Scatter Search aplicado aplicada al problema de la minimización del *profile* de matrices y grafos

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

[1] Dept. Ciencias de la Computación, Universidad Rey Juan Carlos (Spain)
[2] Leeds School of Business, University of Colorado Boulder (USA)
[3] Dept. de Estadística e Investigación Operativa, Universidad de Valencia (Spain)
jesus.sanchezoro@urjc.es, abraham.duarte@urjc.es, laguna@colorado.edu, rafael.marti@uv.es

Resumen En este trabajo se propone un algoritmo basado en *Scatter Search* para resolver el problema de la minimización del *Profile*. Se utiliza la estrategia *Path Relinking* como método base para la generación de nuevas soluciones mediante combinación. El problema del Profile (PMP) es NP-duro y tiene aplicaciones relevantes en técnicas de análisis numérico basadas en la manipulación de matrices de gran tamaño. Fue propuesto a comienzos de los 70 pero en el estado del arte no se encuentra ningún método eficiente para la resolución del problema. La amplia experimentación llevada a cabo en este trabajo demuestra que se ha cumplido el objetivo de establecer un nuevo estándar en la solución del PMP.

Keywords: profile minimization problem, metaheurísticas, scatter search, path relinking

1. Introducción

Se define $G = (V, E)$ como un grafo conexo no dirigido, siendo V el conjunto de vértices ($|V| = n$) y E el conjunto de aristas ($|E| = m$) de G. La asignación de una etiqueta única ($\varphi(v) = \{1, 2, ..., n\}$) a cada uno de los vértices de G conforma una ordenación lineal φ de los vértices de G. Una solución al PMP de un grafo G puede representarse mediante una ordenación lineal de los vértices de G. Siendo $Adj(v)$ el conjunto de vértices adyacentes a v, se define $f(v, \varphi, G)$ como el mínimo entre la posición del vértice adyacente a v con la menor etiqueta en φ y la propia etiqueta de v. En términos matemáticos,

$$f(v, \varphi, G) = \min\left(\varphi(v), \underset{u \in Adj(v)}{\arg\min} \varphi(u)\right).$$

El $VertexCut$ de un vértice v del grafo G en una ordenación φ queda definido como $VC(v, \varphi, G) = \varphi(v) - f(v, \varphi, G)$. El *profile* de una ordenación lineal φ de un grafo G ($P(\varphi, G)$) se define como la suma de los $VertexCut$ de cada vértice

en la ordenación. Es decir,

$$P(\varphi, G) = \sum_{v \in V} VC(v, \varphi, G)$$

El objetivo del PMP es encontrar una ordenación lineal $\varphi^\star$ tal que $P(\varphi^\star, G) = \min_{\varphi \in \phi} P(\varphi, G)$. La Figura 1 muestra un ejemplo de un grafo conexo no dirigido (izquierda) así como una posible ordenación lineal del mismo, $\varphi = \{C, E, B, A, D\}$ (derecha). Además, se muestra la evaluación de los $VertexCut$ de cada vértice. El *profile* de la ordenación mostrada se calcula como la suma de todos los $VertexCut$, es decir: $P(\varphi) = 0 + 1 + 0 + 3 + 3 = 7$.

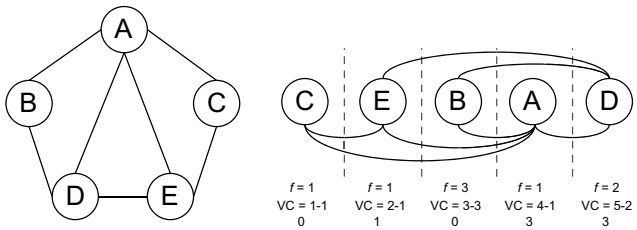

Figura 1. Ejemplo de un grafo (izquierda) y una posible ordenación lineal del mismo, con el cálculo de los $VertexCut$ de cada vértice

El PMP fue propuesto originalmente como una aproximación para reducir el espacio necesario para almacenar matrices dispersas [26], donde se ha probado que es equivalente al problema del *SumCut* [1]. El PMP también tiene aplicaciones en la mejora del rendimiento de operaciones sobre sistemas de ecuaciones no lineales [25], el Proyecto Genoma Humano [14], arqueología [15], recuperación de la información [2] y reconocimiento de patrones [14].

