Búsqueda Dispersa para la minimización del número de cortes en grafos jerárquicos incrementales

A. Martínez-Gavara[1], J. Sánchez-Oro[2], M. Laguna[3], A. Duarte[2], and R. Martí[1]

[1] Departamento de Estadística e Investigación Operativa, Universidad de Valencia, València, Spain,
{gavara, rafael.marti}@uv.es
[2] Department of Computer Science, Universidad Rey Juan Carlos, Móstoles, Spain,
{jesus.sanchezoro, abraham.duarte}@urjc.es
[3] Leeds School of Business, University of Colorado at Boulder, USA,
laguna@colorado.edu

Resumen Este trabajo se centra en el área del dibujo automático de grafos, en la que los algoritmos sitúan los vértices y las aristas de un grafo de un modo que resulte operativo para su manejo. Los métodos de dibujo de grafos que preservan la disposición de dibujos anteriores se denominan incrementales y tienen su campo de aplicación en las áreas de planificación y logística en las que se realizan actualizaciones frecuentes de las diferentes tareas como turnos, horarios o inventarios. Proponemos pues un algoritmo heurístico basado en la metodología Búsqueda Dispersa para la resolución de este problema en el contexto de los grafos jerárquicos, que sirven para modelar cualquier grafo acíclico dirigido. La experiencia computacional realizada muestra la eficiencia del algoritmo y su superioridad frente a propuestas anteriores.

Keywords: dibujado de grafos, búsqueda dispersa, grafo incremental, metaheurística

1. Introducción

La mayoría de los sistemas de información complejos requieren de una representación visual de un grafo, de manera que sea sencillo de interpretar. Los grafos se han convertido en la unidad básica de modelado en numerosas áreas, como por ejemplo la gestión de proyectos, la planificación de producción, el equilibrado de líneas, los procesos de negocio o la visualización de software. El dibujo de grafos (*graph drawing*) es actualmente un área de investigación bien establecida, con numerosas publicaciones, entre la que destacamos la monografía [1]. De entre los diferentes criterios responsables de una buena representación de un grafo, la minimización del número de cortes se considera uno de los más importantes [2]. En este artículo proponemos un algoritmo heurístico basado en la metodología de búsqueda dispersa para obtener soluciones eficientes a dicho problema de optimización.

Actas de la XVII Conferencia de la Asociación Española para la Inteligencia Artificial, pp. 271-280

Este trabajo se centra en la representación de grafos dirigidos acíclicos jerárquicos (*Hierarchical Directed Acyclic Graph* - HDAG), también conocidos como grafos por capas o jerarquías. La representación de un HDAG se lleva a cabo estableciendo los vértices en una serie de líneas verticales equiespaciadas denominadas capas, de manera que todos los arcos siguen la misma dirección. El problema de minimizar el número de cortes que se producen en cada capa en este tipo de grafos es $\mathcal{NP}$-completo incluso si solo existen dos capas [4]. La Figura 1 muestra una posible representación de un HDAG dividido en tres capas, donde la primera capa tiene cuatro vértices y la segunda y tercera presentan tres vértices cada una.

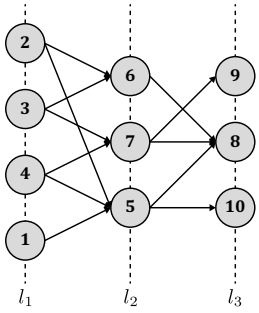
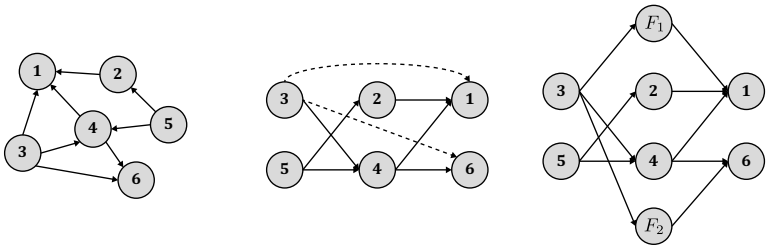


Figura 1: Representación de una jerarquía (HDAG) de tres capas.

