Evaluación del rendimiento de técnicas heurísticas para el S-labeling

Marcos Robles
Universidad Rey Juan Carlos
marcos.robles@urjc.es

Sergio Cavero
Universidad Rey Juan Carlos
sergio.cavero@urjc.es

Eduardo G. Pardo
Universidad Rey Juan Carlos
eduardo.pardo@urjc.es

***Resumen*—El problema de S-labeling es un problema de etiquetado de grafos que asigna etiquetas numéricas a los vértices de un grafo, con el objetivo de minimizar una función objetivo. Concretamente, para cada par de vértices adyacentes se anota la etiqueta más pequeña del par. Una vez que se han revisado todos los pares, la función objetivo se calcula como la suma de todas las etiquetas anotadas. En este trabajo preliminar se realiza una propuesta que utiliza la metaheurística *Greedy Randomized Adaptive Search Procedure*. Concretamente, se analizan un total de cuatro métodos constructivos aleatorizados, dos de ellos tomados de la literatura y dos de esta propuesta. Como método de mejora se proponen dos búsquedas locales y el uso de *Variable Neighborhood Descent* para combinarlas. El mejor algoritmo propuesto se compara con el mejor algoritmo del estado del arte (*Population-based Iterated Greedy*), utilizando un conjunto de instancias de referencia. Los resultados muestran que uno de los métodos constructivos presentados es capaz de obtener soluciones competitivas con una desviación baja. También se comprueba que la fase de mejora es capaz de refinar las soluciones propuestas por el constructivo, obteniendo soluciones similares a las que se consiguen utilizando el algoritmo del estado del arte. Finalmente, se discuten los puntos fuertes y débiles de esta propuesta y se sugieren algunas líneas de investigación futuras.**

***Palabras clave*—Etiquetado de grafos, *Variable Neighborhood Search*, S-labeling**

I. Introducción

El problema de S-labeling es un *Graph Layout Problem* (GLP) [4], [6], [19] cuyo objetivo consiste en asignar una etiqueta numérica a los vértices de un grafo. El S-labeling originalmente aparece relacionado con el empaquetado de matrices-(0,1) [24].

Formalmente, dado un grafo no dirigido $G = (V, E)$ donde V y E son los conjuntos de vértices y aristas respectivamente, se define un etiquetado ϕ asignando una única etiqueta (es decir, un número) $l \in \mathbb{Z}$, tal que $1 \leq l \leq |V|$, a cada vértice. Generalmente, esta relación se realiza mediante la función biyectiva ϕ que, para un vértice v dado, devuelve su etiqueta correspondiente. Dado el etiquetado ϕ de un grafo G, para evaluar la función objetivo de la solución, denotada como número de S-labeling (SL), se calcula, para cada arista e, la etiqueta mínima asociada al par de vértices de sus extremos, y se suman todas las etiquetas mínimas calculadas anteriormente. Formalmente, la función objetivo se define como $SL(\phi, G) = \sum_{(u,v)\in E} \min \phi(u), \phi(v)$.

Este trabajo ha sido parcialmente financiado por los proyectos PID2021-125709OA-C22 y PID2021-126605NB-I00, financiados por MCIN/AEI/10.13039/501100011033 y "ERDF A way of making Europe"; el proyecto CIAICO/2021/224, financiado por la Generalitat Valenciana; el proyecto M2988, financiado por la convocatoria "Proyectos Impulso de la Universidad Rey Juan Carlos 2022"; la "Cátedra de Innovación y Digitalización Empresarial entre Universidad Rey Juan Carlos y Second Episode" (Ref. MCA06); y la "Red Española de optimización heurística 4.0 digitalización" (Ref. RED2022-134480-T).

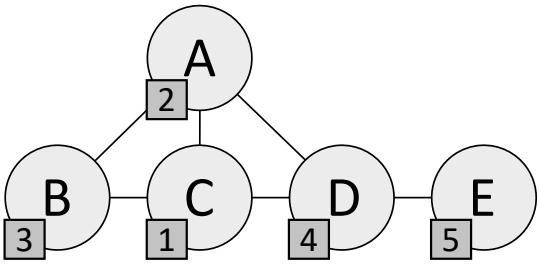

Figura 1. Ejemplo de un etiquetado de un grafo.

