Resolución del problema de los caminos disjuntos en grafos dirigidos usando técnicas metaheurísticas

B. Martín
I.N.E.C.O.
Avenida del Partenón, 4 - 4º
28042 Madrid
bernardo.martin@ineco.es

A. Sánchez
Depto. Ciencias de la Computación
E.S.C.E.T.
Universidad Rey Juan Carlos
28933 Móstoles (Madrid)
angel.sanchez@urjc.es

A. Duarte
Depto. Ciencias de la Computación
E.S.C.E.T.
Universidad Rey Juan Carlos
28933 Móstoles (Madrid)
abraham.duarte@urjc.es

Resumen

Este artículo presenta un trabajo en progreso que compara la aplicación de tres estrategias multiarranque constructivas (MSGA) para resolver de manera aproximada la versión combinatoria del problema de encontrar el mayor número de caminos con arcos disjuntos entre pares de nodos terminales en un grafo dirigido (problema DEDP). Se compara el comportamiento de los tres algoritmos implementados sobre un conjunto estándar de instancias de prueba. Los mejores resultados en calidad se consiguen mediante el algoritmo SP-MSGA.

1. Introducción

El problema de los caminos con arcos disjuntos (*Directed Edge-Disjoint Paths problem*, o abreviadamente problema DEDP) se puede definir de la forma siguiente [2]. Dado un grafo dirigido con pesos en los arcos $G=(V,A)$ y un conjunto T de solicitudes de conexión entre pares de nodos terminales $T = \{(s_1,t_1),(s_2,t_2), \ldots, (s_k,t_k)\}$ de V, se trata de determinar si los pares de nodos terminales (s_i,t_i), para todo $i=1,2,\ldots, k$, se pueden conectar mediante caminos formados por arcos disjuntos. La Figura 1 muestra un ejemplo de un grafo dirigido con 12 nodos y 15 arcos. Para este ejemplo, hay 6 nodos terminales, siendo el conjunto $T = \{(s_1,t_1),(s_2,t_2), (s_3,t_3)\}$. La resolución de este problema consiste en encontrar caminos entre pares terminales de tal forma que no se comparta ningún arco (aunque sí se pueden compartir nodos). En esta Figura 1 se muestra una solución al problema donde se han representado con un trazo más grueso y diferente color los arcos que pertenecen a cada uno de los tres caminos $s_i \xrightarrow{*} t_i$ que, en este ejemplo, forman parte de la solución. Obsérvese que no es necesario ni pasar por todos los nodos del grafo ni utilizar todos los arcos. En general, el número de caminos disjuntos encontrados será muy inferior al número de pares terminales inicialmente establecidos en el problema.

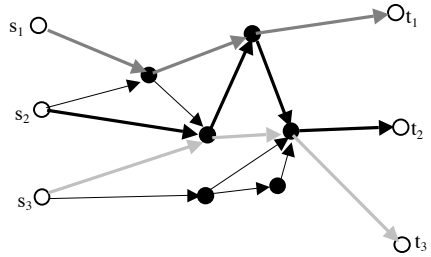

Figura 1. Ejemplo de solución del problema DEDP para un grafo G.

Este problema está bastante relacionado con el de establecer caminos con aristas disjuntas en grafos no dirigidos [8] (*Edge-Disjoint Paths problem*, abreviadamente problema EDP) y también con el problema de establecer caminos con nodos disjuntos (*Vertex-Disjoint Connecting Paths problem*, abreviadamente problema VDP).

