Resolución del problema EDP para Grafos No Dirigidos utilizando algoritmos MSGA y GRASP

Bernardo Martín[1], Abraham Duarte[2] y Ángel Sánchez[2]

[1]INECO. Paseo de la Habana, 138. 28036 Madrid
bernardo.martin@ineco.es
[2]Dpto. Ciencias de la Computación, Universidad Rey Juan Carlos.
28933 Móstoles (Madrid)
{abraham.duarte, angel.sanchez}@urjc.es

Abstract. En este trabajo se comparan diferentes estrategias implementadas basadas en algoritmos voraces multi-arranque (MSGA) y en algoritmos GRASP para el problema *Edge Disjoint-Paths* (EDP), que consiste en encontrar el mayor número de caminos con arcos disjuntos entre pares de nodos terminales dados, en grafos no dirigidos. Los resultados se han evaluado sobre grafos de *benchmarks* estándares definidos para este problema.

Keywords. Problema EDP (*Edge-Disjoint Paths*), metaheurísticas, algoritmos voraces multi-arranque, GRASP, *k-shortest paths*, redes de comunicaciones.

1 Introducción

El problema de encontrar el mayor número de caminos disjuntos en grafos no dirigidos (conocido como *Edge-Disjoint Paths problem*, o abreviadamente EDP), presenta una importante utilidad práctica para redes de comunicaciones [2][7]. En efecto, en este tipo de redes una operación fundamental consiste en el establecimiento de rutas de conexión (es decir, caminos en el grafo correspondiente), que resulten ser simultáneas y disjuntas, para el intercambio de información entre pares de nodos terminales de la red. Ello obedece a que las peticiones de conexión pueden llegar simultáneamente y, también, se trata de establecer en mayor número de rutas entre nodos terminales de manera que no interfieran entre ellas mediante la compartición de *switches* (nodos) y/o de links (aristas) entre los caminos del grafo. Este hecho implicaría el cumplimiento de unas restricciones técnicas y también un mayor grado de paralelismo en la comunicación [1][2].

Este trabajo presenta nuevos algoritmos para la resolución de una versión del problema EDP donde los caminos no pueden compartir aristas. Otras versiones del problema consideran que no se pueden compartir nodos del grafo [2]. Las soluciones planteadas se basan en versiones de algoritmos voraces multi-arranque (*Multi-Start Greedy Algorithms*, abreviadamente MSGA) y de algoritmos GRASP [3]. El problema puede describirse más formalmente como sigue. Sean $G=(V, E)$ un grafo no diri-

gido, que representa a la red de comunicaciones, y sea $T=\{(s_i,t_i) \mid (i = 1, 2, \ldots, |T|) \cap (s_i, t_i \in V)\}$ una colección predefinida de *commodities* o pares de nodos terminales de G que pueden ser conectados a través de un camino. Un conjunto de *commodities* es realizable si existe una colección de caminos entre cada nodo terminal s_i con cada nodo terminal t_i, tales que los caminos correspondientes no comparten arcos. Se considera aquí el problema combinatorio de establecer el mayor número de conexiones entre los dos nodos componentes de *commodities* de T tal que dicho conjunto sea realizable en el grafo original G. Por lo tanto, el valor de la función objetivo $f(S)$ de una solución S factible del problema EDP, se define como: $f(S) =|S|$. Se trata de maximizar el número de pares terminales conectados tal que resulten realizables en G. El problema de decisión relativo a si un conjunto de *commodities* dado T es o no realizable sobre un grafo G es NP-completo [4]. Otras variantes del problema EDP para árboles parciales de orden k [8] o para grafos serie-paralelos [6] también son problemas NP-completos. En la literatura se han propuesto principalmente soluciones basadas en colonias de hormigas (ACO) para el problema considerado [7][1][2].