Por ejemplo, considérese el grafo $G_1(V_1, E_1)$ representado en la Figura 1 con vértices $V_1 = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}\}$ y aristas $E_1 = \{(\mathrm{A}, \mathrm{B}), (\mathrm{A}, \mathrm{C}), (\mathrm{A}, \mathrm{D}), (\mathrm{B}, \mathrm{C}), (\mathrm{C}, \mathrm{D}), (\mathrm{D}, \mathrm{E})\}$. Además, sea ϕ_1 un etiquetado donde $\phi_1(\mathrm{A}) = 2$, $\phi_1(\mathrm{B}) = 3$, $\phi_1(\mathrm{C}) = 1$, $\phi_1(\mathrm{D}) = 4$, y $\phi_1(\mathrm{E}) = 5$. Para este etiquetado, el número de S-labeling se calcula de la siguiente manera:

$$\begin{aligned} SL(\phi_1, G_1) &= \min(\phi_1(\mathrm{A}), \phi_1(\mathrm{B})) + \min(\phi_1(\mathrm{A}), \phi_1(\mathrm{C})) + \\ &\quad \min(\phi_1(\mathrm{A}), \phi_1(\mathrm{D})) + \min(\phi_1(\mathrm{B}), \phi_1(\mathrm{C})) + \\ &\quad \min(\phi_1(\mathrm{C}), \phi_1(\mathrm{D})) + \min(\phi_1(\mathrm{D}), \phi_1(\mathrm{E})) \\ &= \min(2, 3) + \min(2, 1) + \min(2, 4) + \\ &\quad \min(3, 1) + \min(1, 4) + \min(4, 5) \\ &= 2 + 1 + 2 + 1 + 1 + 4 = 11. \end{aligned}$$

El objetivo de este problema es encontrar el etiquetado ϕ^* que minimiza el número de S-labeling para el grafo dado.

Con el fin de explorar propuestas innovadoras para el problema de S-labeling, se proponen nuevos métodos constructivos, optando por la adopción de metaheurísticas basadas en trayectorias. En concreto, en este trabajo se utiliza la metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP) [20], [21] y se plantean dos variantes de *Variable Neighborhood Descent* (VND) [1], [10]. Para evaluar esta propuesta, se lleva a cabo un análisis comparativo con respecto

a las mejores soluciones encontradas por el algoritmo del estado del arte [13].

El resto del documento se estructura como sigue. En primer lugar, se realiza una revisión de la literatura en la Sección II. En la Sección III, se presenta la propuesta algorítmica de este trabajo. A continuación, en la Sección IV se recoge la experimentación preliminar con la que se ajusta cada uno de los componentes del algoritmo. En la Sección V, se compara la propuesta algorítmica con el mejor método del estado del arte. Por último, se presentan las conclusiones en la Sección VI.

II. Trabajos relacionados

El primer trabajo sobre el problema de S-labeling se remonta al año 2018 [7], donde se relaciona con el problema del empaquetado de matrices-(0,1) [24]. En [7] los autores estudian el S-labeling desde un punto de vista teórico, presentando cotas a partir de propiedades, tales como el grado máximo del grafo. Por otro lado, también se relacionan estas propiedades con el problema de *Vertex Cover*, en el que se quiere obtener el subconjunto de vértices de un grafo que abarquen todas las aristas. En [7] también se realiza la propuesta de un algoritmo constructivo voraz, cuyo criterio voraz está basado en el grado de cada vértice, asignando las etiquetas de menor valor a los vértices de mayor grado. Finalmente, también presentan que, para el problema de S-labeling, los grafos de tipo *caterpillar* [11] y *split graph* [17] son resolubles en tiempo polinómico.

En [22] se hizo una propuesta de un algoritmo exacto para grafos sin requerir una estructura definida. En este trabajo se utiliza programación entera mixta utilizando desigualdades, heurísticas y técnicas de ramificación. Con estos componentes se consiguen soluciones óptimas para varios grupos de instancias. Además, los autores también formulan una aproximación con programación por restricciones.

En [13] se hace una propuesta heurística basada en *Population-based Iterated Greedy* (PIG), una extensión de la metaheurística *Iterated Greedy* (IG) [23], que iterativamente destruye y reconstruye la solución. En este caso, como heurística para la destrucción se utiliza un criterio aleatorio, mientras que, para la reconstrucción, se asignan los vértices sin etiqueta de forma voraz basándose en la función objetivo. PIG aplica las operaciones de IG a las soluciones de la población y selecciona las mejores soluciones para obtener la siguiente generación. Como búsqueda local se utiliza un movimiento de intercambio con la estrategia de primera mejora.

III. Propuesta algorítmica

Como esquema general para esta propuesta se utiliza GRASP [20], [21], una metaheurística multiarranque que divide cada iteración en dos etapas, una etapa de construcción y otra de mejora. En el Algoritmo 1 se presenta el pseudocódigo de esta metaheurística. Como se puede observar, en el paso 5, se utiliza un algoritmo constructivo voraz aleatorizado. En este trabajo, se proponen diversos métodos constructivos para este paso, que se presentan en la Sección III-A. A la solución generada se le aplica un proceso de mejora en el paso 6. En este trabajo se proponen, igualmente, un total de cuatro métodos de mejora (dos búsquedas locales y dos VND) que se introducirán en la Sección III-B.