Decidir si un conjunto de k pares de nodos terminales se pueden conectar mediante caminos disjuntos para el problema EDP es NP-completo. Hay que hacer notar que la dificultad de encontrar caminos disjuntos en los problemas EDP y DEDP es bastante diferente. Como una arista entre un nodo u y otro v es equivalente a un par de arcos ($u \rightarrow v$ y $v \rightarrow u$), se puede considerar que en términos de complejidad en tiempo EDP requiere un análisis de un mayor número de aristas. Por otro lado, al existir más arcos en el grafo, existe una probabilidad mayor de encontrar más caminos entre pares terminales. El problema DEDP es más restrictivo (entre cada par de nodos podría haber

un solo arco), resultando incluso ser NP-completo para el caso k=2 según fue demostrado por Fortune, Hopcroft and Wyllie [7]. Para digrafos acíclicos, DEDP es resoluble en tiempo O(knm^k) siendo n=|A| y m=|A| [7].

El problema EDP presenta una formulación similar al problema de Steiner generalizado (*Generalized Steiner Problem*, abreviadamente GSP) [10][12], que permite modelar el diseño de redes de comunicaciones de gran fiabilidad. En el GSP aparecen los mismos elementos que en el EDP y, adicionalmente, existe una matriz de costes asociada a las aristas del grafo. Se trata de encontrar un subgrafo de coste mínimo que conecte un mayor número de pares terminales, respetando la restricción de que los caminos entre pares terminales han de ser disjuntos en aristas.

Otra variante de estos problemas consiste en considerar un grado de congestión acotada en los arcos (problema DirEDPwC o *Directed Edge-Disjoint Paths with Congestion*) [1]. En este tipo de problemas el objetivo es establecer el mayor número de rutas disjuntas entre pares terminales con la restricción de que como máximo $c(N)$ caminos pueden ir a través de los arcos, siendo $c(N)$ un parámetro de congestión de los nodos.

Todos estos problemas son casos particulares de *problemas multiflujo en grafos* [2][14], donde dado un grafo $G=(V,A)$ (con aristas orientadas o no orientadas), un conjunto de terminales $T=\{(s_1,t_1),(s_2,t_2), \ldots, (s_k,t_k)\}$, un conjunto de demandas d_i (i=1,2,…,k) y una colección de capacidades c (enteras o racionales) asociadas a las aristas, se trata de encontrar para cada par terminal (s_i,t_i) un flujo f_i de valor d_i. Toda solución válida debe satisfacer la restricción de capacidad de las aristas $e \in A$:

$$\sum_{i=1}^{k} f_i(e) \le c(e)$$

En los problemas DEDP y EDP se fija $c(e)$=1, para toda $e \in A$.

La versión de optimización combinatoria para el problema DEDP consiste en establecer el mayor número de caminos disjuntos posibles entre elementos del conjunto T. Por lo tanto, el valor de la función objetivo $f(S)$ para una solución S al problema combinatorio considerado se define como: $f(S)=|S|$.

Las aplicaciones del problema DEDP [8] y de sus extensiones (problemas EDP y VDP, respectivamente) están relacionadas con el diseño VLSI, la planificación de tareas, el *routing* en sistemas de comunicaciones, y las redes de transporte, entre otras.

En este trabajo en progreso se comparan tres algoritmos voraces multiarranque, que hacen uso de una lista tabú, para resolver de manera aproximada el DEDP. El primero de ellos usa un algoritmo de caminos mínimos, el segundo ordena de forma decreciente los nodos adyacentes a uno dado en función del número de arcos salientes del nodo considerado, y el tercero ordena dichos nodos adyacentes en orden creciente.

En general, podría decirse que habría dos formas de resolver el problema combinatorio DEDP. La primera opción consiste en obtener soluciones aproximadas (por ejemplo, a través de técnicas metaheurísticas [9][5], que garantizan resultados de calidad en tiempos de ejecución razonables) pero asegurándose que toda solución parcial consistente en un camino entre dos pares terminales cumple la restricción de ser disjunto en arcos con los caminos previamente generados ente otros pares terminales. La segunda opción consiste en olvidarse de la restricción de que los caminos han de ser disjuntos, utilizando la técnica de la relajación de Lagrange. En este caso, se resolvería un problema de caminos mínimos entre múltiples nodos origen y destino. Después, por medio de un proceso iterativo, se resuelven conflictos de caminos no disjuntos, subiendo el coste de los arcos problemáticos (que tenderán a ser evitados en las siguientes iteraciones). En [8] se presenta un trabajo resuelve el problema EDP con este método y se aplica a problemas de tráfico en redes ATM. En nuestro artículo, se plantea una solución del primer tipo usando algoritmos voraces multiarranque.

