# Greedy Randomized Adaptive Search Procedure para el Partial Set Covering Problem

Olga Somalo[1 [0009−0008−9143−9807]], Sergio Pérez-Peló[1 [0000−0002−1915−4160]], and Jesús Sánchez-Oro[1 [0000−0003−1702−4941]]

Dept. Informática y Estadística, Universidad Rey Juan Carlos, 28933, Móstoles, Spain
o.somalo.2021@alumnos.urjc.es,{sergio.perez.pelo,jesus.sanchezoro}@urjc.es
https://grafo.etsii.urjc.es/en/

**Resumen** Este trabajo se centra en la resolución del Problema de Cobertura Parcial de Conjuntos mediante enfoques metaheurísticos. Este problema surge como una variante más realista del clásico Problema de Cobertura de Conjuntos, con aplicaciones directas en logística, localización de instalaciones y análisis de redes, entre otros ámbitos. El problema se ha abordado principalmente mediante métodos exactos y el objetivo de este trabajo es evaluar el rendimiento del modelo matemático implementado en Gurobi, poniendo a prueba sus límites. A continuación, la propuesta metaheurística se compara con el enfoque exacto en aquellas instancias en las que este último logra encontrar el valor óptimo, y se evalúa también en un conjunto de instancias de mayor complejidad en las que el modelo exacto no consigue alcanzar la optimalidad. La experimentación computacional evidencia la necesidad de emplear algoritmos heurísticos, dado que el método exacto no logra encontrar ninguna solución óptima en las instancias complejas, mostrando una elevada desviación respecto a los resultados obtenidos por el algoritmo metaheurístico.

**Keywords:** Partial Set Covering Problem · Greedy Randomized Adaptive Search Procedure · Programación Lineal Entera · Metaheurísticas · Optimización Combinatoria

## 1. Introducción

Esta investigación se centra en la resolución del Problema de Cobertura Parcial de Conjuntos (PSCP, por sus siglas en inglés), que es una generalización del clásico Problema de Cobertura de Conjuntos (SCP). Sea $U = \{1, 2, \ldots, m\}$ un universo finito de elementos y $S = \{S_1, S_2, \ldots, S_n\}$ una colección de subconjuntos, donde $S_j \subseteq U \ \forall \ j \in \{1, \ldots, n\}$. Se define una matriz de incidencia $A = (a_{ij})_{m \times n}$, donde:

$$a_{ij} = \begin{cases} 1 & \text{si } i \in S_j \\ 0 & \text{en caso contrario} \end{cases}$$

El objetivo del SCP es encontrar una colección $C$ de subconjuntos con el coste mínimo, de tal forma que todos los elementos del universo estén cubiertos.

Un elemento se puede considerar cubierto únicamente si existe un $j \in \{1, \ldots, n\}$ tal que el valor de $a_{ij}$ es igual a uno; es decir, si el elemento está contenido en algún $S_j \in C$.

El PSCP busca relajar las restricciones del SCP con el fin de dotar de mayor flexibilidad al problema, permitiendo así modelar escenarios de la vida real donde la cobertura total no es necesaria o, incluso, resulta económicamente inviable. En particular, el PSCP no requiere cubrir la totalidad de los elementos del universo, sino un número mínimo de elementos $p$, con $1 \leq p \leq m$ (cabe señalar que el PSCP se convierte en el SCP cuando $p = m$). El PSCP puede modelarse formalmente como:

$$\text{mín} \quad Z = \sum_{j=1}^{n} x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{ij} x_j \geq y_i, \qquad \forall i = 1, \ldots, m, \tag{2}$$

$$\sum_{i=1}^{m} y_i \geq p, \tag{3}$$

$$x_j \in \{0, 1\}, \qquad \forall j = 1, \ldots, n, \tag{4}$$

$$y_i \in \{0, 1\}, \qquad \forall i = 1, \ldots, m. \tag{5}$$

El modelo requiere dos tipos de variables binarias:

- $x_j$ (Ecuación 4), que toma el valor 1 si se selecciona el subconjunto $j$ y 0 en caso contrario.
- $y_i$ (Ecuación 5), que toma el valor 1 si el elemento $i$ está cubierto y 0 en caso contrario.