El PMP es un problema equivalente al *Interval Graph Completion Problem* [19], el cual es NP-completo, como se demostró en [10]. Sin embargo, la solución óptima de algunos tipos de grafos se puede obtener en tiempo polinómico. Por ejemplo, en [19] se proponen varios algoritmos que encuentran la solución del PMP en tiempo polinómico para grafos de tipo camino, rueda, bipartido completo y árboles D4 (árboles con diámetro 4). De la misma manera, en [12] se presenta un algoritmo que encuentra la solución óptima del PMP para grafos triangulares.

El primer heurístico encontrado en el estado del arte referente al PMP consiste en un procedimiento constructivo [3] conocido como *Reverse Cuthill-McKee Algorithm* (RCM), basado en construir una estructura de nivel de los vértices. En [22] se mejora el algoritmo del RCM cambiando la selección del nodo raíz dentro de una estructura de nivel más general. Este método se conoce como GPS. En [11] se presenta el algoritmo GK que utiliza un pseudo-diámetro para producir una nueva estructura de nivel, mejorando los resultados de GPS en calidad,

pero necesitando un mayor tiempo de cómputo. Finalmente, en [17] se introduce una nueva implementación que mejora la calidad tanto de GK como de GPS. El mejor método heurístico para el PMP encontrado en la literatura aparece en [18] y utiliza la metodología del Recocido Simulado (*Simulated Annealing - SA*) combinada con los métodos constructivos descritos anteriormente.

2. Scatter Search

Scatter Search (SS) [16] es una metaheurística que incluye cinco métodos para construir, mantener y transformar una población de soluciones. Tres de estos métodos (construcción diversa, mejora y combinación) dependen directamente del problema, por lo que sus estrategias se basan en la información presente en el mismo. Por otra parte, la actualización del conjunto de referencia y la generación de subconjuntos tienen implementaciones estándar que son independientes del problema. El estado del arte de SS se mantiene en constante crecimiento con un gran número de nuevos trabajos (ver [8], [20], [6] y [7]). El Algoritmo 1 muestra el marco general de la metodología *Scatter Search*.

Algoritmo 1 Metodología Scatter Search

```
1: Construcción diversa
2: Mejora
3: Actualización del RefSet
4: while criterio de finalización no alcanzado do
5:     Generación de los subconjuntos
6:     Combinación
7:     Mejora
8:     Actualización del RefSet
9: end while
```

El algoritmo comienza con la aplicación del método de construcción diversa (paso 1) y el método de mejora (paso 2), generando una población de P soluciones que se utilizan para construir el conjunto de referencia inicial ($RefSet$) de tamaño b. El $RefSet$ debe contener tanto soluciones de alta calidad como soluciones diversas, por lo que en su actualización (paso 3) se seleccionan las $b/2$ mejores soluciones de P y, tras ello, las $b/2$ soluciones más diversas con respecto al $RefSet$ del conjunto restante ($P \setminus RefSet$). Es importante tener en cuenta que la selección de las soluciones más diversas de un conjunto es un problema NP-duro (ver [5]) y, por lo tanto, este paso se lleva a cabo de manera heurística. En concreto, una vez que se han seleccionado las $b/2$ mejores soluciones al $RefSet$, las siguientes soluciones se añaden una a una. La primera solución seleccionada es la solución más diversa respecto a las soluciones actuales del $RefSet$. La medida de la diversidad se representa mediante una función que maximiza la mínima distancia entre la solución que se está evaluando y el conjunto de soluciones donde va a ser agregada (en este caso, el $RefSet$). La distancia entre

una solución φ y el $RefSet$ se define como la suma mínima de las diferencias absolutas de las etiquetas asignadas a cada vértice en la solución φ y cada una de las soluciones φ' del $RefSet$. En términos matemáticos,

$$d(\varphi, RefSet) = \min_{\varphi' \in RefSet} \left(\sum_{v \in V} |\varphi(v) - \varphi'(v)| \right).$$