Hasta donde sabemos, los algoritmos propuestos en este trabajo son los primeros que intentan resolver el problema DEDP de manera aproximada utilizando técnicas metaheurísticas. Blesa y Blum [3][4] han abordado el problema EDP usando un algoritmo de optimización basado en Colonias de Hormigas (ACO). El problema GSP también ha sido abordado por medio de algunas metaheurísticas; en particular, algoritmos genéticos, recocido simulado y VNS [12][13]. Nosotros hemos explorado, como propuesta inicial para resolver este problema, diferentes enfoques voraces con mecanismo multiarranque (MSGA) por la simplicidad del planteamiento y por ajustarse a la naturaleza del problema ya que

consideramos adecuado ir construyendo una solución de alta calidad satisfaciendo las restricciones impuestas por el problema.

El trabajo está organizado de la siguiente forma. La sección 2 presenta y describe en detalle los tres algoritmos propuestos para el problema DEDP. La sección 3 describe los grafos usados para los experimentos así como el análisis de la eficiencia de los algoritmos propuestos para diferentes instancias del problema. Finalmente, la sección 4 presenta las conclusiones del trabajo y los desarrollos futuros.

2. Algoritmos voraces multiarranque (MSGA) para el DEDP.

Los algoritmos voraces [6] constituyen estrategias sencillas que se emplean para resolver diversos problemas de optimización (p. ej. caminos mínimos en grafos, planificación de tareas, etc.). Una solución se construye incrementalmente, incorporando en cada paso a un elemento candidato que mantenga la factibilidad de la solución. Inicialmente, en conjunto de candidatos se ordena según cierto(s) criterio(s). En general, las estrategias voraces permiten conseguir buenas soluciones con un tiempo de cómputo razonable en algunos problemas.

A los diferentes algoritmos voraces propuestos para resolver el DEDP sobre un grafo $G=(V,A)$, se les añade una etapa multiarranque [9] para realizar una exploración más completa del espacio de búsqueda y para aleatorizar el punto de comienzo del algoritmo. Si disponemos de k pares terminales que queremos conectar mediante caminos con arcos disjuntos, existen $k!$ permutaciones o formas de ordenar dichos pares terminales. Es evidente que si elegimos buscar caminos a partir de dos permutaciones distintas de conjunto T, las soluciones conseguidas serán distintas. Por ello, se plantean N ($N<<k!$) ejecuciones independientes del algoritmo voraz específico. En los tres algoritmos propuestos se hace uso de una lista tabú durante la construcción de caminos entre pares de nodos terminales. El objetivo es evitar volver a visitar nodos (es decir, evitar ciclos) durante la construcción de un camino. A su vez, durante la ejecución del algoritmo se dispone del conjunto $\hat{A}$ de arcos "que se pueden utilizar" para formar caminos (inicialmente, $\hat{A}=A$) y cada vez que se selecciona un arco $a\in\hat{A}$ como parte de un camino se lo quita de $\hat{A}$ (si el camino no es factible, dicho arco se restituye al conjunto). La estructura común de los tres algoritmos MSGA planteados en este trabajo se representa mediante el siguiente pseudocódigo. En todos ellos, la función *ObtenerSolucionParcial* que busca un camino en G entre dos nodos y que es específica de cada algoritmo voraz planteado, se detallará en las subsecciones de este apartado. La salida del algoritmo será aquella solución S_{mejor} que conecte un mayor número de pares terminales satisfaciendo las restricciones del problema.

```
procedure MSGA (G, T, N)
// input: instancia del problema DEDP:
//    G = (V,A) es un grafo dirigido,
//    T = {(s_i,t_i) | ∀i∈1..k: s_i,t_i∈V} pares terminales,
//    N = numero permutaciones de T (multiarranques)
// output: mejor solucion S_mejor al problema MSGA
var
  S_j : solucion parcial iteracion j-esima
  π : conjunto de permutaciones sobre elementos de T
  Â : conjunto factible de arcos para caminos s_i →* t_i
  f : valor de la funcion objetivo para una solucion
  L_tabu : lista (conjunto) tabu de nodos
  P_i : camino entre nodos terminales s_i y t_i
begin
  S_mejor = ∅;
  π={π_1,..., π_|k| | π_i permutación de {(s_i,t_i)|i∈1..k }∈T};
  for j = 1..N do // primeras N permutaciones de π
    S_j = ∅;     // inicializar solucion parcial de iteracion j
    Â = A;       // arcos factibles para construir caminos
    for l = 1..k do // ∀(s_l,t_l)∈T ∧ k = |V|
      L_tabu = ∅;
      L_tabu = L_tabu ∪ {s_l} // incluir nodo s_l en la lista tabú
      P_l = ObtenerSolucionParcial (G, L_tabu, s_l, t_l);
      S_j = S_j ∪ {P_l};   // incluir camino completo en S_j
      Â = Â – {a | a∈P_l};  // quitar de Â los arcos de P_l
    end for;
    if f(S_j) > f(S_mejor) then
      S_mejor = S_j ;    // actualizar major solucion
    end if
  end for
end; // MSGA
```