Algoritmo 1 *Greedy Randomized Adaptive Search Procedure.*

```
1:  function GRASP
2:      Iteración ← 1
3:      φ' ← SoluciónInicialAleatoria()
4:      while Iteración < Iteración_max do
5:          φ_new ← ConstructivoVorazAleatorizado(α, g)
6:          φ'_new ← Mejora(φ_new)
7:          if F.O.(φ'_new) < F.O.(φ') then
8:              φ' ← φ'_new
9:          end if
10:         Iteración ← Iteración + 1
11:     end while
12:     return φ'
13: end function
```

III-A. Métodos constructivos

Las propuestas de métodos constructivos para el S-labeling son limitadas, reduciéndose a solo dos métodos constructivos en la literatura [7], [13]. Uno de los objetivos de este trabajo es ampliar la variedad de algoritmos constructivos disponibles para el S-labeling. En concreto, se presentan dos nuevos constructivos basados en criterios voraces que exploran propiedades del problema. Estos dos constructivos son enmarcados en la metodología GRASP presentada en la sección anterior. También, se plantea la adaptación de los dos constructivos existentes a la metodología GRASP.

Dado que GRASP emplea una aproximación multiarranque [14] es necesario que cada construcción genere una solución distinta para potenciar la exploración. Con este fin, GRASP aplica un grado de aleatoriedad a todos los constructivos, de forma que incluso los que tienen un criterio determinista generen soluciones diferentes. Por lo tanto, los métodos constructivos se componen de un criterio voraz g para seleccionar el mejor vértice a añadir a la solución, y un valor α para regular la aleatoriedad.

En el Algoritmo 2, se presenta el esquema general de los métodos constructivos propuestos. Partiendo de una lista de vértices candidatos, CL, que contiene todos los vértices que todavía no se han añadido a la solución. En cada iteración, se construye una lista de candidatos restringida (RCL). Para construirla se utiliza un criterio voraz g. Este criterio evalúa todos los vértices de CL, para calcular los valores máximos y mínimos ($g_{\min}$ y $g_{\max}$ respectivamente). A continuación, todos los vértices de $c \in CL$ que cumplan la condición $g(c) \leq g_{\min} + \alpha * (g_{\max} - g_{\min})$ pasarán a formar parte de RCL. Por último, se selecciona un vértice al azar de la RCL y se le asigna la etiqueta más pequeña disponible. Este proceso se repite hasta que todos los vértices tienen una etiqueta asignada y, equivalentemente, no quedan candidatos a añadir a la solución.

Cabe destacar que un valor de $\alpha = 0$, implica que la selección del siguiente vértice a añadir a la solución será completamente voraz. Por otro lado, si $\alpha = 1$, el algoritmo será equivalente a una selección completamente aleatoria. Finalmente, una vez generada la RCL, se selecciona un vértice al azar y se añade a la solución.

Algoritmo 2 Constructivo voraz aleatorizado.

```
1: function ConstructivoVorazAleatorizado(α, g)
2:     φ_new ← SoluciónVacía()
3:     CL ← V_G
4:     L ← {1, 2, ..., |V_G|}
5:     while CL ≠ ∅ do
6:         l_min ← mín(L)
7:         RCL ← GenerarRCL(CL, α, g)
8:         c ← SelecciónAleatoria(RCL)
9:         φ_new(c) ← l_min
10:        CL ← CL \ c
11:        L ← L \ l_min
12:    end while
13:    return φ_new
14: end function
```

En esta propuesta se proponen cuatro estrategias para generar la RCL, cada una con un criterio voraz g: i) generación basada en la función objetivo, ii) generación basada en los vértices adyacentes etiquetados, iii) generación basada en el grado de los vértices y iv) generación basada en una medida de centralidad. A continuación se describen cada una de estas estrategias.

El primer constructivo propuesto (**Basic**) utiliza como criterio voraz el valor de la función objetivo al añadir el vértice a la solución. En cada iteración del algoritmo constructivo, se evalúan todos los vértices que todavía no han sido etiquetados. Para ello, se les asigna la mínima etiqueta disponible y se evalúa la función objetivo. El valor obtenido será el valor del criterio voraz para ese vértice. Formalmente, se define como $g_1(\phi', u) = SL(\phi' \cup u, G)$ tal que $\phi'(u) = \text{mín}(L)$. Este método se utiliza en el algoritmo del estado del arte [13] como criterio para reconstruir la solución.

