en p_E, agregar [uv] a p_N
9           si u tiene siguiente w en p_E, agregar [uw] a p_N
10       si no, agregar [u] a p_N
11       fin si
12    fin por cada
13 fin si
14 return p_N

Fin Función
```

Figura 2.4: Algoritmo T2: transforma caminos de GSP-EC en GSP-NC.

man caminos de instancias EC en caminos de instancias NC y viceversa.

El algoritmo T2, presentado en la figura 2.4, recibe un camino p_E sobre el grafo de una instancia P_E del GSP-EC, y devuelve un camino p_N sobre la instancia $P_N = T1(P_E)$ que visita los nodos correspondientes en el mismo orden, haciendo uso de una arista de costo 0 al pasar por cualquier clique que coresponda a nodos de grado al menos 3.

En la figura 2.3 se muestra un ejemplo de la transformación de un camino mediante T2.

El algoritmo T3, presentado en la figura 2.5, recibe un camino p_N sobre el grafo de una instancia P_N del GSP-EC, y devuelve un camino p_E sobre la instancia P_E tal que $P_N = T1(P_E)$; que visita los nodos correspondientes, en el mismo orden, "colapsando" cada clique visitado en un nodo.

En la figura 2.3 también se muestra un ejemplo de la transformación de un camino mediante T3.

2.5.2. Teoremas de Equivalencia

Proposición 2.5.1 *Si p, q son caminos arista-disjuntos con iguales extremos en P_E, entonces $T2(p)$ y $T2(q)$ son nodo-disjuntos (sin considerar sus extremos) en P_N, y tiene sus mismos extremos, o extremos dentro de los cliques eventualmente correspondientes.*

Prueba. Sean p', q' las imágenes según T2 de p, q. Supongamos que p', q' tienen un nodo intermedio $[u]$ en común. Por la forma en que se definió T1 (ver líneas 2-4 y 15-17 de la Figura 2.2), un nodo con etiqueta simple como $[u]$ tiene necesariamente grado menor o igual a 2; entonces ser intermedio a p', q' supone que ambos entran y salen de dicho nodo por sus únicas 2 aristas, lo cual sólo puede darse si en los caminos p, q se compartían dichas aristas, lo

Algoritmo T3 (p_N)**;**
1 $p_N \leftarrow \emptyset$
2 por cada nodo x en orden de p_N hacer
3 si $x = [u]$ agregar u a p_E
4 si no
5 sea $[uv] = x$
6 si último$(p_E) \neq u$ agregar u a p_E
7 fin si
8 fin por cada
9 return p_E

Fin Función

Figura 2.5: Algoritmo T3: transforma caminos de GSP-NC en GSP-EC.

cual es absurdo. Supongamos ahora que p', q' tienen un nodo intermedio $[uv]$ en común. Por la forma en que definimos la transformación T2 (p', q' no son cualesquiera en P_N sino que son producto de aplicar T2), ambos caminos harán uso de la arista $([uv], [v])$ o $([uv], [vu])$ según sea el grado de v. Pero ello significaría que los caminos p, q compartirían una arista (u, v) en P_E; lo cual es absurdo también.

QED

Proposición 2.5.2 *Si p', q' son caminos nodo-disjuntos en P_N, entonces $T3(p')$ y $T3(q')$ son arista-disjuntos.*

Prueba. Sean p, q las imágenes según T3 de p', q'. Supongamos que comparten una arista (u, v). Entonces, por la forma en que definimos a T3, esto sólo puede darse si en p', q' (ambos) hay aristas de la forma $([u], [v]), ([uv], [v]), ([u], [vu]), ([uv], [vu])$; como $[u]$ y $[uv]$ no pueden existir a la vez, ni tampoco $[v]$ y $[vu]$, se sigue que ambos caminos p', q' compartirían nodos en P_N, lo cual es absurdo.

QED

Proposición 2.5.3 *El costo de un camino p en P_E es igual al costo de $T2(p)$ en P_N; el costo de un camino p' en P_N es igual al costo de $T3(p')$ en P_E.*

Prueba. Es trivial considerando la forma en que hemos definido las transformaciones y las funciones de costo; basta ver que las secuencias de aristas fuera de los cliques son las mismas (y tienen idéntico costo por construcción), y las aristas de los cliques no aportan costo alguno.

QED

Notación 2.5.4 *Si S es una solución ejemplo (factible, no necesariamente optimal) de un problema P_E, denotaremos $T2(S)$ al conjunto de caminos obtenidos mediante la aplicación de T2*

a cada camino de S. En forma análoga, si S' es una solución ejemplo factible en P_N, denotaremos $T3(S')$ al conjunto de caminos obtenidos mediante la aplicación de T3 a cada camino de S'.

Podemos ahora presentar uno de los principales resultados de este capítulo.

Teorema 2.5.5 *Si S' es una solución ejemplo factible de un problema P_N, entonces $T3(S')$ es una solución ejemplo factible para los problemas P_E tal que $T1(P_E) = P_N$; y de idéntico costo.*

Prueba. Sea $S = T3(S')$. Cada pareja de terminales en P_N tiene sus requisitos cubiertos mediante suficientes caminos nodo-disjuntos en S'. En virtud de la proposición 2.5.2 también se tendrá entonces la misma cantidad de caminos arista-disjuntos en S para la pareja de terminales que engendró a los terminales de P_N en cuestión; con lo cual queda probado que S es solución ejemplo factible del P_E. En cuanto al costo, puede verificarse fácilmente que las aristas no-clique empleadas en S' pueden ponerse en correspondencia uno-a-uno con las aristas empleadas en S (y de iguales costos), por lo cual el costo del grafo inducido por S y S' son idénticos también en costo.

QED

La importancia de este resultado radica en que, a la hora de buscar soluciones factibles para resolver problemas GSP-EC, permite emplear eventuales algoritmos ya existentes para encontrar soluciones factibles en problemas GSP-NC. En la Figura 2.6 puede verse el ciclo propuesto. Puede generarse soluciones factibles S para los problemas P_E (GSP-EC) mediante algoritmos diseñados para dicha modalidad, como los que desarrollaremos en este trabajo. O puede también transformarse mediante T1 en un problema GSP-NC, resolverlo mediante un algoritmo diseñado para dichos problemas obteniendo una solución S', y finalmente traducir dicha solución hacia una solución S del problema original a través de T3.

También podemos probar el resultado en sentido opuesto (aunque de menor utilidad, puesto que los P_N no cubren todos los problemas GSP-NC sino solamente aquellos que son generables mediante T1 a partir de un problema GSP-EC).

Proposición 2.5.6 *Si S es una solución ejemplo factible de un problema P_E, entonces $T2(S)$ (complementando sus caminos eventualmente con algunas aristas de costo 0 dentro de los cliques) es una solución ejemplo factible para el problema $P_N = T1(P_E)$; y de idéntico costo.*

Prueba. Sea $S' = T2(S)$. Sean (x, y) nodos terminales de P_N. Si se requieren r caminos nodo-disjuntos entre x, y, es porque en P_E se requerían r caminos entre los terminales que engendraron a x, y a través de T1. Cada pareja de terminales en P_E tiene sus requisitos cubiertos mediante suficientes caminos arista-disjuntos en S. En virtud de la proposición 2.5.2 también se tendrá la misma cantidad de caminos nodo-disjuntos en S' uniendo a x, y, o en el peor caso, un nodo del mismo clique que x con un nodo del mismo clique que y. Como todos los nodos de un clique están interconectados con aristas de costo nulo, se puede agregar las aristas entre ellos que sean eventualmente necesarias, a fin de obtener los r caminos deseados entre x, y. Por un argumento similar a la proposición anterior, el costo de las soluciones será el mismo.

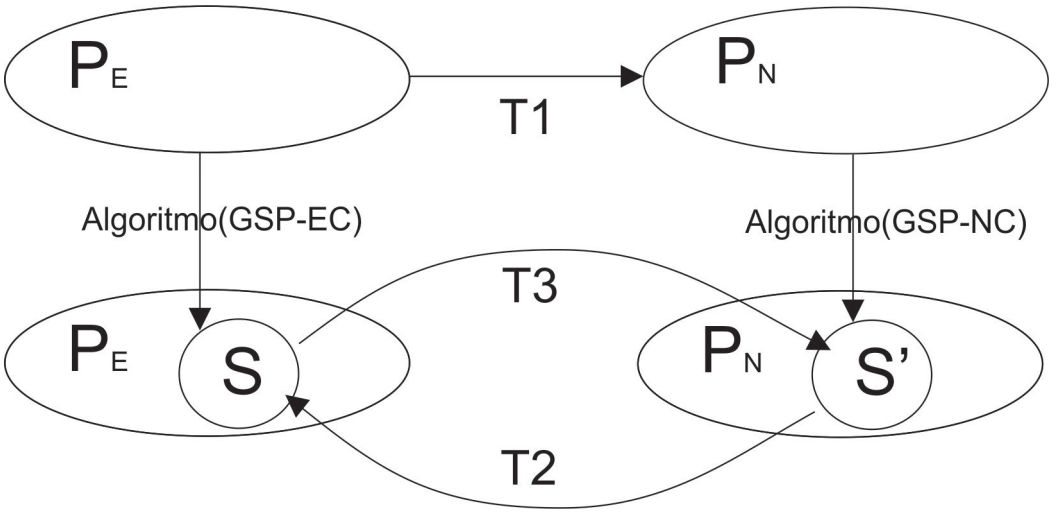


Figura 2.6: Ciclo de Transformaciones

QED

Llegados a este punto podemos ahora culminar este desarrollo mediante un resultado que completa la utilidad del ciclo arriba propuesto.

Teorema 2.5.7 *Si S' es solución ejemplo optimal de $P_N = T1(P_E)$, entonces $S = T3(S')$ es solución optimal de P_E.*

Prueba. S' es optimal en P_N, por lo tanto es factible. Entonces, en virtud del teorema 2.5.2, S es solución ejemplo factible de P_E. Supongamos que no es optimal; debe existir otra solución ejemplo S_{opt} de P_E tal que costo$(S) >$ costo(S_{opt}). Pero entonces, aplicando la proposición 2.5.2, $T2(S_{opt})$ es una solución ejemplo factible de P_N con costo igual al de S_{opt}. Por lo tanto, existiría una solución factible del P_N de menor costo que una óptima, lo cual es absurdo.

QED

De este modo, mientras más nos acerquemos a la optimalidad generando soluciones en P_N, nos estaremos acercando en igual medida a la optimalidad en P_E.

2.5.3. Consideraciones de Complejidad

Hemos propuesto un mecanismo para resolver problemas GSP-EC mediante algoritmos que resuelven problemas GSP-NC. Analicemos ahora la complejidad adicional que introduce el ciclo propuesto. En primer lugar, es necesario traducir la instancia de problema P_E en P_N (algoritmo T1), lo cual requiere una cantidad de operaciones de orden no mayor a $|E|^2$; mucho menor al orden de cualquier algoritmo que genere soluciones en cualquiera de las modalidades del problema. Lo mismo puede decirse respecto al algoritmo T3 que traducirá la solución del

P_N al P_E. Sin embargo, existe un potencial incremento importante en la complejidad del grafo de conexiones factibles sobre el que se debe trabajar al pasar a P_N, dado por los siguientes factores:

- Por cada nodo que engendra un clique, se agregan tantos nodos como indica su grado menos 1.
- Cada clique K_n agrega $n(n-1)/2$ aristas al problema.

En los algoritmos y demostraciones arriba dadas hemos transformado en cliques los nodos a partir de grado 3 inclusive. Puede probarse que no es necesario dividir los nodos de grado 3 (basta observar que dos caminos que empleen un mismo nodo de grado 3 como intermedio, necesariamente comparten aristas); debiéndose sí engendrar cliques para los nodos a partir de grado 4 y mayores. En definitiva entonces, la cantidad de nodos y aristas adicionales (n^+ y m^+) que se agregarán están dadas por:

$$n^+ = \sum_{v:\delta(v)>3} (\delta(v) - 1) \tag{2.8}$$

$$m^+ = \sum_{v:\delta(v)>3} \frac{\delta(v)(\delta(v)-1)}{2} \tag{2.9}$$

Hemos visto entonces cómo la aplicación del ciclo propuesto lleva a un aumento en la complejidad de la instancia a resolver, en términos de las cantidades de nodos y aristas a tratar.

En el siguiente capítulo presentaremos una metaheurística conocida como GRASP, para luego diseñar algoritmos basados en la misma para general soluciones del GSP.

Capítulo 3

La Metaheurística GRASP

3.1. Introducción a GRASP

GRASP (Greedy Randomized Adaptive Search Procedure por su sigla en inglés, o Procedimiento de Búsqueda Adaptativa Glotón y Aleatorio) es una metaheurística de desarrollo relativamente reciente ([8], [29], [31], [28], [9]) para la resolución de problemas de optimización combinatoria. Podemos caracterizarla como de "optimización local multiarranque". La misma propone la aplicación repetida e independiente de 2 fases a la resolución de estos problemas, a saber:

1. generación "rápida" de una solución factible al problema con elementos que aporten aleatoriedad a la construcción y algoritmos "voraces" (greedy).

2. mejora iterativa de la solución generada mediante procedimientos de optimización local, acotada ya sea por una cantidad fija de pasos, umbral de mejora en iteraciones sucesivas, o hasta el alcance de un óptimo local. La optimización local se logra al efectuar "movimientos" de la solución de partida hacia soluciones "vecinas", definiéndose de un cierto modo el concepto de vecindad, usualmente a través de modificaciones menores en la solución, y evaluando entre las soluciones vecinas cuál es la que mayor mejora significa en la función objetivo buscada para "moverse hacia ella"; si no la hay, entonces se habrá hallado un óptimo local.

El nombre de GRASP obedece entonces a los siguientes factores:

- **G - Greedy**: Porque en la fase de construcción de soluciones factibles se aplica alguna técnica o enfoque "greedy" (voraz o glotón)

- **R - Randomized**: Porque en la fase de construcción se hace intervenir, al menos de una forma, el azar para elegir entre varios candidatos el próximo elemento que se incorpore a la solución que se está construyendo

- **A - Adaptive**: Porque cada elección hecha al ir construyendo una solución, puede modificar el costo incremental de cada uno de los otros elementos a agregar en forma posterior

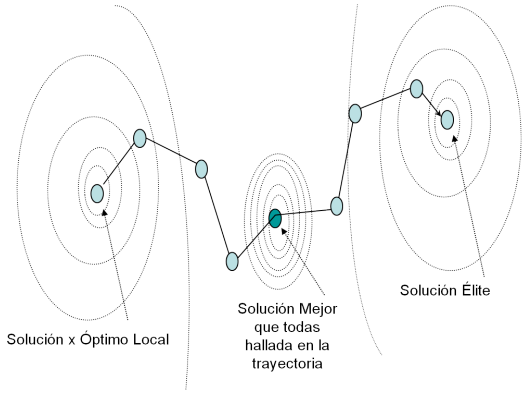


Figura 3.1: GRASP y Path-Relinking

- **S - Search**: Debido a la fase de búsqueda local que sigue a la de construcción
- **P - Procedure**: Procedimiento, algoritmo.

La metaheurística GRASP suele ser complementada con una técnica conocida como path-relinking ([30]), la cual básicamente consiste en tomar dos buenas soluciones entre las halladas hasta cierto momento, y mediante "movimientos" (pequeñas variantes sucesivas) transformar una solución en la otra, analizando si en dicho camino se obtiene una solución desconocida hasta el momento, mejor que ambas. Usualmente una de las dos soluciones de partida -tomada de un conjunto de soluciones de buena calidad- es conocida como "solución elite" y se avanza hacia la misma desde un óptimo local hallado en una iteración cualquiera de la metaheurística. En la Figura 3.1 se ilustra este mecanismo, donde los puntos rellenos representan soluciones en el "espacio de soluciones" representado por el plano, y las curvas cerradas representan curvas de nivel del costo de las soluciones.

En una metaheurística GRASP pura, la fase de construcción se puede esquematizar como muestra la Figura 3.2

La "RCL" (Restricted Candidate List) consiste en una lista de posibles elementos candidato a agregarse a la solución que se está conformando, obtenidos mediante algún mecanismo voraz cuyas características dependerán del problema en cuestión. En las formas más puras de la metaheurística estos elementos deberán ser tales que no se pierda factibilidad al agregarlos, esto es, que al agregarlos se obtenga una solución en formación que pueda evolucionar hacia una solución al problema. Usualmente el elemento aleatorio es introducido a través de alguna forma de ponderación de probabilidades, que haga más propensos a ser elegidos a aquellos elementos que de acuerdo a cierto criterio heurístico y de fácil evaluación se espera que conduzcan a mejores soluciones.

En cuanto a la fase de búsqueda local, existen básicamente dos enfoques ligeramente diferentes:

- **Best-improving**: se analizan todos los vecinos y luego se elige aquél que supone la

Algoritmo Construcción_GRASP();
1 $sol \leftarrow \emptyset$
2 evaluar costo incremental de elementos candidatos a incorporar a sol
3 mientras sol no sea una solución completa, hacer
4 construir una lista restringida de candidatos (RCL)
5 elegir un elemento s de RCL aleatoriamente
6 $sol \leftarrow sol \cup \{s\}$
7 reevaluar los costos incrementales
8 fin mientras
9 return sol

Fin Función

Figura 3.2: Algoritmo Construcción_GRASP

mayor mejora para "moverse" hacia el mismo, recomenzando después el análisis, hasta que no se encuentren vecinos "mejores".

- **First-improving**: se analizan vecinos hasta encontrar el primero que suponga una mejora; entonces se hace el movimiento hacia el mismo, para recomenzar el análisis, hasta que no se encuentren vecinos "mejores".

En cuanto al concepto de vecindad, queda definido una vez que se diseña una función que para cada solución factible al problema devuelva un conjunto formado por todas y sólo sus soluciones vecinas. Distintas vecindades pueden ser combinadas (como lo haremos más adelante) para obtener mejores resultados; esta es la base de otra metaheurística conocida como VNS (Variable Neighborhood Search) basada en esta misma idea. Dos atributos que es altamente deseable que tenga una vecindad para obtener un buen desempeno de un algoritmo GRASP son:

- Que sea eficiente en términos de cómputo la determinación de los vecinos de una solución factible
- Que sea eficiente en términos de cómputo la determinación de sus costos (por ejemplo, con un cálculo mínimo a partir del costo -ya conocido- de la solución cuyos vecinos se está evaluando)

3.2. Estrategia a Seguir

En el presente trabajo nos basaremos en la metaheurística GRASP para proponer algoritmos que generen soluciones factibles al GSP en su versión EC (arista-conectividad), con el objetivo de que logren bajos costos en tiempos de ejecución razonables. En primer lugar analizaremos la fase de construcción, proponiendo algoritmos para construir rápidamente soluciones factibles de partida, y veremos qué modificaciones introducir a efectos de lograr algunas

propiedades deseables que pueden presentar.

A continuación atacaremos la fase de optimización local, para la cual definiremos vecindades y movimientos basados en una descomposición estructural de las soluciones factibles en caminos simples y unas estructuras a las que llamaremos "key-stars" que combinan varios de dichos caminos. Luego propondremos algoritmos que combinan estos movimientos para buscar óptimos locales.

Finalmente someteremos a prueba los algoritmos con un conjunto de casos de prueba, en su mayoría tomados de casos publicados en bibliotecas de problemas clásicos sobre grafos, para evaluar su desempeño y establecer conclusiones.

Parte II

CONSTRUCCIÓN DE SOLUCIONES FACTIBLES

Capítulo 4

Algoritmos para la Construcción de Soluciones Factibles

En este capítulo abordaremos la fase de generación de soluciones factibles de la heurística GRASP, que alimentarán la entrada de la fase posterior de optimización local. Comenzaremos adaptando un algoritmo propuesto en [33] para nuestro caso arista disjunto. Propondremos un par de variantes a efectos de poder garantizar ciertas propiedades de interés que definiremos. Finalizaremos proponiendo un algoritmo diferente que puede ser útil, entre otros casos, cuando ya se conocen muchos caminos entre parejas de nodos terminales.

4.1. Introducción y Algoritmos Previos

Los algoritmos Construcción1 y Construcción2 que proponemos son adaptaciones del algoritmo "ConstPhase" de [33], que a su vez está inspirado en una heurística de Takahashi-Matsuyama [38] para la resolución del SPG (Problema de Steiner en Grafos). En la Figura 4.1 reproducimos dicho algoritmo, empleando nuestra notación. El algoritmo recibe el grafo de conexiones factibles $G = (V, E)$, la matriz de costos C, el conjunto de terminales T, la matriz de requerimientos de conectividad R y un parámetro k de aleatorización de GRASP.

Básicamente el algoritmo procede del siguiente modo:

- 1 - comienza con una solución ejemplo S vacía (todos los conjuntos P_{ij} vacíos)
- 2 - escoge al azar una pareja i, j de terminales para la cual falta determinar al menos un camino disjunto a los ya encontrados (o sea para el cual aún no se conocen r_{ij} caminos disjuntos)
- 3 - determina k caminos que unan a i, j, considerando las aristas de caminos ya agregados a la solución como de costo 0 (ya que es "gratis" su reutilización)
- 4 - escoge aleatoria e uniformemente uno de los k caminos y lo agrega a la solución S; si no es posible hallar k caminos, elimina de S los caminos ejemplo para la pareja i, j y vuelve al paso 2

Algoritmo ConstPhase (G, C, T, R, k)**;**
1 $G_{sol} \leftarrow (T, \emptyset); m_{ij} \leftarrow r_{ij} \forall i, j \in T; P_{ij} \leftarrow \emptyset \forall i, j \in T; A_{ij} \leftarrow 0 \forall i, j \in T$
2 mientras $\exists m_{ij} > 0 : A_{ij} <$MAX_ATTEMPT hacer
3 sean i, j dos terminales al azar tales que $m_{ij} > 0$
4 $G' \leftarrow G \setminus P_{ij}$
5 sea $C' = (c'_{uv}) : c'_{uv} \leftarrow [0$ si $(u, v) \in G_{sol}, c_{uv}$ si no]
6 $L \leftarrow$ k-caminos-más-cortos(G', C', k, i, j)
7 si $L = \emptyset$
8 $A_{ij} \leftarrow A_{ij} + 1; P_{ij} \leftarrow \emptyset; m_{ij} \leftarrow r_{ij}$
9 si no
10 si $\exists p' \in L$,costo$(p')_{C'} = 0$ entonces $p \leftarrow p'$
11 si no $p \leftarrow$ Select_Random$(L); G_{sol} \leftarrow G_{sol} \cup \{p\}$
12 $P_{ij} \leftarrow P_{ij} \cup \{p\}; m_{ij} \leftarrow m_{ij} - 1$
13 $[P, M] \leftarrow$ General_Update_Matrix(G_{sol}, P, M, p, i, j)
14 fin si
15 fin mientras
16 return G_{sol}, P
Fin Función

Figura 4.1: Algoritmo ConstPhase.

- 5 - vuelve al paso 2 a menos que todos los requisitos r_{ij} hallan sido cubiertos
- 6 - devuelve S

La matriz A lleva la cuenta de la cantidad de veces que se llegó a una situación en la cual no es posible encontrar un nuevo camino, disjunto de los anteriores, para unir cierta pareja de terminales; cuando se supera un umbral dado por el natural MAX_ATTEMPTS, se aborta la construcción considerándola infructuosa. La invocación a la función General_Update_Matrix determina si gracias a la introducción del nuevo camino, se acaba de conectar también uno de sus extremos con otro nodo terminal parte del mismo, para registrarlo como camino ejemplo y reducir en uno los requisitos pendientes para esta última pareja de terminales.

Este algoritmo presenta las siguientes características de la heurística GRASP:

- **GREEDY**: se emplea el enfoque "voraz" de buscar el camino más corto (en realidad los k más cortos y elegir uno) para unir dos terminales dados;
- **RANDOMIZED**: se escoge en forma aleatoria el orden en el cual se van trazando los caminos (o sea qué terminales ir conectando) y también se escoge en forma aleatoria cuál de los k caminos adoptar en la línea 11;
- **ADAPTIVE**: las aristas que ya fueron usadas en caminos anteriores al que hay que trazar son consideradas "de peso 0" al momento de calcularlo; de modo que "importa la historia" previa al cálculo de cada camino.

Este algoritmo puede aplicarse perfectamente al caso del GSP-EC (o sea en su versión arista disjunta). La única diferencia con la versión NC (nodo disjunta) se da en la implementación del algoritmo k-caminos-más-cortos. Para ello se pueden emplear algoritmos como el originalmente propuesto por Suurballe y variantes del mismo ([36], [37], [34], [22]), aunque nosotros escogimos para la implementación y testeo un algoritmo equivalente (y de implementación más simple) propuesto en [6].

4.2. Algunas Propiedades de Interés

Existen dos propiedades que pueden ser de interés a la hora de construir una solución; ante un algoritmo propuesto nos interesará saber si verifica tales propiedades. A continuación las enunciamos y veremos que el algoritmo ConstPhase no cumple ninguna de ambas; para posteriormente proponer variantes que sí lo hacen.