2 Algoritmos voraces multi-arranque (MSGA) implementados

2.1 Descripción de estrategias plantadas

La resolución del problema mediante los algoritmos voraces multi-arranque se ha realizado a partir de las implementaciones previas realizadas por los autores en [5], pero, en esta ocasión, se usan grafos no dirigidos. En aquel estudio, se implementó el algoritmo voraz siguiendo tres criterios a la hora de buscar soluciones en cada una de las iteraciones a ejecutar:

- Estrategia *sp-MSGA* (*shortest paths MSGA*): Se obtiene el camino entre cada par terminal $(s_i,t_i)\in T$ buscando el camino mínimo por medio del algoritmo de Dijkstra.
- Estrategia *dod-MSGA* (*decreasing out-degree MSGA*): Para crear el camino entre pares terminales $(s_i,t_i)\in T$, toma para cada nodo que pertenezca al camino, el nodo adyacente con mayor grado de adyacencia posible, y lo añade a dicho camino.
- Estrategia *iod-MSGA* (*increasing out-degree MSGA*): Similar al anterior, en este caso para construir el camino se elige el nodo con menor grado de adyacencia posible.

Además de las estrategias anteriores, en este trabajo se presenta la opción de precarga o precálculo *off-line* de resultados mediante el algoritmo *k-shortest paths* (que llamaremos estrategia *ksp-MSGA*).

2.2 Implementación del algoritmo según la estrategia *ksp-MSGA*

Teniendo en cuenta que los algoritmos de las estrategias *sp-MSGA*, *dod-MSGA* e *iod-MSGA* son similares a los desarrollados para los grafos dirigidos presentados en

[5], en este caso se detallará únicamente la estrategia *ksp-MSGA*. Básicamente, el algoritmo se divide en dos partes bien diferenciadas. Por un lado, se encuentra la función general `MSGA(N,solKSP)` que realiza las labores de diversificación de las tareas de búsqueda, creando para cada arranque una lista ordenada y aleatoria de los pares terminales a conectar. Por otro lado, se usa la función auxiliar `obtenerSolucionParcial(G,`L_{tabu}`,s,`t_1`)` que realiza las labores de búsqueda local para cada par terminal (s_i, t_i). Esta función toma los posibles caminos asignables a dicho par de la información que le proporciona el algoritmo *k-shortest paths*. A continuación, se muestra el pseudocódigo de ambas funciones.

```
function MSGA(N,solKSP=AlgoritmoKSP(G, T, K))
  input:
    N= número de permutaciones de T para multi-arranque
    G=(V,A), grafo dirigido
    T={(s_i,t_i)| ɣiϵ1..m: s_i,t_i ϵ V pares terminales
    K número caminos calculados por ksp para cada par terminal(s_i,t_i).
    solKSP= {c1..cm}|c_i={d_1..d_r} | ɣiϵ1..m /
         s_i,t_i ϵ V, pares terminales de c_i
         d_1..d_r ϵ ci, lista de caminos con longitud creciente |r| < K
  output: mejor solución S_mejor al problema EDP con MSGA
  var
    S_j: solución parcial iteración j-ésima
    Π: conjunto de permutaciones sobre elementos de T
    f: valor de la función objetivo para una solución
    P_i: camino entre los nodos terminales s_i y t_i
    f_mejor: valor de la función objetivo para S_mejor
  begin
    S_mejor=nulo;
    f_mejor=-infinito;
    Π={Π_1..Π_|k|| Π_i permutación de {(s_i,t_i)|iϵ1..k}ϵT};
    for j=1..N do // primeras N iteraciones de Π
      S_j=nulo; // inicializa solución parcial para iter. j
      for l=1..Q do // ɣ(s_l,t_l)ϵT ^ Q=|V|
        P_i= obtenerSolucionParcial(solKSP_l);
        S_j=S_j+P_i; // incluir solución parcial al total
        Â= Â-{a|aeP_j}; //eliminar de Â los arcos de P_i
      end for
      if f(S_j) > f(S_mejor) then
        S_mejor = S_j // actualiza la mejor solución
      end if
    end for
end function
```

```
function obtenerSolucionParcial(c1)
  input: c1, caminos solución ksp en longitud creciente del par (s1,t1)
  output: camino P1 con menor longitud entre s1 y t1
  var Ltabu, lista tabú de caminos
  begin
       Ltabu = nulo;
       do
         di= primerCamino(c1) | di ¬∈ Ltabu;
         if di ∈ Â then
           return di;
         else
           Ltabu=Ltabu+di;
       while (di!=nulo);
end function
```