Como las etiquetas de una solución representan posiciones, el cálculo de la distancia es equivalente a la distancia posicional [4]. Una vez que se ha seleccionado una solución φ del conjunto $P \setminus RefSet$, ésta se añade al $RefSet$ y el proceso se repite hasta completar el $RefSet$ con b soluciones, eligiendo en cada paso la solución $\varphi^\star$ con la máxima distancia a las soluciones actuales del $RefSet$. Es decir,

$$\varphi^\star = \arg\max_{\varphi \in P \setminus RefSet} d(\varphi, RefSet)$$

En este trabajo, durante la etapa de generación de subconjuntos (paso 5) se generan todos los pares de soluciones que no han sido combinados anteriormente. La actualización del $RefSet$ del paso 8 es diferente a la del paso 3. Para mantener la diversidad entre las soluciones, una solución φ se admite si mejora la mejor solución del $RefSet$, o si mejora la peor solución y además la distancia de φ con la solución más cercana es mayor que un umbral predefinido $dthresh$. Si finalmente la solución φ puede entrar al $RefSet$, lo hace sustituyendo la solución más cercana con un valor de función objetivo menor o igual que φ. El proceso finaliza cuando no hay nuevas soluciones que puedan formar parte del $RefSet$. Es decir, si durante una iteración determinada del método, el $RefSet$ no cambia tras la actualización del paso 8, el algoritmo termina.

3. Construcción de soluciones diversas

En este trabajo se presentan cuatro métodos de construcción de soluciones diversas para construir una población P de soluciones, basados en la metodología GRASP [9,23,24]. Se define U como el conjunto de vértices no etiquetados y $L = V \setminus U$ como el conjunto de vértices ya etiquetados. Inicialmente, todos los vértices están en U. El primer método, C1, comienza seleccionando el vértice $v \in U$ de menor grado, seleccionando al azar en caso de empate. El vértice elegido v recibe la primera etiqueta, por lo que $\varphi(v) = 1$, y se actualizan los conjuntos U y L ($U = U \setminus \{v\}$ y $L = L \bigcup \{v\}$). Tras ello, se construye una lista de candidatos CL formada por los vértices adyacentes a v que no han sido etiquetados. En términos matemáticos, $CL = \{u : (v, u) \in E | v \in U\}$.

Para cada vértice v se evalúa la siguiente función voraz: $g_1(v) = |N_L(v)| - |N_U(v)|$, donde $N_L(v) = \{u \in L : (v, u) \in E\}$ y $N_U(v) = \{u \in U : (v, u) \in E\}$. La función voraz $g_1(v)$ mide la necesidad de dar al vértice v la siguiente etiqueta. La función considera que es más urgente etiquetar un vértice que tiene a la mayoría de sus adyacentes ya etiquetado. En la metodología GRASP las

construcciones son semi-voraces, y la implementación contiene una lista de candidatos restringida (RCL). La RCL incluye a los mejores candidatos de la CL que van a tener la misma probabilidad de ser elegidos:

$$RCL = \left\{v \in CL : g_1(v) > g_1^{min} + \alpha_1 \left(g_1^{max} - g_1^{min}\right)\right\}$$

donde g_1^{min} y g_1^{max} son el mínimo y el máximo valor de $g_1(v) \forall v \in CL$, respectivamente. Se elige un vértice v de la RCL de forma aleatoria y, tras asignarle la siguiente etiqueta disponible, los conjuntos U y L se actualizan. CL también se actualiza añadiendo los vértices sin etiquetar adyacentes a v. La construcción termina cuando todos los vértices han sido etiquetados.

El segundo método constructivo (C2) es semejante a C1, pero modificando la implementación de la selección semi-voraz. En este caso, la RCL se construye mediante una muestra aleatoria de vértices de la CL, y, tras ello, se selecciona el vértice v con mayor valor de $g_1(v)$. El tamaño de la RCL se establece con un parámetro α_2, que representa un porcentaje del tamaño de la lista de candidatos. Es decir, la RCL tendrá un tamaño de $\alpha_2 \cdot |CL|$ vértices. Como en C1, una vez que un vértice ha sido elegido, se le asigna la siguiente etiqueta y se actualizan los conjuntos U y L.

La función voraz $g_1(v)$ no tiene en cuenta los valores de las etiquetas asignadas a los vértices adyacentes a v. Por ejemplo, si i es la siguiente etiqueta disponible, $g_1(v)$ no utiliza información acerca de las etiquetas que los elementos de $N_L(v)$ han recibido (entre 1 y $i-1$). Claramente, es más importante etiquetar el vértice v si sus adyacentes han recibido etiquetas que son mucho menores que i. La nueva función voraz $g_2(v)$ sí que aprovecha esta información:

$$g_2(v) = (|N_L(v)| - |N_U(v)|) \sum_{u \in N_L(v)} |\varphi(v) - \varphi(u)|$$

Esta nueva función voraz se utiliza para crear dos nuevos métodos constructivos: C3 y C4. C3 es similar a C1, y C4 es similar a C2, pero utilizando g_2 como función voraz.