Cabe destacar que existen numerosos algoritmos para transformar cualquier grafo dirigido acíclico (*Directed Acyclic Graph* - DAG) en un HDAG, de manera que pueda ser representado como tal [6]. Un método estándar consiste en disponer los vértices en capas, de manera que todas las aristas vayan en el mismo sentido (de una capa inferior a una superior). Algunos algoritmos minimizan los cruces y el número de aristas que atraviesan varias capas al realizar esta disposición. Después, para eliminar las aristas que atraviesan varias capas, crean un nodo ficticio por cada capa atravesada por la arista, obteniendo un HDAG. Por lo tanto, el algoritmo propuesto es aplicable a cualquier grafo acíclico dirigido.

La Figura 2 muestra un ejemplo de la transformación de un DAG a un HDAG. En concreto, se parte del DAG representado en la Figura 2a con seis vértices y ocho aristas dirigidas. En la Figura 2b los vértices han quedado dispuestos en capas, de manera que la primera capa contiene a los vértices 3 y 5, la segunda capa a los vértices 2 y 4 y, por último, la tercera capa a los vértices 1 y 6. Como se puede observar, hay dos aristas que atraviesan más de una capa (representadas con línea discontinua): las aristas (3,1) y (3,6). Para eliminar estas aristas, se crea un nodo ficticio por cada capa que atraviesan: F_1 y F_2 para las aristas (3,1) y (3,6), respectivamente.

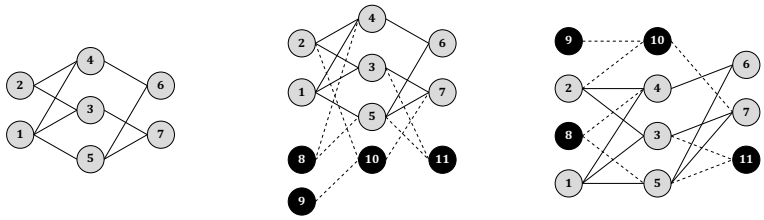
En este trabajo abordamos el problema del dibujo incremental, que consiste en añadir unos vértices y aristas a una jerarquía bien representada (con un



(a) (b) (c)

Figura 2: Proceso de conversión de un grafo acíclico dirigido (a) a un grafo jerárquico (c)

número de cortes relativamente bajo). El problema consiste en ubicar los nuevos vértices de modo que se minimice el número total de cortes de las aristas sin alterar el orden relativo entre los vértices de la jerarquía original.



(a) Grafo original (b) Grafo incremental con 4 nuevos vértices (c) Solución para el grafo incremental

Figura 3: Creación de un grafo incremental y una posible solución al mismo.

La Figura 3 muestra el proceso de creación de una jerarquía incremental, así como una posible solución (dibujo) de la misma. En concreto, la Figura 3a muestra un dibujo de la jerarquía original donde se producen dos cortes. La Figura 3b muestra la creación de la jerarquía incremental, añadiendo cuatro nuevos vértices (representados con fondo negro) y sus correspondientes aristas (representadas con líneas discontinuas). Los nuevos vértices se añaden, inicialmente, al final de cada capa, resultando en un dibujo con catorce cortes. La Figura 3c representa un dibujo diferente (optimizado) de la misma jerarquía, resultante de la reordenación de los vértices de cada capa con el objetivo de reducir el número de cortes totales, obteniendo un total de nueve. Es importante destacar que el orden relativo de los vértices originales se mantiene.

Aunque el problema general de minimizar cortes de aristas en jerarquías ha sido muy estudiado, el problema incremental ha recibido poca atención. Hasta la fecha solo conocemos un trabajo [10] para el caso concreto de grafos bipartidos (jerarquías de dos capas) En él se propone un algoritmo exacto basado en la ramificación y poda para el caso bipartido, así como un método heurístico basado en GRASP (*Greedy Randomized Adaptive Search Procedure*) para resolver ejemplos de tamaño grande en tiempos razonables. Nuestro primer objetivo es extender el algoritmo GRASP presentado en el trabajo previo al caso general de jerarquías con más de dos capas. Después proponemos un algoritmo heurístico basado en búsqueda dispersa para obtener soluciones de mejor calidad para este problema. Presentamos finalmente una comparativa entre ambos algoritmos.