Algoritmo 1. Algoritmo voraz multiarranque (MSGA) propuesto para el problema DEDP.

A continuación, se describen las tres estrategias voraces propuestas (incluidas dentro de la plantilla del MSGA y correspondientes a la función *ObtenerSolucionParcial*): la primera se basa en la obtención de caminos más cortos en

número de arcos (*shortest paths*, o abreviadamente SP) entre pares de nodos terminales; la segunda, consiste en seleccionar los nodos adyacentes a uno dado, basándose en la ordenación decreciente según su número de arcos de salida (*Decreasing Out-Degree*, o abreviadamente DOD); finalmente, la tercera estrategia es similar a la segunda pero basándose en una ordenación creciente de los arcos de salida (*Increasing Out-Degree*, o abreviadamente IOD).

2.1. Estrategia SP-MSGA.

Esta estrategia voraz calcula el camino más corto entre cada par terminal de nodos $(s_l, t_l) \in T$, $\forall l \in \{1..k\}$. Dicho camino mínimo puede obtenerse por medio del algoritmo de Dijkstra [6].

```
P_l = function ObtenerSolucionParcial (G, L_tabu, s, t_l);
//    G = (V,A) es un grafo dirigido,
//    L_tabu es la lista tabú de nodos
//    s es nodo actual,
//    t_l es el nodo final del par terminal (s_l,t_l)∈T
// output: camino más corto P_l entre s y t_l
var
  a : conjunto de nodos adyacentes uno dado
begin
  if (s == t_l) then
     return P_l;   // camino encontrado
  else if s ∈ NodoFinal(G) then // nodo sin sucesores
     return ∅;     // no hay camino
  else // algoritmo de Dijkstra de caminos mínimos
     |P_l| = ∞;
     a = NodosAdyacentes(s);
     for i = 1..|a| do
       L_tabu = L_tabu ∪ (a_i};   // nodo a_i en lista tabu
       P_i = ObtenerSolucionParcial (G, L_tabu,. a_i, t_l.);
       if |p_i| < |p_l| then
         L_tabu = L_tabu − {n | n∈P_l};  // n se borran de L_tabu
         P_l = P_i ;    // se actualiza la mejor solucion
       end if
     end for
   end if
end; // ObtenerSolucionParcial SP-MSGA
```