4. Búsqueda local

En este artículo se proponen dos métodos de búsqueda en una vecindad: uno definido por intercambios y otro por inserciones. El $intercambio(i, j)$ de dos nodos v, u tal que $\varphi(v) = i$ y $\varphi(u) = j$ se define como la modificación de las etiquetas de dichos nodos, de manera que el vértice v recibe la etiqueta de u y viceversa. Por otra parte, una $insercion(i, j)$ es el movimiento del vértice v situado en la posición i ($\varphi(v) = i$) a la posición j. Como ambos movimientos producen una vecindad de tamaño $O(n^2)$, es necesario evaluar los movimientos de manera eficiente para identificar el mejor de ellos.

En la evaluación del movimiento $insercion(i, j)$ para $j > i$, es sencillo ver que solo hay que reevaluar el $VertexCut$ de los vértices v tal que $i \leq \varphi(v) \leq n$. Por tanto, si se almacenan los $VertexCut$ de todas las posiciones, el cálculo de

un movimiento se basa en actualizar aquellas posiciones relevantes y el valor de la suma de los $VertexCut$. Además, la actualización de los $VertexCut$ es acumulativa. Es decir, si primero evaluamos el movimiento $insercion(i, j)$, debemos reevaluar el $VertexCut$ de aquellos vértices situados entre las etiquetas i y n. Si tras ese movimiento evaluamos $insercion(i, j + 1)$ sin tener en cuenta el anterior movimiento, necesitaríamos evaluar otra vez los $VertexCut$ de los mismos vértices. Sin embargo, observando los cambios en los $VertexCut$ en estos movimientos, se puede determinar que los únicos cambios tras el segundo movimiento se producen en los vértices v con etiquetas $\varphi(v) \geq j + 1$.

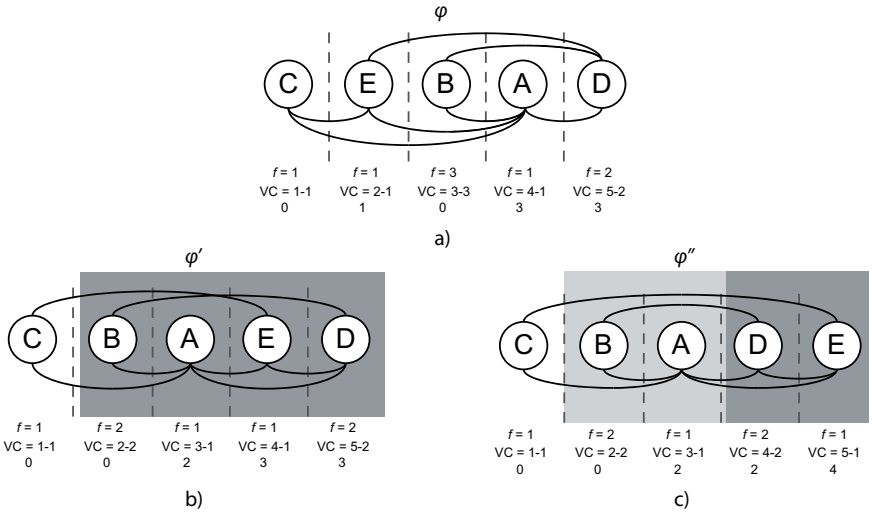

Figura 2. Ejecución de dos inserciones consecutivas: $insercion(2, 4)$ (b) y $insercion(2, 5)$ (c) a partir de una ordenación lineal (a). Se resalta con un sombreado oscuro las posiciones de las que hay que recalcular el $VertexCut$ y con sombreado claro las posiciones que no es necesario calcular gracias a la implementación propuesta.