3 Algoritmos GRASP implementados

3.1 Implementaciones según estrategias *sp-GRASP*, *dod-GRASP* e *iod-GRASP*

Los algoritmos GRASP según las estrategias *sp-GRASP*, *dod-GRASP* e *iod-GRASP*, estrategias descritas en la sección anterior, se han implementado como algoritmos voraces multi-arranque, con la particularidad de incluir aleatoriedad en la elección de los "mejores nodos", según el valor dado a α que es el parámetro usado para el cálculo de la lista restringida $RCL\alpha$ en el GRASP.

En el pseudocódigo presentado a continuación, se muestra solamente la parte de búsqueda local del algoritmo para el criterio *sp-GRASP*. Como se puede apreciar, para esta estrategia la variable CL es una lista de caminos ordenados por longitud creciente. Los algoritmos para las estrategias *dod-GRASP* e *iod-GRASP* son similares al anterior. Para estos casos, CL es una lista de caminos, ordenados por la suma del grado de adyacencia de los nodos que forman el camino, en orden decreciente y creciente, respectivamente.

```
function obtenerSolucionParcial(G,Ltabu,s,t1)
  inputs:
    G=(V,A), grafo no dirigido; s, nodo actual
    Ltabu, lista tabú de nodos
    t1, nodo final del par terminal (s1,t1)∈T
  output: camino más corto p1 entre s y t1
  var
    a: conjunto de nodos adyacentes a uno dado
    CL: Lista de caminos, ordenados por longitud
        creciente, candidatos a generar solución parcial
    RCLα: Lista restringida de los α mejores caminos de CL
```

```
  begin
    L_tabu= L_tabu+s;
    if (s= t_1) then
      return t_1; //camino encontrado
    else if s ϵ nodoFinal(G) then //nodo sin sucesores
      return nulo; //no encuentra camino
    else // algoritmo de Dijkstra de caminos mínimos
      a=nodosAdyacentes(s);
      for i=1..|a| do
        if a_i ┐ϵ L_tabu then // si no se encuentra en L_tabu
          P_i= obtenerSolucionParcial(G,L_tabu,a_i,t_1);
          CL=CL+P_i;
        end if
      end for
      P_1=obtenerCaminoAleatoriamente(RCL_α);
      //actualiza L_tabu, eliminando caminos de la lista excepto escogida
      L_tabu= L_tabu-{n|n ϵ (RCL-P_1)};
      return P_1;
    end if-else
end function
```