El segundo constructivo se propuso originalmente en [2], y está inspirado en un criterio voraz previamente publicado para otros GLP [3], [16]. Denotado en este trabajo como **Adjacency**, este constructivo consiste en evaluar cada vértice no etiquetado y darle un valor numérico basándose en sus vértices adyacentes que ya están etiquetados ($V_d = v \in \phi \; \forall \; u \in V$) y los que no lo están ($V_f = v \notin \phi \; \forall \; u \in V$). Este método ha destacado anteriormente por su flexibilidad, ya que cada uno de los componentes de la evaluación está afectado por un peso independiente $w_i \in [-1, 1]$, que se ajusta al problema concreto utilizando un *software* especializado. En este trabajo se ha utilizado *irace* [12] para realizar el ajuste de los pesos. Formalmente, el criterio voraz del constructivo Adjacency se define como $g_2(\phi, u) = w_1 * |V_d(u)| + w_2 * |V_f(u)|$. La principal desventaja de los constructivos Basic y Adjacency es que dependen del estado actual de la solución y, por lo tanto, tienen que evaluar la solución por cada vértice en cada paso de la construcción.

El tercer constructivo (**Degree**) utiliza el grado de un vértice para seleccionar el mejor candidato a añadir a la solución, eligiendo el que tenga el mayor grado en cada paso. El criterio voraz de este constructivo se define formalmente como $g_3(u) = -|u|$, y puede considerarse como una variación del constructivo Adjacency en el que se tienen los pesos $w_1 = -1$, $w_2 = -1$. Por ejemplo, en el etiquetado de la Figura 1, los vértices A, C y D tienen un grado de 3, el vértice de B un grado de 2, y el vértice E tiene un grado de 1. Una idea heurística sencilla lleva a priorizar la asignación de etiquetas pequeñas a los vértices A, C y D, ya que al tener un grado mayor tienen un mayor peso en la función objetivo. En el caso del vértice E, al tener un grado de 1 se le pueden asignar etiquetas grandes, ya que tiene un menor peso en la función objetivo.

El último constructivo, denotado como **Betweenness**, surge al analizar las soluciones generadas con el constructivo Degree. Se llegó a la conclusión de que la idea del constructivo Degree se podía extender a vértices con un grado mayor a 1 al utilizar una medida de centralidad, la intermediación (conocida en inglés como *Betweenness centrality*) [8]. La intermediación es una medida basada en los caminos más cortos dentro de un grafo. Se calcula contando el número de estos caminos que pasan a través de un vértice. Si se da que la medida de intermediación de un vértice es 0, entonces se puede determinar que independientemente de su grado, ese vértice es poco relevante para el etiquetado. Por lo tanto, en este constructivo se asignan los vértices de mayor a menor valor de intermediación. Este criterio voraz se define como $g_4(u) = -Betweenness(u)$. Por ejemplo, en el etiquetado de la Figura 1, los vértices B y E tienen una intermediación de 0, y los vértices A, C y D tienen una intermediación distinta de 0. En el caso de A, C y D el camino más corto entre $\{\text{B}\}$ y $\{\text{D}, \text{E}\}$ pasa o por A o por C, y el camino más corto entre $\{\text{A}, \text{B}, \text{C}\}$ y $\{\text{E}\}$ pasa por D. Por otro lado, B y E no aparecen en el camino más corto para llegar a ningún otro nodo. Consecuentemente, utilizando en esta métrica, se le asigna una mayor importancia a los vértices A, C y D, que recibirán las etiquetas más pequeñas. Por otro lado, los vértices B y E se consideran poco relevantes y, por lo tanto, se les asignarán las etiquetas mayores.

En contrapartida con los constructivos de Basic y Adjacency, los constructivos de Degree y Betweenness no dependen del estado actual de la solución y, por lo tanto, no requieren evaluaciones intermedias.

III-B. Métodos de mejora

Dada una solución factible, se aplica un proceso de mejora basado en una búsqueda local, que aplica cambios a la solución hasta que se alcanza un óptimo local. En este trabajo se proponen dos búsquedas locales basadas en movimientos de intercambios e inserción, respectivamente.

El movimiento de intercambio (**swap**) involucra dos vértices, u y v, con etiquetas $\phi(u) = l_1$ y $\phi(v) = l_2$. El movimiento de intercambio $swap(\phi, u, v)$ da como resultado una nueva

solución ϕ' con la asignación de etiquetas $\phi'(u) = l_2$ y $\phi'(v) = l_1$. Por ejemplo, para el etiquetado ϕ_1 representado en la Figura 1, si se hace el intercambio entre los vértices D y C, que tienen las etiquetas 4 y 1 respectivamente, las etiquetas de la solución pasarían a ser $\phi'_1(\text{A}) = 2$, $\phi'_1(\text{B}) = 3$, $\phi'_1(\text{C}) = 4$, $\phi'_1(\text{D}) = 1$, y $\phi'_1(\text{E}) = 5$.