- **Propiedad de Construcción 1 (PC1)**: $\lim_{intentos \to \infty} p(\text{lograr un óptimo}) = 1$; en otras palabras, que podamos garantizar cualquier nivel deseado de certidumbre de alcanzar una solución óptima dado que se pueden hacer tantos intentos como sea necesario para ello.

- **Propiedad de Construcción 2 (PC2)**: El algoritmo garantiza la creación de soluciones siempre que los r_{ij} requeridos no superen la nodo (resp. arista) conectividad de la pareja i, j en G. En ciertas situaciones, puede ser deseable obtener los niveles más elevados posibles de conectividad que el grafo de conexiones factibles pueda proporcionar para una, varias o todas las parejas i, j.

El algoritmo ConsthPhase no verifica la PC1, como se demuestra con un contraejemplo en la Figura 4.2. La misma muestra una instancia del GSP, en la cual hay 3 nodos terminales (negros) y uno opcional (blanco), y 9 aristas factibles con sus costos. Supongamos que se requiere 1 camino disjunto entre cualquier pareja de nodos terminales. Si se ejecuta ConstPhase con $k = 1$, los 1-caminos-más-cortos serán siempre caminos de longitud 1 dados por las aristas de costo 10. Por lo tanto la solución construída será invariablemente la mostrada en (b), con costo 20. Por otra parte, si se ejecuta ConstPhase con $k = 2$, los 2-caminos-más-cortos serán siempre caminos de longitud 1 dados por las aristas de costo 10 y 11. Por lo tanto, la solución construida será siempre de costo mayor o igual a 20, algunas de las posibles se muestran en (c). En ninguno de dichos casos se hallará la solución óptima mostrada en (d), de costo 18. Aún más, ninguna de las soluciones mostradas en (b) o (c) pueden ser transformadas en la solución óptima mediante ninguno de los mecanismos para la fase de optimización local propuestos en [33] y/o en este trabajo (reemplazo de key-paths, key-trees o key-stars, tratados más adelante). En resumen, existen instancias (algunas como esta incluso muy elementales) del GSP y parametrizaciones del ConstPhase para las cuales es imposible construir una solución óptima (y para ciertos movimientos, también imposible que las generadas "muten" a través de los mismos hacia una solución optimal).

Por otra parte, el algoritmo ConstPhase no cumple con la PC2 puesto que su enfoque voraz para el trazado de los r_{ij} caminos disjuntos entre una pareja i, j de terminales suele llevar a

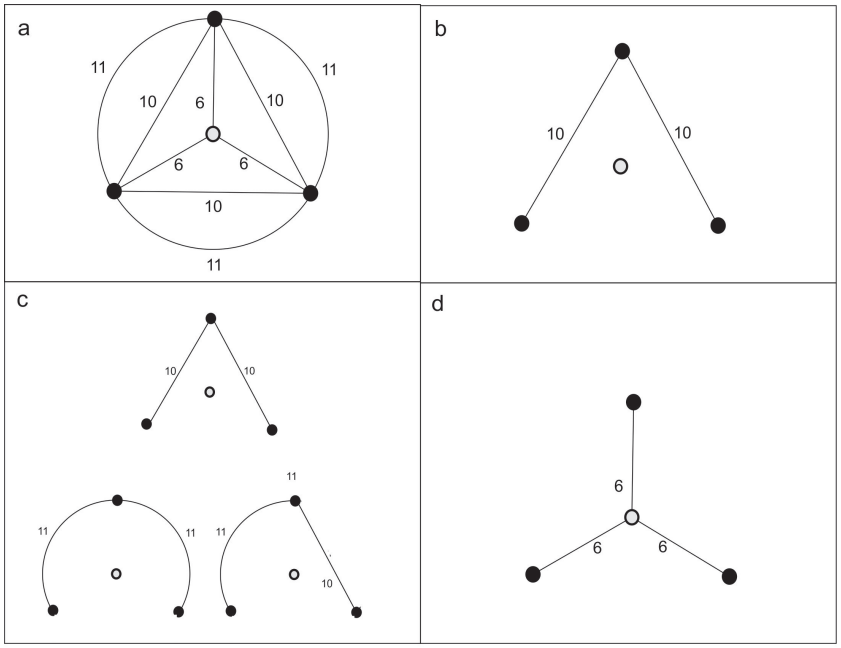


Figura 4.2: Contraejemplo para PC1

situaciones en las que, a pesar de que se ha determinado una cantidad menor de caminos que la conectividad mutua de dichos nodos, ya no es posible trazar uno nuevo disjunto. Esto se ilustra con un ejemplo muy elemental en la Figura 4.3. En la misma, el nivel de conectividad (tanto arista como nodo disjunta) es 2 entre los nodos u, v. Sin embargo, la determinación del camino más corto (o en términos del algoritmo, el "1-caminos-más-cortos" con $k = 1$), construiría el camino mostrado en (b), que inhabilita todo camino adicional disjunto, imposibilitando la obtención de una solución como la mostrada en (c). Pueden construirse ejemplos similares para cualquier valor de k. Si bien ConstPhase reintenta MAX_ATTEMPTS veces al enfrentar una situación de este tipo, no hay garantías de que se obtenga el nivel máximo de conectividad, y en situaciones como la ejemplificada ($k = 1$) con ningún valor de MAX_ATTEMPTS, por mayor que sea, se tendrá chances de alcanzar el nivel máximo de conectividad.

4.3. Algoritmos Propuestos

En esta sección proponemos 3 algoritmos posibles para la fase de construcción: Construcción1 y Construcción2 (pequeñas variantes de ConstPhase) y Construcción3 (basado en un enfoque diferente).

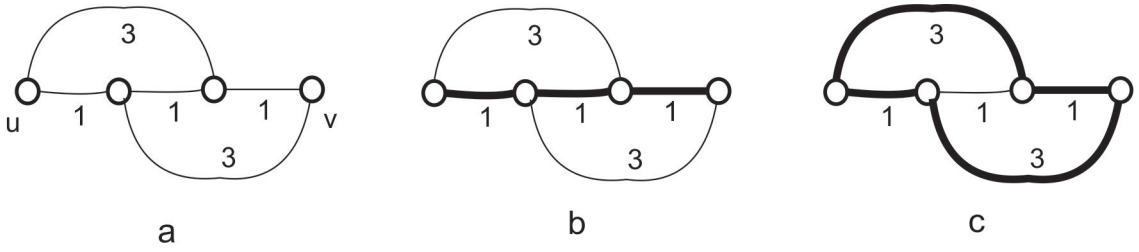


Figura 4.3: Contraejemplo para PC2

4.3.1. Construcción1

A efectos de lograr que se cumpla la PC1, proponemos un enfoque diferente para la aleatorización en la construcción de caminos para unir una pareja dada de terminales. El mismo consiste en introducir "ruido" en los costos, alterándolos en forma aleatoria al principio de la ejecución del algoritmo, para luego seleccionar siempre el camino más corto posible ponderado según estos costos alterados; empleando a tales efectos el algoritmo de Dijkstra. Las diferencias a introducir respecto al algoritmo ConstPhase son entonces:

- Al principio del algoritmo, se alterarán los costos de todas las aristas mediante una función disenada de modo que puedan oscilar entre $(0, +\infty)$ y valor esperado igual al costo original de la arista
- En vez de generar k-caminos-más-cortos para luego elegir uno al azar, se buscará siempre el camino más corto (algoritmo de Dijkstra)

Una ventaja adicional del algoritmo propuesto es que en cada paso se invoca al algoritmo camino-más-corto en vez de k-caminos-más-cortos con la consiguiente ventaja en tiempo de procesamiento cuando $k \neq 1$. En la Figura 4.4 se presenta el algoritmo con estas modificaciones incluidas.

4.3.1.1. Alteración de Costos

Tanto a efectos de introducir aleatoriedad en el algoritmo de construcción, como de evitar las "trampas" de ciertos óptimos locales que impiden que siempre se tenga probabilidades de hallar una solución optimal, el algoritmo comienza alterando los costos mediante una función que llamaremos **altera_costos**. Intuitivamente, la idea consiste en dejar sin costo eventuales aristas factibles que tienen costo cero, a la vez de alterar el costo de las demás, pero de algún modo que se vea influido por el costo original, de modo de que sea lo más "frecuente" que aristas con costos menores a otras obtengan costos alterados menores que éstas. A continuación formalizaremos estas ideas y daremos una posible definición para la función.

Procuraremos que las siguientes propiedades sean verificadas por la función altera_costos:

- Si el costo de la arista original es 0, el costo alterado también será 0.

Algoritmo Construcción1 (G, C, T, R)**;**
1 $G_{sol} \leftarrow (T, \emptyset); m_{ij} \leftarrow r_{ij} \forall i, j \in T; P_{ij} \leftarrow \emptyset \forall i, j \in T; A_{ij} \leftarrow 0 \forall i, j \in T$
2 $C \leftarrow$altera_costos(C)
3 mientras $\exists m_{ij} > 0 : A_{ij} <$MAX_ATTEMPT hacer
4 sean i, j dos terminales al azar tales que $m_{ij} > 0$
5 $G' \leftarrow G \setminus P_{ij}$
6 sea $C' = (c'_{uv}) : c'_{uv} \leftarrow [0$ si $(u, v) \in G_{sol}; c_{uv}$ si no]
7 $p \leftarrow$ camino-más-corto(G', C', i, j)
8 si $\nexists p$
9 $A_{ij} \leftarrow A_{ij} + 1; P_{ij} \leftarrow \emptyset; m_{ij} \leftarrow r_{ij}$
10 si no
11 $G_{sol} \leftarrow G_{sol} \cup \{p\}$
12 $P_{ij} \leftarrow P_{ij} \cup \{p\}; m_{ij} \leftarrow m_{ij} - 1$
13 $[P, M] \leftarrow$ General_Update_Matrix(G_{sol}, P, M, p, i, j)
14 fin si
15 fin mientras
16 return G_{sol}, P
Fin Función

Figura 4.4: Algoritmo Construcción1.

- Si el costo de la arista original es distinto de 0:
  - El costo de la arista será alterado en forma independiente a las demás aristas.
  - El costo alterado será un valor cualquiera de $(0, +\infty)$ gracias a una cierta distribución de probabilidad que asigne probabilidades no nulas a cualquier subintervalo abierto de $(0, +\infty)$. Denotaremos como $\hat{C}_{ij}$ a la variable aleatoria que siga esta distribución para una cierta arista (i, j).
  - El valor esperado será el costo original de la arista.
  - $(c_{ij} < c_{hk}) \rightarrow (p(\hat{C}_{ij} < \hat{C}_{hk}) > \frac{1}{2})$ para cualquier par de aristas $(i, j), (h, k)$

donde $f_{ij}(x)$ es la función de densidad de probabilidad de la distribución de $\hat{C}_{ij}$ para la arista (i, j); y c_{ij} es el costo original de una arista (i, j).

Como una alternativa posible proponemos el empleo de la "distribución exponencial", definida del siguiente modo (con un parámetro λ):

$$f_\lambda(x) = \lambda e^{-\lambda x}$$

Como parámetro emplearemos el inverso del costo original de las aristas; para cada arista (i, j) con costo original c_{ij} la función altera_costos propuesta devolverá entonces:

- 0 si $c_{ij} = 0$
- un valor generado por una variable con distribución $f_{1/c_{ij}}$ si $c_{ij} > 0$

La distribución propuesta asigna valores en $(0, +\infty)$ para cada arista con costo no nulo, verifica la propiedad del límite, y tiene valor esperado igual a dicho costo, puesto que el valor esperado de una distribución exponencial es el inverso de su parámetro, es decir, c_{ij} en nuestro caso.
También puede verse que la distribución propuesta verifica la propiedad de orden. Analizando la distribución conjunta:

$$p(\hat{C}_{ij} < \hat{C}_{kh}) = \iint_{x<y} (1/c_{ij}) e^{-x/c_{ij}} (1/c_{kh}) e^{-y/c_{kh}} \, dx \, dy = \frac{c_{ij}^{-1}}{c_{ij}^{-1} + c_{hk}^{-1}}$$

vemos que la probabilidad de que (i, j) obtenga un valor menor al de (h, k) es mayor cuanto menor sea c_{ij} respecto de c_{hk}.

La distribución exponencial asigna probabilidades no nulas a cualquier subintervalo abierto de $(0, +\infty)$; esto es de vital importancia para la demostración de la propiedad PC1 que sigue a continuación.
Dejamos propuesto el trabajo de explorar el efecto sobre el algoritmo del uso de otras distribuciones.

4.3.1.2. Espacio de Soluciones Construibles

Demostraremos a continuación que el algoritmo Construcción1 verifica la propiedad PC1.

Teorema 4.3.1 *Para toda instancia del GSP, el algoritmo Construcción1 siempre incluye en su espacio de soluciones algunas optimales y tiene una probabilidad mayor estricta que cero de alcanzar una de ellas; en otras palabras, verifica la propiedad PC1.*

Prueba. Sea S una solución ejemplo cualquiera cuyo costo es el optimal para el problema. Supongamos que la probabilidad p de que una ejecución del algoritmo construya exactamente dicha solución es positiva estricta, $p \in (0, 1]$. Entonces, la probabilidad de que el algoritmo no encuentre dicha solución en ninguna de k ejecuciones (que son independientes) será $(1 - p)^k$, y la probabilidad de construir dicha solución al menos una vez al cabo de k ejecuciones será:

$$p_k(S) = 1 - (1 - p)^k > 0$$

Por lo tanto

$$\lim_{k \to \infty} p_k(S) = 1$$

tal como queremos demostrar.

Veamos ahora que efectivamente, dada cualquier instancia del problema y una solución ejemplo optimal S, la probabilidad p es no nula. Sea $(S)_i = s_1, s_2, ..., s_r$ con $r = \sum_{i,j \in T} r_{ij}$ un ordenamiento de los caminos de S obtenido del siguiente modo:

- s_1 es el (o cualquiera de los) caminos de S que tiene la menor cantidad de aristas con costo no nulo;
- s_k es el (o cualquiera de los) caminos que tienen la menor cantidad de aristas con costo no nulo que no son parte de ninguno de los caminos anteriores $s_1 ... s_{k-1}$;

Denotando como $l^*(s)$ la cantidad de aristas de costo no nulo de un conjunto s, y $l_1^*, ..., l_r^*$ a los resultados de aplicar l^* a un cierto ordenamiento de caminos, formalmente $(S)_i$ es cualquier ordenamiento tal que

$$l^*\left(s_k \setminus \bigcup_{i=1...k-1} s_i\right) = \min_{s \in S \setminus \{s_1,...,s_{k-1}\}} l^*\left(s \setminus \bigcup_{i=1...k-1} s_i\right)$$

La cantidad de ordenamientos posibles (arreglos con repeticiones) de la secuencia de parejas de terminales (i, j) que el algoritmo puede seguir (aleatoriamente) para generar los caminos está dada por

$$\frac{\left(\sum_{i<j} r_{ij}\right)!}{\prod_{i<j}(r_{ij}!)}$$

y por lo tanto, la probabilidad de que el algoritmo genere los caminos ejemplo siguiendo un orden de terminales extremos que coincida con el de $(S)_i$ será

$$p_{orden}(S) = \frac{\prod_{i<j}(r_{ij}!)}{\left(\sum_{i<j} r_{ij}\right)!} > 0$$

Asumamos ahora que el algoritmo generará los caminos con el mismo orden de terminales dado por $(S)_i$. Veremos a continuación que la probabilidad de que el camino generado en la posición i sea justamente el s_i (u otro con idéntico costo) es no nula; lo cual permitirá completar nuestra demostración hallando una cota inferior para la probabilidad p. Para ello construiremos una ponderación alterada de costos $\hat{C}$ que hará que el algoritmo genere exactamente los caminos deseados, y veremos que la probabilidad de que la función de alteración de los costos originales resulte en $\hat{C}$ es no nula.

Dados un cierto ϵ tal que $0 < \epsilon < \frac{1}{2|E|}$, y un cierto $k \in \mathbb{N}$, sea $\hat{C}_k = (\hat{c}_{k,ij})$ una ponderación (asignación de costos) cualquiera de aristas factibles que verifique lo siguiente:

- $\hat{c}_{k,ij} = 0$ si $c_{ij} = 0$
- $\hat{c}_{k,ij} \in (k - \epsilon, k + \epsilon)$ si $c_{ij} > 0$

En lo que resta trabajaremos con $\hat{C}_k$ y llamaremos $\hat{c}_k(q)$ al costo de un camino q según $\hat{C}_k$. En general un camino s con $l = l^*(s)$ aristas de costo no nulo tendrá un costo $\hat{c}_k(s) \in (kl - l\epsilon, kl + l\epsilon)$ y por lo tanto $\hat{c}_k(s) \in (kl - 1/2, kl + 1/2)$. Se sigue que dados 2 caminos q y q' con $l^*(q) < l^*(q')$ siempre se cumple que $\hat{c}_k(q) < \hat{c}_k(q')$.

Construyamos la ponderación de costos $\hat{C}$ del siguiente modo. A todas las aristas de s_1 se le asigna un costo según $\hat{C}_1$. A todas las aristas de $s_2 \setminus s_1$ se le asigna costo según $\hat{C}_2$. Se continúa de igual modo, asignando a las aristas de $s_k \setminus (s_1 \cup s_2 \cup ... \cup s_{k-1})$ costos según $\hat{C}_k$. Finalmente a las aristas de $G \setminus S$ se les asigna un costo cualquiera mayor a todos los ya asignados. Con esta matriz de costos alterados $\hat{C}$, el primer camino que generará el algoritmo será necesariamente s_1; puesto que cualquier camino diferente tendrá por lo menos la misma cantidad de aristas con costo no nulo, y por cada arista que se "cambie" entre s_1 y éste, se estaría aumentando el costo. Del mismo modo, el segundo camino generado no puede ser otro que s_2, puesto que el costo agregado será mínimo sólo si se agregan sus aristas. Continuando el razonamiento, vemos que el algoritmo generará exactamente la secuencia $(S)_i$ de caminos ejemplo.
Tenemos entonces:

- Una probabilidad no nula $p_{orden}(S)$ de "sortear" un orden de parejas de terminales igual al de $(S)_i$
- Una probabilidad no nula $p_{alter}(S)$ de que la función de alteración de costos asigne costos a las aristas factibles de G que verifiquen nuestra definición de $\hat{C}$ (puesto que para cualquier arista con costo original no nulo, la probabilidad de que nuestra función de alteración le asigne un costo en $(k - \epsilon, k + \epsilon)$ es no nula).

La probabilidad de que se den ambos sucesos es de $p_{orden}(S)p_{alter}(S) > 0$, y siendo estos casos un subconjunto del espacio muestral de "sorteos y alteraciones" que producen a S, se sigue que hemos hallado una cota inferior mayor que 0 para la probabilidad p; lo cual concluye nuestra demostración.

QED

4.3.2. Construcción2

Para situaciones donde se desea que se verifique la PC2, proponemos una modificación a la Construcción1, basada en calcular los r_{ij} caminos ejemplo necesarios para conectar la pareja de terminales i, j en un solo paso, mediante el algoritmo de k-caminos-más-cortos. Los algoritmos hasta aquí presentados generan k (parámetro GRASP) caminos candidatos para luego elegir uno, y más adelante se vuelven a efectuar esto para agregar otro camino a la misma pareja de terminales. Hemos visto que esto puede llevar a situaciones donde ya no pueden agregarse caminos para esos terminales a pesar de no haberse alcanzado su nivel máximo de conectividad (esto se da cuando para todo corte cardinal-minimal entre los nodos, siempre existe al menos un camino de los ya encontrados que emplea más de una arista del corte). En la Figura 4.5 se presenta el algoritmo Construcción2 con la modificación propuesta. La invocación al algoritmo k-caminos-más-cortos garantiza que se alcance el máximo nivel de conectividad si así

Algoritmo Construcción2 (G, C, T, R)**;**
1 $G_{sol} \leftarrow (T, \emptyset); m_{ij} \leftarrow r_{ij} \forall i, j \in T; P_{ij} \leftarrow \emptyset \forall i, j \in T$
2 $C \leftarrow$altera_costos(C)
3 mientras $\exists m_{ij} > 0$ hacer
4 sean i, j dos terminales al azar tales que $m_{ij} > 0$
5 sea $C' = (c'_{uv}) : c'_{uv} \leftarrow [0$ si $(u, v) \in G_{sol}, c_{uv}$ si no]
6 $L \leftarrow$ k-caminos-más-cortos(G, C', r_{ij}, i, j)
7 si $|L| < r_{ij}$
8 error "Requerimientos de R son imposibles"
9 si no
10 $G_{sol} \leftarrow G_{sol} \cup L$
11 $P_{ij} \leftarrow L; m_{ij} \leftarrow 0$
12 fin si
13 fin mientras
14 return G_{sol}, P
Fin Función

Figura 4.5: Algoritmo Construcción2.

es requerido por la matriz R para cualquier pareja de terminales. En este caso no se invoca a la función General_Update_Matrix puesto que aunque se introduzcan posibles caminos para (i, w) o (w, j) al agregar los caminos para (i, j) a la solución ejemplo (siendo w un terminal cualquiera intermedio de estos últimos caminos), en general no tienen por qué ser parte de un conjunto de k-caminos-más-cortos para (i, w) o (w, j), por lo cual no pueden "reconocerse" en esta etapa.

El hecho de sortear el orden en que se van creando, de a uno, los caminos ejemplo en ConstPhase, hace que el espacio de soluciones factibles generables por el algoritmo sea mayor que si se generan en un solo paso los r_{ij} caminos. Pero nuestros algoritmos Construcción1 y Construcción2 introducen la aleatoriedad no sólo en el orden de generación sino también en la alteración de costos, lo cual constribuye a paliar dicha pérdida de "diversidad".

El algoritmo Construcción2 también cumple la propiedad PC1. Para probarlo puede emplearse un enfoque muy similar al usado para Construcción1, basado en determinar rangos de costos a asignar a las aristas de forma que al evaluarse en un cierto orden dado las parejas de terminales, se generen caminos que conformen exactamente un subgrafo solución optimal dado. La diferencia estriba en que el orden de parejas de terminales deberá ser escogido de forma que la cantidad de aristas de costo no nulo que agregan **en forma conjunta** los caminos optimales para la "próxima" pareja de terminales sea la menor posible.

Por último, una variante a introducir puede ser que la cantidad de caminos a encontrar al invocar a k-caminos-más-cortos sea igual a $max(r_{ij}, k)$ siendo k el parámetro de GRASP de los algoritmos ConstPhase y Construcción1; en caso de que $r_{ij} < k$, se habrá de elegir luego exactamente r_{ij} de esos caminos para incorporar a la solución en formación.

4.3.3. Construcción3

4.3.3.1. Un Enfoque Diferente

En este apartado propondremos un algoritmo que hace uso de una técnica diferente. Los algoritmos hasta aquí presentados proceden de una forma "constructiva" y "adaptativa"; parten de un conjunto nulo de caminos para luego ir agregando paulatinamente caminos ejemplo, cuyo diseño va siendo influido por los caminos previamente escogidos. El algoritmo que propondremos a continuación opera de forma muy diferente (casi opuesta), a saber, "restrictiva" y "egoísta". Proponemos básicamente los siguientes pasos:

- Generar los r^*_{ij}-caminos-más-cortos sobre el grafo y costos originales, para cada pareja de terminales i, j, siendo r^*_{ij} una cantidad al menos igual a r_{ij} en cada caso.
- Escoger exactamente r_{ij} caminos de cada uno de los conjuntos, en forma aleatoria, ponderando con mayor probabilidad a los caminos que presentan máyor probabilidad de compartir aristas con caminos de otros terminales; esto a fin de privilegiar la reutilización de aristas.

El aspecto "restrictivo" viene dado porque en una primera fase se genera un superconjunto de caminos, para restringirse en una fase posterior a algunos de ellos. El aspecto "egoísta" viene dado por el cómputo de los caminos más cortos para cada pareja de terminales que no toma en cuenta en modo alguno los caminos computados para las demás parejas de terminales. A su vez, los aspectos "Greedy" y "Randomized" de GRASP estarán presentes en el algoritmo del siguiente modo:

- La selección de caminos en el segundo paso se basa en un score calculado para cada camino, que toma en consideración tanto su costo como la probabilidad de que comparta aristas con caminos ejemplo que unen a otras parejas de terminales; este es un mecanismo heurístico para encontrar un conjunto globalmente conveniente de caminos con un enfoque "voraz" en lugar de hallar determinísticamente el conjunto óptimo - que como veremos es un complejo problema de optimización combinatoria.
- En el primer paso, la cantidad r^*_{ij} de caminos encontrados para cada par de terminales i, j será determinada aleatoriamente por un parámetro α; mientras que en el segundo paso, la selección de los r_{ij} caminos requeridos se hace según probabilidades ponderadas según el score arriba mencionado de cada camino. Estas son dos formas de incorporar la característica "randomized" inherente a GRASP.