La Ecuación 2 garantiza que los elementos cubiertos ($y_i = 1$), estén contenidos en al menos un subconjunto seleccionado. La Ecuación 3 asegura que se cubran al menos $p$ elementos. El objetivo en el PSCP (Ecuación 1) es el mismo que en el SCP: minimizar el número de subconjuntos seleccionados.

Para ilustrar el problema, consideremos como ejemplo un universo $U = \{1, 2, 3, 4, 5\}$ y una colección de subconjuntos $S = \{S_1, S_2, S_3, S_4\}$, donde el número mínimo de elementos a cubrir es $p = 4$. La cobertura de cada subconjunto se representa en la Figura 1.

Ahora analizamos dos posibles soluciones para el PSCP sobre esta instancia de ejemplo. La primera solución es $C_1 = \{S_2\}$. En este caso, los elementos cubiertos son $S_2 = \{2, 3, 4\}$, por lo que la solución resultante no es factible. La segunda solución es $C_2 = \{S_1, S_3\}$, por lo que los elementos cubiertos son $S_1 \cup S_3 = \{1, 2, 4, 5\}$, lo que resulta en una solución factible dado que el número de elementos cubiertos es mayor o igual que $p$, es decir, $|S_1 \cup S_3| \geq p$.

La relevancia del PSCP ha crecido continuamente entre la comunidad científica y los profesionales, principalmente debido a su mejor capacidad para modelar

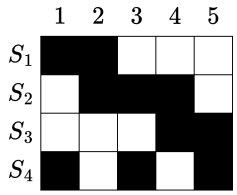



Figura 1: Instancia de ejemplo con un universo de cinco elementos y cuatro subconjuntos. Un subconjunto $S_j$ cubre a un elemento $i$ si la celda correspondiente a la fila $j$ y columna $i$ está coloreada en negro. Nótese que, por lo tanto, $a_{ij}$ valdrá 1 ya que el subconjunto $S_j$ cubre al elemento $i$.

escenarios donde los recursos son limitados en comparación con el SCP original. El problema fue introducido originalmente en [3], utilizando el parámetro $p$ para establecer una cobertura mínima para los elementos del universo. Este nuevo enfoque abrió una nueva línea de investigación en problemas de cobertura flexible, con aplicaciones directas en logística, localización de instalaciones, planificación de emergencias y análisis de redes, entre otros. Se ha demostrado que el PSCP es $\mathcal{NP}$-Difícil [7] al ser una generalización del SCP.

Desde una perspectiva teórica, el PSCP se ha abordado mediante modelos de Programación Lineal Entera (ILP). Específicamente, [8] mostró que algunas instancias geométricas específicas del PSCP podían transformarse en una versión de SCP con relajaciones lineales y teoría de aproximación. La mayor parte de la literatura se centra en la propuesta de algoritmos heurísticos para su resolución, debido a la complejidad del problema. [1] propuso una Búsqueda Tabú Iterada proporcionando soluciones de alta calidad, mientras que [10] presentó un algoritmo de Optimización por Colonia de Hormigas (ACO) que integra mecanismos adaptativos basados en sistemas bioinspirados para escapar de óptimos locales. Recientemente, en [12] se propuso un nuevo algoritmo basado en población denominado PSSC (Population-based Set Covering Solver), que combina estrategias de búsqueda tabú y *path-relinking*, logrando superar los mejores resultados en la literatura hasta la fecha.

La investigación actual sobre PSCP muestra un equilibrio entre procedimientos exactos y heurísticos. Por un lado, los métodos exactos están aprovechando los avances en solvers como Gurobi o CPLEX en los últimos años, aumentando el número de instancias resueltas de manera óptima y reduciendo la brecha en aquellas instancias donde el óptimo aún no se conoce. Por otro lado, las heurísticas y metaheurísticas están mejorando continuamente los resultados obtenidos mientras aumentan la eficacia de los algoritmos.