2. Notación y definiciones

Se define un grafo jerárquico $H = (V, E, k, L)$ como un grafo acíclico dirigido $G = (V, E)$ donde V y E representan el conjunto de vértices y aristas respectivamente, k es el número de capas y $L(v) : V \to \{1, 2, \ldots, k\}$ es la función que asigna a cada vértice el número de la capa en la que se encuentra. Notar que $\forall (u, v) \in E, L(v) - L(u) = 1$. La función L define, implícitamente, el conjunto $L_i = \{v \in V : L(v) = i\}$, con $1 \leq i \leq k$, que llamaremos la capa i de la jerarquía. De este modo, el conjunto de vértices V queda particionado como la unión del conjunto de capas:$V = \bigcup_{i=1}^{k} L_i$.

Dado que las aristas se representan como la línea recta que une dos vértices de capas consecutivas, un dibujo del grafo queda determinado por una ordenación de los vértices de cada capa. Se define pues un dibujo de la jerarquía H como $D = (H, \Phi)$ donde $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ y φ_i es la ordenación (permutación) de la capa L_i. Es decir, $\varphi_i(v)$ es la posición de v en la capa L_i.

El problema de optimización de minimizar el número de cruces en una jerarquía puede ser reformulado como el de encontrar las ordenaciones óptimas de cada capa. En este sentido, podemos hablar del dibujo óptimo $D^\star$ como aquel de modo que ningún otro dibujo D de la jerarquía H tiene un número menor de cruces.

Un cruce se produce en la intersección de dos aristas $(u, v), (w, t) \in E$ donde $u, w \in L_i$ y $v, t \in L_{i+1}$, y tal que $\varphi_i(u) < \varphi_i(w)$ y $\varphi_{i+1}(v) > \varphi_{i+1}(t)$ ó $\varphi_i(u) > \varphi_i(w)$ y $\varphi_{i+1}(v) < \varphi_{i+1}(t)$

Dados dos vértices $u, w \in L_i$, donde $\varphi(u) < \varphi(w)$, se define $K_s(u, w)$ como el número de cruces que se producen entre las capas L_i y L_{i+1} debido a las aristas incidentes con ambos vértices (y con sus sucesores). Más formalmente,

$$K_s(u, w) = \sum_{v \in N(u) \cap L_{i+1}} \left| t \in N(w) \cap L_{i+1} : \varphi_{i+1}(t) < \varphi_{i+1}(v) \right|,$$

donde $N(u) = \{v \in V : (u, v) \in E\}$ es el conjunto de vértices adyacentes a u. De manera similar, se define $K_p(u, w)$ como el número de cruces que se producen entre las capas L_{i-1} y L_i con aristas incidentes a u y w (y con sus predecesores).

Utilizando estas definiciones, el número de cruces que se producen entre las capas i e $i+1$ de un dibujo $D = (G, \Phi)$ se calcula como:

$$K_s(D, i) = \sum_{\substack{u,w \in L_i \\ \varphi_i(u) < \varphi_i(w)}} K_s(u, w).$$

El número de cortes totales que se producen entre aristas en la solución se define como $K(D) = \sum_{i=1}^{k-1} K_s(D, i)$. El objetivo de este trabajo es encontrar aquella solución $D^\star$ que minimiza el número de cortes entre aristas.

Desde hace más de una década, el método estándar para minimizar el número de cruces en una jerarquía es el denominado **algoritmo del baricentro** aplicado capa a capa [1]. Este algoritmo considera como fijo el orden φ_i en la capa L_i y determina la ordenación de la capa L_{i+1} aplicando el algoritmo del baricentro. Éste consiste en calcular la posición de un vértice $u \in L_{i+1}$ como el promedio de las posiciones de sus adyacentes en L_i. Con un "barrido por capas de izquierda a derecha" se sitúan todos los vértices. El algoritmo aplica ahora el mismo procedimiento en la dirección contraria (desde la última capa hasta la primera). En este caso, la posición de un vértice u en la capa L_i se calcula como el baricentro de sus adyacentes en la capa L_{i+1}. El algoritmo se detiene tras varios "barridos hacia delante y hacia atrás" cuando los vértices ya no cambian de posición.