El segundo movimiento estudiado es el movimiento de inserción (**insert**), que puede verse como una serie de movimientos de intercambio consecutivos a la etiqueta más cercana en la solución. Dado un vértice u y una etiqueta l tal que $\phi(u) \neq l$, el vértice u se intercambia con el vértice $\phi(u) + 1$ si $l > \phi(u)$ o $\phi(u) - 1$ si $\phi(u) > l$ hasta que la etiqueta l se asigna a u, obteniendo un nuevo etiquetado. Por ejemplo, volviendo al etiquetado ϕ_1 representado en la Figura 1, si se hace la inserción del vértice A en la posición 4, se tendría que hacer en primer lugar el intercambio entre A y B, ya que A tiene la etiqueta 2 y B la etiqueta 3. A continuación, habría que hacer otro intercambio con el vértice D, que tiene la etiqueta 4. Dado que la etiqueta de A ya es 4, se da el movimiento por completado. En este caso las etiquetas de la solución pasarían a ser $\phi'_1(\text{A}) = 4$, $\phi'_1(\text{B}) = 2$, $\phi'_1(\text{C}) = 1$, $\phi'_1(\text{D}) = 3$, y $\phi'_1(\text{E}) = 5$.

Para explorar estos dos vecindarios, se utiliza un procedimiento de Búsqueda Local (BL). Una BL puede seguir una estrategia de primera mejora (FI) o mejor mejora (BI). Experimentalmente, se ha determinado que en esta propuesta, la vecindad de intercambio se explora siguiendo la estrategia de FI. Es decir, se realiza el primer intercambio que mejore la solución actual. Esta BL se denota como BL_{swap}. Por otro lado, la BL de inserción, denotada como BL_{insert}, utiliza una combinación de FI y BI, puesto que se realiza la mejor inserción para el primer vértice que mejora la solución actual.

En este trabajo se propone, además, combinar estas búsquedas locales en un método de mejora siguiendo las ideas y principios de *Variable Neighborhood Search* (VNS) [9], [10], [18]. En concreto, se plantea el uso de la variante VND [5]. VND consiste en la repetición sistemática de varias BL, de manera que, la solución resultante, será óptimo local de todas las vecindades consideradas.

El pseudocódigo del VND propuesto, presentado en el Algoritmo 3, combina las dos BL planteadas anteriormente. En concreto, VND toma como parámetros una solución inicial y el número de vecindades máximo a explorar k_{max}, que coincide con el número de búsquedas locales. Antes de iniciar el procedimiento, se establece $k = 1$ (Paso 2), lo que indica que se examinará la vecindad más cercana, definida por la primera BL. Una vez aplicada la BL a la solución (Paso 6), si el nuevo óptimo local supera a la solución inicial (Paso 10), esta última se actualiza y se reinicia $k = 1$ (Pasos 11 y 12); de lo contrario, se incrementa k en 1 unidad (Paso 14). Este proceso se repite hasta que k alcanza el valor máximo $k_{\max}$ (Paso 4), lo que indica que se han explorado $k_{\max}$ vecindades sin encontrar una solución mejor. En este punto, se considera que el proceso ha concluido, obteniendo un óptimo local con respecto a todas las vecindades examinadas.

Algoritmo 3 *Variable Neighborhood Descent.*

```
1:  function VND(φ'_0)
2:      k ← 1
3:      k_max ← 2
4:      while k ≤ k_max do
5:          if k = 1 then
6:              φ'' ← BL_1(φ'_0)
7:          else
8:              φ'' ← BL_2(φ'_0)
9:          end if
10:         if F.O.(φ'') < F.O.(φ'_0) then
11:             φ'_0 ← φ''
12:             k ← 1
13:         else
14:             k ← k + 1
15:         end if
16:     end while
17:     return φ'_0
18: end function
```

IV. Experimentación preliminar

En esta sección, en primer lugar, se realiza una experimentación exhaustiva para ajustar los pesos del método constructivo de Adjacency en el contexto del S-labeling, y los valores de α de todos los métodos constructivos. En segundo lugar, se realizan los experimentos necesarios para determinar la mejor combinación de BL para el VND. Se prueba la propuesta sobre un subconjunto de 20 instancias, extraídas del estado del arte para el problema [13]. Para la selección de estas instancias se ha utilizado un *software* especializado que analiza las propiedades de todas las instancias y devuelve un subconjunto representativo de estas [15]. Las instancias se dividen en seis tipos: 3DMesh, Grids, Hamming, Harwell Boeing, Random y Toroidal mesh.