Sin embargo, aún existen algunos desafíos abiertos para el PSCP, como la ausencia de un banco de instancias estándar para facilitar comparaciones futuras o la falta de estudios comparativos entre enfoques exactos y heurísticos, entre otros. Esta investigación pretende avanzar en estas líneas. Por un lado, generando un nuevo banco de pruebas de instancias desafiantes. Por otro lado,

proponiendo un nuevo algoritmo metaheurístico basado en *Greedy Randomized Adaptive Search Procedure*, comparándolo con un enfoque exacto para el PSCP con el fin de evaluar la eficiencia y eficacia de ambos enfoques.

El resto del artículo se organiza de la siguiente manera: la Sección 2 presenta la propuesta algorítmica para resolver el PSCP; la Sección 3 muestra los resultados experimentales realizados para comparar el método exacto con la propuesta heurística; y, finalmente, la Sección 4 presenta las conclusiones y las líneas futuras de investigación derivadas de este trabajo.

## 2. Greedy Randomized Adaptive Search Procedure

El algoritmo propuesto para obtener soluciones de alta calidad para el PSCP se basa en el marco del *Greedy Randomized Adaptive Search Procedure* (GRASP), originalmente propuesto por [5] y posteriormente definido formalmente en [6]. GRASP es una metaheurística basada en trayectorias, compuesta por dos fases diferenciadas: construcción y mejora local. La primera tiene como objetivo generar una solución factible de alta calidad, mientras que la segunda se encarga de encontrar un óptimo local con respecto a un vecindario previamente definido. Ambas fases se ejecutan iterativamente hasta alcanzar un criterio de parada, que suele ser un número máximo de iteraciones o un tiempo máximo de cómputo. Esta metodología se ha aplicado con éxito en trabajos recientes [2,9,11].

La principal ventaja de GRASP reside en su capacidad para equilibrar la exploración y la explotación durante el proceso de búsqueda. En particular, en lugar de considerar un procedimiento constructivo completamente voraz, GRASP propone un procedimiento constructivo semivoraz, controlado por el único parámetro del algoritmo, habitualmente denominado $\alpha$. Este procedimiento constructivo se describe en la Sección 2.1, mientras que el método de búsqueda local diseñado para encontrar un óptimo local se presenta en la Sección 2.2.

El criterio de parada utilizado en esta investigación es una combinación entre un número máximo de iteraciones y un tiempo máximo de cómputo. En particular, el algoritmo construye iteravamente nuevas soluciones y posteriormente las mejora hasta alcanzar el número máximo de iteraciones. Sin embargo, si el proceso supera el tiempo máximo de cómputo permitido, el algoritmo se detiene incluso aunque todavía no se haya alcanzado el número máximo de iteraciones.

### 2.1. Fase de construcción

El objetivo de la fase constructiva es generar una solución inicial de alta calidad en tiempos de cómputo reducidos. En los procedimientos voraces tradicionales, el método selecciona iterativamente, de un conjunto de candidatos, aquel con el mejor valor según un determinado criterio voraz. Por el contrario, en la fase constructiva de GRASP, el candidato seleccionado no es el mejor, sino uno de los más prometedores, de forma que en cada iteración se construyen soluciones distintas y se introduce diversificación en la búsqueda. El grado de aleatoriedad se controla mediante un parámetro $\alpha \in [0, 1]$.

El Algoritmo 1 presenta el pseudocódigo del método constructivo propuesto.

**Algoritmo 1** `Construir`$(I = (U, S, p), \alpha)$

1: $S_f \leftarrow RND(S)$
2: $C \leftarrow \{S_f\}$
3: $CL \leftarrow S \setminus \{S_f\}$
4: $Cov \leftarrow S_f$
5: **while** $|Cov| < p$ **do**
6: $\quad g_{\text{máx}} \leftarrow \underset{S_i \in CL}{\text{máx}}\, g(S_i)$
7: $\quad g_{\text{mín}} \leftarrow \underset{S_i \in CL}{\text{mín}}\, g(S_i)$
8: $\quad \mu \leftarrow g_{\text{máx}} - \alpha \cdot (g_{\text{máx}} - g_{\text{mín}})$
9: $\quad RCL \leftarrow \{S_i \in CL : g(S_i) \geq \mu\}$
10: $\quad S_i \leftarrow RND(RCL)$
11: $\quad C \leftarrow C \cup \{S_i\}$
12: $\quad Cov \leftarrow Cov \cup S_i$
13: $\quad CL \leftarrow CL \setminus \{S_i\}$
14: **end while**
15: **return** $C$