4.3.3.2. Algoritmo Propuesto

El algoritmo sugerido recibe las siguientes entradas:

- el grafo $G = (V, E)$ de conexiones factibles
- el conjunto T de nodos terminales

Algoritmo Construcción3(G, C, R, α)**;**

1 $G_{sol} \leftarrow (T, \emptyset)$; $S \leftarrow \emptyset$; $P_{ij} \leftarrow \emptyset \forall i \neq j \in T$
2 para cada pareja de nodos $k, l \in T$
3 $r^*_{kl} \leftarrow$ cantidad-aleatoria(r_{kl}, α)
4 $S^{kl} \leftarrow$ k-caminos-más-cortos(G, C, r^*_{kl}, k, l)
5 fin por cada
6 $X \leftarrow$ EligeCaminos(S, C, R)
7 por cada camino $s \in S$
8 si todas las aristas de s están etiquetadas con 1 en X
9 $G_{sol} \leftarrow G_{sol} \cup \{s\}$
10 $P_{ij} \leftarrow P_{ij} \cup \{s\}$ siendo i, j los nodos extremos de s
11 fin si
12 fin por cada
13 $P \leftarrow$ GeneralUpdateMatrix(G_{sol}, P)
14 return G_{sol}, P
End Procedure

Figura 4.6: Algoritmo Construcción3.

- la matriz de costos $C_{|V|\times|V|}$
- la matriz de requerimientos de conexiones $R_{|T|\times|T|}$
- el parámetro de aleatoriedad α

En el contexto de este algoritmo denotaremos como S al conjunto de caminos generados a través del algoritmo k-caminos-más-cortos, que serán candidatos a ser miembros de la solución a construir. Este conjunto será particionado en diversos subconjuntos según sus nodos extremos; denotaremos como S^{ij} al subconjunto de caminos que arrancan y terminan en i, j. También denotaremos como $X \in \{0, 1\}^{|E|}$ a un "bitmap" de 0's y 1's que indicará respectivamente la ausencia o presencia de cada arista en un conjunto de aristas ordenado; a su vez denotaremos como X_{ij} al "bit" que indica la ausencia o presencia de la arista (i, j).

El algoritmo, mostrado en la Figura 4.6, procede del siguiente modo. Las líneas 2-5 computan los caminos más cortos arista disjuntos para cada pareja de nodos terminales. La cantidad de caminos a generar para un par k, l es computada por la función cantidad-aleatoria del siguiente modo:

$$r^*_{kl} = r_{kl} \times (1 + \lfloor \alpha RND() \rfloor)$$

donde $RND()$ es un número real aleatorio uniforme en el rango $[0, 1)$. Luego, la línea 4 invoca el algoritmo k-caminos-más-cortos para encontrar los r^*_{kl} caminos más cortos entre k, l (o tantos como sea posible; ya que r^*_{kl} podría superar al nivel de conectividad entre ambos nodos).

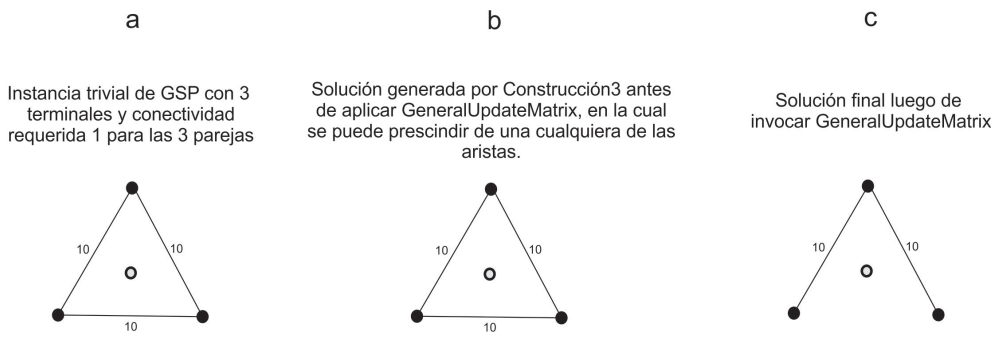


Figura 4.7: Construcción3: Generación de Caminos Sobrantes

La línea 6 invoca al procedimiento $EligeCaminos$ (descrito más adelante) para elegir exactamente r_{kl} caminos para cada pareja k, l, tratando de minimizar el costo global del conjunto de aristas resultante.
Una vez elegidas las aristas que formarán la solución (indicadas por X), las líneas 7-12 construyen G_{sol} y P agregando cada camino cuyas aristas estén todas "presentes" en X.
Finalmente la línea 13 busca (y elimina) caminos que se pueda eliminar sin violar ningún requisito de conectividad, que podrían darse de forma similar a la vista en los algoritmos anteriores. Por ejemplo, en la Figura 4.7 se muestra un ejemplo trivial con 3 terminales y 1 camino requerido solamente para cada pareja; los caminos más cortos coinciden con las aristas a, b, c y serán por lo tanto los elegidos, aunque puede prescindirse de cualquiera de ellos sin violar los requerimientos; esto es precisamente lo que hace en este caso GeneralUpdateMatrix.

4.3.3.3. Selección Restrictiva de Caminos

El algoritmo Construcción3 se basa en computar un conjunto S que tiene más caminos que los necesarios, para luego escoger exactamente las cantidades necesarias mediante el subalgoritmo EligeCaminos. El mismo debe determinar un subconjunto de costo mínimo de S (el conjunto de caminos generados por k-caminos-más-cortos) que cumpla con los requerimientos de conectividad dados por R.[1]. Este problema puede ser modelado como un problema de programación entera empleando la siguiente notación:

- C: Matriz de costos
- R: Matriz de requerimientos de conexión
- X: Conjunto de aristas representado como se describió anteriormente
- X_{ij}: bit $(0, 1)$ indicador de la ausencia o presencia de la arista (i, j) en X
- S: conjunto de caminos a considerar

[1] Asumimos, sin pérdida de generalidad, que el grafo G es compatible con todos los requerimientos especificados por R

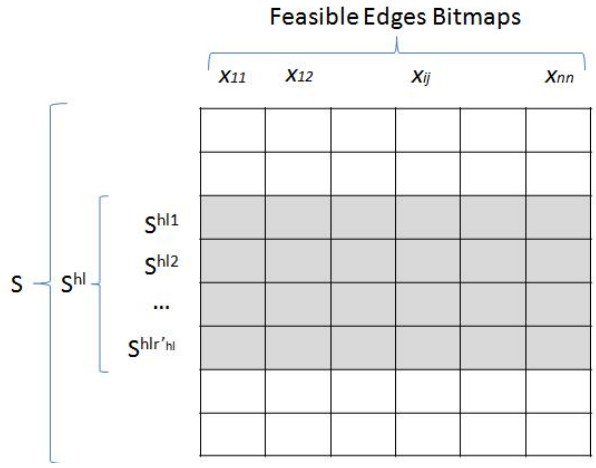


Figura 4.8: Modelo de Programación Entera para EligeCaminos

- S^{hl}: subconjunto de todos los caminos en S con extremos h, l
- S^{hlk}: camino número k en el conjunto ordenado S^{hl}
- S^{hlk}_{ij}: bit $(0, 1)$ indicador de la ausencia o presencia de la arista (i, j) en S^{hlk}

Problema: encontrar $x_{ij}, (i, j) \in E \times E$ de forma de

$$minimizar \sum_{i,j \in E} c_{ij} x_{ij} \tag{4.1}$$

sujeto a las siguientes restricciones

$$\sum_{i,j \in E} x_{ij} S^{hlk}_{ij} \geq y_{hlk} \sum_{i,j \in E} S^{hlk}_{ij} \tag{4.2}$$

$$\sum_{k=1}^{r^*_{ij}} y_{ijk} \geq r_{ij} \tag{4.3}$$

$$x_{ij} \in \{0, 1\} \forall i, j \in E, i < j \tag{4.4}$$

$$y_{hlk} \in \{0, 1\} \tag{4.5}$$

$$\forall h, l \in T, h < l, k = 1..r^*_{hl}$$

El conjunto auxiliar de variables binarias y_{hlk} indica si un cierto camino S^{hlk} es o no parte de la solución. La condición 4.2 establece que todas las aristas de caminos que tienen $y_{hlk} = 1$ deben estar en la solución. La condición 4.3 indica que al menos r_{ij} caminos deben ser incluidos para cada pareja de nodos terminales i, j.

Función Score(s)**;**

1 $score \leftarrow 0; [m, n] \leftarrow$ extremos(s)
2 por cada arista (i, j) en s
3 $\quad overlap \leftarrow 0$
4 $\quad$ por cada par de nodes $h, k \in T : (\{h, k\} \neq \{m, n\} \wedge (i, j) \in S^{hk})$
5 $\quad\quad overlap \leftarrow overlap + \frac{min(r_{hk}, |S^{hk}|)}{|S^{hk}|}$
6 $\quad$ fin por cada
7 $\quad score \leftarrow score + c_{ij}(\frac{1}{1+overlap})$
8 end por cada
9 return $\frac{1}{score}$

End Function

Figura 4.9: Función de Scoring de Caminos.

En forma acorde a la filosofía GRASP proponemos un algoritmo voraz para elegir un conjunto de caminos que cumpla los requisitos de conectividad intentando minimizar su costo global. La idea central se basa en que es deseable que un camino elegido presente las siguientes dos características:

- que tenga bajo costo;
- que comparta muchas aristas con otros caminos que tienen al menos un extremo distinto - elevando así la probabilidad de que se compartan aristas con otros caminos que serán parte de la solución final.

y por lo tanto trataremos de escoger caminos que presenten ambas características[2]. El algoritmo procederá computando un score para cada pareja como se muestra más abajo, y luego eligiendo la cantidad requerida de caminos para cada par de terminales con probabilidades proporcionales a sus scores (introduciendo así un segundo factor de aleatoriedad). El score para un camino s se computará mediante el algoritmo mostrado en la Figura 4.9.

La variable $overlap$ se usa para computar el valor esperado de la cantidad de ocurrencias de la arista (i, j) en caminos con extremos distintos a i, j. Luego, como la arista (i, j) es parte del camino s, el valor esperado total de las ocurrencias es $1 + overlap$ y entonces s será "penalizado" solamente con su fracción correspondiente de c_{ij}.
Una vez computados todos los scores, el algoritmo EligeCaminos eligirá las cantidades requeridas por la matriz R, según probabilidades proporcionales a los scores, elevando así las chances de elegir caminos convenientes.

[2]Que por su naturaleza tienden a ser antagónicas.

4.3.3.4. Propiedades de Construcción

Proposición 4.3.2 *El algoritmo Construcción3 verifica la propiedad PC2.*

Prueba. El algoritmo comienza generando conjuntos S^{hk} con al menos r_{hk} caminos para cada pareja de terminales h, k cuya conectividad disjunta no sea menor que r_{hk}. Por lo tanto, para tales parejas, el algoritmo EligeCaminos podrá elegir exactamente r_{hk} caminos, con lo cual se verifica la propiedad PC2 tal como la hemos definido.

QED

Respecto a la propiedad PC1, queda pendiente demostrar ya sea que siempre se verifica (o sea, que dada una instancia GSP y un nivel de certeza deseado de que se obtendrá un óptimo, siempre pueda hallarse α y una cantidad de iteraciones tales que se garantice dicha probabilidad); o de lo contrario encontrar un contraejemplo que demuestre que no siempre es posible hacerlo.

4.3.4. Comentarios Finales

Hemos visto que el algoritmo ConstPhase puede ser empleado indistintamente para el caso GSP-NC o GSP-EC, siempre y cuando el algoritmo empleado para determinar los k-caminos-más-cortos opere en una u otra modalidad. Hemos visto también un par de propiedades que pueden resultar de interés para la construcción de soluciones factibles, junto con variantes (Construcción1, Construcción2) al algoritmo que permiten verificar ambas propiedades. Para la sección de Pruebas de Laboratorio, haremos uso del algoritmo Construcción1, ya que en los casos habituales de aplicación sobre redes reales, la conectividad factible de las parejas de terminales sobrepasa cómodamente a los requisitos de conectividad.

Finalmente presentamos una técnica diferente para diseño de soluciones factibles, basada en un enfoque "egoísta" y "restrictivo". Dicho enfoque puede ser especialmente apropiado para ser complementado por una fase posterior de optimización local orientada a la **consolidación** de caminos, esto es, a introducir movimientos que consideren conjuntos de caminos con extremos distintos 2 a 2 y reemplacen parte de los mismos por otros conjuntos de aristas (buscando que las compartan entre sí y también eventual reutilización de aristas ya presentes en otros caminos). Dejaremos planteado el estudio y evaluación de esta alternativa como trabajo futuro de investigación.

En la sección siguiente pasaremos a estudiar la fase de optimización local de GRASP que tomará como insumo las soluciones factibles creadas por los algoritmos ya vistos para generar soluciones al GSP.

Parte III

OPTIMIZACIÓN LOCAL DE SOLUCIONES FACTIBLES

Capítulo 5

Estudio de Movimientos

En este capítulo analizaremos algunas formas de definir vecindarios y movimientos para la heurística GRASP, a partir del reemplazo de conjuntos de aristas de una solución factible por otro conjunto de aristas de modo que la solución siga siendo factible (y de menor costo). Luego analizaremos una forma específica de descomponer estructuralmente una solución, empleada en [33] para proponer algunos movimientos, viendo cómo se adaptan al caso arista-disjunto y proponiendo algunas extensiones. Luego, en la sección siguiente, formularemos los algoritmos que serán objeto de testing en la siguiente parte de este trabajo.

5.1. Generalidades

Sea G_{fac} una solución factible, siendo S una solución ejemplo de la misma, es decir, una evidencia de que aquélla es solución factible. Sea $H \subseteq E_{G_{fac}}$ un conjunto de aristas que, por efecto de un movimiento, será reemplazado por otro conjunto $H' \subseteq E_G$. En general, el grafo $G_{fac} \setminus H$ perderá algunos caminos de S (o sea, faltarán aristas que eran parte de dichos caminos), mientras que conservará otros caminos intactos.

Sea p_{ijk} el k-ésimo camino (en una cierta enumeración arbitraria) del conjunto P_{ij} de caminos ejemplo para los nodos terminales i, j. El efecto sobre p_{ijk} de la supresión de las aristas de H será la supresión de cero, uno o más "subcaminos" de p_{ijk}, en otras palabras, $p_{ijk} \setminus H$ puede seguir siendo un camino conexo, o pasar a tener 2 o más componentes conexas. Introducimos entonces la siguiente definición:

Definición 5.1.1 *Dado un camino p_{ijk} y un conjunto de aristas H llamaremos **segmento** a cada subcamino de p_{ijk} que coincida con una componente conexa de $p_{ijk} \cap H$, o sea, cada subcamino de longitud maximal que resulte suprimido de p_{ijk} al eliminar las aristas de H.*

Notación 5.1.2 *Denotaremos como $seg(p_{ijk}, H)$ al conjunto de los segmentos resultantes del camino p_{ijk} al suprimir las aristas de H, como $seg(P_{ij}, H)$ al conjunto de segmentos análogamente a un conjunto de caminos P_{ij}, y como $seg(S, H)$ al conjunto de todos los segmentos*

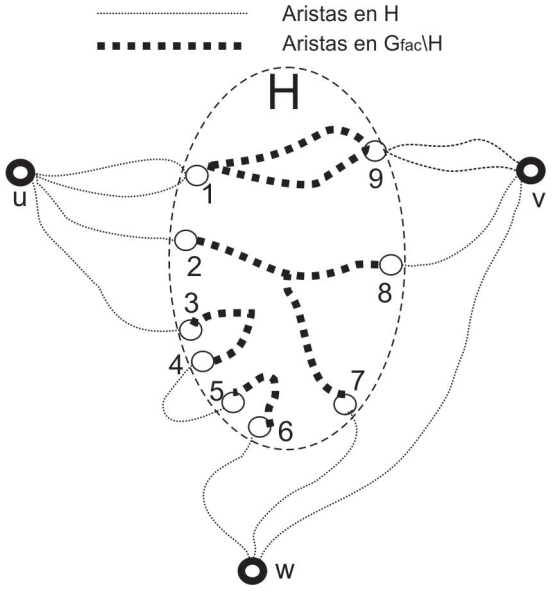


Figura 5.1: Segmentos

asociados a los caminos de una solución S. Tendremos entonces que

$$seg(p_{ijk}, H) = \{s_{ijkl}\}$$

donde s_{ijkl} *es cada uno de los segmentos resultantes de cada camino* p_{ijk}*, enumerados según* l*.*

Esto se ilustra con un ejemplo muy general en la Figura 5.1. En la misma los nodos u, v, w son terminales. Tenemos 4 caminos arista-disjuntos entre u y v, 3 caminos arista-disjuntos entre u y w, y 3 caminos arista-disjuntos entre v y w. Algunos de estos caminos (no todos) tienen varias de sus aristas en el conjunto H. En este ejemplo, se configuran los siguientes segmentos:

- 2 segmentos entre los nodos 1-9
- 1 segmento entre cada una de las parejas de nodos 2-8, 3-4, 5-6, 7-8

A su vez vemos que los segmentos 2-8 y 7-8 comparten algunas aristas.

En base a las definiciones y notaciones dadas podemos verificar que la intersección de una solución ejemplo S con el conjunto H coincide con la superposición de las aristas de todos los segmentos resultantes al suprimir las aristas de H de los caminos de S:

$$S \cap H = \left(\bigcup_{i,j \in T} P_{ij} \right) \cap H = \left(\bigcup_{i,j \in T} \bigcup_{k(i,j)} p_{ijk} \right) \cap H =$$

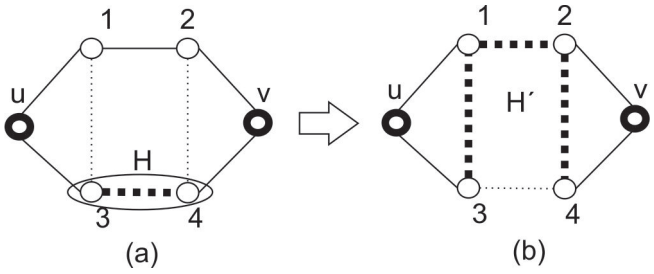


Figura 5.2: Segmentos de reemplazo y pérdida de conectividad

$$\bigcup_{i,j \in T} \bigcup_{k(i,j)} (p_{ijk} \cap H) = \bigcup_{i,j \in T} \bigcup_{k(i,j)} seg(p_{ijk}, H) = \bigcup_{i,j \in T} seg(P_{ij}, H)$$

Un movimiento, en el contexto de nuestros algoritmos, será el reemplazo en una solución factible G_{fac} del conjunto H por otro conjunto $H' \subset E_G$, tal que se conserva la factibilidad, esto es, $(G_{fac} \setminus H) \cup H'$ es también una solución factible al GSP. Nos interesa concebir movimientos cuya factibilidad o bien esté garantizada por la estructura de los mismos, o bien sea fácilmente verificable en términos del cómputo requerido. Para ello, nos centraremos en analizar movimientos que "reparen" los caminos de una solución ejemplo S afectados por la supresión del conjunto H, reemplazándola por otra solución ejemplo S', cuyos caminos ejemplo sean idénticos a los de S salvo en los segmentos $seg(S, H)$, que son reemplazados uno a uno por caminos contenidos en H' con iguales nodos extremo que los segmentos que reemplazan. Como veremos a continuación, deben tenerse en cuenta ciertas precauciones al momento de construir los segmentos de reemplazo.

5.2. Restricciones en la Selección de Segmentos de Reemplazo

Veamos la situación ilustrada en la Figura 5.2, en la cual las líneas llenas son aristas de la solución factible, las punteadas gruesas también lo son (y además conforman los conjuntos H y H') y las líneas punteadas débiles son aristas del grafo que representa los links posibles, pero no usadas por la solución factible en consideración. Supongamos que se requieren 2 caminos arista-disjuntos entre u y v. En (a), se cumple el requerimiento, como lo muestra la solución ejemplo (u-1-2-v y u-3-4-v); y definimos un conjunto H, formado por una sola arista, la 3-4. En (b) se efectuó un reemplazo del segmento 3-4 por otro segmento con los mismos extremos (3-1-2-4). Pero nótese que en (b) ya no existen 2 caminos arista disjuntos entre u y v; a lo sumo puede obtenerse uno. Obsérvese que en el conjunto H', el segmento que reemplaza al 3-4 de H no es arista-disjunto con el camino de la solución ejemplo u-1-2-v (comparten la arista 1-2).

Es natural preguntarse: ¿siempre se pierden niveles de conectividad cuando se dan superposiciones entre los segmentos de reemplazo y porciones de caminos de la solución ejemplo exteriores al conjunto H? El ejemplo mostrado en la Figura 5.3 muestra que la respuesta es negativa, esto no siempre sucede. En la parte (a) se observa que u y v están 2-conectados, como lo evidencia la solución ejemplo (u-1-2-v y u-3-4-5-v); a su vez, 2 es el máximo nivel

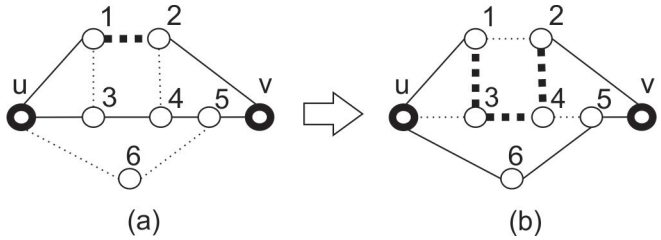


Figura 5.3: Segmentos de reemplazo y pérdida de conectividad

de arista conectividad. En la parte (b) se observa que, a pesar de un reemplazo de segmento que se superpone a uno de los caminos de la solución ejemplo (arista 3-4) es posible conformar una nueva solución ejemplo que testimonia la 2-conectividad (siendo la misma la pareja u-1-3-4-2-v y u-6-5-v).

Al momento de considerar un cierto conjunto H' para reemplazo de un conjunto H de aristas, se puede entonces aplicar dos criterios diferentes:

- Evitar este tipo de superposiciones (descartar conjuntos H' que las inducen)
- Considerar también conjuntos que induzcan superposiciones, verificando si se conserva el nivel de conectividad requerido para todo par de terminales con caminos que sufren este tipo de superposición (criterio más costoso en cómputo, pero más inclusivo).

En lo que resta de este trabajo optaremos por el primer criterio, es decir, evitaremos este tipo de superposición “riesgosa”.

5.3. Propuesta y Estudio de Movimientos

5.3.1. Descomposición de Soluciones

Al momento de diseñar movimientos, dos atributos deseables a perseguir son: que sea sencillo definir los conjuntos H de aristas a reemplazar, y que sea sencillo encontrar un reemplazo para el mismo que reponga factibilidad (de una cierta solución ejemplo). Por ”sencillo” en este contexto entendemos que se requiera un esfuerzo de cómputo reducido.
Una descomposición de la solución factible que puede emplearse en el contexto del SPG es la descomposición en **key-paths** y **key-trees** ([39]); los movimientos que propondremos harán uso de esta descomposición para determinar conjuntos de aristas H convenientes. A continuación definimos los conceptos y la notación necesarios:

Definición 5.3.1 *Dada una solución factible a una instancia del GSP, un* ***key-nodo*** *es todo nodo no terminal al cual inciden aristas de dicha solución y cuyo grado es igual o mayor a 3 en la misma.*

Definición 5.3.2 *Dada una solución factible a una instancia del GSP, un* ***key-path*** *es cualquier camino simple de la misma, cuyos extremos son ambos terminales, o ambos key-nodos, o uno*

terminal y el otro key-nodo; y cuyos nodos intermedios son no terminales con grado 2 en dicha solución.

Definición 5.3.3 *Dada una instancia del GSP, una solución factible G_T, y un nodo $v \in V_{G_T}$ que es terminal o key-nodo, definimos como la **key-star** asociada a v en G_T al subgrafo de G_T que se forma mediante la unión de todos los key-paths que parten desde v.*

Las siguientes definición y proposición permiten dar una caracterización estructural de las key-stars.

Definición 5.3.4 *Llamaremos **multipath** a un conjunto de caminos simples arista-disjuntos, tales que todos empiezan y terminan en la misma pareja de nodos u, v. En particular, un camino simple es un multipath trivial.*

Proposición 5.3.5 *La key-star asociada a un key-nodo v en una solución factible G_T, es la unión de uno o más multipaths arista-disjuntos entre sí, y tales que el nodo v está en todos, y es el único nodo compartido entre cualquier par de multipahts.*

Prueba. Si existe un único key-path que parte de v, la demostración es trivial. Supongamos ahora que existen al menos dos. Sea X el conjunto de los extremos de los key-paths de la key-star distintos a v, y sea $k_x = \{\text{key-paths } v...x\}$ con $x \in X$. Para cada conjunto de caminos k_x se cumple que:

- todos sus caminos tienen iguales extremos v y x;
- todos son caminos simples;
- ninguna pareja de ellos tienen nodos en común salvo v y x puesto que los demás nodos tienen todos grado 2, dado por 2 aristas incidentes de un mismo camino del conjunto

Por lo tanto, los k_x cumplen con la definición de multipath. Por otro lado, si $x_1, x_2 \in X, x_1 \neq x_2$, entonces por idéntico argumento (grados intermedios 2) k_{x_1} y k_{x_2} solamente comparten el nodo v; y hemos considerado todos los key-paths que parten de v pues todos terminan en uno de los elementos de X. De este modo hemos demostrado que la key-star consiste en un conjunto $\{k_x, x \in X\}$ de multipaths arista-disjuntos entre sí cuyo único nodo en común es v, tal como queríamos demostrar.