Los algoritmos se codificaron en Java 21 y todos los experimentos se llevaron a cabo en un sistema equipado con un procesador Intel Xeon Gold 6226R funcionando a 2,90 GHz y con 119 GB de RAM. Como métricas para evaluar el rendimiento de las propuestas se utilizan el promedio de la función objetivo (F.O.), el tiempo promedio de ejecución (T. CPU) medido en segundos, la desviación promedio (Desv.) respecto a las mejores soluciones de ese experimento y el porcentaje de mejores soluciones que obtuvo esa propuesta (%Mejores). En este trabajo se calcula la desviación respecto a una instancia como Desv. $= \frac{V_{actual} - V_{min}}{V_{min}}$, donde V_{actual} es el valor obtenido por el método, y V_{min} es el valor mínimo de la función objetivo entre los métodos comparados. Para determinar el %Mejores se calcula el porcentaje de instancias en las que el método tiene una desviación de 0.

En primer lugar, se realiza un ajuste de los parámetros de los constructivos propuestos. En concreto, es necesario determinar el valor de α, que regula el grado de aleatoriedad del constructivo, y los pesos w_1 y w_2 del constructivo Adjacency. Comenzando por el ajuste de los pesos w_1 y w_2, se utiliza

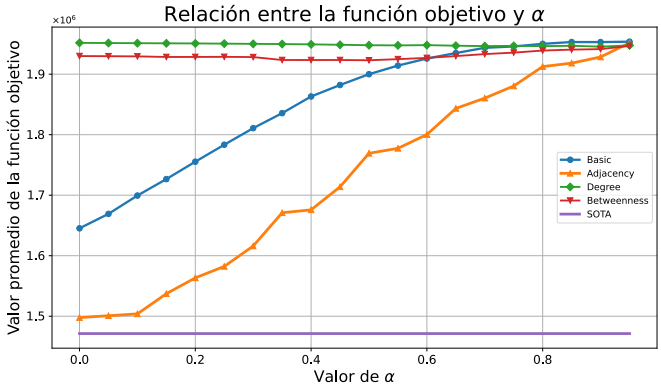

Figura 2. Evolución del valor de la función objetivo en los métodos constructivos en relación con el valor de α. Se incluye el algoritmo del estado del arte como punto de referencia (SOTA).

la herramienta *irace* [12], y se obtiene como mejores pesos los valores $w_1 = -0,1$ y $w_2 = -1,0$. Estos valores implican que la clave para construir buenas soluciones es asignar las etiquetas más bajas a aquellos vértices que tienen una mayor cantidad de vértices adyacentes fuera de la solución.

Para ajustar el valor de α se prueban 19 valores entre 0,00 y 0,95 inclusive con una diferencia de 0,05 entre cada par de valores, siendo 0 el criterio puramente voraz y 1 un criterio totalmente aleatorio. Esta evolución se muestra en la Figura 2. Se puede observar que los métodos constructivos Degree y Betweenness no se ven afectados por el valor de α, debido a que hay muchos vértices que al ser evaluados por el criterio voraz obtienen el mismo valor. Por otro lado, en los métodos constructivos Basic y Adjacency el valor de α sí que afecta al rendimiento, obteniendo mejores resultados a con valores bajos de α. En concreto, se obtienen resultados muy cercanos a los del algoritmo del estado del arte con el constructivo de Adjacency para los valores entre 0 y 0,1. Comparando el constructivo de Adjacency con los otros métodos constructivos, se puede observar que este encuentra los mejores resultados independientemente del valor de α.

Teniendo en cuenta estos resultados, para experimentaciones posteriores se mantiene el valor de α de 0,15. Este valor se ha elegido en lugar de uno menor debido a que se busca obtener soluciones distintas en cada iteración, que permitan diversificar la búsqueda durante la fase de mejora.

Para complementar este experimento, se realiza una comparativa de los cuatro métodos constructivos. En esta comparativa se realizan 100 construcciones sin límite de tiempo, y se devuelve la mejor solución generada. Como punto de referencia se incluye en la comparativa un constructivo aleatorio (**Random**) que genera 100 soluciones aleatorias y devuelve la mejor.

Los resultados de este experimento se muestran en la Tabla I. Se puede observar que los resultados obtenidos por el constructivo de Adjacency son mejores en todas las métricas evaluadas, permitiendo seleccionarlo como el mejor método constructivo de la comparativa. El principal inconveniente de

Tabla I
RESULTADOS DE LAS DIFERENTES PROPUESTAS DE MÉTODOS CONSTRUCTIVOS.