2.2. Estrategia DOD-MSGA.

Esta estrategia voraz va calculando incrementalmente el camino entre pares terminales $(s_l, t_l) \in T$. Para ello, ordena los nodos adyacentes al nodo actual en orden decreciente al número de arcos salientes de dichos nodos adyacentes (grado de salida), y se elige aquel nodo con mayor grado de salida.

```
P_l = function ObtenerSolucionParcial (G, , L_tabu , s, t_l);
//    G = (V,A) es un grafo dirigido,
//    L_tabu es la lista tabú de nodos
//    s es nodo actual,
//    t_l es el nodo final del par terminal (s_l,t_l)∈T
// output: camino de adyacencia decreciente P_l de s a t_l
var
  a : conjunto de nodos adyacentes uno dado
begin
  if (s == t_l) then
     return P_l;   // camino encontrado
  else if s ∈ NodoFinal(G) then // nodo sin sucesores
     return ∅;     // no hay camino
  else // algoritmo adyacencias decrecientes
     a = NodosAdyacentes(s);
     a' = OrdenarDecrecienteGradoSalida (a);
     l = 1;
     do // si no hay camino y hay nodos adyacentes
        L_tabu = L_tabu ∪ (a'_l};
        P_l = ObtenerSolucionParcial (G, L_tabu,. a'_l, t_l.);
        if ⌐Camino(P_l) then
          L_tabu = L_tabu − {n | n∈P_l};  // n se borran de L_tabu
        end if
     while (⌐Camino(P_l)) ^ (l<|a'|)
  end if
end; // ObtenerSolucionParcial DOD-MSGA
```

2.3. Estrategia IOD-MSGA.

Esta estrategia voraz es muy similar a la anterior. La diferencia reside en la ordenación en orden creciente al número de arcos salientes del nodo actual (grado de salida), y se elige por lo tanto aquel nodo adyacente con menor grado de salida.

```
P_l = function ObtenerSolucionParcial (G, , L_tabu , s, t_l);
//    G = (V,A) es un grafo dirigido,
//    L_tabu es la lista tabú de nodos
//    s es nodo actual,
//    t_l es el nodo final del par terminal (s_l,t_l)∈T
// output: camino de adyacencia decreciente P_l de s a t_l
var
  a : conjunto de nodos adyacentes uno dado
begin
  if (s == t_l) then
     return P_l;   // camino encontrado
  else if s ∈ NodoFinal(G) then // nodo sin sucesores
     return ∅;     // no hay camino
  else // algoritmo adyacencias crecientes
     a = NodosAdyacentes(s);
     a' = OrdenarCrecienteGradoSalida (a);
     l = 1;
     do // si no hay camino y hay nodos adyacentes
        L_tabu = L_tabu ∪ (a'_l};
        P_l = ObtenerSolucionParcial (G, L_tabu,. a'_l, t_l.);
        if ⌐Camino(P_l) then
          L_tabu = L_tabu − {n | n∈P_l};  // n se borran de L_tabu
        end if
     while (⌐Camino(P_l)) ^ (l<|a'|)
```

```
  end if
end; // ObtenerSolucionParcial IOD-MSGA
```

3. Resultados experimentales

Para la experimentación del problema DEDP se han usado algunos de los grafos del *benchmark* construido por Blesa y Blum [3][4] para el problema EDP. En nuestro caso, para resolver el problema DEDP se ha considerado que solo existen arcos dirigidos en un sentido (en lugar de aristas, que corresponden a pares de arcos en ambos sentidos). Para una descripción detallada de dichas grafos, remitimos al lector al trabajo [4] de dichos autores para resolver el problema EDP. Algunos de sus grafos han sido creados usando el generador de grafos BRITE [11].