QED

En la Figura 5.4 se muestra un ejemplo de key-star con 3 multipaths que unen al nodo v con los nodos x_1, x_2, x_3, que se supone son nodos terminales y/o key-nodos de una solución factible de la cual esta key-star es parte de la descomposición.

En el contexto de un problema GSP-NC, dada una solución factible minimal en terminos de key-paths, esto es, en la cual no puede suprimirse ningún key-path sin pérdida de factibilidad, se cumple que las key-stars son árboles, es decir, key-stars acíclicas, en las que todos los multipaths son triviales; les llamaremos **key-trees** al igual que en la literatura previa. (Cabe observar que en [33] sólo se define el concepto de key-tree con nodos de Steiner como centro; nosotros extenderemos el concepto a nodos de Steiner y terminales).

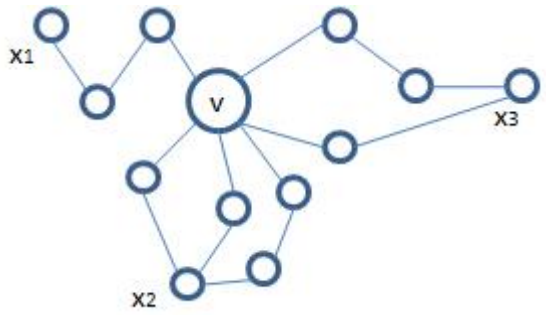


Figura 5.4: Ejemplo de key-star con 3 multipaths

5.3.2. Movimientos Propuestos

A continuación analizaremos diferentes mecanismos para la fase de optimización local del algoritmo GRASP, que a partir de una solución factible, mediante movimientos hacia soluciones vecinas, alcance una solución factible de menor costo que no pueda ser mejorada por posteriores movimientos. Para ello definiremos en cada caso el concepto de "vecindad", basados en la estructura ya vista de key-paths y key-stars. Luego formularemos algoritmos para implementar la fase de optimización local combinando distintas vecindades, sometiéndolos posteriormente a testing.
A efectos de definir las vecindades deberemos especificar cuáles son los eventuales conjuntos candidatos H de aristas a reemplazar en cada movimiento. Veremos como, al construir estos conjuntos a partir de la unión de key-paths, la determinación de los conjuntos de reemplazo H' se simplifica, consistiendo en general en la determinación de nuevos key-paths para reemplazar los eliminados. Para apoyar esta afirmación damos la siguiente proposición.

Proposición 5.3.6 *Dada una instancia del GSP, una solución factible G_{fac} con su key-descomposición (o sea su descomposición en key-paths), y una solución ejemplo S, todo segmento resultante de la eliminación de un conjunto de aristas H formado a partir de la unión de uno o más key-paths, consiste precisamente en unión de uno o más de dichos key-paths.*

Prueba. Sea s uno cualquiera de los caminos de la solución ejemplo S. Dicho camino consiste en una secuencia de key-paths, que se recorren (y se obtienen) del siguiente modo:

- se parte del nodo origen del camino recorriendo sus aristas hacia el nodo destino
- cada vez que se visita un nodo n: si es terminal o tiene grado mayor a 2, se acaba de recorrer un key-path (y se comienza a recorrer uno diferente); si es no terminal de grado menor que 3, se está ante un nodo interno de un key-path.

Siendo entonces los caminos de S secuencias de key-paths, y siendo que el conjunto H a reemplazar consiste en la unión de uno o más key-paths, se sigue inmediatamente el resultado propuesto.

QED

En adelante entonces definiremos las vecindades en función de conjuntos H compuestos por diversas formas de combinar key-paths; en virtud de la proposición 5.3.6 sabemos que los segmentos resultantes serán también key-paths, lo cual simplificará la tarea de formular conjuntos de reemplazo H' para H.

5.3.2.1. Optimización1: reemplazo simple de key-paths

En términos intuitivos, la idea de este primer "movimiento" consiste en reemplazar un key-path con extremos u, v por otro key-path con los mismos extremos pero cuyo costo sea menor.

Definición 5.3.7 *Dada una instancia del GSP y una solución factible G_{fac}, definimos como vecina a toda solución factible G'_{fac} que se pueda obtener reemplazando a cualquiera de sus key-paths k por otro camino simple con su mismo origen y extremo.*

Como ya hemos visto, deben tomarse ciertas precauciones al reemplazar un segmento por otro con iguales extremos, puesto que podrían perderse niveles de conectividad. En este primer movimiento propuesto, restringiremos el concepto de vecindad del siguiente modo.

Definición 5.3.8 *Dada una instancia del GSP y una solución factible G_{fac}, definimos como **vecindad1** al conjunto de todos los grafos que se obtienen reemplazando a cualquiera de sus key-paths k por otro camino simple de menor costo, con su mismo origen y extremo, que no emplee ninguna arista de G_{fac} salvo algunas eventualmente del propio k.*

Dada una solución G_{fac}, el conjunto de soluciones vecinas dadas por **vecindad1** es un subconjunto del antes definido; pero para el cual puede demostrarse fácilmente que se mantiene la factibilidad (evitándose el problema de pérdida de niveles de conectividad ya visto) como muestra la siguiente proposición.

Proposición 5.3.9 *Dada una solución factible G_{fac} y un key-path a reemplazar k, cualquiera de los grafos de su vecindad son también soluciones factibles.*

Prueba. Sea S una solución ejemplo que da evidencia de que G_{fac} es solución factible. Sea p un camino cualquiera de la misma. Si el camino no incluía al key-path k, entonces estará intacto en cualquier grafo de la vecindad, puesto que las únicas aristas que se pierden al pasar a un vecino son algunas (eventualmente todas) las de k. Si el camino sí incluía a k, entonces será reemplazado por otro camino que es idéntico, salvo en el subcamino k que pasa a ser un subcamino k'; y por la definición de **vecindad1** dicho camino no emplea ninguna arista presente en otro camino de S que una los mismos extremos. Como sea entonces vemos que cualquier grafo vecino tendrá tantos caminos ejemplo como tenía S, con lo que queda probado que serán soluciones factibles.

QED

Algoritmo Optimiza1 (G, C, T, S)**;**

```
1  mejorar ← TRUE
2  κ ← k_descomponer(S)
3  mientras (mejorar) hacer
4    mejorar ← FALSE
5    por cada kpath k ∈ κ con extremos u, v hacer
6      G' ← el subgrafo inducido de G por E(k) ∪ (E \ E(S))
7      k' ← camino_más_corto (G', C, u, v)
8      si costo(k') < costo(k)
9        S ← (S \ k) ∪ k'
10         κ ← κ \ {k} ∪ {k'}
11         mejorar ← TRUE
12         abortar por cada
13       fin si
14   fin por cada
15 fin mientras
16 return S

Fin Función
```

Figura 5.5: Algoritmo Optimiza1.

En la Figura 5.5 se presenta un algoritmo de búsqueda local **Optimiza1** basado en la **vecindad1**. El mismo toma como entrada un grafo G (grafo de conexiones factibles del GSP), la matriz de costos C, los terminales T y una solución ejemplo S; y devuelve como resultado otra solución ejemplo con tantos caminos como S, con un costo menor o igual al de S (un mínimo local alcanzado según la secuencia de movimientos que se hagan). El loop 5-14 va probando el reemplazo de cada key-path, hasta que encuentra un reemplazo que disminuye el costo. En la línea 6 se computa el grafo G' que se obtiene al suprimir de G las aristas de la solución actual S menos las del key-path k a reemplazar; sobre este grafo G' luego se busca el camino más corto empleando el algoritmo de Dijkstra que una los mismos extremos que k. Nótese que la descomposición en key-paths se debe hacer solamente al principio, puesto que en adelante cada camino de la solución ejemplo mantendrá la misma secuencia de key-paths. Esta descomposición tiene orden $O(sm)$ siendo s la cantidad de caminos ejemplo (la sumatoria de los requerimientos del problema) y m la cantidad de aristas de G, ya que requiere recorrer una vez sola cada camino de S.
El algoritmo finaliza cuando se prueban todos los key-paths de la solución ejemplo actual y no es posible mejorar el costo de ninguno de ellos.

En caso de que se reemplace un key-path por otro de menor costo (línea 10) algunas aristas del mismo pasan a quedar sin emplear en la solución; con lo cual, eventualmente, pueden ser aprovechadas para mejorar key-paths que ya fueron evaluados sin éxito previamente. Es por ello que se aborta la recorrida de key-paths para volver a recomenzarla luego de cada mejora

encontrada. Así planteado, nuestro algoritmo es del tipo "first improving".

5.3.2.2. Optimización2: reemplazo complejo de key-paths

El algoritmo Optimización1 hace uso de una definición de vecindad para la cual es sencillo demostrar que se preserva la factibilidad, pero que deja de lado la posibilidad de emplear las aristas de la solución actual con lo cual el espacio de soluciones vecinas probadas se ve reducido. En la práctica se observa que esta restricción afecta sensiblemente los resultados obtenidos, es decir, se detiene al alcanzar costos sensiblemente mayores a los que se puede lograr si se consideran dichas aristas, como en el algoritmo que propondremos a continuación.

Nuevamente, en términos intuitivos, la idea de este segundo movimiento consiste en reemplazar un key-path con extremos u, v por otro key-path con los mismos extremos, cuyo costo sea menor, eligiendo sus aristas de un conjunto cuidadosamente determinado, más inclusivo que el de la Optimización1. Para establecer este conjunto necesitamos introducir cierta notación previa.

Notación 5.3.10 *Sea una solución ejemplo S y uno de sus key-paths k. Denotaremos como $J_k(S)$ al conjunto de caminos dado por*

$$J_k(S) = \{p \in S : k \subseteq p\}$$

En otras palabras, es el conjunto de los caminos ejemplo que emplean (contienen) al key-path k.

Notación 5.3.11 *Sea una solución ejemplo S y uno de sus key-paths k. Denotaremos como $\chi_k(S)$ al conjunto de aristas dado por*

$$\chi_k(S) = \bigcup_{q=i\ldots j \in J_k(S)} E(P_{ij} \setminus q)$$

El conjunto $\chi_k(S)$ contiene todas las aristas de la actual solución ejemplo que podrían ocasionar problemas de pérdida de niveles de conectividad en caso de usarse para reemplazar al key-path k, como ya lo vimos al introducir el capítulo de movimientos. En primer lugar se determina cuáles son los caminos $J_k(S)$ que emplean a ese key-path. Luego se identifica las aristas de aquellos caminos que unen iguales extremos que alguno de los caminos q de $J_k(S)$ como potencialmente problemáticas, aunque dejando de lado las del propio camino q. Estamos ahora en condiciones de definir la segunda vecindad.

Definición 5.3.12 *Dada una instancia del GPS y una solución factible G_{fac}, definimos como **vecindad2** al conjunto de todos los grafos que se obtienen reemplazando a cualquiera de sus key-paths k por un camino de costo menor, con su mismo origen y extremo, que solamente emplee aristas de $E_G \setminus \chi_k(S)$ y/o del propio k.*

Nótese que $\chi_k(S) \subseteq E(S)$, es decir que en esta definición se emplean al menos las mismas y en general más aristas para buscar el reemplazo de k que en la **vecindad1**. En la práctica,

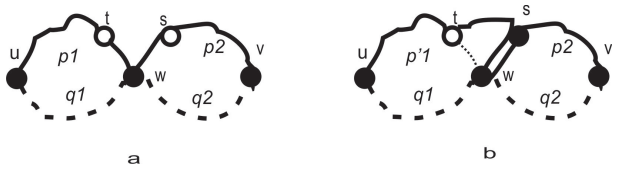


Figura 5.6: Cambio en estructura de k-descomposición

como veremos, esto implica niveles de optimización sensiblemente mejores.

Veamos que también en este caso los grafos de la vecindad son soluciones factibles.

Proposición 5.3.13 *Dada una solución factible G_{fac} y un key-path a reemplazar k, cualquiera de los grafos de su vecindad son también soluciones factibles.*

Prueba. Es análoga a la correspondiente para la **vecindad1**. Cualquier camino p que empleara al key-path k tendrá nuevas aristas elegidas fuera de las aristas que componían al resto de los caminos ejemplo con iguales extremos que p; de este modo, se sigue teniendo tantos caminos ejemplo que unen los extremos de p como había en S, lo cual prueba la proposición.

QED

En la Figura 5.7 se presenta un algoritmo de búsqueda local **Optimiza2** basado en la **vecindad2**. El mismo toma como entrada un grafo G (grafo de conexiones factibles del GSP), la matriz de costos C, los terminales T y una solución ejemplo S; y devuelve como resultado otra solución ejemplo con tantos caminos como S, con un costo menor o igual al de S (un mínimo local alcanzado según la secuencia de movimientos que se hagan). El loop 5-19 va probando el reemplazo de cada key-path, hasta que encuentra un reemplazo que disminuye el costo. En la línea 6 se computa el grafo G' que se obtiene al suprimir de G las aristas dadas por el conjunto $\chi_k(S)$ pero sí permitiendo el uso de las aristas del key-path k. Al igual que en la vecindad anterior, sobre este grafo G' luego se busca el camino más corto empleando el algoritmo de Dijkstra que una los mismos extremos que k. A diferencia del caso anterior, aquí se permite el reuso de algunas aristas ya presentes en la solución; es por ello que se construye una matriz de costos C' modificada, en la cual se asigna costo 0 a las aristas ya presentes en otros caminos ejemplo, en forma similar a como hicimos en los algoritmos de la fase de construcción.

En el caso de este algoritmo, puede suceder que se creen o eliminen key-paths (más allá del mero reemplazo) puesto que pueden aparecer o desaparecer nodos de Steiner, como se ilustra en la Figure 5.6. En dicha figura, parte (a), se muestra 2 key-paths p_1, q_1 que unen los key-nodos u, w y 2 key-paths p_2, q_2 que unen los key-nodos w, v. En la parte (b) se ha reemplazado el key-path p_1 por otro key-path p'_1 que tiene un primer tramo común al "viejo" key-path p_1, luego un tramo con aristas nuevas (que no estaban usadas en la solución) y finalmente un tramo que ya era parte de la solución (parte del key-path p_2). Obsérvese que ahora el nodo s pasa a ser un nuevo key-nodo y $w...s$ un nuevo key-path. A su vez, si la secuencia $p_1 - p_2$ era parte

Algoritmo Optimiza2 (G, C, T, S)**;**
1 $mejorar \leftarrow$ TRUE
2 $\kappa \leftarrow$ k_descomponer(S)
3 mientras $(mejorar)$ hacer
4 $mejorar \leftarrow$ FALSE
5 por cada kpath $k \in \kappa$ con extremos u, v hacer
6 $G' \leftarrow$ el subgrafo inducido de G por $E(k) \cup (E \setminus \chi_k(S))$
7 $C' \leftarrow (c'_{ij})/c'_{ij} = 0$ si $(i,j) \in S \setminus k, c'_{ij} = c_{ij}$ si no
8 $k' \leftarrow$ camino_más_corto (G', C', u, v)
9 si costo$(k') <$ costo(k)
10 $mejorar \leftarrow$ TRUE
11 actualizar $S : \forall p \in J_k(S) p \leftarrow (p \setminus k) \cup k'$
12 si $\exists z \in V(k'), z \notin \{u, v\}$, grado$(z) \geq 3$ en S
13 quitar_ciclos $(J_k(S))$
14 $\kappa \leftarrow$ k_descomponer(S)
15 si no
16 $\kappa \leftarrow \kappa \setminus \{k\} \cup \{k'\}$
17 fin si
18 fin si
19 fin por cada
20 fin mientras
21 return S

Fin Función

Figura 5.7: Algoritmo Optimiza2.

de un camino ejemplo de la solución ejemplo, la nueva secuencia $p'_1 - p_2$ introduce aristas repetidas (se recorre 2 veces el nuevo key-path), por lo que es necesario eliminar este ciclo del camino ejemplo en cuestión. En consecuencia, en estos casos (cuando se identifica la aparición de nodos de Steiner, línea12) se debe recomputar la k-descomposición de la solución actual, y tomar la precaución de eliminar eventuales ciclos que se hayan formado en los caminos ejemplo.
Al igual que antes, el algoritmo finaliza cuando se prueban todos los key-paths de la solución ejemplo actual y no es posible mejorar el costo de ninguno de ellos.

5.3.2.3. Optimización3: reemplazo de key-stars

La potencialidad de mejora local del reemplazo de key-paths se ve rápidamente limitada por casos donde es necesario trabajar con estructuras más grandes (conjuntos H más complejos) para lograr la introducción de mejoras en el costo de una solución factible. Por ejemplo, la Figura 5.8 muestra un ejemplo sencillo; en la misma, los nodos negros son terminales y los blancos son nodos opcionales, las aristas llenas son las de la solución actual y las punteadas son aristas no empleadas del grafo de conexiones factibles. La parte (a) muestra una solución

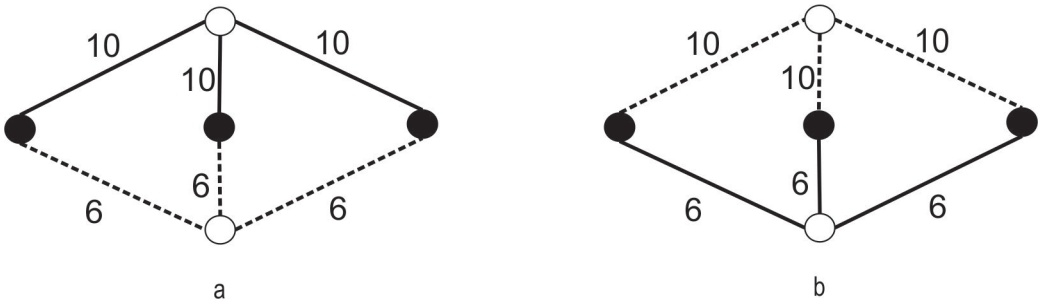


Figura 5.8: Ej. de movimientos introducidos gracias al reemplazo de key-stars

factible de costo 30, en la cual no es posible reemplazar ningún key-path por otro de menor costo, aún reutilizando aristas de la solución. Pero la key-star puede ser reemplazada como un todo por la mostrada en la parte (b), con un costo total de 18.

Resulta interesante entonces plantearse el reemplazo de key-stars en un paso, esto es, movimientos en los cuales el conjunto H consiste en una key-star. En dichos casos, los segmentos resultantes de suprimir la key-star consistirán en uno de los siguientes casos:

- 2 key-paths que comparten el nodo central de la key-star; los otros extremos de dichos key-paths pueden ser key-nodos y/o nodos terminales;
- 1 key-path que tiene al nodo central de la key-star como uno de sus extremos; nótese que esto sólo puede suceder si dicho nodo central es un nodo terminal

En el contexto de este trabajo permitiremos que el centro de una key-star sea ocupado por un nodo terminal. Esto habilita casos como el segundo arriba mencionado, extendiendo así la potencialidad de los movimientos propuestos.

Para el caso del GSP-NC, en [33] se propone el reemplazo de cualquier key-tree por otro árbol cualquiera con idénticas hojas, mostrándose que esto preserva la factibilidad; se sugiere entonces un algoritmo que incluye una heurística local para generar árboles con tales características intentando encontrar uno que reduzca el costo. El hecho de vedar la repetición de nodos en caminos disjuntos deja de lado algunos problemas que sí se presentan en el caso del GSP-EC.

Un problema que puede surgir es el de la pérdida de niveles de conectividad ya visto. En la Figura 5.9 se muestra un ejemplo de ello. En la parte (a) se ilustran 2 caminos arista disjuntos que unen a los nodos terminales u, v, a saber, $ueifv$ y $ugihv$. Consideremos el reemplazo de la key-star con centro i y hojas e, f, g, h. En la parte (b) se muestra un eventual reemplazo por una estructura de árbol con las mismas hojas y 2 nodos internos j, k. Nótese que en este caso se ha perdido un nivel de conectividad, ya que los 2 caminos ahora compartirían una arista, la (j, k). Este sencillo ejemplo muestra que, para el reemplazo de key-stars en el caso arista disjunto,

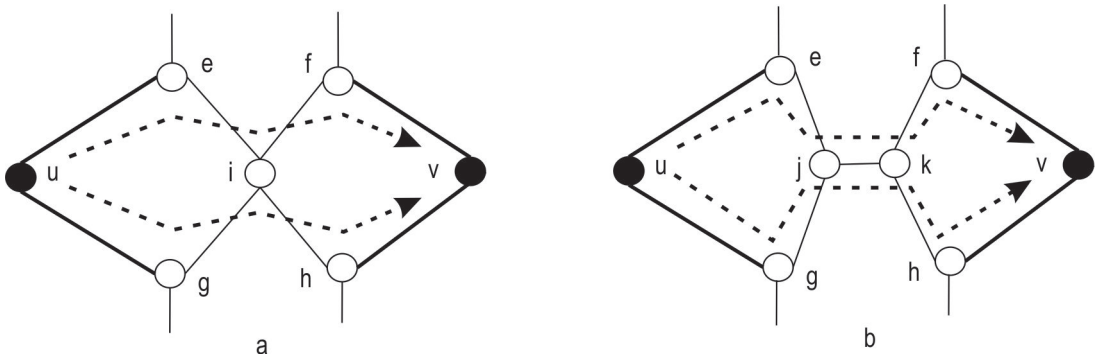


Figura 5.9: Potenciales problemas al reemplazar key-stars

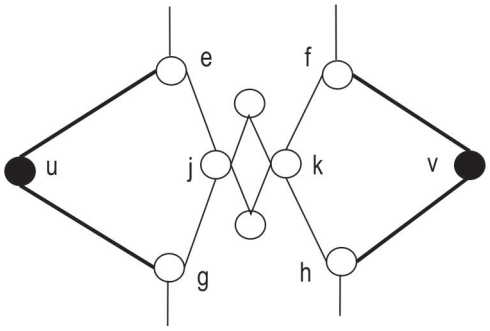


Figura 5.10: Reemplazo cíclico para key-star acíclica

deben tomarse las precauciones ya vistas al introducir los movimientos en forma general. De todos modos es interesante mencionar un par de observaciones:

- puede darse que una key-star que sea un árbol (o sea una key-star acíclica o key-tree) encuentre su mejor reemplazo en forma de estructura cíclica; por ejemplo la figura Figura 5.10 podría eventualmente ser el reemplazo óptimo y sin pérdida de caminos para el caso planteado en la Figura 5.9.

- en el caso de que la key-star a reemplazar tenga tres “hojas”, este problema no se manifiesta, puesto que es imposible que dicha key-star estuviera en uso por más de un camino ejemplo para la misma pareja de nodos terminales.

Pueden plantearse heurísticas similares a las empleadas en [33] para diseñar estructuras que reemplacen a la key-star; pero en general, presentan las siguientes características:

- siendo heurísticas propondrán un movimiento “aceptablemente bueno”, pero en general no óptimo

- es necesario comprobar que no introduzcan aristas como la (j, k) de la Figura 5.9; como veremos en una sección posterior, diseñar algoritmos que intrínsecamente las eviten consiste en un complejo problema muy similar al GSP, que de hecho es una extensión del mismo en el sentido de que introduce nuevas restricciones.

Nosotros propondremos a continuación un movimiento que en forma determinística y con orden polinomial logra reemplazar a cualquier key-star por la key-star de menor costo posible que permita reparar los caminos ejemplo. Para ello definimos en forma precisa la noción de vecindad asociada al movimiento a continuación.

Definición 5.3.14 *Dada una instancia del GSP y una solución factible G_{fac}, definimos como* ***vecindad3*** *al conjunto de todos los grafos que se obtienen reemplazando a cualquiera de sus key-stars k por otra key-star de menor costo que preserve la misma conectividad 2 a 2 entre las hojas de k y nodos terminales involucrados; empleando la key-star de reemplazo solamente aristas no usadas en G_{fac} salvo eventualemente algunas de k.*

Como ya hemos mencionado, nosotros proporcionaremos un algoritmo determinista y de tiempo polinómico para determinar la key-star que debe utilizarse para reemplazar una cualquiera. A continuación introducimos cierta notación necesaria y discutiremos los distintos casos que pueden plantearse.

Notación 5.3.15 *Denotaremos como θ_k al nodo central de una key-star k dada.*

Notación 5.3.16 *Denotaremos como ψ_k al conjunto de nodos hoja de una key-star k dada.*

Notación 5.3.17 *Dada una key-star k y uno de sus nodos centro u hoja m denotaremos como $\hat{\delta}_{k,m}$ a la máxima cantidad de key-paths que unen a m con otro nodo centro u hoja.*

Pueden presentarse distintos casos en función de que el nodo central de la key-star a reemplazar k sea o no terminal, y de que el nodo central de la key-star reemplazante k' sea o no una de las hojas de k; las 4 posibilidades se ilustran en la Figura 5.11.