	F.O.	T. CPU (s)	Desv. (%)	% Mejores
Random	1952162,95	0,15	33,92	0
Basic	1726608,65	242,35	24,06	0
Adjacency	**1537199,80**	475,50	**0,00**	**100**
Degree	1928423,55	**1,80**	20,15	0
Betweenness	1950978,50	10,50	24,56	0

este método es su coste computacional, siendo el que requiere más tiempo de todos. Respecto al resto de métodos, se puede ver que los métodos Degree y Betweenness son mucho más ligeros computacionalmente que los métodos Adjacency y Basic, debido principalmente a la ausencia de evaluaciones intermedias, sin embargo, obtienen soluciones de peor calidad, haciendo que estos sean los peores métodos en cuanto al valor de la función objetivo, siendo similares a los resultados del constructivo Random.

El siguiente experimento tiene como objetivo comparar los métodos de BL propuestos. En concreto, en este trabajo se han presentado la búsqueda local de intercambios (BL_{swap}) y de inserciones (BL_{insert}). Además, estas BL son combinadas en un esquema de VND. Este experimento también tiene como objetivo determinar el orden en el que se ejecutarán estas búsquedas en el VND: VND_1: primero BL_{insert} y luego BL_{swap} y VND_2, primero BL_{swap} luego BL_{insert}.

A continuación, se comparan estos métodos. Para ello se ejecutan durante un tiempo máximo de 600 segundos, al alcanzar el tiempo máximo se detiene la ejecución y se retorna el resultado obtenido. Cada vez que se encuentra un mínimo local, el proceso se reinicia desde una nueva solución hasta que se agota el tiempo. Las soluciones de partida se generan mediante el mejor método constructivo identificado, el constructivo de Adjacency con los pesos $w_1 = -0,1$ y $w_2 = -1,0$, y un valor de α de 0,15. Nótese que todos los métodos de mejora parten de soluciones iniciales idénticas al emplear el mismo método constructivo, dado que este último posee una semilla fija. Esto significa que el comportamiento del método constructivo será el mismo en todos los experimentos, lo que garantiza que los métodos de mejora sean comparables.

En la Tabla II se presentan los resultados del experimento. Se puede observar que BL_{swap} obtiene resultados mejores que BL_{insert}. Además, VND_2 obtiene mejores resultados que VND_1. Es destacable, también, que VND_1 obtiene peores soluciones que la BL_{swap} por sí sola en el tiempo máximo establecido. Como resultado, se selecciona VND_2 como el mejor método de mejora entre los propuestos.

V. RESULTADOS FINALES

Por último, se compara la mejor propuesta algorítmica de este trabajo, el constructivo de Adjacency con VND_2, con el mejor método del estado del arte [13]. Se recuerda que esta propuesta, denotada como PIG, es un algoritmo poblacional basado en IG. Este algoritmo ha sido proporcionado por los propios autores, está programado en el lenguaje C y ha sido

Tabla II
RESULTADOS DEL CONSTRUCTIVO DE ADJACENCY, Y BL_{swap}, BL_{insert}, VND_1 Y VND_2.

	F.O.	T. CPU (s)	Desv. (%)	% Mejores
Adjacency	1537199,80	475,50	2,16	0
BL_{insert}	1501652,95	601,00	0,43	0
BL_{swap}	1489662,45	600,90	0,05	55
VND_1	1501226,30	601,05	0,38	10
VND_2	**1489363,45**	600,85	**0,02**	**65**

Tabla III
COMPARACIÓN DEL CONSTRUCTIVO ADJACENCY CON VND_2 Y LOS RESULTADOS OBTENIDOS POR EL ALGORITMO PIG DEL ESTADO DEL ARTE [13].

	F.O.	T. CPU (s)	Desv. (%)	% Mejores
Adjacency	1537199,80	475,50	2,82	0
VND_2	1489363,45	600,85	0,66	0
PIG [13]	**1471313,60**	**201,21**	**0,00**	**100**

ejecutado en el mismo entorno de experimentación que esta propuesta.

Los resultados se muestran en la Tabla III, en la que se incluyen el constructivo de Adjacency, el método de VND_2, y el algoritmo PIG del estado del arte. Se puede observar que el algoritmo formado por el nuevo método constructivo complementado con VND obtiene resultados competitivos cuando se compara con el algoritmo PIG, aunque los mejores resultados los obtiene el método del estado del arte. Un aspecto destacable de estos resultados es la baja desviación que presentan las soluciones generadas por el constructivo Adjacency sin el método de mejora, demostrando su eficacia.