A partir de un grafo jerárquico $H = (V, E, k, L)$, se define un grafo incremental $IH = (IV, IE, k, IL)$ como el grafo jerárquico resultante de añadir nuevos vértices y aristas al grafo original sin modificar el número k de capas, de manera que $V \subseteq IV$, $E \subseteq IE$ y $IL(v) : IV \to \{1, 2, \ldots, k\}$, con $IL(v) = L(v)\ \forall v \in V$.

Dada una jerarquía incremental IH de una jerarquía H, y un dibujo D de H, el problema del dibujo de grafos incrementales (*Incremental Graph Drawing Problem*, IGDP) consiste en encontrar un dibujo ID de IH de manera que se minimice el número de cruces de aristas, respetando el orden relativo de los vértices originales de H en cada capa. Notar que el dibujo incremental tiene sus orígenes en la necesidad de respetar la disposición original del grafo a la vez que buscamos una buena ubicación y trazado de los nuevos elementos.

A partir de un dibujo $D = (H, \Phi)$ de la jerarquía H, se define un dibujo incremental ID de la jerarquía IH como $ID = (IH, \Pi)$ donde $\Pi = \{\pi_1, \pi_2, \ldots \pi_k\}$, π_i es la ordenación (permutación) de la capa L_i y el orden relativo de los vértices de H se mantiene en IH. Es decir, para dos vértices $u, w \in V \cap L_i : \varphi_i(v) < \varphi_i(w)$, se ha de cumplir que $\pi_i(v) < \pi_i(w)$.

3. Búsqueda Dispersa

Búsqueda Dispersa [9] es una metaheurística poblacional que consta de cinco procedimientos principales con el objetivo de construir, mantener y transformar una población de soluciones. La Figura 4 presenta el esquema clásico de un algoritmo que implementa búsqueda dispersa.

El algoritmo comienza con la generación de un conjunto de soluciones diversas (Sección 3.1), las cuales se someterán al método de mejora descrito en la

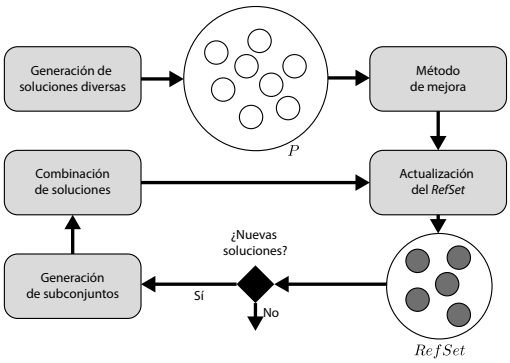


Figura 4: Esquema de la metodología Búsqueda Dispersa

Sección 3.2. El tamaño del conjunto inicial de soluciones P es un parámetro del algoritmo, aunque en este diseño preliminar consideraremos que es igual a 100. A continuación, se crea el conjunto de referencia (*RefSet*) con un tamaño predeterminado b, igual a 10, que contiene a las mejores y más diversas soluciones de entre las generadas en el paso anterior. Cabe destacar que la selección de las soluciones más diversas es un problema $\mathcal{NP}$-difícil en sí mismo, por lo que se suele llevar a cabo de manera heurística. En el algoritmo propuesto se seleccionan en primer lugar las $b/2$ mejores soluciones y se incluyen en el *RefSet*. Para completar el *RefSet* se elige, en cada iteración, la solución más diversa del conjunto inicial con respecto al *RefSet*, que será aquella que presente una distancia máxima con respecto a las soluciones ya presentes en el *RefSet*. Una solución para el IGDP se representa como la ordenación de cada una de las capas de la jerarquía, por lo que definimos la distancia entre dos soluciones como el número de elementos que ocupan posiciones diferentes en cada una de las capas. La distancia respecto al *RefSet* se define como la mínima distancia entre la solución candidata a entrar y el resto de soluciones del *RefSet*. El resultado de esta etapa es un conjunto de referencia con un total de b soluciones.