Todos los algoritmos se han codificado en C++ y se han ejecutado en un PC con procesador Intel(R) Pentium(R) 3. En la tabla 1 se presentan los resultados obtenidos para los tres algoritmos constructivos multiarranque propuestos en este trabajo. En la primera columna aparece el nombre del grafo (*Graph*), en la segunda el número de pares terminales o *commodities* (#c) y en las tres siguientes los resultados obtenidos por cada algoritmo (SP-MSGA, DOD-MSGA e IOD-MSGA, respectivamente). En relación a los grafos usados en los experimentos [3][4]: *G1* corresponde la grafo denominado **bl-wr2-wht2.10-50.sdeg.bb** en el *benchmark* de Blesa y Blum para el problema EDP; *G2* corresponde al grafo **graph3.bb**; *G3*, al grafo **AS-BA.R-Wax.v100e217.bb** y *G4*, al grafo **mesh25x25.bb**, respectivamente.

Teniendo en cuenta que los algoritmos descritos en ese trabajo tienen una componente aleatoria, cada uno de ellos se ha ejecutado 10 veces independientes para cada instancia del problema. Para conocer su robustez y fiabilidad de las implementaciones, para cada algoritmo mostrado en la Tabla 1 se presenta la media del mejor valor de las 10 ejecuciones (*mean_q*), el mejor valor obtenido (*max_q*) en esas 10 iteraciones, el menor (*min_q*) y, finalmente, el tiempo medio de ejecución en segundos *T*(*s*) para cada instancia (*Graph*, *#c*) del problema DEDP. En la Tabla 1, se ha marcado en negrita el mejor valor en número de pares terminales encontrados (*max_q*) para cada instancia.

		SP-MSGA				DOD-MSGA				IOD-MSGA			
Graph	#c.	mean_q	max_q	min_q	T(s)	mean_q	max_q	min_q	T(s)	mean_q	max_q	min_q	T(s)
	50	5,8	**10**	2	12,9	5,0	9	2	15,2	4,9	9	2	15,3
	125	11,3	**15**	8	21,0	9,7	12	7	24,9	9,7	13	7	26,1
G1	200	17,1	24	14	26,6	16,5	**31**	13	35,2	15,8	21	13	34,0
	16	9,1	**11**	7	3,4	8,5	10	7	3,4	8,1	10	7	3,2
	41	12,2	**14**	11	4,1	11,6	13	10	4,2	11,8	**14**	9	3,9
G2	65	14,8	**18**	11	4,6	13,8	16	11	4,5	13,0	15	11	4,3
	10	3,7	**7**	1	0,5	3,0	**7**	1	0,5	3,0	**7**	1	0,5
	25	8,9	**11**	6	0,8	8,0	9	5	0,8	8,1	10	5	0,8
G3	40	10,8	**15**	4	1,0	9,2	13	3	1,2	9,4	13	3	1,2
	62	17,5	**25**	13	104,0	17,0	23	13	105,0	16,7	23	13	106,0
	156	35,0	**43**	32	192,8	33,1	37	30	191,3	33,6	37	31	186,8
G4	250	45,1	**48**	41	220,6	41,7	44	39	218,3	42,4	44	39	214,6

Tabla 1: Resultados experimentales para los tres algoritmos multiarranque constructivos (MSGA).

Los resultados experimentales muestran una ligera superioridad en calidad del algoritmo constructivo multiarranque SP-MSGA con respecto a los otros dos algoritmos desarrollados (DOD-MSGA e IOD-MSGA, respectivamente). Se observa que para la mayor parte de las instancias consideradas, dicho algoritmo obtiene un valor más elevado tanto de la media de los pares terminales que es capaz de encontrar (*mean_q*), como para el máximo número de pares terminales (*max_q*) en las 10 ejecuciones realizadas. Por otro lado, los tiempos de ejecución son bastante similares para los tres algoritmos al resolver cada instancia considerada del problema.