VI. CONCLUSIONES

En este trabajo se ha estudiado el problema de S-labeling. Se proponen varios métodos constructivos novedosos para este problema. El más destacable es el de Adjacency, que es capaz de obtener soluciones con una desviación baja respecto a los mejores valores obtenidos por el algoritmo del estado del arte. Se amplía la propuesta probando múltiples métodos de búsqueda local y VND. Se comprueba de forma práctica la efectividad del movimiento de intercambio frente al de inserciones cuando se intenta mejorar una solución. Como resultado, se mejora las soluciones obtenidas por el constructivo de Adjacency, reduciendo la desviación con las soluciones obtenidas con PIG [13].

Para trabajos futuros se sugiere, en primer lugar, la exploración de nuevos vecindarios, con el objetivo de intentar obtener resultados mejores u obtenerlos en un tiempo menor. En segundo lugar, la introducción de mecanismos basados en la memoria, que podría mejorar el rendimiento del algoritmo. También se propone intentar optimizar el proceso de mejora, reduciendo el número de evaluaciones o reduciendo el tamaño del vecindario explorado. Por último, se propone emplear el constructivo de Adjacency en el algoritmo PIG y comprobar si hay una mejora en el rendimiento.

REFERENCIAS

[1] Cavero, S., Pardo, E.G., Duarte, A.: A general variable neighborhood search for the cyclic antibandwidth problem. Computational Optimization and Applications **81**(2), 657–687 (2022)
[2] Cavero, S., Pardo, E.G., Glover, F., Martí, R.: Strategic oscillation tabu search for improved hierarchical graph drawing. Expert Systems With Applications **243**, 122668 (2024)
[3] Cavero, S., Pardo, E.G., Laguna, M., Duarte, A.: Multistart search for the cyclic cutwidth minimization problem. Computers & Operations Research **126**, 105116 (2021)
[4] Díaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Computing Surveys (CSUR) **34**(3), 313–356 (2002)
[5] Duarte, A., Mladenovic, N., Sánchez-Oro, J., Todosijevic, R.: Variable Neighborhood Descent. In: Handbook of Heuristics, pp. 341–367. Springer International Publishing (Aug 2018)
[6] Dujmović, V., Wood, D.R.: On linear layouts of graphs. Discrete Mathematics and Theoretical Computer Science **6**(2), 339–358 (2004)
[7] Fertin, G., Rusu, I., Vialette, S.: The S-labeling problem: An algorithmic tour. Discrete Applied Mathematics **246**, 49–61 (Sep 2018)
[8] Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry pp. 35–41 (1977)
[9] Hansen, P., Mladenovic, N.: Variable Neighborhood Search, pp. 211–238. Springer US, Boston, MA (2005)
[10] Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M.: Variable neighborhood search. Springer (2019)
[11] Harary, F., Schwenk, A.J.: The number of caterpillars. Discrete Mathematics **6**(4), 359–365 (1973)
[12] López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)
[13] Lozano, M., Rodriguez-Tello, E.: Population-based iterated greedy algorithm for the S-labeling problem. Computers & Operations Research **155**, 106224 (Jul 2023)
[14] Martí, R.: Multi-start methods. Handbook of metaheuristics pp. 355–368 (2003)
[15] Martín-Santamaría, R., Cavero, S., Herrán, A., Duarte, A., Colmenar, J.M.: A practical methodology for reproducible experimentation: an application to the double-row facility layout problem. Evolutionary Computation pp. 1–36 (2023)
[16] Mcallister, A.J.: A new heuristic algorithm for the linear arrangement problem. Tech. rep., New Brunswick, CA: University of New Brunswick (1999)
[17] Merris, R.: Split graphs. European Journal of Combinatorics **24**(4), 413–430 (2003)
[18] Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & operations research **24**(11), 1097–1100 (1997)
[19] Pardo, E.G., Martí, R., Duarte, A.: Linear layout problems. In: Handbook of Heuristics, pp. 1025–1049. Springer (2018)
[20] Resende, M.G., Ribeiro, C.C.: Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. Handbook of metaheuristics pp. 283–319 (2010)
[21] Resende, M.G., Ribeiro, C.C.: Optimization by GRASP. Springer (2016)
[22] Sinnl, M.: Algorithmic expedients for the S-labeling problem. Computers & Operations Research **108**, 201–212 (Aug 2019)
[23] Stützle, T., Ruiz, R.: Iterated greedy. Handbook of heuristics pp. 547–577 (2018)
[24] Vialette, S.: Packing of (0, 1)-matrices. RAIRO-Theoretical Informatics and Applications **40**(4), 519–535 (2006)