3.2 Implementación con estrategia de precarga (*ksp-GRASP*)

Para el caso de la estrategia *ksp-GRASP*, se aplica el algoritmo GRASP sobre una lista CL obtenida a partir de los resultados obtenidos de la ejecución previa del algoritmo *k-shortest paths*. Dicha lista contiene los caminos en orden creciente de longitud. De esta manera, será más sencillo obtener el umbral y la lista restringida RCL_α.

```
function MSGA(N,CL=ordenacionCaminosPorLongitud(solKSP=
                  AlgoritmoKSP(G, T, K)))
  inputs:
    N= número de permutaciones de T para multi-arranque
    G=(V,A), grafo dirigido
    T={(s_i,t_i)| ɣiϵ1..m: s_i,t_i ϵ V pares terminales
    K es una constante que indica el número de caminos
   calculados por KSP para cada par terminal (s_i,t_i).
    solKSP= {c_1..c_m}|c_i={d_1..d_r} | ɣiϵ1..m /
         s_i,t_i ϵ V, pares terminales de c_i
         d_1..d_r ϵ ci, lista de caminos en orden de longitud
        creciente |r| < K
    CL= { d_1..d_u | ɣd_iϵ{c_1..c_m} /  |d_i-1|≤ |d_i|≤ |d_i+1|
         lista de caminos solución ordenados de KSP
          ordenados por longitud creciente
```

```
  output: mejor solución S_mejor al problema EDP con GRASP
  var
    S_j: solución parcial iteración j-ésima
    f: valor de la función objetivo para una solución
    f_mejor: valor de la función objetivo para S_mejor
  begin:
    S_mejor=nulo;
    f_mejor=infinito;
    for j=1..N do // primeras N iteraciones de Π
      S_j=nulo; // inicializa solución para it. j
      S_j= obtenerSolucionIteracion(CL);
      if f(S_j) > f(S_mejor) then
        S_mejor = S_j // actualiza la mejor solución
      end if
    end for
end function

function obtenerSolucionIteracion (CL)
  input:
    CL: { d_1..d_u | γd_i∈{c_1..c_m} / |d_i-1|≤ |d_i|≤ |d_i+1|
  output: conjunto de caminos S_i obtenidos del conjunto CL
          a partir de una posición inicial en la lista Π_i
  var
    RCL_α: Lista restringida de los α mejores caminos de CL
    umbral: longitud máxima para caminos c ∈ CL / c ∈ RCL_α
    Π={Π_1..Π_|CL|| Π_i permutación de RCL_α}
    L_tabu, lista tabú de caminos
    S_j: solución parcial iteración
  begin
      S_j = nulo; // iniciar solución de la iteración
      L_tabu  = nulo; // iniciar L_tabu
      do
      // Calcular umbral: camino long. mín y máx de CL.
      Umbral= |d_1| + α*( |d_u|- |d_1|)/ d_1,d_u ∈ CL;
      //calcular RCL_α
      RCL_α= c_x ∈ CL / |c_x|< Umbral;
        //tomar camino aleatorio de RCL_α
        d_i =random({d_1..d_u} ∈ RCL_α) ^ d_i ┐∈ L_tabu;
        // si el camino es factible: de momento no
        // pertenece, a la solución, ni ésta contempla los
        // pares terminales del camino.
```

```
            if (di ┐∈ Sj) ^ ((si,ti)∈T)^((si,ti)∈ di)/
               ((si,ti) ┐∈ Sj) then
               Sj = Sj + di; // se añade a la solución de la it.
               CL = CL – di; // se elimina camino de CL. Se
                   //actualiza dl y du si se diese el caso.
            end if
            //se actualiza la lista para impedir que se tome
            //varias veces el mismo camino
            Ltabu  =Ltabu+ di;
         // mientras que no se haya seleccionado todos los
         // elementos de RCLα
         while ┐(∀di ∈ RCLα | di ∈ Ltabu);
         return Sj;
end function
```

4 Resultados

Los grafos utilizados en las pruebas de los algoritmos desarrollados, que corresponden a un subconjunto de los presentados en los trabajos [1] y [2], son los siguientes: *graph3.bb* (renombrado abreviadamente en las sucesivas tablas de resultados como *G1*), *graph4.bb* (renombrado como *G2*), *AS-BA.R-Wax.v100e217.bb* (renombrado como *G3*), *bl-wr2-wht2.10-50.sdeg.bb* (renombrado como *G4*) y *mesh25x25.bb* (renombrado como *G5*, respectivamente). El parámetro *#c* representa el número de *commodities* o pares terminales establecidos para cada grafo.

Para cada estrategia algorítmica implementada aparecen tres columnas:

- La columna q_{max} es el valor máximo de función objetivo obtenida mediante la aplicación del algoritmo sobre un grafo dado para el total de las *commodities* o instancias de pares terminales.
- La segunda de las columnas, q_{med}, muestra la media de los valores de función objetivo obtenida mediante la aplicación del algoritmo sobre un grafo dado para el total de las *commodities* o instancias de pares terminales.
- La tercera de las columnas *t(s)* indica el valor medido del tiempo de ejecución del algoritmo (en segundos) para un conjunto o instancia de pares terminales y un grafo concreto.

Para todos los experimentos se han usado los siguientes valores de parámetros: N=100 y K=10, respectivamente, donde N representa el número de permutaciones de T para la ejecución multi-arranque y K es una constante que indica el número de caminos precalculados por el algoritmo *k-shortest paths* para cada par terminal (s_i,t_i). Los experimentos se han realizado en un ordenador Intel Pentium Dual-Core E5800 a 3,20GHz y con 4GB RAM.