La Figura 4 muestra un ejemplo de inserción de un vértice en dos posiciones consecutivas. En concreto, la Figura 4.b muestra la inserción del vértice E en la posición 4, partiendo de la ordenación $\varphi = \{C, E, B, A, D\}$ de la Figura 4.a. Por otro lado, la Figura 4.c muestra la inserción del vértice B en la posición 5, partiendo de la misma ordenación (Figura 4.a). En ambas inserciones, es necesario volver a calcular el $VertexCut$ de las posiciones $p \geq \varphi(E)$. Sin embargo, la implementación eficiente propuesta en este trabajo permite aprovechar los cálculos realizados para la evaluación de los $VertexCut$ en la Figura 4.b, de forma que en la segunda inserción (Figura 4.c) tan solo es necesario evaluar las posiciones $p \geq \varphi'(E)$. Es decir, se evalúan dos posiciones en lugar de cuatro. Por lo tanto, para aprovechar esta evaluación eficiente de la inserción, el movimiento

$insercion(i, j)$ se implementa como una secuencia de intercambios de vértices en posiciones consecutivas.

5. Métodos de combinación

Se propone un método de combinación (CM1) y una variante (CM2). El método propuesto sigue la estrategia de *Path Relinking* (PR) [13]. PR trata de construir caminos entre dos soluciones donde la función objetivo es solo uno de los elementos utilizados para determinar la dirección. Aprovecha el principio de que la vecindad de una solución buena posiblemente contenga otras soluciones de alta calidad si se explora de una manera diferente.

Se define φ_s como la solución inicial y φ_t como la solución guía o destino. CM1 comienza identificando el conjunto D de vértices que tienen etiquetas diferentes en φ_s y φ_t, es decir, $D = \{v : \varphi_s(v) \neq \varphi_t(v)\}$. Se genera un conjunto de soluciones intercambiando las etiquetas del vértice v y el vértice que tiene la etiqueta $\varphi_t(v)$ en la solución inicial, para todos los vértices $v \in D$. Este movimiento hace que φ_s se acerque a φ_t. Este paso genera $|D|$ soluciones de las cuáles seleccionamos la mejor teniendo en cuenta el valor de la función objetivo. La solución elegida se convierte en la nueva solución inicial y continúa el proceso hasta que $D = \emptyset$, es decir, hasta que $\varphi_s = \varphi_t$. Este procedimiento, llamado *Greedy Path Relinking* en [24] devuelve la mejor solución encontrada en el camino.

CM2 es una variante de CM1, donde se modifica el criterio de selección de soluciones en el camino generado. En este caso, en lugar de elegir la mejor solución del conjunto de soluciones generado en un paso del camino, se elige una solución aleatoria para favorecer la diversificación. Al igual que CM1, CM2 devuelve la mejor solución encontrada en el camino.

6. Resultados experimentales

Esta sección describe los experimentos computacionales que se han llevado a cabo para probar los algoritmos propuestos. Todos los algoritmos se han desarrollado en Java SE 6 y ejecutados en un Intel Core i7 2600 a 3.4 GHz con 2 GB de memoria RAM. Los resultados de los algoritmos RCM [3] y SA [18] del estado del arte se han obtenido ejecutando el código C original proporcionado por los autores. El conjunto de instancias utilizado consiste en 262 instancias divididas en tres subconjuntos, y están disponibles en `www.optsicom.es/pmp`. Los tres subconjuntos de instancias se describen a continuación:

- **HB:** Se compone de 73 instancias de la colección *Harwell-Boeing Sparse Matrix Collection* [21]. Contiene instancias obtenidas de problemas de sistemas lineales, mínimos cuadrados y cálculo de autovalores en una amplia variedad de disciplinas científicas. El número de vértices de las instancias varía entre 24 y 960 y el número de aristas entre 34 y 3721.
- **K-Graphs:** Este conjunto contiene 98 grafos bipartidos con un número de vértices comprendido entre 4 y 142 y un número de aristas entre 3 y 5016.

El conjunto de vértices V de un grafo bipartido se puede dividir en dos subconjuntos V_1 y V_2 de forma que no existen aristas entre dos nodos del mismo subconjunto de vértices. El valor óptimo de estos grafos se conoce por construcción y viene determinado por $n_1 n_2 + \frac{1}{2} n_1(n_1 - 1)$ para $n_1 \leq n_2$, donde $n_1 = |V_1|$ y $n_2 = |V_2|$

- **D4-Trees:** Consta de 91 instancias basadas en árboles con diámetro 4, un rango de vértices entre 10 y 100 nodos y un rango de aristas comprendido entre 9 y 99. Las soluciones óptimas se conocen por construcción, y su valor es $|E| + \sum_{i=3}^{k} (\gamma(v_i) - 1)$.