El método comienza seleccionando de manera aleatoria el primer subconjunto a considerar para favorecer la diversidad (paso 1). A continuación, se crea la solución (paso 2) y la lista de candidatos ($CL$) (paso 3). Los elementos cubiertos inicialmente son el conjunto de elementos cubiertos por el primer subconjunto seleccionado, $S_f$ (paso 4). Luego, el método itera hasta cubrir el número mínimo de elementos $p$, seleccionando un nuevo subconjunto en cada iteración (pasos 5-14). En cada iteración, se evalúan los valores máximo y mínimo de un determinado criterio voraz (pasos 6-7). En este caso, el criterio voraz se calcula como el número de elementos nuevos que se cubrirían si se selecciona el subconjunto $S_i$. En términos matemáticos,

$$g(S_i) = \left| \{j \in S_i : j \notin \bigcup_{S_k \in C} S_k\} \right|$$

La justificación de este criterio es que cuanto mayor sea el número de elementos nuevos cubiertos, menor será el número de subconjuntos necesarios para conformar una solución factible. A continuación, se evalúa $\mu$ para definir un umbral que permita considerar un subconjunto como prometedor. Nótese que este umbral depende directamente del parámetro $\alpha \in [0, 1]$, el cual es responsable de equilibrar la aleatoriedad del proceso constructivo. En particular, si $\alpha = 0$, entonces $\mu = g_{\text{máx}}$, por lo que se consideran únicamente aquellos subconjuntos con el mayor valor voraz. Por el contrario, $\alpha = 1$ da lugar a un procedimiento constructivo totalmente aleatorio, ya que $\mu = g_{\text{mín}}$ y, por lo tanto, se consideran todos los subconjuntos. Posteriormente, se construye la Lista Restringida

de Candidatos (*RCL*, por sus siglas en inglés) con todos los subconjuntos candidatos cuyo valor de la función voraz es mayor o igual al umbral $\mu$ (paso 9). El siguiente subconjunto se selecciona de manera aleatoria de la *RCL* (paso 10) y se añade a la solución actual (paso 11). También se actualiza el conjunto de elementos cubiertos *Cov* (paso 12), así como la *CL* (paso 13). El método finaliza devolviendo la solución construida (paso 15).

### 2.2. Fase de mejora

La solución construida no es necesariamente un óptimo local con respecto a ningún vecindario, por lo que puede mejorarse en la segunda fase de GRASP. En esta investigación, el vecindario explorado está conformado por todas las soluciones que pueden alcanzarse mediante un único operador de movimiento. Dicho operador consiste en añadir un nuevo subconjunto a la solución y eliminar dos subconjuntos ya seleccionados. La solución resultante, en caso de ser factible, supondrá siempre una mejora con respecto al valor de la función objetivo, es decir, al número de subconjuntos seleccionados. Por lo tanto, cualquier solución factible encontrada en el vecindario se considerará una mejora.

La estrategia utilizada para recorrer el vecindario es *first improvement*, la cual presenta una menor exigencia computacional que la versión *best improvement*. Esto es debido a que la estrategia *best improvement* evalúa todos los movimientos posibles para realizar el que considera mejor, mientras que *first improvement* realiza el primero que consigue una mejora [4].

Para recorrer el vecindario, en primer lugar, se añade un subconjunto aleatorio a la solución. Luego se buscan dos subconjuntos que se puedan eliminar, de forma que la solución siga siendo factible, es decir, de forma que la cobertura de la solución siga estando por encima del umbral. Si se encuentran dichos subconjuntos, se eliminan de la solución y se reinicia la búsqueda. Si no se encuentran, se vuelve a la solución inicial, se añade otro subconjunto de forma aleatoria y se vuelven a buscar dos subconjuntos para eliminar.

## 3. Resultados computacionales

Esta sección tiene como objetivo evaluar el rendimiento del algoritmo propuesto. La experimentación se ha dividido en dos partes: la experimentación preliminar, diseñada para la calibración de parámetros y el análisis de cada componente; y la experimentación final, con el objetivo de comparar los resultados con el modelo matemático, implementado en Gurobi 13.