4.1 Algoritmos MSGA

Resultados de las estrategias de distancia mínima *sp* y de precarga *ksp*

		sp-MSGA			*ksp-MSGA*		
Grafo	*#c*	q_{max}	q_{med}	*t(s)*	q_{max}	q_{med}	*t(s)*
	16	16	15	5,46	16	14	0,02
G1	41	29	25	6,30	30	25	0,03
	65	37	29	6,67	37	32	0,04
G2	43	38	35	91,93	35	30	0,17
	10	9	7	0,55	9	7	0,01
G3	25	15	12	0,64	15	12	0,02
	40	19	16	0,66	20	16	0,02
	50	24	19	25,10	28	20	0,26
G4	125	39	33	28,85	43	36	0,30
	200	54	43	31,73	57	46	0,36
	62	26	20	37,16	42	34	0,41
G5	156	32	27	39,19	61	56	0,47
	250	34	30	39,67	75	68	0,56

Table 1. Resultados para las estrategias *sp-MSGA* y *ksp-MSGA*

Resultados de las estrategias de máxima y mínima adyacencia

		dod-MSGA			*iod-MSGA*		
Grafo	*#c*	q_{max}	q_{med}	*t(s)*	q_{max}	q_{med}	*t(s)*
	16	11	9	2,13	16	15	11,43
G1	41	16	12	2,33	30	26	12,96
	65	17	14	2,44	37	30	13,44
G2	43	18	14	19,12	41	37	190,01
	10	9	6	0,49	10	8	1,15
G3	25	13	10	0,61	15	12	1,35
	40	16	12	0,67	19	16	1,48
	50	19	15	20,10	26	20	54,43
G4	125	30	24	24,98	45	35	69,44
	200	35	29	28,18	60	47	80,74
	62	15	12	22,62	29	24	122,18
G5	156	21	17	25,06	38	32	127,36
	250	26	20	26,29	42	37	128,68

Table 2. Resultados para las estrategias *dod-MSGA* e *iod-MSGA*

4.2 Algoritmos GRASP

Resultados de las estrategias de distancia mínima *sp* y de precarga *ksp*

		sp-GRASP			*ksp-GRASP*		
Grafo	*#c*	q_{max}	q_{med}	*t(s)*	q_{max}	q_{med}	*t(s)*
	16	12	10	0,89	15	13	1,08
G1	41	15	13	1,04	29	24	12,91
	65	18	15	1,10	35	31	47,88
G2	43	18	15	8,17	33	29	15,32
	10	8	6	0,22	8	6	0,33
G3	25	13	10	0,27	15	12	3,33
	40	16	13	0,29	20	16	12,07
	50	17	13	6,35	26	20	23,10
G4	125	28	22	8,63	42	35	331,71
	200	35	28	10,13	56	45	1360,53
	62	17	13	11,70	40	32	42,87
G5	156	21	18	13,28	61	54	647,15
	250	25	21	14,24	73	68	2695,39

Table 3. Resultados para las sstrategias *sp-GRASP* y *ksp-GRASP*

Resultados de las estrategias de máxima y mínima adyacencia

		dod-GRASP			*iod-GRASP*		
Grafo	*#c*	q_{max}	q_{med}	*t(s)*	q_{max}	q_{med}	*t(s)*
	16	13	10	0,90	13	10	1,70
G1	41	16	13	1,91	17	13	1,92
	65	18	15	2,00	18	15	2,04
G2	43	17	15	15,37	19	15	15,45
	10	9	7	0,44	9	7	0,43
G3	25	13	10	0,63	13	10	0,55
	40	18	13	0,64	18	13	0,62
	50	21	16	17,78	21	16	17,91
G4	125	32	26	22,90	31	26	22,90
	200	36	31	26,69	36	32	26,67
	62	18	13	14,66	20	14	14,76
G5	156	23	18	16,94	22	18	17,11
	250	25	21	18,83	27	22	19,07

Table 4. Resultados para las estrategias *dod-GRASP* e *iod-GRASP*

Comparando los resultados para estos grafos con los presentados por Blesa y Blum en [2], usando un algoritmo ACO extendido, puede resumirse que nuestros resultados en relación con el valor de la función objetivo (valores q_{max} y q_{med}) son ligeramente inferiones a los presentados en [2]. Sin embargo, los correspondientes tiempos de ejecución conseguidos *t(s)* por nuestros algoritmos para dichos grafos mejoran considerablemente (en promedio, en un 98,94%) a los presentados por Blesa y Blum.