Es importante destacar que los resultados mostrados en esta tabla no son directamente comparables con los publicados en Blesa y Blum [3][4]. Estos autores resolvieron el problema para grafos no dirigidos. Por ese motivo, el número de pares terminales que ellos encuentran es bastante más elevado para cualquier instancia. Esto se debe a que el problema EDP (como se comentó en la Sección 1 de este trabajo) es más sencillo de resolver que el problema DEDP. Por contra, los tiempos que obtiene nuestro algoritmo son menores que los que obtenían Blesa y Blum. De nuevo, esto se debe a que en grafos dirigidos (problema DEDP) existe un número significativamente menor de caminos que explorar, con lo que el problema DEDP se resuelve en menos tiempo que el equivalente DEDP.

4. Conclusión

Se ha presentado un trabajo preliminar que compara la aplicación de tres estrategias multiarranque constructivas (denominadas SP-MSGA, DOD-MSGA e IOD-MSGA, respectivamente) para resolver de manera aproximada el problema DEDP. Se han analizado y comparado los algoritmos implementados sobre un conjunto adaptado estándar de instancias de prueba para el problema EDP. Aunque los resultados en tiempo son similares para los tres algoritmos considerando las mismas instancias a resolver, los mejores resultados cualitativos se consiguen mediante el algoritmo SP-MSGA. Como futuro trabajo, pretendemos completar los algoritmos introduciendo una etapa de mejora sobre la solución construida (a modo similar que lo hace la metaheurística GRASP).

Referencias

[1] Andrews, M., Chuzhoy, J., Sanjeev K. y Zhang, L., "Hardness of the undirected edge-disjoint paths problem with congestion", *Proc. 46th Annual IEEE Symposium on Foundations of Computer Science* (FOCS 2005), págs. 226-241, 2005.

[2] Bang-Jensen, J. and Gutin, G., *Digraphs: Theory, Algorithms and Applications*, Springer Monographs in Mathematics, Springer, 2001.

[3] Blesa, M. y Blum, C., "Ant Colony Optimization for the Maximum Edge-Disjoint Paths Problem", G.R. Raidl et al. (eds.): EvoWorkshops 2004, *LNCS 3005*, págs. 160-169, 2004.

[4] Blesa, M. y Blum, C., "Finding edge-disjoint paths in networks by means of artificial ant colonies", *Journal of Mathematical Modelling and Algorithms*, 2007. (en prensa)

[5] Blum, C. y Roli, A., "Metaheuristics in combinatorial optimization: Overview and conceptual comparison", *ACM Computing Surveys,* vol. 35, no. 3, págs. 268-308, 2003.

[6] Brassard, G. y Bratley P., *Fundamentals of Algorithmics*, Prentice Hall, 1996.

[7] Fortune, S., Hopcroft, J.E., y Wyllie, J., "The directed subgraph homeomorphism problem", *Theoretical Computer Science*, v. 10, págs. 111-121, 1980.

[8] Gawrilow, E. y otros, "Disjoint routing in telecommunication networks", *Technical Report*, TU Berlin, 2005. URL: http://www.math.tu-berlin.de/coga/projects/routing/disjointrouting

[9] Glover, F. y Kochenberger, G. (editores), *Handbook of Metaheuristics*, Kluwer, 2003.

[10] V. Kahn, V. Crescenzi, P., *A compendium of NP optimization problems*. URL: www.nada.kth.se/theory/problemlist.html.

[11] Medina, A., Lakhina, A., Matta, I. y Byers, J., *BRITE: Boston University Representative Internet Topology Generator*, 2001. URL: http://cs-pub.bu.edu/brite.

[12] Nesmachnow, S., Cancela, H. y Alba, E., "Técnicas Evolutivas Aplicadas al Diseño de

Redes de Comunicaciones Confiables", *Actas del Tercer Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados* (MAEB'04), págs. 388-395, 2004.

[13] Nesmachnow, S., "Evaluating Simple Metaheuristics for the Generalizad Steiner Problem", *Journal of Computer Science & Technology*, vol. 5, no. 5, págs. 285-291, 2005.

[14] M.L.H. Rasmussen, *Disjoint paths in directed networks with length and distribution criteria*, Master Thesis, Technical University of Denmark, 2005.