Los experimentos llevados a cabo se pueden dividir en dos partes. En la primera parte se lleva a cabo un análisis factorial de los algoritmos propuestos sobre un subconjunto de las instancias utilizadas para seleccionar la mejor combinación de constructivo, búsqueda local y método de combinación. En concreto, el subconjunto de instancias de la experimentación preliminar se compone de las 30 instancias con menos de 250 vértices obtenidas del conjunto HB.

La mejor combinación resultante del experimento factorial es la del constructivo C1 junto con la búsqueda basada en inserciones y el método de combinación CM1, obteniendo una desviación del 0.80 %, 20 de las mejores soluciones encontradas y un tiempo de ejecución de 6.59 segundos, seguido de C3+inserciones+CM1, con un 0.75 % de desviación, 17 mejores soluciones encontradas y 6.78 segundos de tiempo de ejecución. El algoritmo utilizado para el experimento final es por tanto C1+inserciones+CM1.

En el experimento final se comparan los mejores algoritmos encontrados en el estado del arte con el algoritmo propuesto en este trabajo (SS) utilizando el conjunto completo de instancias. En concreto, se utilizan los algoritmos Reverse Cuthill-McKee (RCM) y Simulated Annealing (SA). El experimento reporta información acerca de el valor promedio de la función objetivo (F.O.), la desviación estándar respecto a la mejor solución conocida (Dev), el número de mejores soluciones encontradas (#Mejores) y el tiempo de ejecución en segundos. La Tabla 1 muestra los resultados obtenidos en el experimento.

Algoritmo	**F.O.**	**Dev(%)**	**#Mejores**	**Tiempo**
RCM	2886.31	69.00	107	69.78
SA	2837.06	18.62	135	111.27
SS	2346.52	0.10	234	60.22

Tabla 1. Comparativa del algoritmo propuesto con el estado del arte

Como se puede observar, el algoritmo propuesto obtiene mejores resultados en todas las estadísticas. Además, la baja desviación del método propuesto indica que, en el reducido número de instancias donde nuestro algoritmo no encuentra la mejor solución conocida, proporciona soluciones de alta calidad. Analizando los conjuntos de instancias por separado (D4-Trees, K-Graphs y HB) se observa que el conjunto de los K-Graphs no parece ser un reto para ninguno de los algoritmos,

ya que en todos los casos obtienen el valor óptimo (salvo el algoritmo SA en una instancia) en un intervalo de tiempo reducido (menos de un segundo en el caso de SS y RCM, y 6 segundos en el caso de SA). Sin embargo, en el conjunto D4-Tree el algoritmo SS ya muestra su potencial, obteniendo 89 de los 91 valores óptimos, seguido del SA (31), y RCM (2). En el conjunto más difícil de los tres, HB, donde no existe óptimo conocido, el algoritmo SS sigue dominando al resto de algoritmos, obteniendo 67 de las mejores soluciones y una desviación del 0.38 %, seguido del SA (7 mejores soluciones, 28.93 %) y el RCM (7 mejores soluciones, 31.31 %). El tiempo de ejecución requerido por SS y RCM es similar, siendo SS ligeramente inferior, mientras SA necesita aproximadamente el doble de tiempo.

Finalmente, para validar los resultados obtenidos, se aplican test no paramétricos a los algoritmos utilizados para comprobar si existen diferencias significativas entre ellos. En primer lugar se aplica el test de Friedman, obteniendo un p-valor de 0.000, lo que claramente indica que existen diferencias significativas entre los métodos propuestos. Los rankings producidos por el test son 1.16 (SS), 2.37 (SA) y 2.81(RCM). A continuación se aplica el test de Wilcoxon y de signos para comparar los mejores algoritmos del experimento, SS y SA. En ambos casos se obtiene un p-valor de 0.000, lo que demuestra que sí existen diferencias significativas entre ambos algoritmos.

7. Conclusiones

En este trabajo se presentan diferentes algoritmos basados en la metodología *Scatter Search* y *Path Relinking* para resolver el problema del *Profile*. En concreto, se proponen cuatro métodos constructivos, dos búsquedas locales y dos métodos de combinación de soluciones. Se presenta una experimentación factorial para elegir la mejor combinación de métodos para el algoritmo (constructivo, búsqueda local y método de combinación). Los resultados obtenidos muestran que el objetivo del trabajo se ha cumplido, mejorando los resultados encontrados en el estado del arte. Los resultados están respaldados por diferentes test estadísticos no paramétricos que confirman la calidad del algoritmo propuesto.