En dicha figura, los nodos t, u, v son hojas de una key-star k que será reemplazada por otra key-star k'. Veamos los casos posibles:

- **Caso (a):** $\theta_k \notin T \wedge \theta_{k'} \in \psi_k$. En este caso una de las hojas pasa a oficiar como nuevo centro de key-star (de k'). Desde dicho nodo deberán surgir tantos key-paths hacia cada una de las otras hojas como éstas tenían con el centro de k.

- **Caso (b):** $\theta_k \notin T \wedge \theta_{k'} \notin \psi_k$. En este caso el nuevo centro es un nodo cualquiera, distinto de las hojas de k; se debe mantener la misma cantidad de key-paths centro-hoja para cada una.

- **Caso (c):** $\theta_k \in T \wedge \theta_{k'} \in \psi_k$. En este caso el nuevo centro es una de los nodos que era hoja en k; pero para mantener el nivel de conectividad entre las hojas, se debe dotar a cada hoja m de k' de $\hat{\delta}_{k,m}$ key-paths que la unan con el nuevo centro. En la figura se ilustra esto para mantener doble conectividad entre t y θ_k.

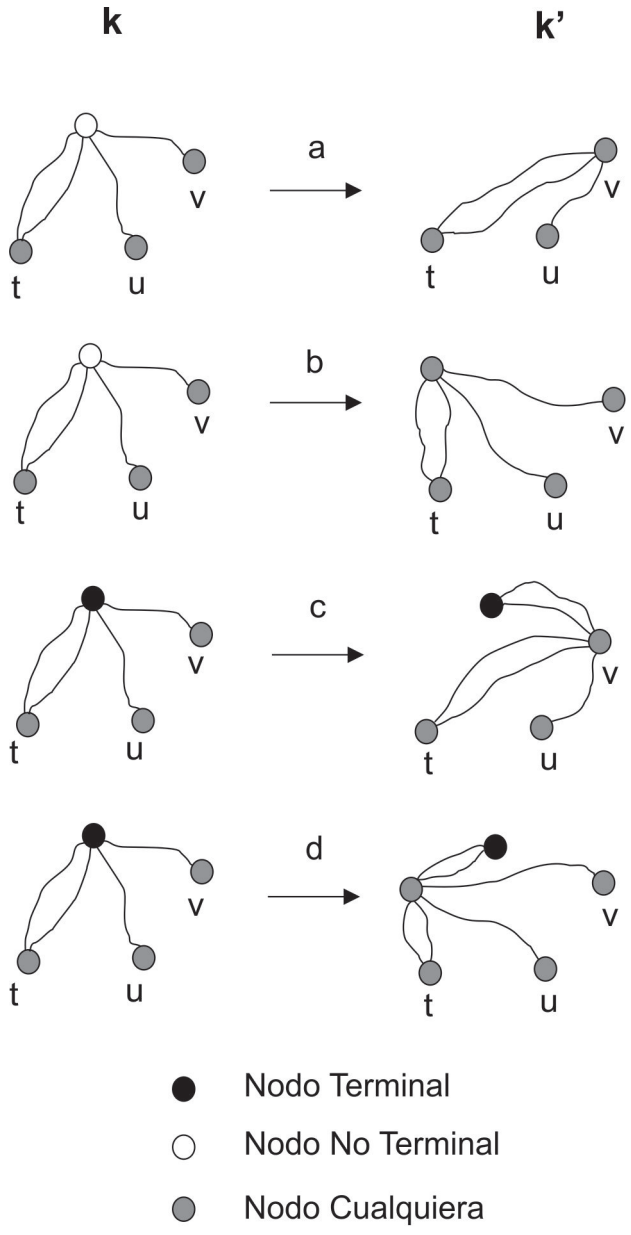


Figura 5.11: 4 casos de movimientos de key-stars

- **Caso (d):** $\theta_k \in T \wedge \theta_{k'} \notin \psi_k$. En este caso, el nodo θ_k es terminal y por lo tanto no puede ser "descartado"; surge un nuevo centro que puede o no ser terminal, con lo cual se pasa a tener una key-star k' con una hoja más que k. Nuevamente vemos que es necesario mantener $\hat{\delta}_{k,m}$ key-paths partiendo de cada hoja de k'. En la figura se ilustra esto para mantener doble conectividad entre t y θ_k.

Basados en estas ideas, a efectos de determinar cuál es la mejor key-star k', proponemos emplear una técnica basada en el uso de algoritmos (antes comentados) para determinar los n caminos de menor peso entre dos nodos de un grafo con aristas ponderadas. En la Figura 5.13 se muestra el algoritmo para tales efectos, que recibe como entrada la instancia del problema G, los costos C, los terminales T, la solución ejemplo S para la solución factible actual, y la key-star k a reemplazar. Se ilustra un ejemplo en la parte (a) de la Figura 5.12. La línea 1 genera un grafo G' a partir de G suprimiendo las aristas usadas en S salvo las de la propia key-star k cuyo uso se permitirá. La línea 2 agrega a G' un nodo virtual w que funcionará a modo de "sumidero" como veremos más adelante. Las líneas 3 y 4 generan el conjunto de nodos que será necesario interconectar en la nueva key-star. En las líneas 5-7 se conectan dichos nodos al nodo especial w, como se ilustra en la parte (b), siendo éstas las únicas aristas con que contará dicho nodo, cuya cantidad denotamos como $\delta_{G,w}$. Luego las líneas 9-14 iteran los posibles nodos z candidatos a ser el centro de la nueva key-star, que puede ser cualquier nodo (salvo w), inclusive los de Ω. Entonces, se efectúa la búsqueda de $\delta_{G,w}$ caminos arista disjuntos que unan los nodos z y w ("fuente" y "sumidero") en la línea 10; esto se ilustra en la parte (c), donde se solicitan 5 caminos. Esto sólo será posible hallando caminos arista disjuntos que unan los nodos de Ω con z, tantos para cada nodo como aristas se les agregó con w; así se conformará una nueva key-star (si se ignora el nodo w). Si fue posible hallarlos, y en caso de ser su costo agregado menor al de la mejor key-star hallada hasta el momento, se está ante una nueva "mejor key-star" (líneas 11-13), que se obtiene descartando el nodo w y sus aristas como ilustra la parte (d). De este modo contamos con un algoritmo con tiempo de ejecución polinómico para determinar el reemplazo óptimo de la key-star.

Finalmente, mostramos ahora la estructura que presentaría un algoritmo de búsqueda local basado en la **vecindad3** en la Figure 5.14; en realidad los mejores resultados se obtienen combinando estos movimientos con los de la **vecindad2**, por lo cual en la fase de testing emplearemos un algoritmo combinado que en cada paso agota primero la posibilidad de reemplazo de key-paths para luego intentar reemplazo de key-stars.

5.3.2.4. Optimización4: reemplazo complejo de key-stars

Luego de formular la **vecindad1**, en la **vecindad2** procuramos explotar la reutilización de aristas ya usadas en la solución actual. Podemos hacer algo análogo para la **vecindad3**, definiendo entonces una **vecindad4** que busque la mejor key-star, con uso de aristas ya introducidas en la solución que no ocasionen potenciales problemas de pérdida de niveles de conectividad. Previamente, redefinimos nuestros operadores J y χ para poder aplicarlos sobre key-stars.

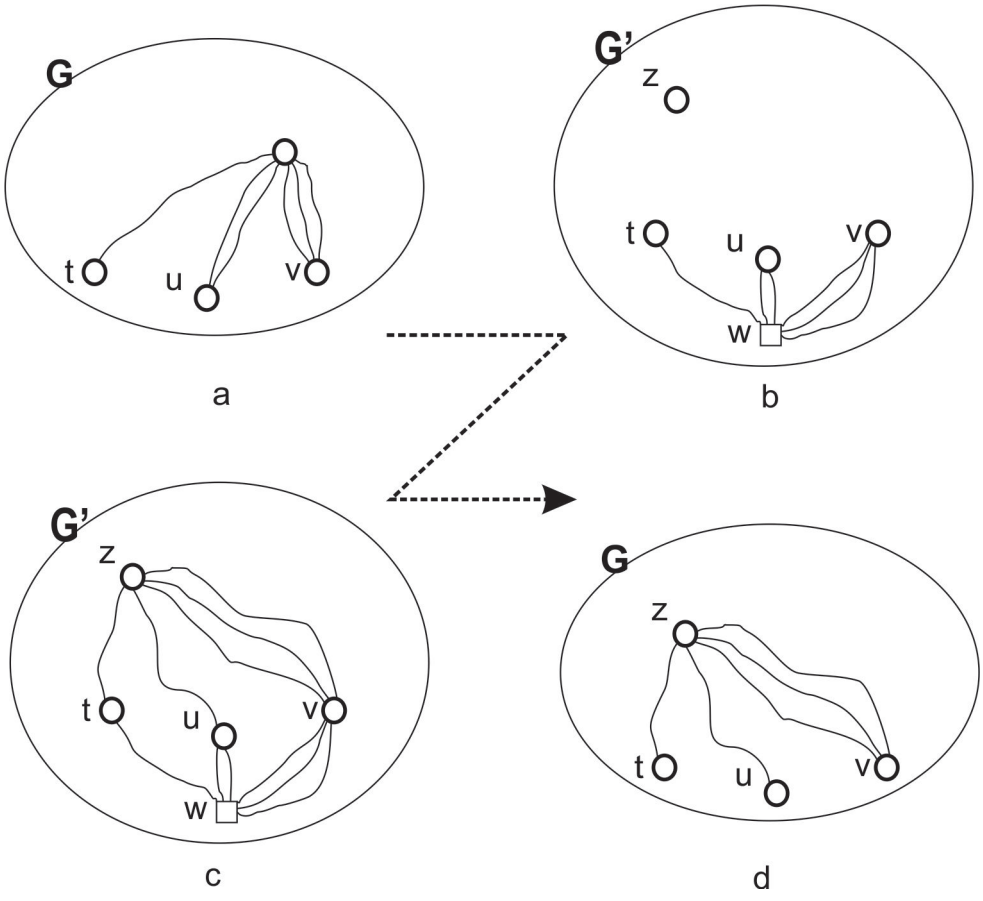


Figura 5.12: Funcionamiento del Algoritmo MejorKeyStar

Algoritmo MejorKeyStar (G, C, T, S, k)**;**
1 $G' \leftarrow G - E(S) + E(k)$
2 agregar un nodo virtual w a G'
3 $\Omega \leftarrow \psi_k$
4 si $\theta_k \in T$, $\Omega \leftarrow \Omega \cup \{\theta_k\}$
5 por cada $m \in \Omega$
6 agregar $\hat{\delta}_{k,m}$ aristas paralelas (w, m) a G' con costo 0
7 fin por cada
8 $c_{min} \leftarrow 0$; $k_{min} \leftarrow k$
9 por cada $z \in V(G)$
10 $k' \leftarrow$ caminos_más_cortos$(G', \delta_{G,w}, z, w)$
11 si k' tiene $\delta_{G,w}$ caminos $\wedge$ costo$(k') < c_{min}$
12 $c_{min} \leftarrow$costo(k'); $k_{min} \leftarrow k'$
13 fin si
14 fin por cada
15 return k_{min}

Fin Función

Figura 5.13: Algoritmo MejorKeyStar.

Algoritmo Optimiza3 (G, C, T, S)**;**
1 $mejorar \leftarrow$ TRUE
2 $\kappa \leftarrow$ k_descomponer(S)
3 mientras ($mejorar$) hacer
4 $mejorar \leftarrow$ FALSE
5 por cada kstar $k \in \kappa$ hacer
6 $k' \leftarrow$ MejorKeyStar(G, C, T, S, k)
7 si costo$(k') <$ costo(k)
8 $mejorar \leftarrow$ TRUE
9 reemplazar k por k' en los caminos de S
10 $\kappa \leftarrow$ k_descomponer(S)
11 fin si
12 fin por cada
13 fin mientras
14 return S

Fin Función

Figura 5.14: Algoritmo Optimiza3.

Notación 5.3.18 *Sea una solución ejemplo S y una de sus key-stars k. Denotaremos como $J_k(S)$ al conjunto de caminos dado por*

$$J_k(S) = \{p \in S : p \cap k \neq \emptyset\}$$

En otras palabras, es el conjunto de los caminos ejemplo que hacen uso de la key-star k.

Notación 5.3.19 *Sea una solución ejemplo S y una de sus key-stars k. Denotaremos como $\chi_k(S)$ al conjunto de aristas dado por*

$$\chi_k(S) = \bigcup_{q=i...j \in J_k(S)} E(P_{ij} \setminus q)$$

Definición 5.3.20 *Dada una instancia del GSP y una solución factible G_{fac}, definimos como **vecindad4** al conjunto de todos los grafos que se obtienen reemplazando a cualquiera de sus key-stars k por la key-star de menor costo posible que preserve la misma conectividad 2 a 2 entre las hojas de k y nodos terminales involucrados; empleando la key-star de reemplazo aristas de $E_G \setminus \chi_k(S)$ y/o de la propia k.*

Para utilizar la **vecindad4** se requiere una modificación en el algoritmo MejorKeyStar: las aristas ya presentes en la solución actual, que no sean parte de la key-star a reemplazar k, se considerarán **de peso 0** en el grafo G' al momento de aplicar el algoritmo de caminos_más_cortos.

Para reducir la cantidad de candidatos a nodo central de la key-star reemplazante que serán evaluados, pueden aplicarse diversos criterios heurísticos. Por ejemplo, en contextos donde la desigualdad triangular de costos se verifique (o esté cerca de cumplirse) -situación habitual en contextos de aplicación del GSP- puede considerarse sólo el subconjunto de nodos que se encuentran dentro de un cierto "radio" de distancia de los nodos Ω.

Hemos hasta aquí propuesto algoritmos basados en la metaheurística GRASP para generar soluciones al GSP. En el capítulo siguiente formularemos una extensión al problema GSP, que nos permitirá proponer un esquema de resolución recursivo para el GSP, en el cual se podría emplear virtualmente cualquier descomposición estructural para determinar conjuntos de aristas a reemplazar en la fase de optimización local.

Capítulo 6

El Problema General de Steiner Extendido

En este capítulo formularemos una extensión del GSP motivada por la definición general de "movimientos" y el concepto de "segmentos" introducidos en el capítulo anterior. Veremos también como dicho problema extendido admite una resolución heurística recursiva.

6.1. Motivación y Formulación del Problema

6.1.1. Motivación

Volvamos sobre la Figura 5.1. En la misma es posible encontrar 4 caminos arista disjuntos que conectan a los nodos terminales u, v, a saber:

- a) $u..,1..,9...v$
- b) $u..,1..,9...v$ (3 tramos paralelos a otros sendos tramos)
- c) $u..,2..,8...v$
- d) $u..,3..,4..,5..,6...w...v$

Supongamos que estos caminos son los de una solución ejemplo para una cierta heurística en marcha que intenta mejorar la solución actual. Estamos ante el problema de reemplazar el conjunto de aristas H (punteadas gruesas) por otro conjunto H' como parte de un movimiento de optimización local de forma de preservar los caminos ejemplo. Para restablecer los caminos (a) y (b), claramente se necesitan 2 caminos arista disjuntos en H' que conecten a los nodos 1 y 9. A su vez, para restablecer el camino (c) es necesario conectar los nodos 2 y 8 en H'; pero ello debe hacerse en forma también disjunta a los nuevos caminos $1..,9$. Finalmente, para restablecer el camino ejemplo (d) es necesario hallar caminos en H' que conecten a 3 con 4 y a 5 con 6; y una vez más, ambos deberán ser arista disjuntos respecto a los 3 antes mencionados. El no hacerlo así supondría que los 4 caminos ejemplo restablecidos ya no serían arista disjuntos 2 a 2, con la consiguiente pérdida (potencial) de nivel de conectividad que se explicó en el

punto 5.2.

En la sección anterior introdujimos un mecanismo para determinar cuáles aristas de la solución actual pueden reutilizarse sin peligro de introducir "conflictos" entre caminos que deben ser arista disjuntos (con la consiguiente reducción del nivel de conectividad), para el caso de key-paths y key-stars. Generalicemos ahora esta idea para el caso de conjuntos H cualesquiera y sus segmentos asociados. Para ello extenderemos una vez más nuestras definiciones de J y χ, ahora para el caso de segmentos.

Notación 6.1.1 *Sea una solución ejemplo S y uno de sus segmentos s. Denotaremos como $J_s(S)$ al conjunto de caminos dado por*

$$J_s(S) = \{p \in S : s \subseteq p\}$$

En otras palabras, es el conjunto de los caminos ejemplo que hacen uso del segmento s, o sea, se ven "afectados" por el reemplazo de dichos segmentos.

Notación 6.1.2 *Sea una solución ejemplo S y uno de sus segmentos s. Denotaremos como $\chi_s(S)$ al conjunto de aristas dado por*

$$\chi_s(S) = \bigcup_{q=i\ldots j \in J_s(S)} E(P_{ij} \setminus q)$$

A su vez, dos segmentos que participan en S de caminos que unen los mismos terminales, no podrán compartir aristas; con lo cual aparece una restricción adicional que involucra exclusión mutua entre ciertos segmentos 2 a 2. Agregamos para ello la siguiente notación:

Notación 6.1.3 *Sea una solución ejemplo S y uno de sus segmentos s. Denotaremos como $\sigma_s(S)$ al conjunto de segmentos dado por*

$$\sigma_s(S) = \{s' \in seg(H) : (\exists p_{ij} \neq p'_{ij} \in S : s \subseteq p_{ij} \wedge s' \subseteq p'_{ij})\}$$

Hemos así establecido, dado un segmento s, cuáles son las aristas de la solución actual que podrían ocasionar conflictos en caso de ser reutilizadas para reemplazar a dicho segmento, y también con cuáles segmentos deberá ser arista-disjunto. Tomando estas precauciones, cada segmento podrá ser reemplazado por cualquier camino simple que una sus mismos extremos, obteniéndose así un posible conjunto H' de reemplazo. Tenemos entonces:

- un conjunto de segmentos $seg(H)$
- un conjunto de aristas a evitar al recalcular cada segmento s, $\chi_s(S)$
- un conjunto de segmentos "arista incompatibles" para cada segmento s, $\sigma_s(S)$
- para cada pareja de "extremos de segmento" un cierto número de segmentos que los une
- costos para las aristas involucradas (algunos puestos en 0 como hicimos en la sección anterior)

Estamos ahora en condiciones de modelar esta situación enunciando el "Problema General de Steiner Extendido".

6.1.2. Formulación del Problema

Comenzamos definiendo la siguiente notación para poder trabajar cómodamente con parejas de nodos.

Notación 6.1.4 *Dado un conjunto A denotaremos como $2_i^A, i \in \mathbb{N}$ al conjunto de todos los subconjuntos de A que tienen cardinalidad i, o sea, $2_i^A = \{a \subseteq A : |a| = i\}$.*

Ahora definimos el problema; hacemos uso del concepto de "solución ejemplo S" y las notaciones P_{ij} y p_{ij} antes definidas.

Definición 6.1.5 ***Problema General de Steiner Extendido***. *Dados:*

- $G = (V, E, C)$ *: grafo de conexiones factibles con aristas ponderadas con costos (C) no negativos;*
- $T \subseteq V$ *: subconjunto de nodos que llamaremos "terminales";*
- $R_{|V|\times|V|} = (r_{ij})$ *: matriz simétrica de enteros no negativos con diagonal principal 0;*
- $\hat{\chi} : 2_2^T \to 2^E$ *: subconjuntos de aristas a evitar al conectar una pareja de terminales dada*
- $\hat{\sigma} : 2_2^T \to 2^{2_2^T}$ *: conjuntos de parejas de terminales cuyos caminos deben ser arista-incompatibles con una pareja de terminales dada*

Encontrar un subgrafo G_T de costo mínimo sobre G sobre el cual exista un conjunto de caminos S tales que:

- *(1)* $(\forall i, j \in T)(|P_{ij}| \geq r_{ij})$
- *(2)* $(\forall p_1 \neq p_2 \in P_{ij})(p_1 \cap p_2 = \emptyset)$
- *(3)* $(\forall i, j \in T)(\forall p \in P_{ij})(p \cap \hat{\chi}(\{i, j\}) = \emptyset)$
- *(4)* $(\forall i, j, h, k \in T)(\{h, k\} \in \hat{\sigma}(\{i, j\}) \to (\forall p \in P_{ij})(\forall q \in P_{hk})(p \cap q = \emptyset)$

La matriz R representa niveles de conectividad arista disjunta que es necesario alcanzar para cada pareja de terminales. Las funciones $\hat{\chi}$ y $\hat{\sigma}$ representan respectivamente las aristas que se deben evitar al conectar una pareja de terminales dada, y los conjuntos de parejas de terminales cuyos caminos ejemplo deben ser arista disjuntos con los de una pareja de terminales dada.
A su vez, las condiciones (1) a (4) expresan lo siguiente en términos intuitivos:

- (1) Hay tantos caminos como exige R para cada pareja de terminales ...
- (2) ... los cuales son arista disjuntos 2 a 2.
- (3) Cada camino de S evita emplear aristas del conjunto de aristas vedadas a su pareja de terminales extremos.

```
Algoritmo EGSP (G, C, T, R, χ̂, σ̂, S);
1  mejorar ← TRUE
2  mientras (mejorar) hacer
3    mejorar ← FALSE
4    ℍ ←estructuras_candidatas_a_reemplazar(H)
5    por cada H ∈ ℍ hacer
6      (G', C', T', R', χ̂', σ̂', S') ← formular_subproblema_para(H)
7      H' ←EGSP(G', C', T', R', χ̂', σ̂', S')
8      si costo(H') < costo(H)
9        mejorar ← TRUE
10         reemplazar H por H' en los caminos de S
11         abortar "por cada"
12       fin si
13   fin por cada
14 fin mientras
15 return S

Fin Función
```

Figura 6.1: Algoritmo EGSP.

- (4) Cada camino de S evita emplear aristas de cualquier otro camino de S cuyos extremos sean "incompatibles"

En el apartado siguiente, bosquejamos un algoritmo heurístico para resolver el GSP mediante invocaciones recursivas al EGSP, haciendo uso de nuestro concepto de movimientos de reemplazo $H \rightarrow H'$.

6.2. Esquema de una Heurística Recursiva General

En la Figura 6.1 se presenta la estructura de un algoritmo recursivo que explota los conceptos vistos hasta aquí. La función "estructuras_candidatas_a_reemplazar" genera varios conjuntos de aristas para los cuales se intentará el reemplazo (o sea, nuestros conjuntos H). Aquí no especificamos la forma de determinarlo, de ahí el carácter "general" del algoritmo; más adelante damos algunas consideraciones al respecto. En la línea 6, ya con un conjunto específico H, se construyen los parámetros de entrada para una invocación recursiva, como se detalla más abajo. Una vez retornado de dicha invocación, el algoritmo verifica si se encontró una mejora en el costo y de ser así, se "adopta" el reemplazo H', recomenzándose la búsqueda de reemplazos, hasta que no sea posible encontrar una mejora.

Los parámetros para la invocación recursiva se determinan del siguiente modo. Una vez fijado H y sus segmentos sobre la solución ejemplo S (que será S'), el conjunto T' será el de todos los nodos que son extremo de algún segmento. Luego, a partir de la cantidad de segmentos que unen cada pareja de dichos nodos, se construye la matriz R'. Analizando los caminos de S que unen cierta pareja de terminales de T, se determinan los segmentos que deben ser

arista disjuntos; a partir de dicha información y de la función $\hat{\sigma}$ se conforma la función $\hat{\sigma}'$. A su vez, se extiende la función $\hat{\chi}$ mediante el análisis de $\chi_s(s)$ para cada segmento $s \in S$. Finalmente se modifican los costos (C') poniendo en 0 el de aquellas aristas que de todas formas seguirán siendo parte de la solución, como hemos hecho en casos anteriores.

Dada una instancia del problema GSP y una solución ejemplo S, mediante sus parámetros (G, C, T, R, S), la fase de búsqueda local se ejecutaría mediante la siguiente invocación "raíz" a la recursión EGSP:

$$S = EGSP(G, C, T, R, \emptyset, \emptyset, S)$$

donde las funciones $\hat{\chi}$ y $\hat{\sigma}$ tienen preimagen nula.

En la Figura 6.2 se ilustra la forma en que procede el algoritmo. Ante una instancia dada, se fija el conjunto de estructuras cuyo reemplazo será intentado como se ilustra en la parte (b), y luego se invoca para cada elemento de dicho conjunto una instancia del mismo problema en la cual los extremos de segmentos asociados a dicho elemento son tratados como terminales.