A continuación, el algoritmo aplica el método de generación de subconjuntos. Este método genera todos los pares de soluciones pertenecientes al *RefSet* que no hayan sido combinadas con anterioridad. Entonces, a cada par se le aplica el método de combinación descrito en la Sección 3.3, y a la solución resultante, el método de mejora. La solución mejorada pasa a ser una candidata a entrar en *RefSet*. Una solución entrará en *RefSet* si y solo si presenta una calidad superior a la peor solución del *RefSet* actual. En ese caso, sustituirá a la solución del *RefSet* con la que guarde la mínima distancia de entre aquellas soluciones de peor calidad. En caso de reemplazar una o más soluciones del *RefSet*, el algoritmo continua aplicando el método de combinación entre aquellos pares de soluciones no combinados durante la ejecución. El método que presentamos finaliza cuando no se ha producido ningún cambio en *RefSet* tras aplicar el método de combinación y evaluar todas las posibles soluciones candidatas.

De los cinco métodos que constituyen el algoritmo búsqueda dispersa, dos de ellos, la construcción de subconjuntos del *RefSet* y la actualización del *RefSet* los hemos implementado según el diseño estándar. Esto es, nos limitamos a considerar todas las parejas del *RefSet* no combinadas anteriormente, y entran en el *RefSet* las nuevas soluciones según su calidad. Los otros tres métodos restantes que definen la metodología se describen a continuación.

3.1. Método constructivo

El método de generación de soluciones diversas propuesto se basa en la metodología GRASP [3]. Inicialmente se considera que los vértices del grafo original, V, están situados según la ordenación del dibujo original D. A continuación, se añaden los vértices de $IV \setminus V$, siguiendo como criterio voraz el grado del vértice candidato con respecto a los vértices ya incluidos en el dibujo. Es decir, se selecciona, al azar, un vértice de una lista de candidatos restringida formada por aquellos cuyo grado supere un umbral definido como $g_{max} - \alpha \cdot (g_{max} - g_{min})$, donde g_{min} y g_{max} son el grado mínimo y máximo, respectivamente, y $\alpha \in [0, 1]$ es un parámetro que controla la voracidad / aleatoriedad del método. El vértice seleccionado se coloca en la posición correspondiente a su baricentro.

3.2. Método de mejora

En esta sección se describe el método de mejora propuesto dentro del marco de la búsqueda dispersa. La búsqueda local itera sobre cada una de las capas en orden lexicográfico y, dentro de cada capa, se examinan los vértices en el orden del dibujo o solución actual. Para cada capa i y cada vértice v, el método calcula el número de cortes que se producirían si éste se insertara en la posición anterior, $\pi_i(v) - 1$, y en la siguiente, $\pi_i(v) + 1$, a la actual. A continuación se lleva a cabo el mejor de los dos movimientos evaluados, siempre y cuando éste produzca una mejora en el valor de la función objetivo de la solución actual. Si durante la exploración de las capas hemos realizado algún movimiento de mejora, el algoritmo comienza de nuevo, hasta que no se encuentre ningún movimiento capaz de mejorar la solución actual.

3.3. Método de combinación

Nuestro método de combinación se basa en la metodología *Path Relinking* (PR) [5], originalmente propuesta en el contexto de *Tabu Search*. Se basa en explorar trayectorias que conectan soluciones de alta calidad, generando en el camino soluciones intermedias que podrían superar a la mejor solución encontrada en la búsqueda. PR puede considerarse como una generalización de los métodos clásicos de combinación, en el sentido de que a partir de dos soluciones genera un conjunto de soluciones intermedias.