Agradecimientos

Este trabajo ha sido parcialmente financiado por la Comunidad de Madrid a través del proyecto con referencia S2009/TIC-1542 y por el Ministerio de Educación y Ciencia a través del proyecto con referencia (TIN2009-07516). Nos gustaría agradecer al Prof. Robert R. Lewis su colaboración compartiendo el código fuente de los métodos RCM y SA.

Referencias

1. Agrawal A, Klein P, Ravi R. Ordering problems approximated: single-processor scheduling and interval graph completion. Automata, Languages and Programming, 510:751–762 (1991).

2. Botafogo RA. Cluster analysis for hypertext systems. Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, 116–125 (1993).
3. Cuthill E, McKee J. Reducing the bandwitdh of sparse symmetric matrices. ACM'69 Proceedings of the 1969 24th national conference, 157–172 (1969).
4. Das A, Roberts M. Metric distances of permutations. `www.cra.org/Activities/craw\_archive/dmp/awards/2004/Das/paper.ps`, (2004).
5. Duarte A, Martí R. Tabu Search and GRASP for the Maximum Diversity Problem. European Journal of Operational Research, 178:71–84 (2007).
6. Duarte A, Martí R, Pardo EG, Pantrigo JJ. Scatter Search for the cut width minimization problem. Annals of Operations Research, 199(1):285–304 (2012).
7. Duarte A, Glover F, Martí R, Gortázar F. Hybrid scatter stabu search for unconstrained global optimization. Annals of Operations Research, 183:95–123 (2011).
8. Gallego M, Duarte A, Laguna M, Martí R. Hybrid heuritics for the maximum diversity problem. Computational Optimization and Applications, 44(3):411-426 (2009).
9. Feo TA, Resende MGC. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters, 8(2):67–71 (1989).
10. Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H.Freeman and Co., (1979).
11. Gibbs NE. A hybrid profile reduction algorithm. ACM Trans. Math. Softw., 2(4):378–387 (1976).
12. Guan G, Williams KL. Profile minimization of triangulated triangles. Discrete Mathematics, 260:69-76 (2003).
13. Glover F, Laguna M. Tabu Search. Kluwer Academic Publishers: Boston, (1997).
14. Karp R. Mapping the genome: some combinatorial problems arising in molecular biology. STOC'93 Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, 278–285 (1993).
15. Kendall R. Incidence matrices, interval graphs and seriation in archaeology. Pacific Journal of Mathematics, 28:565–570 (1992).
16. Laguna M, Martí R. Scatter Search: methodology and implementations in C. Clubber Academic Publisher (2003).
17. Lewis J. The Gibbs-Poole-Stockmeyer and Gibbs-King algorithms for reordering sparse matrices. ACM Transactions on Mathematical Software (TOMS), 8:190–194 (1982).
18. Lewis R. Simulated annealing for profile and fill reduction. International Journal for Numerical Methods in Engineering, 37(6):905-925 (1994).
19. Lin Y, Yuan J. Profile minimization problem for matrices and graphs. Acta Math. Appl. Sinica, English Series, Yingyong Shuxue-Xuebas, 10(1):107–112 (1994).
20. Martí R, Duarte A, Laguna M. Advanced scatter search for the max-cut problem. INFORMS Journal on Computing, 21(1):26-38 (2009).
21. `http://math.nist.gov/MatrixMarket/`
22. Poole W, Stockmeyer P, Gibbs N. An algorithm for reducing the bandwitdh and profile of a sparse matrix. SIAM Journal on Numerical Analysis, 13:236–250 (1976).
23. Resende MGC, Smith SH, Feo TA. A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42:860–878 (1994).
24. Resende MGC, Martí R, Gallego M, Duarte A. GRASP and path relinking for the max-min diversity problem. Computers and Operations Research, 37:498-508 (2010).
25. Saad Y. Iterative methods for sparse linear systems. SIAM, ISBN-13:978-0-898715-34-7 (2003).
26. Tewarson, R. Sparse matrices. Academic Press: New York (1973).