El comportamiento del algoritmo estará básicamente determinado por la forma de construir el conjunto de estructuras_candidatas_a_reemplazar. En general dependerá del tamaño de la instancia en cuestión. Por ejemplo, podría considerarse en cada fase determinar conjuntos H cuya cantidad de aristas se reduzca a medida que se profundiza el nivel de recursión; es natural que en niveles más cercanos a la raíz se intente el reemplazo de conjuntos "grandes", para ir reduciéndolos a medida que avanza la recursión. En cierto punto pueden usarse estructuras como las que hemos visto (key-paths, key-stars, parejas de key-stars conectadas) para las cuales se empleen localmente algoritmos diferentes como los ya vistos.

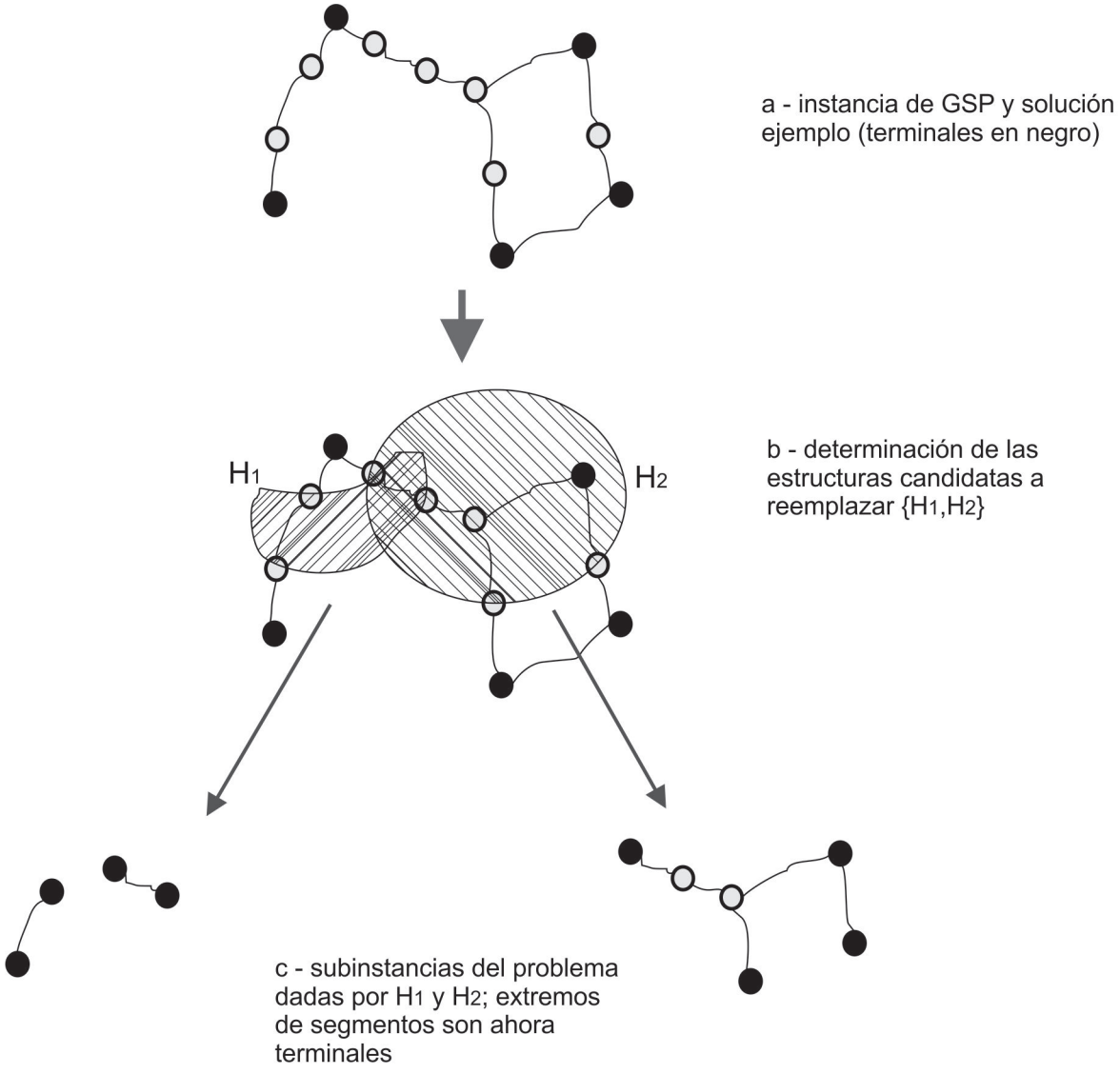


Figura 6.2: Funcionamiento del Algoritmo Recursivo

Parte IV

PRUEBAS

Capítulo 7

Pruebas de Desempeño

7.1. Algoritmos GRASP a Probar

En esta sección presentaremos los algoritmos que serán puestos a prueba con un mismo conjunto de instancias del GSP-ED. Hasta ahora hemos presentado:

- Algoritmos cuya ejecución genera **una** solución factible a una instancia del problema
- Algoritmos que dada una solución factible, la mejoran mediante movimientos locales aplicando **un tipo** específico de movimiento, hasta que ya no son posibles mejoras posteriores

Nuestro algoritmo GRASP ejecutará una cierta cantidad de veces, la generación de una solución factible seguida de una fase de optimización local, que estará eventualmente basada en la combinación de tipos de movimientos. En la Figura 7.1 se muestra el algoritmo general para llevar a cabo esta tarea, en el caso de que se emplee solamente un tipo de movimiento de optimización local.

Las función Construcción es una referencia general a cualquiera de las construcciones propuestas; nuestras pruebas están basadas en el algoritmo Construcción1. A su vez, la función Optimiza es una referencia a cualquiera de los algoritmos de optimización local antes presentados (Optimiza1, Optimiza2, Optimiza3); emplearemos en esta modalidad a Optimiza1 y Optimiza2.

El algoritmo itera (líneas 2-10) tantas veces como indique el parámetro $iters$. En cada iteración se construye una solución factible mediante el algoritmo Construcción (heurístico glotón y aleatorizado). Dicho algoritmo genera una solución y devuelve un conjunto S de caminos ejemplo junto con el grafo asociado a dicha solución. Dependiendo de la matriz de requerimientos R, la topología del grafo, e incluso del algoritmo en cuestión y factores aleatorios dentro del mismo, puede darse que la solución generada no verifique todos los requisitos especificados por R. En este caso hemos tomado el siguiente enfoque:

- Se devolverá una solución que tenga la menor cantidad posible de requerimientos no cubiertos de R;

```
Algoritmo GRASP_GSP1(G, C, T, R, iters);
1  c_min ← ∞; S_opt ← ∅
2  para i = 1..iters hacer
3     [G_fac, S] ←Construcción(G, C, T, R)
4     si |S| ≥ |S_opt|
5        [G_sol, S] ←Optimiza(G, C, S)
6        si costo(S) < c_min
7           c_min ←costo(S); S_opt ← S
8        fin si
9     fin si
10 fin para
11 return S_opt

Fin Función
```

Figura 7.1: Algoritmo GRASP_GSP1 (con un solo movimiento local).

- Dentro de tal conjunto, se devolverá la que tenga menor costo

Luego de creada la solución factible, y solamente si no hay pérdida de requerimientos de conectividad respecto a soluciones antes encontradas, en la línea 5 se procede a la fase de optimización local. La línea 6 verifica si se ha logrado una mejora en el mejor costo hallado hasta el momento; de ser así, en la línea 7 se registra la nueva solución en S_{opt} y su costo en c_{min}. Finalmente, al cabo de las $iters$ iteraciones, se devuelve la mejor solución encontrada.

Veamos ahora el esquema de un algoritmo que combine 2 tipos de movimientos. En la Figura 7.2 se presenta el pseudocódigo correspondiente.

Luego de construída una solución en la línea 3, y siempre que no sea con pérdida de requerimientos de conectividad, se procede a un ciclo de mejoras alternadas dadas por la aplicación de 2 tipos de movimiento. La línea 6 invoca a una optimización local a la cual nos referimos aquí como OptimizaA; luego se procederá a aplicar la segunda optimización (OptimizaB) si es la primera vez que se hace esta secuencia para la solución creada en la línea 3, o si en la línea 6 se logró una reducción del costo. A su vez, se retornará a la línea 6 solamente si OptimizaB logró introducir mejoras. De este modo los algoritmos trabajan en forma combinada, hasta el momento en que uno de los dos no logra modificar la solución; evitándose así una invocación posterior inútil al otro algoritmo.

Para las pruebas emplearemos como fase de construcción la Construcción1 puesto que en la mayoría de las aplicaciones de interés la cantidad de requisitos de conectividad entre pares de terminales es menor (o mucho menor) que la conectividad factible entre los mismos. A su vez, para la fase de optimización local, compararemos las alternativas ilustradas en el Cuadro 7.1.

Los algoritmos a testear GSPTest1 y GSPTest2 harán empleo de un único tipo de movimiento, respectivamente “reeplazo simple de key-paths” y “reemplazo complejo de key-paths”. A

Algoritmo GRASP_GSP2$(G, C, T, R, iters)$**;**
1 $c_{min} \leftarrow \infty; S_{opt} \leftarrow \emptyset$
2 para $i = 1..iters$ hacer
3 $[G_{fac}, S] \leftarrow$Construcción(G, C, T, R)
4 si $|S| \geq |S_{opt}|$
5 $primera \leftarrow$TRUE
6 $[G_{sol}, S'] \leftarrow$OptimizaA(G, C, S)
7 si $primera \vee$costo$(S') <$costo(S)
8 $primera \leftarrow$FALSE; $[G_{sol}, S''] \leftarrow$OptimizaB$(G, C, S')$
9 si costo$(S'') <$costo(S') ir al paso 6
10 fin si
11 si costo$(S'') < c_{min}$
12 $c_{min} \leftarrow$costo$(S''); S_{opt} \leftarrow S$
13 fin si
14 fin si
15 fin para
16 return S_{opt}

Fin Función

Figura 7.2: Algoritmo GRASP_GSP2 (combinando 2 movimientos locales).

	GSPTest1	**GSPTest2**	**GSPTest3**	**GSPTest4**
Construcción	Construcción1	Construcción1	Construcción1	Construcción1
OptimizaA	Optimiza1	Optimiza2	Optimiza2	Optimiza2
OptimizaB	—	—	Optimiza3	Optimiza4

Cuadro 7.1: Algoritmos GRASP a probar

su vez, GSPTest3 y GSPTest4 combinarán el reemplazo complejo de key-paths con "reemplazo simple de key-stars" y "reemplazo complejo de key-stars".

7.2. Casos de Prueba

En la literatura relacionada con el GSP no hemos encontrado instancias de "benchmark" para las cuales se conozca el costo optimal (que entonces permitirían saber qué tan cerca del óptimo están las soluciones dadas por un algoritmo determinado). A efectos de probar el desempeño de nuestros algoritmos hemos definido un conjunto de casos de prueba, basados en 2 bibliotecas, a saber:

- steinlib [10]: contiene instancias varias del problema de Steiner, conociéndose en varios de ellos las soluciones óptimas, y en otros casos la mejor hallada hasta el momento.
- tsplib [20]: contiene instancias de varios problemas de teoría de grafos; en particular hay una sección de "problemas del agente viajero", de la cual tomamos 2 casos geográficos reales.

En los casos tomados de Steinlib, se cuenta con el costo de la solución optimal para el problema de Steiner, que es equivalente a nuestro problema GSP-ED cuando los requerimientos de conectividad r_{ij} son todos iguales a 1; esto nos permitirá comparar los resultados de nuestros algoritmos contra dichos óptimos. En el resto de los casos haremos un análisis de gap entre la mejor solución que hallemos y las mejores soluciones halladas por los demás algoritmos a probar.

El Cuadro 7.2 presenta los casos de prueba junto con sus características principales.

Para cada caso se da la cantidad de nodos $|V|$, la cantidad de aristas $|E|$, la cantidad de nodos terminales $|T|$, la cantidad de nodos de Steiner (opcionales) $|St|$, el nivel de conectividad requerido entre cualquier par de terminales del caso, y el costo optimal para el problema en caso de ser conocido.

Los problemas b01, b03, b05, b11, b17 están tomados del juego de problemas "B" de Steinlib. Son casos generados aleatoriamente con costos enteros que oscilan entre 1 y 10. Resolveremos instancias con nivel de conectividad 1 (problema de Steiner, de las cuales sabemos la solución óptima) e instancias con nivel de conectividad 2. En varios casos de estos últimos no será posible cumplir con todos los requerimientos (por ejemplo, algunos terminales tienen grado 1), y tal como los definimos nuestros algoritmos entregarán la solución de menor costo entre aquellas que alcancen la mayor cantidad de requerimientos cumplidos.

El caso cc3-4p está tomado del juego de problemas "C" de Steinlib; para este problema resolveremos la instancia de nivel de conectividad 1, y también con nivel de conectividad 3.

Los casos bayg29 y att48 están tomados de la biblioteca Tsplib, y ambos corresponden a casos geográficos reales:

Caso de Origen	V	E	T	St	Requer. Conect.	Opt
b01-r1	50	63	9	41	1-arista-conexo	82
b01-r2	50	63	9	41	2-arista-conexo	ND
b03-r1	50	63	25	25	1-arista-conexo	138
b03-r2	50	63	25	25	2-arista-conexo	ND
b05-r1	50	100	13	37	1-arista-conexo	61
b05-r2	50	100	13	37	2-arista-conexo	ND
b11-r1	75	150	19	56	1-arista-conexo	88
b11-r2	75	150	19	56	2-arista-conexo	ND
b17-r1	100	200	25	75	1-arista-conexo	131
b17-r2	100	200	25	75	2-arista-conexo	ND
cc3-4p-r1	64	288	8	56	1-arista-conexo	2338
cc3-4p-r3	64	288	8	56	3-arista-conexo	ND
bayg29-r2	29	406	11	18	2-arista-conexo	ND
bayg29-r3	29	406	11	18	3-arista-conexo	ND
att48-r2	48	300	10	38	2-arista-conexo	ND

Cuadro 7.2: Casos de prueba

- bayg29: son 29 ciudades de la región de Baviera, Alemania; tomaremos 11 de ellas como terminales a interconectar con niveles 2 y 3 de conectividad, considerando el resto como opcionales
- att48: son 48 ciudades de los Estados Unidos de América; tomaremos 10 de ellas como terminales a interconectar con nivel 2 de conectividad.

En la Figura 7.3 se ilustran los grafos de conexiones factibles (nodos terminales en negro, opcionales vacíos) correspondientes a los casos b01, b03, b05, b11. La Figura 7.4 ilustra el grafo correspondiente al caso b17, y la Figura 7.5 ilustra el grafo asociado al caso cc3-4p.

En la Figura 7.6 se ilustran las 29 ciudades sobre el contorno de Baviera que conforman a nuestro caso bayg29. Finalmente, la Figura 7.7 se ilustran las 48 ciudades sobre el contorno de los Estados Unidos de América que conforman a nuestro caso att48. Para todos los casos, más adelante, se presentan los grafos correspondientes a la mejor solución obtenida por nuestros algoritmos, con la misma disposición ("layout") de nodos y aristas.

7.3. Resultados de las Pruebas

En el Cuadro 7.3 se presentan los resultados numéricos obtenidos de la ejecución de los casos de prueba. Los algoritmos fueron implementados en lenguaje C y C++ y fueron ejecutados en un equipo con procesador Intel Core 2 Duo con 2 GB de memoria RAM, corriendo Microsoft Windows Vista. Cada algoritmo, para cada caso de prueba, fue ejecutado con 100 iteraciones (parámetro **iters**).
Las columnas t1, t2, t3, t4 muestran el tiempo promedio (en milisegundos) de cada iteración.

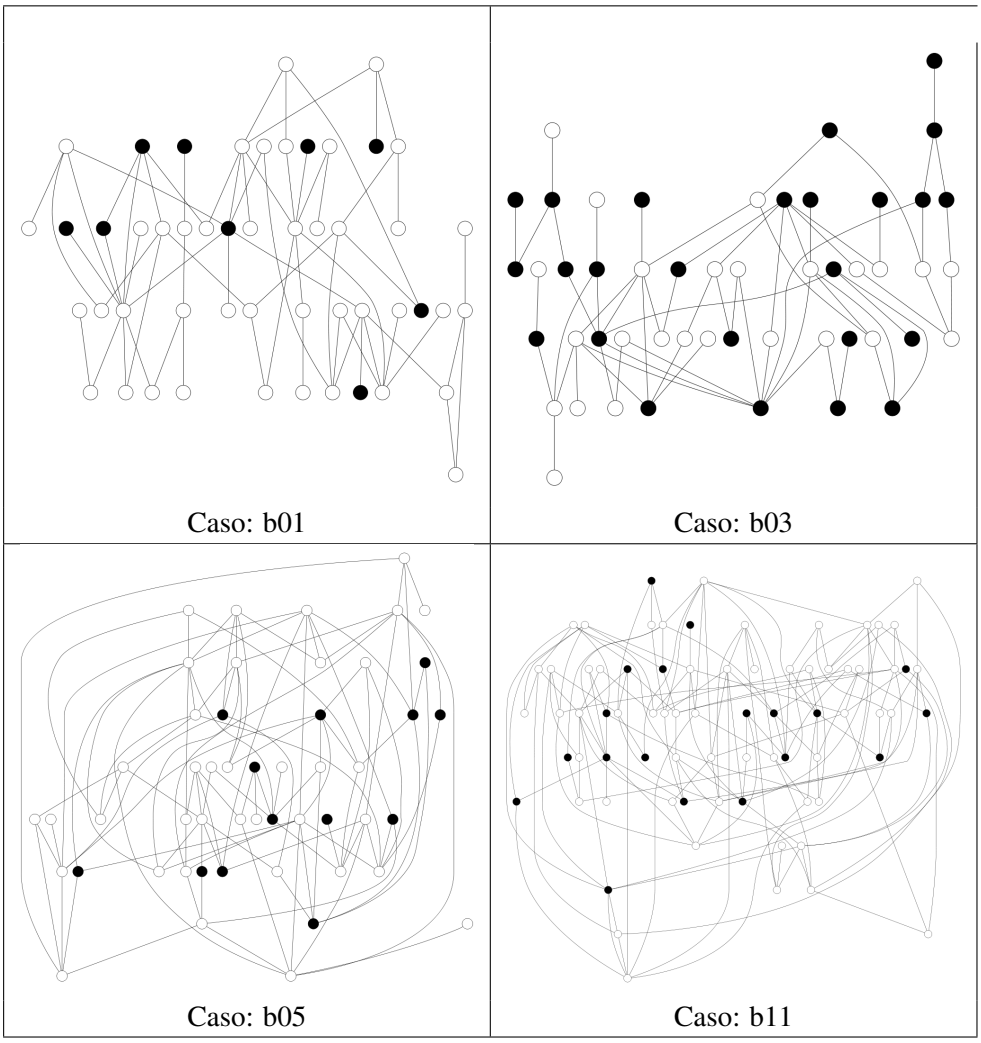


Figura 7.3: Casos b01 a b11

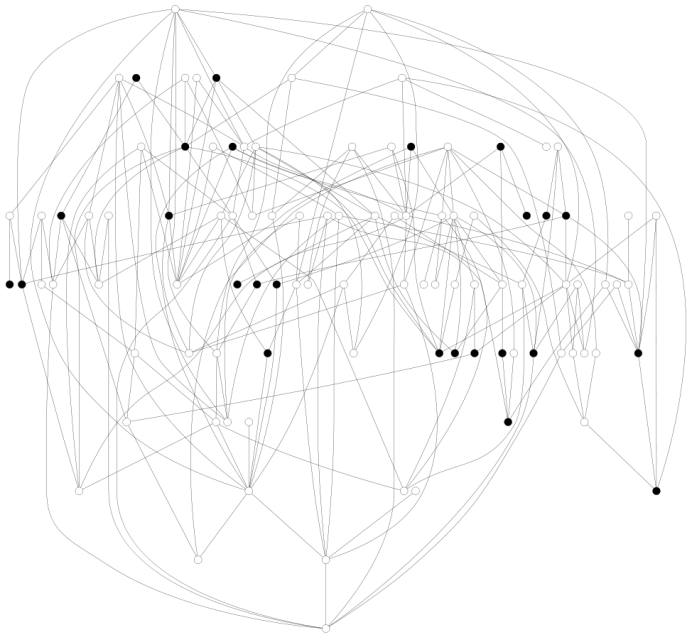


Figura 7.4: Caso b17

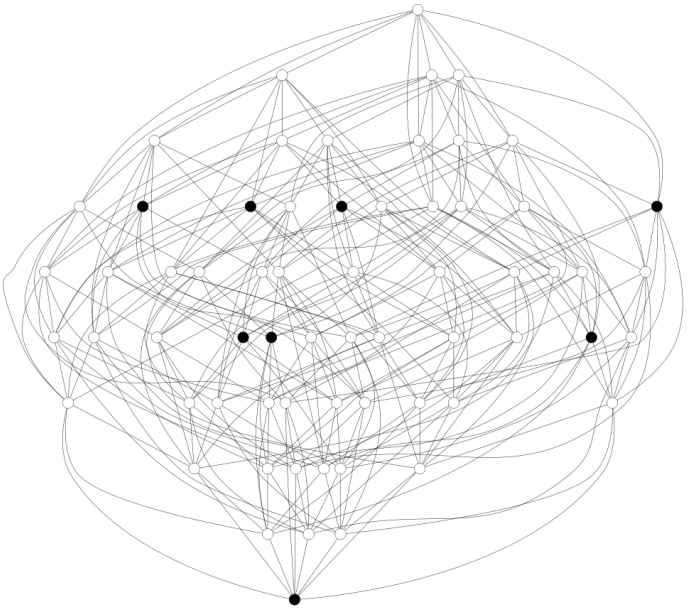


Figura 7.5: Caso cc3-4p

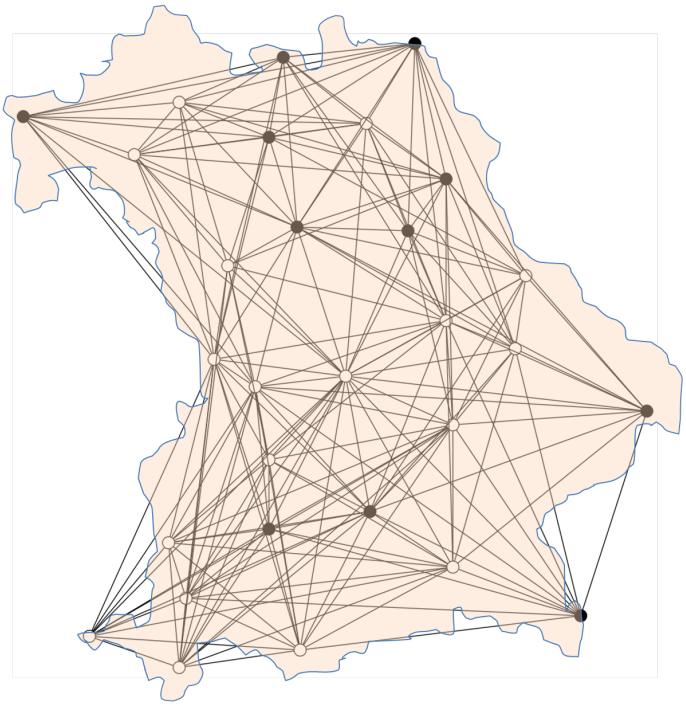


Figura 7.6: Caso bayg29

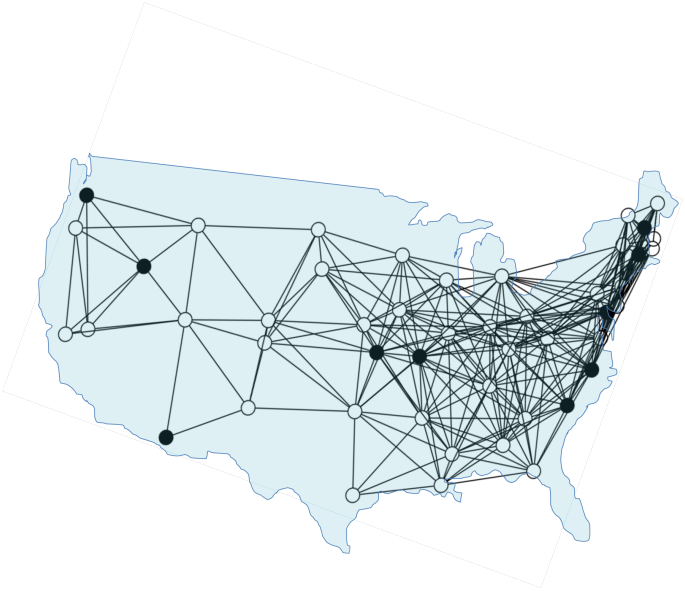


Figura 7.7: Caso att48

Caso	Reqs	R.Sat.	t1(ms)	t2(ms)	t3(ms)	t4(ms)	opt1	opt2	opt3	opt4	opt
b01-r1	36	36	32	42	69	77	82	82	82	82	82
b01-r2	72	42	30	39	62	80	98	98	98	98	ND
b03-r1	300	300	586	1741	2477	2611	138	138	138	138	138
b03-r2	600	378	1314	1998	2725	3108	196	188	188	188	ND
b05-r1	78	78	80	100	245	298	61	61	61	61	61
b05-r2	156	144	262	398	936	1389	122	120	120	120	ND
b11-r1	171	171	332	575	1288	1477	89	88	88	88	88
b11-r2	342	324	1340	1838	4108	4901	195	184	183	180	ND
b17-r1	300	300	1332	2365	5827	6214	133	131	131	131	131
b17-r2	600	531	4210	7224	13166	15143	247	247	244	244	ND
cc3-4p-r1	28	28	64	89	299	388	2356	2339	2338	2338	2338
cc3-4p-r3	84	84	296	1100	1982	2221	6384	6168	6032	5991	ND
bayg29-r2	110	110	198	216	737	975	7576.95	7323.18	6997.88	6856.88	ND
bayg29-r3	165	165	274	491	1969	2413	12545.4	12215.77	11808.4	11722	ND
att48-r2	90	90	166	232	917	1313	24482	23234	23234	23214	ND

Cuadro 7.3: Resultados de las pruebas

Las columnas opt1, opt2, opt3, opt4 muestran el costo de la solución hallada por cada uno de los 4 algoritmos; la columna opt muestra el costo optimal en los casos en que se lo conoce. Las columnas Reqs y R.Sat. muestran respectivamente la cantidad de requerimientos de conectividad solicitados y la cantidad de satisfechos por dichas soluciones.