Dadas dos soluciones ID_s y ID_g, *Path Relinking* comienza desde la solución inicial ID_s y la transforma gradualmente hasta obtener la solución guía, ID_g. La

transformación se lleva a cabo a través de movimientos que introducen en ID_s elementos presentes en ID_g. Se propone como movimiento para la generación de soluciones intermedias la copia de una capa completa de ID_g sobre ID_s. El algoritmo genera la primera solución intermedia, ID_s^1, copiando sobre ID_s la primera capa de ID_g. A continuación, se copiará la segunda capa de ID_g sobre ID_s^1, generando la solución ID_s^2 y así sucesivamente, hasta generar la solución ID_s^{k-1} en $k-1$ pasos, que compartirá con ID_g las ordenaciones de todas las capas salvo la última. El algoritmo devolverá la mejor solución intermedia encontrada durante la generación del camino.

Tal y como se ha documentado en aplicaciones previas de *Path Relinking* [8], es posible que las soluciones intermedias no mejoren a las soluciones que las originaron; sin embargo, pueden ser un punto de partida excelente para la exploración de un área prometedora del espacio de soluciones. Por ello, se recomienda la aplicación de algún tipo de búsqueda local. Notar, sin embargo, que la aplicación de la búsqueda local a todas las soluciones intermedias sería computacionalmente muy costosa y podría producir óptimos locales repetidos. Por ello, limitamos la aplicación de la búsqueda local a la mejor solución del camino.

4. Resultados experimentales

En esta sección se presentan los resultados experimentales obtenidos durante el desarrollo del trabajo. Todos los experimentos se han ejecutado sobre un Intel Core i7 920 (2.67 GHz) con 8 GB de RAM. Hemos generado un conjunto de instancias originales basándonos en el método propuesto en [7], utilizando un número de capas en el rango $\{2, 6, 13, 20\}$ y variando la densidad del grafo en el rango $\{6.5\,\%, 17.5\,\%, 30\,\%\}$.

Una vez generadas las instancias originales, hemos situado sus vértices dentro de cada capa aplicando sucesivamente el algoritmo del baricentro hasta no poder reducir más el número de cortes de aristas. Después, hemos añadido un conjunto de vértices y aristas para obtener el grafo incremental que constituye la instancia final sobre la que aplicaremos nuestro algoritmo. En concreto, el porcentaje de vértices y aristas añadidos al grafo original ha sido de 20 % y 60 %.

Además, el número de vértices en cada capa se elige de manera aleatoria en el rango $[5, 30]$. Para cada combinación de valores de los parámetros anteriores se ha creado una instancia, generando un total de 24 instancias. La Tabla 1 muestra los resultados obtenidos por los diferentes métodos analizados sobre este conjunto. En concreto, P representa a la mejor solución obtenida en el conjunto inicial (es decir, realizando únicamente 100 construcciones, sin aplicar el método de mejora); *RefSet* representa la mejor solución contenida en el *RefSet* inicial, tras el método de mejora; y *SS* representa la mejor solución obtenida al finalizar la ejecución del algoritmo de búsqueda dispersa completo.

Tal y como se ha descrito en la introducción, el mejor método heurístico previo encontrado en la literatura es un algoritmo GRASP [10], que se compone de un método constructivo aleatorizado y una búsqueda local, ambos basados en

el algoritmo del baricentro. Los resultados del método constructivo se muestran en la columna *Const. previo* mientras que los del algoritmo completo (constructivo junto con la búsqueda local) se presentan en la columna *GRASP previo*. Siguiendo las instrucciones de los autores, el algoritmo previo se ha ejecutado hasta completar un total de 100 construcciones junto con 100 mejoras. Hemos tratado de equiparar el tiempo de ejecución del *SS* con el del método *GRASP* previo.

El conjunto de instancias se ha dividido en función del número de capas de cada instancia, de manera que hay 6 instancias para cada valor de $k = \{2, 6, 13, 20\}$. De cada método se presenta el promedio de las desviaciones respecto a los mejores valores conocidos. Además, se introducen dos filas adicionales en la Tabla 1 que contienen la desviación promedia sobre las 24 instancias, Desv. (%), y el tiempo de ejecución medio, Tiempo (s) del algoritmo sobre el total de instancias.