5 Conclusión

Este trabajo compara experimentalmente diferentes versiones de algoritmos de tipo MSGA y GRASP para el problema EDP. Para decidir cuál es el mejor algoritmo desarrollado, hay que tener en cuenta el valor alcanzado por cada algoritmo para la función objetivo y también el tiempo de ejecución correspondiente. Con respecto al primer criterio, se puede apreciar que para los algoritmos MSGA, los mejores resultados corresponden a las estrategias *iod* y *ksp*. La estrategia *iod*, funciona bien para todos los grafos analizados, excepto para la instancia *mesh25×25*. La estrategia *ksp* obtiene buenos resultados para todos los grafos analizados. En el caso de los algoritmos GRASP, la estrategia *ksp* es la que genera los valores de la función objetivo más altos. Se presentan buenos resultados en todas las instancias, mostrando los mejores resultados para los grafos *AS-BA.R-Wax.v100e217* y *mesh25×25*. En cambio, GRASP no funciona bien si se decide por utilizar alguna de las estrategias restantes. En resumen, con respecto a la función objetivo, MSGA obtiene los mejores resultados cuando siguen las estrategias *ksp* o *iod*. Entre los GRASP, la estrategia *ksp* produce los mejores resultados. Si se introduce el tiempo en la decisión de mejor resultados, entonces *ksp-MSGA* aparece como ganador absoluto, obteniendo el mejor tiempo para todos los grafos seleccionados. Como futuro trabajo se plantea abordar el problema *Directed Edge-Disjoint Paths* (DEDP) usando un enfoque similar al propuesto en este trabajo.

Agradecimientos

Este trabajo ha sido financiado parcialmente por el proyecto TIN2011-29827-C02-01 del Ministerio de Ciencia e Innovación.

Bibliografía

1. Blesa, M. y Blum C.: Ant Colony Optimization for the Maximum Edge-Disjoint Paths Problem, G.R. Raidl et al. (eds.): *EvoWorkshops 2004*, *LNCS 2005*, págs. 160-169, 2004.
2. Blesa, M. y Blum C.: Finding edge-disjoint paths in networks by means of artificial ant colonies, *Journal of Mathematical Modelling and Algorithms* 6(3), págs. 361-391, 2007.
3. Blum C. y Roli A.: Metaheuristics in combinational optimization: Overview and conceptual comparison, *ACM Computing Surveys* 35(3), págs. 268-308, 2003.
4. Karp, R.: Complexity of computer computations. *Chapter Reducibility among Combinatorial Problems*, Plenum Press, New York, págs. 85-103, 1972.
5. Martin B., Sánchez A. y Duarte A.: Resolución del problema de los caminos disjuntos en grafos dirigidos usando técnicas metaheurísticas. *CEDI 2007*. *JAEM 2007*, 2007.
6. Nishizeki, T., Vygen J. y Zhou, X.: The edge-disjoint paths problem is NP-complete for series–parallel graphs. *Discrete Applied Mathematics* 115, págs. 177–186, 2001.
7. Nowé, A., Verbeeck, K. y Vrancx, P.: Multi-Type Ant Colony: The Edge Disjoint Paths Problem. M. Dorigo et al. (eds.): *Ants 2004, LNCS 3172*, págs. 202-213, 2004.
8. Zhou, X. y Nishizeki, T.: The Edge-Disjoint Paths Problem is NP-Complete for Partial k-Trees, *Proceedings of the 9th International Symposium on Algorithms and Computation* (ISAAC'98), págs. 417-426, 1998.