Hemos verificado (aplicando algoritmos de flujo máximo entre todos los pares de terminales) que la cantidad de requerimientos satisfechos (R.Sat.) en todos los casos alcanzó el valor máximo posible, esto es, las soluciones obtenidas respetan tantos requerimientos como la topología del grafo de conexiones factibles G permite en cada caso. En casos como por ejemplo el **b01_r2**, se solicitan más caminos disjuntos para algunos pares de terminales que los que es posible trazar (nótese por ejemplo que hay nodos terminales con grado 1 y se solicita conectividad 2).

Los resultados obtenidos por el algoritmo GSPTest4 son superiores o iguales en todos los casos a los GSPTest1, 2 y 3. Lo mismo puede decirse de GSPTest3 respecto de GSPTest1 y 2, y de GSPTest2 respecto de GSPTest1, como era dable esperar puesto que cada algoritmo de la secuencia GSPTest1, 2, 3 y 4 cubre todos los posibles movimientos de los anteriores más algunos posibles nuevos movimientos.

En todos los casos con requerimiento de conectividad igual a 1 para los pares de terminales (casos para los cuales se conoce además el costo optimal) se alcanzaron las soluciones óptimas (árboles de costo mínimo) con los algoritmos GSPTest3 y GSPTest4; lo propio puede decirse del algoritmo GSPTest2 salvo para el caso cc3-4p donde se alcanzó "casi" un óptimo (costo 2239 en vez de 2238).

Respecto de los casos con requerimientos superiores a 1, el Cuadro 7.4 muestra el porcentaje de sobrecosto (conocido como "gap" en inglés) de las soluciones obtenidas por cada algoritmo respecto a la dada por GSPTest4.

Caso	%Gap-1-4	%Gap-2-4	%Gap-3-4
b01-r2	0.00	0.00	0.00
b03-r2	4.26	0.00	0.00
b05-r2	1.67	0.00	0.00
b11-r2	8.33	2.22	1.67
b17-r2	1.23	1.23	0.00
cc3-4p-r3	6.56	2.95	0.68
bayg29-r2	10.50	6.80	2.06
bayg29-r3	7.02	4.21	0.74
att48-r2	5.46	0.09	0.09

Cuadro 7.4: Porcentaje de sobrecostos relativos a GSPTest4

Estos ejemplos sugieren que la diferencia porcentual en los costos se torna más relevante cuando los casos adquieren mayor complejidad; posiblemente debido a que por la misma, son casos donde es posible encontrar más estructuras complejas que convenga reemplazar (como el caso de las "mejores key-stars"). A su vez (en relación al reemplazo de estructuras con reuso de aristas empleadas en la solución) queda manifiesto que la mejora introducida por GSPTest2 respecto a GSPTest1 es mucho más importante que la introducida por GSPTest4 respecto a GSPTest3; o sea, dicho reemplazo de estructuras tiene mayor efecto al aplicarlo sobre reemplazo de key-paths que sobre el reemplazo de key-stars.

El Cuadro 7.5 muestra, para cada algoritmo, el porcentaje de reducción de costo que la fase de optimización local logra respecto al costo dado por la fase de construcción. Se presenta para cada caso y algoritmo el promedio obtenido en las iteraciones. Como la tabla muestra, las mejoras locales introducidas por las variantes de los algoritmos resultan ser progresivamente mayores.

En cuanto a los tiempos de ejecución, es claro el efecto sobre los mismos del cómputo extra requerido por cada algoritmo GSPTest(n) respecto al GSPTest(n-1). En salto mayor en tiempo de ejecución se observa al encadenar las fases de reemplazo de key-paths y reemplazo de key-stars; el tiempo adicional que éste último reemplazo supone es un orden mayor que el de los key-paths, por lo cual se hace muy importante la necesidad de identificar criterios razonables para descartar ("podar") key-stars que no tenga sentido (o muy baja probabilidad) de aportar mejoras, así como técnicas para evitar (o reducir al mínimo) el trabajo del recálculo de la key-descomposición de la solución luego de los reemplazos, siempre que sea posible.

En las Figuras 7.8, 7.9, 7.10, 7.11, 7.12 y 7.13 se ilustran las mejores soluciones halladas para cada uno de los casos de prueba planteados; puede observarse que en todos los casos son soluciones factibles y minimales respecto a la eliminación de aristas.

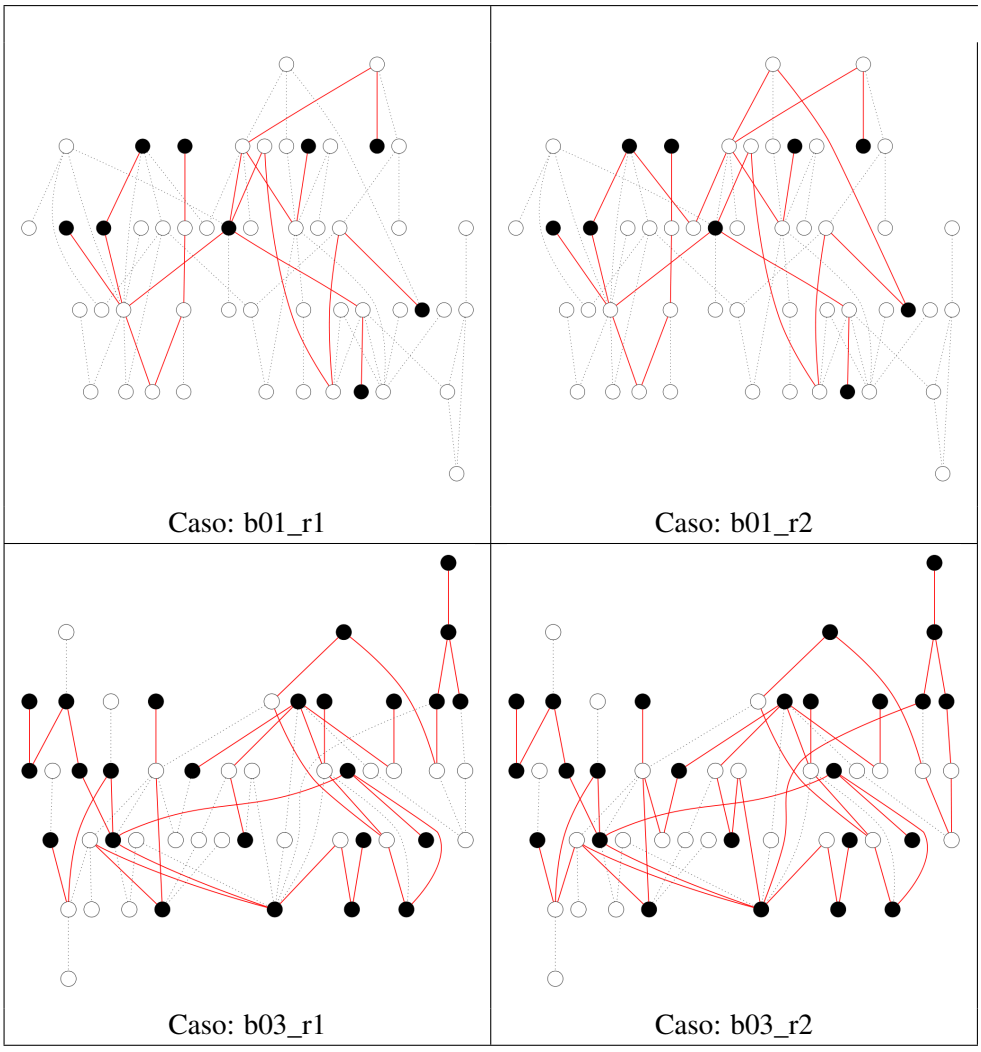


Figura 7.8: Mejores Soluciones de Casos b01 y b03

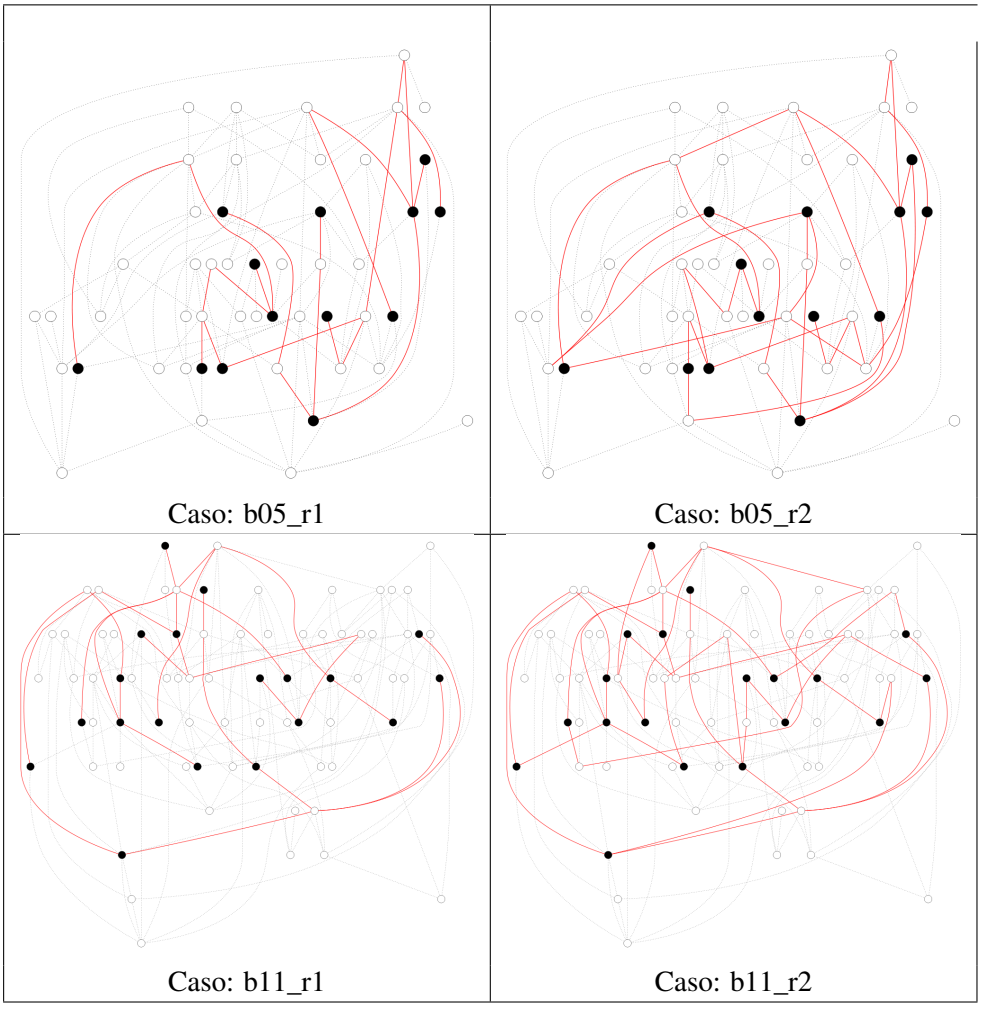
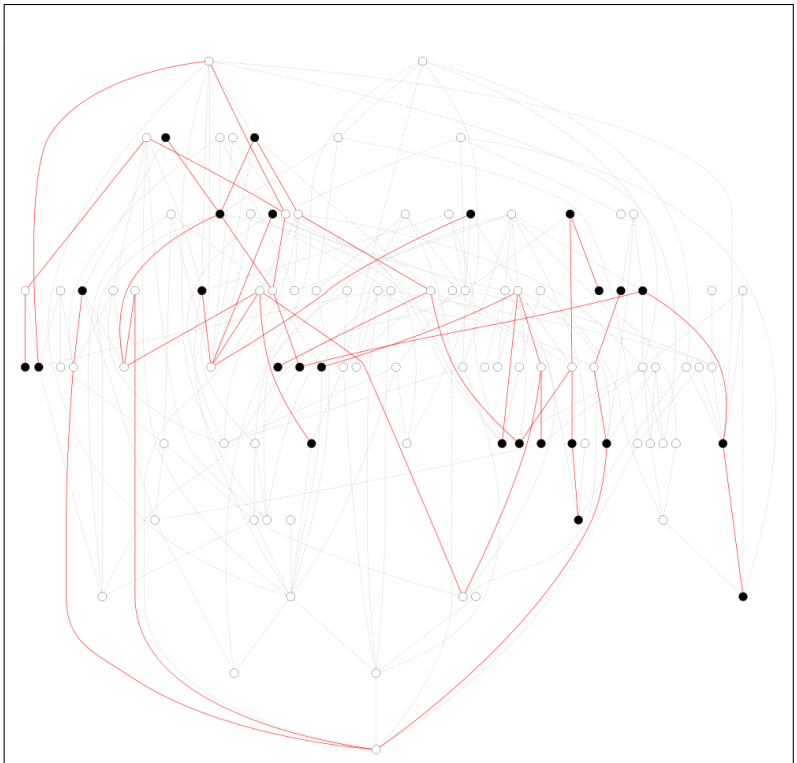
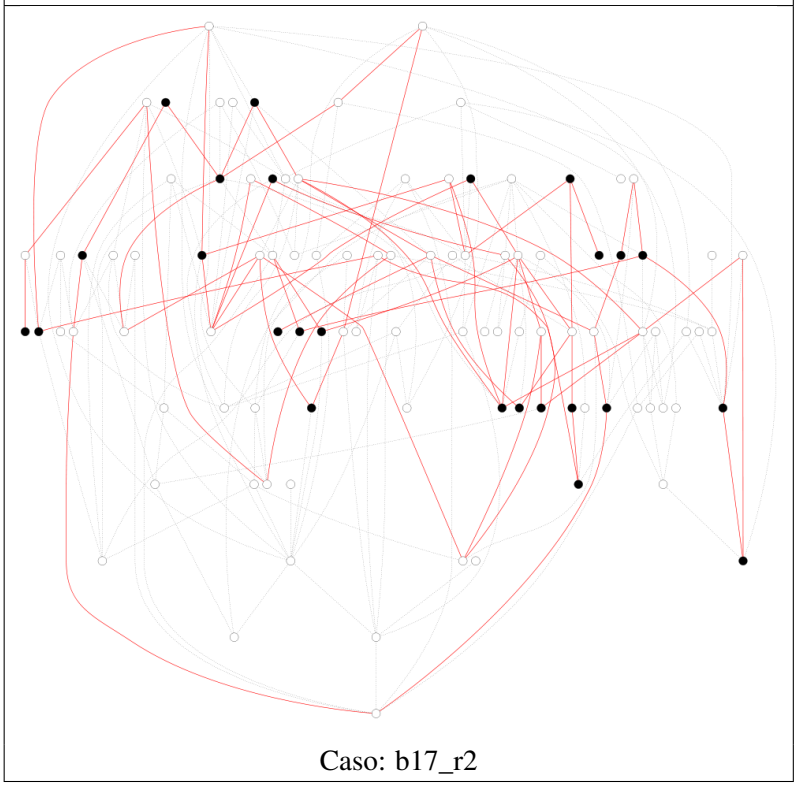


Figura 7.9: Mejores Soluciones de Casos b05 y b11



Caso: b17_r1



Caso: b17_r2

Figura 7.10: Mejores Soluciones de Casos b17

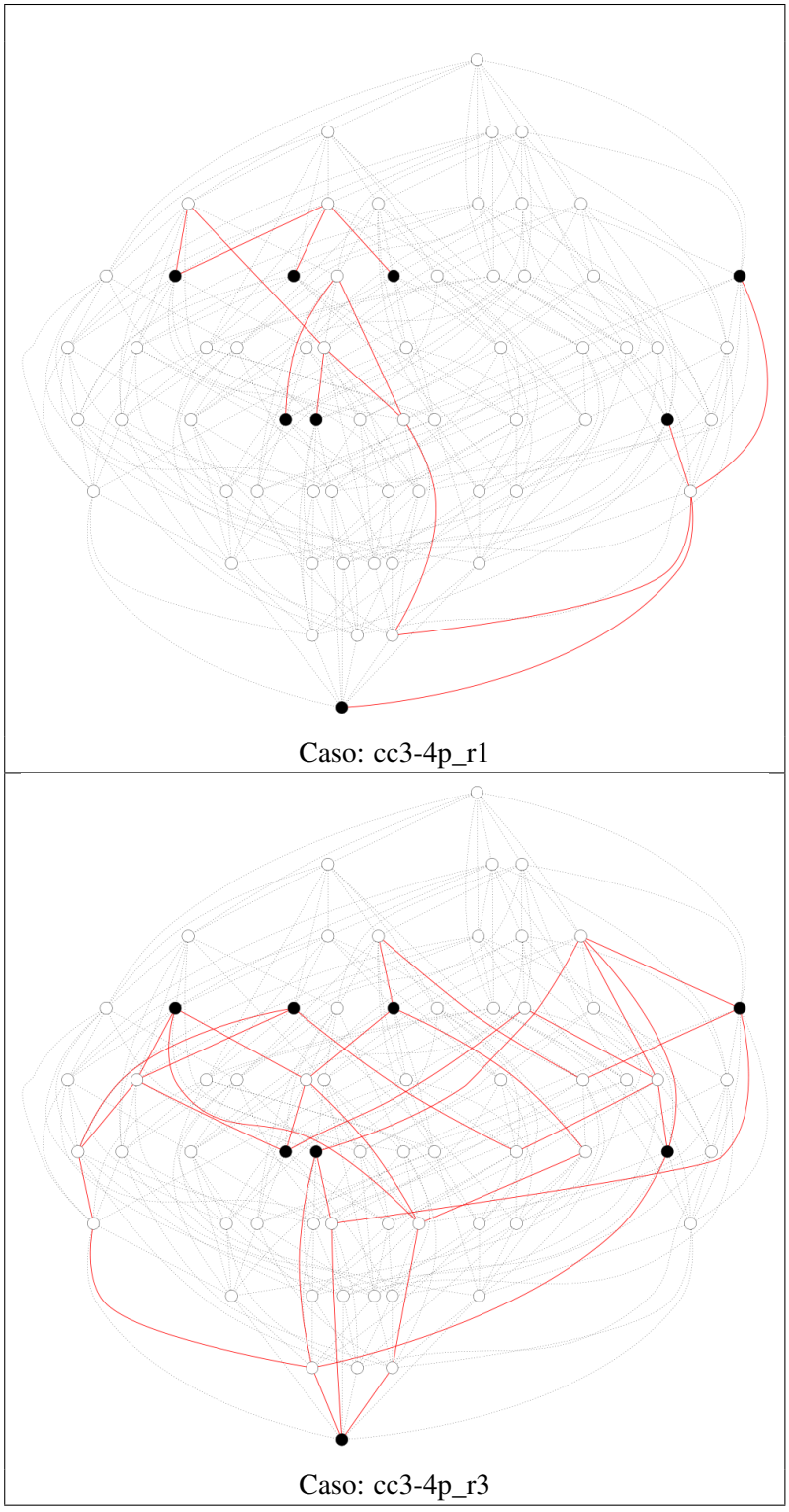
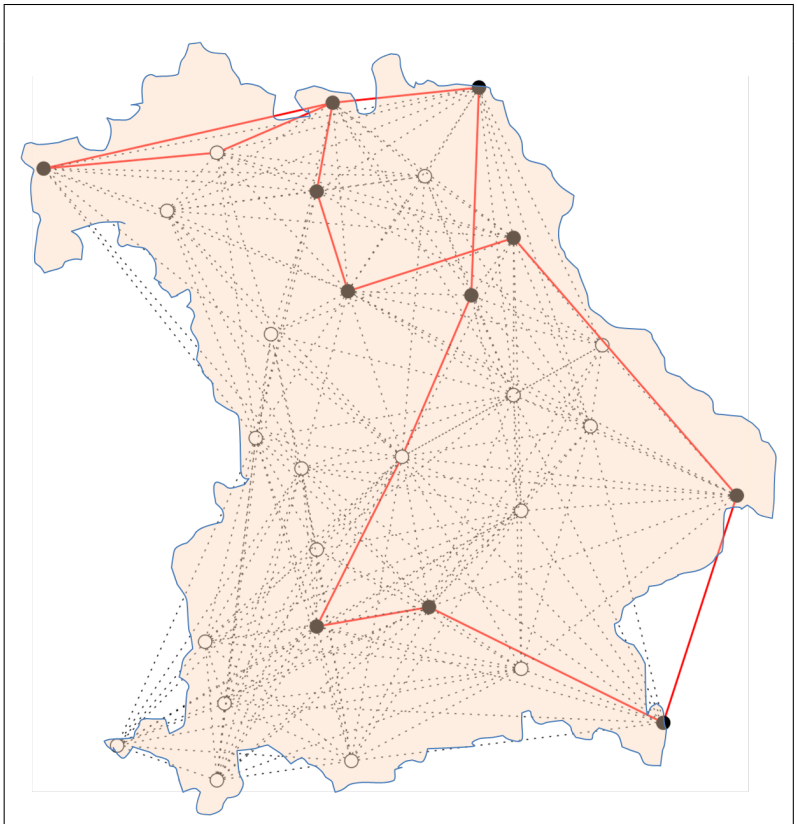
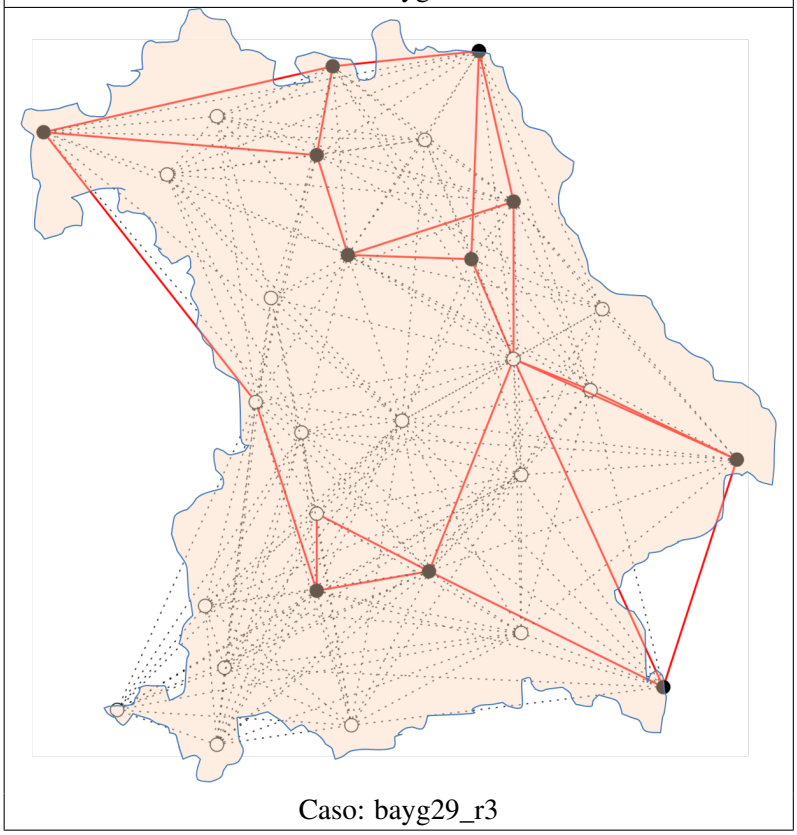


Figura 7.11: Mejores Soluciones de Casos cc3-4p



Caso: bayg29_r2



Caso: bayg29_r3

Figura 7.12: Mejores Soluciones de Casos bayg29

Caso	%Mejora 1	%Mejora 2	%Mejora 3	%Mejora 4
b01-r1	0.0	2.2	2.4	3.0
b01-r2	0.0	1.8	2.8	3.4
b03-r1	0.0	9.6	9.8	10.6
b03-r2	0.0	0.8	1.4	4.1
b05-r1	0.0	6.6	7.4	9.2
b05-r2	0.0	3.2	4.8	5.2
b11-r1	0.4	11.2	13.2	13.8
b11-r2	1.2	2.6	2.8	3.4
b17-r1	0.8	8.4	9.4	10.2
b17-r2	1.2	2.6	2.8	3.0
cc3-4p-r1	2.0	4.0	7.6	10.0
cc3-4p-r3	1.0	2.2	3.6	4.6
bayg29-r2	1.6	3.0	3.4	4.6
bayg29-r3	0.6	1.4	2.4	4.2
att48-r2	2.4	5.6	11.6	13.0

Cuadro 7.5: Mejoras sobre fase de construcción

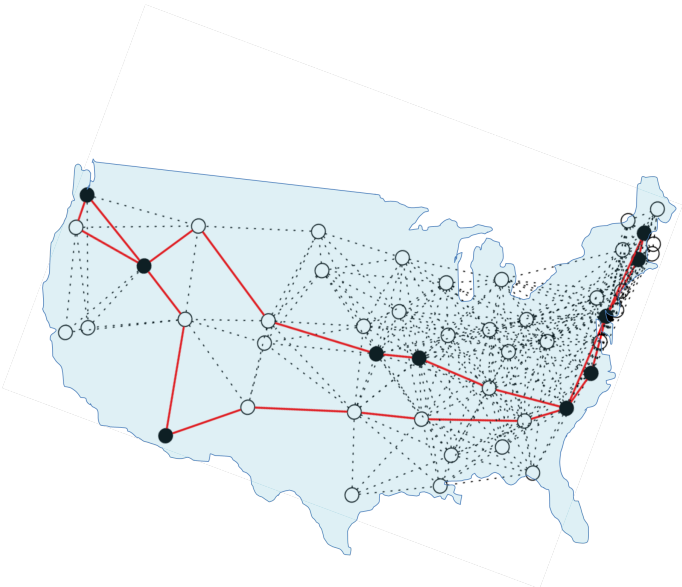


Figura 7.13: Mejor Solución de Caso att48

Parte V

CONCLUSIONES Y EXTENSIONES PROPUESTAS

Capítulo 8

Conclusiones y Trabajos Futuros

A continuación planteamos algunas cuestiones de interés que surgieron durante la concepción de este trabajo así como a raíz de las pruebas efectuadas, y hacemos referencia a otras que ya fueron mencionadas en secciones previas. Las mismas forman la base de una extensión futura del presente trabajo de investigación que entendemos de interés.