Tabla 1: Comparativa de los resultados obtenidos por el método propuesto (SS) y el mejor algoritmo del estado del arte (GRASP previo), con sus respectivos algoritmos constructivos.

	k	*P*	*RefSet*	*SS*	Const. previo	GRASP previo
Desv. (%)	**2**	74.85	0.51	0.00	174.07	91.91
	6	41.66	0.91	0.00	48.09	23.50
	13	25.60	3.59	0.00	37.34	21.39
	20	19.05	2.05	0.00	38.93	22.20
Desv. (%)		40.29	1.76	0.00	74.61	39.75
Tiempo (s)		0.15	0.36	1.83	0.31	16.09

Como se puede observar, la búsqueda dispersa se sitúa como el mejor método de entre todos los evaluados, con una desviación del 0 %, lo que significa que siempre encuentra la mejor solución conocida. Además, cabe destacar que el método constructivo propuesto supera los resultados obtenidos por el método constructivo previo (40.29 % frente a 74.61 %), necesitando la mitad del tiempo de ejecución (0.15 frente a 0.31 segundos). El algoritmo GRASP previo tiene una desviación de 39.75 %, necesitando un tiempo de computación 8 veces mayor que el resto. También es relevante mencionar que el tiempo de ejecución es en todos los casos muy reducido, incluso para las instancias más grandes (20 capas de vértices).

Para comprobar estadísticamente que los resultados son significativos, se han utilizado tests no paramétricos. En concreto, se ha utilizado el test de Friedman sobre los resultados de todos los algoritmos, resultando en un p-valor inferior a 0.0001, lo que indica que existen diferencias significativas entre los métodos considerados. Adicionalmente consideramos una prueba no paramétrica para dos muestras emparejadas (test de los signos) que confirma, con un p-valor inferior a 0.0001, la superioridad de nuestra propuesta frente a las propuestas anteriores.

5. Conclusiones

En este trabajo hemos abordado el problema incremental del dibujo de grafos, en el que reducimos el número de cortes de las aristas añadidas a un grafo ya dibujado. Hemos aplicado la metodología de búsqueda dispersa para diseñar un algoritmo de resolución del problema, proponiendo un diseño básico que es capaz de obtener soluciones competitivas en tiempos muy pequeños. En concreto, proponemos un método constructivo basado en el baricentro, una búsqueda local basada en un intercambio en posiciones consecutivas, y un método de combinación basado en *Path Relinking*. El algoritmo resultante mejora el algoritmo *GRASP* previamente publicado tanto en calidad de la solución encontrada como en tiempos de computación.

Agradecimientos

Este trabajo está parcialmente financiado por los proyectos TIN2015-65460-C2-1-P y TIN2015-65460-C2-2-P (MINECO / FEDER) del Ministerio de Economía y Competitividad, y por la Comunidad Autónoma de Madrid, con el proyecto S2013ICE-2894.

Referencias

1. Di Battista, G., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edn. (1998)
2. Carpano, M.J.: Automatic Display of Hierarchized Graphs for Computer-Aided Decision Analysis. IEEE Transactions on Systems, Man, and Cybernetics 10(11), 705–715 (1980)
3. Feo, T.A., Resende, M.G.C., Smith, S.H.: A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. Operations Research 42(5), 860–878 (1994)
4. Garey, M.R., Johnson: Crossing Number is NP-Complete. SIAM Journal on Algebraic Discrete Methods 4(3), 312–316 (1983)
5. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA (1997)
6. Healy, P., Nikolov, N.S.: Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers, chap. How to Layer a Directed Acyclic Graph, pp. 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
7. Manuel Laguna, Rafael Martí, Vicente Valls: Arc crossing minimization in hierarchical digraphs with tabu search. Computers & Operations Research 24(12), 1175 – 1186 (1997)
8. Laguna, M., Martí, R.: GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. INFORMS Journal on Computing 11(1), 44–52 (1999)
9. Laguna, M., Martí, R.: Scatter search: Methodology and implementations in C. Kluwer Academic Publisher, Norwell, MA, USA (2002)
10. Rafael Martí, Vicente Estruch: Incremental bipartite drawing problem. Computers & Operations Research 28(13), 1287 – 1298 (2001)