Todos los experimentos se han ejecutado en una máquina con procesador AMD EPYC 7282 con 16 núcleos y 16GB de RAM.

Las tablas muestran las siguientes métricas: F.O., el valor promedio de la función objetivo; T(s), el tiempo medio de cómputo requerido por cada algoritmo para finalizar; Desv.( %), la desviación promedio con respecto a la mejor solución encontrada en el experimento; y #Mejores, el número de veces que un algoritmo

alcanza la mejor solución del experimento. En todas las tablas se resaltan en negrita los mejores valores de cada métrica.

El banco de pruebas de instancias se divide en dos subconjuntos: las instancias de la OR-Library y las recién generadas. El primero comprende un subconjunto de 75 instancias derivadas directamente de la literatura con tamaños que van de 200 a 28.160 elementos y de 192 a 11.264 subconjuntos. El segundo está conformado por 165 instancias más desafiantes con tamaños de 30.000 a 50.000 elementos y de 10.000 a 20.000 subconjuntos, diseñadas para evaluar la escalabilidad de la propuesta y del modelo matemático.

### 3.1. Resultados preliminares

La fase preliminar tiene como objetivo principal determinar la configuración más adecuada del algoritmo GRASP. Para ello, se ha seleccionado un subconjunto representativo del 20 % de las instancias disponibles y se ha fijado el umbral de cobertura al 90 %, que es el umbral utilizado en la literatura reciente. Se han seleccionado 48 instancias de distintos tamaños y densidades.

El primer experimento tiene como objetivo estudiar la influencia del parámetro $\alpha$ sobre la calidad de las soluciones generadas por la fase de construcción voraz aleatorizada, ejecutada de forma aislada, es decir, sin aplicar la fase de búsqueda local posterior. Se evalúan los valores de $\alpha \in \{0; 0,25; 0,5; 0,75; RND\}$. Son 4 valores fijos de $\alpha$ y un valor $RND$, que indica que en cada iteración del algoritmo se le asigna a $\alpha$ un valor aleatorio en el intervalo $[0, 1]$. Para cada configuración y cada instancia del subconjunto preliminar, se han realizado 200 ejecuciones independientes del constructivo, registrando la mejor solución obtenida. Los resultados obtenidos se presentan en la Tabla 1.

| $\alpha$ | F.O. | T(s) | Desv.(%) | #Mejores |
|---|---|---|---|---|
| **0,00** | **488,52** | **3,42** | **0,00** | **48** |
| 0,25 | 500,98 | 3,76 | 4,82 | 2 |
| 0,50 | 561,08 | 5,84 | 21,84 | 0 |
| 0,75 | 712,48 | 7,84 | 48,19 | 0 |
| RND | 492,60 | 8,84 | 1,94 | 10 |

Tabla 1: Resultados del constructivo GRASP para distintos valores de $\alpha$ sobre las 48 instancias preliminares.

Analizando los resultados, se puede observar que la configuración con $\alpha$=0 obtiene la mejor solución del experimento en todas las instancias. Además, a medida que el valor de $\alpha$ aumenta, se observa una degradación progresiva de la calidad de las soluciones.

Dado que el mejor valor de $\alpha$ obtenido es $\alpha = 0$, el siguiente experimento compara ese constructivo GRASP con $\alpha$=0 con un constructivo voraz puro.

Ambos enfoques comparten la misma función voraz, que es seleccionar en cada paso el subconjunto que más elementos nuevos aporta a la solución. Sin embargo, el constructivo voraz puro es un algoritmo determinista de una sola pasada. El constructivo GRASP con $\alpha$=0, por el contrario, opera dentro del esquema iterativo multiarranque del GRASP. El parámetro $\alpha$=0 restringe la RCL a los mejores candidatos y selecciona uno de ellos de forma aleatoria. Así, distintas iteraciones pueden generar soluciones diferentes, y el mecanismo multiarranque retiene la mejor de todas ellas. Los resultados obtenidos se recogen en la Tabla 2.

| **Algoritmo** | **F.O.** | **T(s)** | **Desv.(%)** | **#Mejores** |
|---|---|---|---|---|
| Constructivo voraz | 492,81 | **0,04** | 1,64 | 13 |
| **Constructivo GRASP** | **488,52** | 3,42 | **0,00** | **48** |

Tabla 2: Comparación entre el constructivo voraz y el constructivo GRASP sobre las 48 instancias preliminares.