8.1. Arista-Minimalidad de Soluciones

Cabe plantearse la siguiente pregunta: ¿tienen los algoritmos propuestos la propiedad de ser arista-minimales? O sea, queremos saber si dan siempre grafos solución tales que si se elimina cualquier arista de dicho grafo éste deja de ser una solución (o, en forma más amplia, pasa a ser una "semi" solución que pierde al menos un grado de conectividad entre al menos un par de nodos terminales). Dejamos fuera el caso de que la solución tenga aristas "innecesarias" de costo 0 puesto que en rigor las mismas no afectan la "calidad" (minimizar costo) de la solución hallada tal como definimos nuestro problema GSP.

La respuesta a la pregunta es negativa; de hecho existen contraejemplos muy sencillos que ilustran no solamente la no-minimalidad de la fase de construcción (tanto Construcción1 como Construcción2) sino también de las soluciones halladas con la posterior aplicación de cualquiera de los algoritmos de optimización local propuestos. En la Figura 8.1 se muestra uno de tales ejemplos.

En la misma los nodos negros son terminales y el blanco es opcional, y se requiere 2-arista-conectividad entre las parejas (u, v) y (w, v) (no importa el requisito (u, w) para el ejemplo). Si por producto del azar, los dos primeros caminos a ser diseñados resultan ser los que verifiquen los requisitos para (u, v), se obtendrán probablemente los caminos mostrados en la parte (b). (Decimos probablemente dado que no es seguro, en vista del "ruido" que introdujimos en los costos de aristas; pero puede darse y de hecho es la pareja de caminos más probable). Una vez empleada la arista (z, v) la única solución posible que puede dar la fase de construcción es la mostrada en (c). Nótese que en la misma, si se prescinde de la arista (z, v), se continúa teniendo una solución al problema, y de menor costo. Ni el reemplazo de key-paths ni el reemplazo de key-stars como los hemos planteado lograrían deshacerse de dicha arista. En conclusión

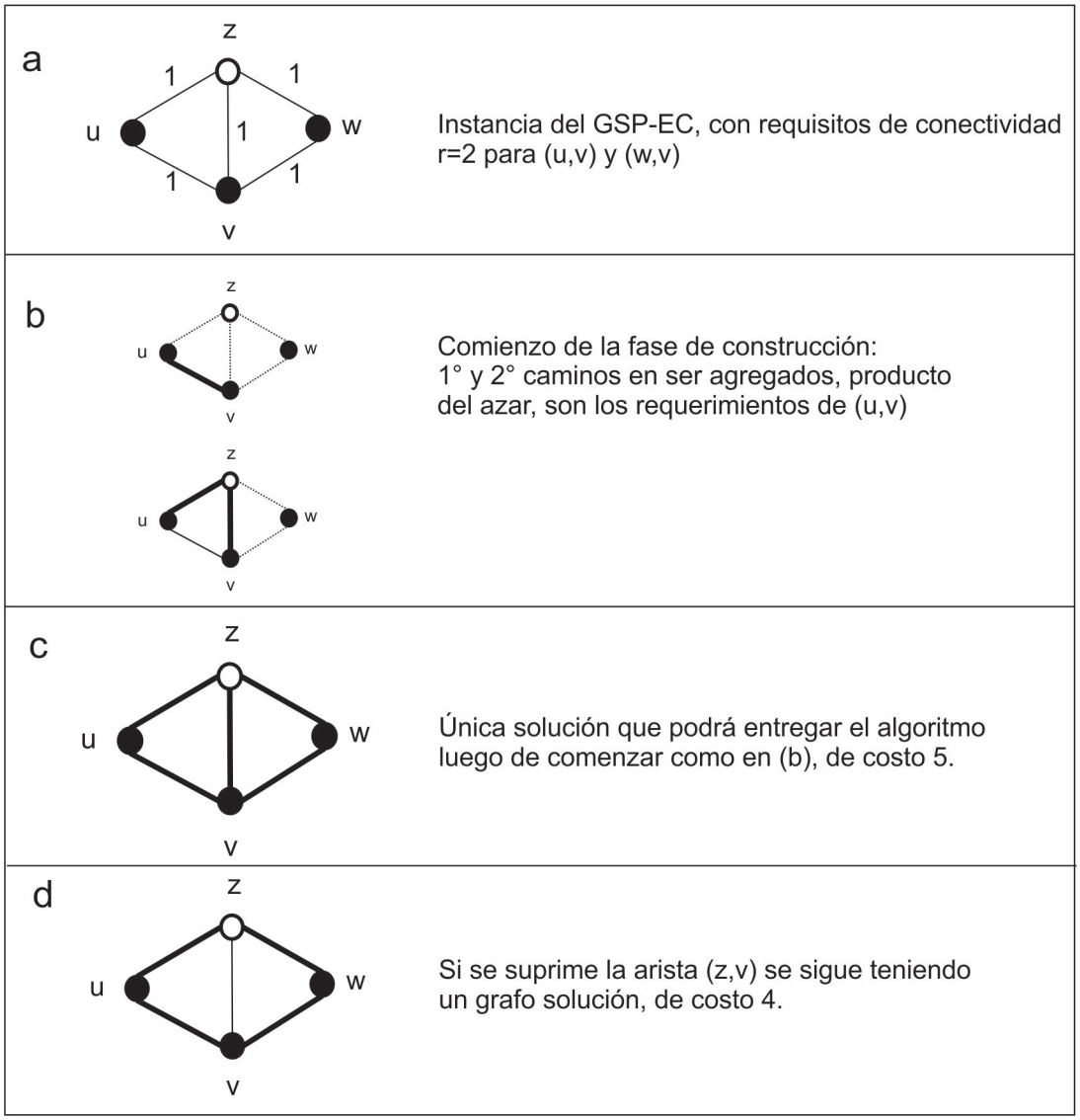


Figura 8.1: Ejemplo de No-Minimalidad

Algoritmo AristaMinimalizar(G, T, R, S)**;**
1 $mejorar \leftarrow$ TRUE
2 mientras $mejorar$ hacer
3 $mejorar \leftarrow$ FALSE
4 por cada arista $e \in E_G$ tal que e no incide en ningún nodo u con grado(u) =máx_requisito(R, u) hacer
5 $[G_{sin\_e}, S_{sin\_e}] \leftarrow$ ConstrucciónSimplificada(G, T, R)
6 si se encontró solución respetando los mismos requisitos
7 $mejorar \leftarrow$ TRUE
8 $G \leftarrow G_{sin\_e}$
9 $S \leftarrow S_{sin\_e}$
10 abortar por cada
11 fin si
12 fin por cada
13 fin mientras
14 return G, S

Fin Función

Figura 8.2: Algoritmo AristaMinimalizar.

estamos ante un contraejemplo que muestra que no está garantida la arista-minimalidad de las soluciones que hallamos. Obsérvese también que este ejemplo es válido para el algoritmo Construcción2 puesto que hemos cubierto en forma simultánea los requisitos para la pareja (u, v).

Analizando varias soluciones halladas durante las pruebas efectuadas encontramos que este tipo de situaciones son frecuentes. A su vez, no es sencillo identificar este tipo de aristas para su supresión. El problema básicamente consiste en, dado un conjunto de caminos ejemplo, reemplazar partes de algunos de ellos empleando aristas ya presentes en la solución, de forma de no perder niveles de conectividad y a la vez lograr que se "liberen" aristas.

En la Figura 8.2 proponemos un mecanismo para transformar una solución factible en arista-minimal (dejando de lado momentáneamente preocupaciones de recursos de cómputo requeridos) que puede aplicarse inmediatamente luego de la fase de construcción, o en cualquier etapa posterior.

El algoritmo recibe como entradas el grafo solución factible a minimalizar, los nodos terminales, la matriz de requisitos de conectividad R y la solución ejemplo conocida actualmente S. Las líneas 4-12 intentan eliminar una arista del grafo y encontrar aún así una solución ejemplo sobre el grafo resultante. No se consideran aristas que inciden en nodos cuyo máximo requisito de conectividad sea igual a su grado, puesto que en ese caso, ya sabemos que la arista es "imprescindible". El algoritmo ConstrucciónSimplificada invocado en la línea 5 es una versión simplificada de nuestros algoritmos de construcción. En el mismo no es relevante el costo de las aristas, y tampoco se requiere hallar el camino más corto, o los k-caminos más cortos entre

parejas de terminales, sino simplemente uno o k caminos cualesquiera entre dichos nodos. Si se logra hallar una solución ejemplo habiendo prescindido de la arista, se la elimina del grafo solución, y vuelve a intentarse la eliminación de otra arista, hasta que esto ya no sea posible. Finalmente se retorna el grafo minimal junto con la solución ejemplo que muestra su factibilidad como solución al GSP.

Invocar repetidas veces (tantas como aristas se intente eliminar) a ConstrucciónSimplificada puede tener un importante costo de cómputo; sin embargo los siguientes factores hacen esperable que la relación costo de cómputo vs beneficios obtenidos sea favorable a su uso en muchos casos:

- ConstrucciónSimplificada no necesita hallar los caminos "más cortos" sino caminos cualesquiera, problema éste de menor orden de ejecución
- Trabaja sobre un grafo que en general será mucho más simple (en cantidad de nodos y aristas) que el grafo de conexiones factibles original sobre el cual trabaja la fase de construcción; esto se acentúa si la proporción de nodos opcionales es alta, y también en particular si los requisitos de conectividad son bajos en relación al grado promedio del grafo, lo cual hace que el porcentaje de aristas de la solución factible sea muy bajo en relación al grafo de conexiones factible

Por supuesto que puede hacerse uso de criterios heurísticos para evitar la prueba de eliminación de todas las aristas, concentrándose solamente en algunas. Formas posibles de encarar esta tarea son:

- Focalizarse en aristas con baja cantidad de ocurrencias en los caminos del conjunto de caminos solución ejemplo conocido, asumiendo que aquellas aristas con alta ocurrencia tienden a ser aristas "altamente necesarias"
- Modificar los algoritmos de forma de mantener más caminos ejemplo que los necesarios (requeridos por R) para cada pareja de terminales; pudiendo entonces elegirlos de forma conveniente al momento de intentar la eliminación de una cierta arista

Dejamos planteada la evaluación del uso de estas técnicas sobre distintos tipos de instancias del GSP como parte de un futuro trabajo de investigación.

8.2. Alteración Aleatoria de Costos

Hemos visto que la alteración de costos bajo ciertas hipótesis garantiza que el algoritmo Construcción1 cumpla la propiedad que llamamos PC1 (probabilidad tan alta como se desee de obtener una solución optimal mediante la cantidad de iteraciones que sea necesaria). Hemos mostrado que la distribución exponencial cumple dichas hipótesis y puede por lo tanto ser empleada - como lo hicimos en la sección de pruebas.

Sin embargo resta investigar qué otras distribuciones que cumplan las hipótesis citadas puede emplearse, con eventuales mejores resultados. La distribución exponencial toma en cuenta solamente el valor esperado del costo de la arista como parámetro; esto deja de lado otros posibles factores que caracterizan al problema en cuestión, como por ejemplo la diversidad (dispersión) de los costos. Durante la fase de pruebas encontramos que en casos donde los costos de las aristas difieren poco entre sí la distribución exponencial genera valores muy dispersos, que tienden a producir soluciones "demasiado aleatorias". En varios de dichos casos se lograron mejores resultados con el empleo de otras formas de aleatorización, como sortear un costo en forma uniforme en un cierto radio alrededor del costo real de cada arista; esta forma de aleatorización no verifica las hipótesis necesarias para nuestra demostración referente a la PC1, pero en la práctica generó mejores resultados en varios casos.

Resulta por lo tanto deseable la introducción de otras distribuciones de probabilidad para alterar los costos, que reflejen de alguna forma la naturaleza de los costos originales; por ejemplo, que se parametrice según el costo real de la arista y la desviación estándar de todos los costos.

8.3. Recursión y Exploración de Nuevas Vecindades

Las vecindades de las que nos hemos valido para definir los movimientos vistos están basadas en la k-descomposición de grafos en key-paths y estructuras a las que hemos llamado key-stars, similares a los key-trees empleados en trabajos previos. A raíz de las pruebas efectuadas hemos encontrado otro tipo de estructuras pasibles de ser optimizadas pero mediante movimientos más complejos, o sea, no realizables mediante la composición secuencial de reemplazos de key-paths y key-stars. A su vez, hemos visto que el cómputo determinístico de reemplazos óptimos para tales estructuras es una tarea que se torna rápidamente compleja al considerar estructuras más complejas; por ejemplo en el caso de las key-stars, hemos propuesto un reemplazo por otra key-star, y para hallar la óptima es necesario recurrir al algoritmo que genera los k caminos disjuntos que suman un costo mínimo, lo cual supuso un notable aumento en el tiempo de ejecución. Por lo tanto entendemos que es interesante concebir mecanismos de vecindad que aúnen criterios heurísticos con reemplazo de estructuras complejas. Fue para ello que hemos planteado una extensión del GSP a la cual denominamos EGSP, mediante el cual bosquejamos la estructura de un algoritmo metaheurístico, que permitiría encontrar soluciones al GSP mediante invocaciones recursivas. Dicho algoritmo fue planteado en términos de particiones arbitrarias de una solución en subconjuntos de aristas, y como tal permite manejar estructuras tan complejas como se desee.

Al investigar esta dirección interesará entonces analizar aspectos como:

- Qué tipos de estructura es conveniente definir a fin de particionar la solución actual para las invocaciones recursivas; en función de topologías reales de difícil optimización mediante reemplazo de estructuras simples como las vistas.
- A partir de qué momento en la recursión es conveniente el uso directo de algoritmos

como los que hemos implementado.

8.4. Enfoque Alternativo de Construcción

Hemos presentado un enfoque diferente para la construcción de soluciones factibles, que parte de la idea de generar "muchos" caminos para cada pareja de terminales, y luego elegir los necesarios para cada pareja, con el objetivo "global" de que los caminos finales supongan el menor costo posible; para ello era deseable privilegiar a la vez caminos de bajo costo y caminos con aristas muy usadas por otros caminos. Vimos que este es un complejo problema de programación lineal entera, para el cual planteamos un mecanismo heurístico basado en scoring de caminos.

Si bien los resultados obtenidos en pruebas preliminares, empleando esta técnica de construcción en combinación con los movimientos estudiados Optimiza1/2/3, fueron en general inferiores (mayor costo) a los obtenidos mediante el uso de Construcción1, creemos que pueden obtenerse mejores resultados si se complementa esta técnica con una fase posterior de optimización local orientada a la **consolidación** de caminos. Por ella entendemos la introducción de movimientos que consideren conjuntos de caminos con extremos distintos 2 a 2, y reemplacen parte (o todo) de los mismos por otros conjuntos de aristas (buscando que las compartan entre sí y también eventual reutilización de aristas ya presentes en otros caminos); proponemos explorar la definición de funciones de energía asociadas a los caminos cuya minimización esté alineada a la consolidación de los mismos, y entonces hacer uso de mecanismos de optimización basados en buscar configuraciones que minimicen dichas funciones de energía como los propuestos por ejemplo en ([12], [13], [17], [18]), empleando modelos de redes neuronales aleatorias como los presentados en ([16], [14], [15]).

8.5. Relación entre las versiones Edge-Connected y Node-Connected

Hemos mencionado diversos aspectos comparativos entre la resolución del problema GSP en sus versiones EC y NC. Hemos visto que, para los mismos grafos de factibilidad, terminales y requerimientos de conectividad, es más complejo resolver el problema en versión EC, por descartarse menos aristas al fijar caminos, y por enfrentar estructuras más complejas en la descomposición de soluciones factibles; así como también que el espacio de soluciones factibles del problema NC está contenido propiamente en el homólogo del problema EC.

También hemos propuesto y demostrado la validez de una transformación inyectiva que pone en correspondencia a toda instancia del problema EC con una instancia de problema NC; de forma tal que las soluciones del último pueden transformase en soluciones del primero de igual costo, y en particular, que si son óptimas en el segundo, lo son también en el primero. Esta transformación, en ambos sentidos (problema EC en problema NC, solución NC en solución EC) es de muy bajo costo (de orden $max(|V|, |E|)$). Esta transformación fue encontrada estudiando la relación entre ambos problemas, buscando responder a la cuestión de si estamos realmente ante dos problemas "diferentes" o "equivalentes". En función de este resultado es

natural preguntarse si tiene sentido desarrollar técnicas para resolver ambos tipos de problema, más apropiadas para uno u otro caso, o si tiene sentido enfocarse solamente en la resolución del problema en versión NC (ya que el EC puede ser fácilmente llevado al mismo). Esto último no puede afirmarse a priori, puesto que como también hemos visto, el problema NC que corresponde a un problema EC queda planteado con el mismo conjunto de terminales y requerimientos de conectividad, pero sobre un grafo de conexiones factibles con más aristas y nodos que el del problema original EC. Dejamos planteada esta duda como futuro tema de investigación.

Otra cuestión que queda planteada es si existe una transformación inversa, es decir, si todo problema NC puede ser llevado en forma inyectiva a un problema EC a bajo costo, de forma que soluciones de éste generen soluciones de aquél de igual costo (y eventual optimalidad).

Bibliografía

[1] Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, 24(3):440–456, 1995.

[2] M. Baïou and A.R. Mahjoub. Steiner 2-edge connected subgraph polytope on series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 10(1):505 – 514, 1997.

[3] Mourad Baïou. *Le problème du sous-graphe Steiner 2-arête connexe : Approche polyédrale*. PhD thesis, Université de Rennes I, Rennes, France, 1996.

[4] Mourad Baïou. On the dominant of the Steiner 2-edge connected subgraph polytope. *Discrete Applied Mathematics*, 112(1-3):3 – 10, 2001.

[5] F. Barahona and A.R. Mahjoub. On 2-connected subgraph polytopes. *Discrete Mathematics*, 147(1):19–34, 1995.

[6] R. Bhandari. Optimal physical diversity algorithms and survivable networks. In *Computers and Communications, 1997. Proceedings., Second IEEE Symposium on*, pages 433 –441, July 1997.

[7] C.R. Coullard, A. Rais, D.K. Wagner, and R.L. Rardin. Linear-time algorithms for the 2-Connected Steiner Subgraph Problem on Special Classes of Graphs. *Networks*, 23(1):195 – 206, 1993.

[8] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[9] P. Festa. Greedy randomized adaptive search procedures. *AIROnews*, 7(4):7–11, 2003.

[10] Thorsten Koch. Konrad-Zuse-Zentrum fÃ¼r Informationstechnik Berlin. Steinlib test data library. http://steinlib.zib.de/steinlib.php.

[11] G.N. Frederickson and Jájá J. On the relationship between the biconnectivity augmentation and traveling salesman problem. *Theoretical Computer Science*, 13(1):189–201, 1982.

[12] E. Gelenbe. Hopfield energy of the random neural network. In *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, volume 7, pages 4681 –4686 vol.7, June 1994.

[13] E. Gelenbe, V. Koubi, and F. Pekergin. Dynamical random neural network approach to the traveling salesman problem. In *Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on*, pages 630 –635 vol.2, October 1993.

[14] Erol Gelenbe. Random neural networks with negative and positive signals and product form solution. *Neural Computation*, 1(4):502–510, 1989.

[15] Erol Gelenbe. Stability of the random neural network model. In Luis Almeida and Christian Wellekens, editors, *Neural Networks*, volume 412 of *Lecture Notes in Computer Science*, pages 56–68. Springer Berlin / Heidelberg, 1990.

[16] Erol Gelenbe and Andreas Stafylopatis. Global behavior of homogeneous random neural systems. *Applied Mathematical Modelling*, 15(10):534 – 541, 1991.

[17] Stafylopatis A. Gelenbe E. Minimum cost graph covering with the random neural network. *Computer Science and Operations Research, O. Balci (ed.)*, pages 139 – 147, 1992.

[18] Anoop Ghanwani. Neural and delay based heuristics for the steiner problem in networks. *European Journal of Operational Research*, 108(2):241 – 265, 1998.

[19] F. Harary. The maximum connectivity of a graph. *Proc. Nat. Acad. Sciences USA*, 48(1):1142–1146, 1962.

[20] Ruprecht-Karls-Universitat Heidelberg. Tsplib network optimization problems library. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95.

[21] H. Kerivin and A. Mahjoub. Design of Survivable Networks: A survey. *Networks*, 46(1):1 – 21, 2005.

[22] Alexander A. Kist, Er A. Kist, and Richard J. Harris. A heuristic to generate all best partially disjoint paths in a communications network. Technical report, Eighth International Conference on Communication Systems (ICCS 2002, 2002.

[23] J. Krarup. The generalized steiner problem. Technical report, DIKU, University of Copenhagen, 1979.

[24] A. Mahjoub. Two-edge connected spanning subgraphs and polyhedra. *Mathematical Programming*, 64(1):199 – 208, 1994.

[25] A.R. Mahjoub and C. Nocq. On the linear relaxation of the 2-node connected subgraph polytope. *Discrete Applied Mathematics*, 95(1):389–416, 1999.

[26] A.R. Mahjoub and P. Pesneau. On the Steiner 2-edge connected subgraph polytope. *RAIRO Operations Research*, 42(1):259–283, 2008.

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

[28] L.S. Pitsoulis and M.G.C. Resende. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 178–183. Oxford University Press, 2002.

[29] M.G.C. Resende. Greedy randomized adaptive search procedures (GRASP). In C.A. Floudas and P.M. Pardalos, editors, *Encyclopedia of Optimization*, volume 2, pages 373–382. Kluwer Academic Publishers, 2001.

[30] M.G.C. Resende and C.C. Ribeiro. GRASP and path-relinking: Recent advances and applications. In T. Ibaraki and Y. Yoshitomi, editors, *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, pages T6–1 – T6–6, 2003.

[31] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.

[32] F. Robledo and E. Canale. Designing backbone networks using the generalized steiner problem. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, volume 1, pages 327 – 334, October 2009.

[33] Franco Robledo. *GRASP heuristics for Wide Area Network design*. PhD thesis, IRISA, Université de Rennes I, Rennes, France, february 2005.

[34] V. Rutenburg. Efficient distributed algorithms for computing shortest pairs of maximally disjoint paths in communication networks. In *INFOCOM '91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s., IEEE*, pages 1214 –1219 vol.3, April 1991.

[35] Mechthild Stoer. *Design of Survivable Networks (Lecture Notes in Mathematics)*. Springer, 1983.

[36] J. W. Suurballe. Disjoint paths in a network. *Networks.*, 4(2):125–145, March 1974.

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

[38] H. Takahashi and A. Matsuyama. An approximate solution for the steiner problem in graphs. *Mathematica Japonica*, 24:537–577, 1980.

[39] M.G.A. Verhoeven and M.E.M. Severens. Parallel local search for steiner trees in graphs. *Annals of Operations Research*, 90:185–202, 1999. 10.1023/A:1018908614375.

[40] Pawel Winter. Steiner problem in networks: A survey. *Networks.*, 17(2):129–167, 2006.

Índice de figuras