Vemos que el constructivo GRASP obtiene la mejor solución del experimento en todas las instancias y el constructivo voraz solo es capaz de igualarlo en 13 instancias.

El siguiente experimento preliminar tiene como objetivo encontrar el mejor valor para el parámetro $\alpha$, que es el único parámetro de entrada del algoritmo GRASP propuesto. En particular, se evalúan los mismos valores de $\alpha \in \{0; 0,25; 0,5; 0,75; RND\}$ sobre el GRASP completo. La Tabla 3 muestra los resultados obtenidos al generar 200 soluciones con GRASP.

| $\alpha$ | **F.O.** | **T(s)** | **Desv.(%)** | **#Mejores** |
|---|---|---|---|---|
| **0** | **487,56** | 547,11 | **0,00** | **48** |
| 0,25 | 498,44 | **418,75** | 4,25 | 2 |
| 0,5 | 534,21 | 779,60 | 13,57 | 2 |
| 0,75 | 552,52 | 968,07 | 16,57 | 2 |
| *RND* | 491,58 | 959,75 | 1,84 | 12 |

Tabla 3: Comparación de GRASP sobre el subconjunto preliminar de 48 instancias considerando diferentes valores de $\alpha$.

Los resultados muestran que la propuesta más voraz, con $\alpha = 0$, obtiene los mejores resultados en términos de calidad, alcanzando la mejor solución en todas las instancias. Analizando estos resultados, se realiza un nuevo experimento con el objetivo de comparar los resultados con $\alpha = 0$ frente a los obtenidos mediante un procedimiento completamente voraz, que consiste en un constructivo

voraz seguido de búsqueda local (denominado Constructivo+LS). Los resultados obtenidos se presentan en la Tabla 4.

| **Algoritmo** | **F.O.** | **T(s)** | **Desv.(%)** | **#Mejores** |
|---|---|---|---|---|
| Constructivo+LS | 491,75 | **4,88** | 1,77 | 13 |
| **GRASP** | **487,56** | 547,11 | **0,06** | **47** |

Tabla 4: Comparación entre el constructivo voraz con búsqueda local (Constructivo+LS) y el GRASP completo sobre las 48 instancias preliminares.

Los resultados muestran que romper empates de manera aleatoria permite a GRASP superar a la propuesta voraz, obteniendo 47 mejores soluciones encontradas y una desviación del 0,06 %, mientras que la propuesta voraz solo alcanza 13 mejores soluciones. El coste de esta mejora es un incremento significativo en el tiempo de ejecución. No obstante, en el contexto de este trabajo, donde el punto de referencia es el solver exacto Gurobi con un límite de 1800 segundos, el tiempo del GRASP se sitúa dentro del presupuesto computacional disponible. Por lo tanto, mantenemos GRASP con $\alpha = 0$ para toda la experimentación final.

### 3.2. Resultados finales

Los experimentos finales están diseñados para evaluar la propuesta exacta implementada en Gurobi frente al GRASP propuesto. El límite de tiempo establecido para Gurobi es de 1800 segundos para proporcionar una solución. GRASP se ejecutó una única vez sobre cada instancia, con un total de 200 iteraciones. Se han utilizado umbrales de cobertura del 90 % y del 95 % del total de elementos.

Los resultados de la comparación sobre el conjunto de 75 instancias de la OR-Library pueden observarse en la Tabla 5.

| **Cobertura** | **Algoritmo** | **F.O.** | **T(s)** | **Desv.(%)** | **#Mejores** |
|---|---|---|---|---|---|
| *90 %* | **GRASP** | **102,79** | **5,70** | **0,91** | **56** |
| | Gurobi | 107,61 | 1169,34 | 2,05 | 53 |
| *95 %* | **GRASP** | **116,33** | **5,46** | **1,17** | 50 |
| | Gurobi | 119,37 | 1334,70 | 2,32 | **52** |

Tabla 5: Comparación de GRASP($\alpha = 0$) y Gurobi 13 con las 75 instancias originales de la OR-Library.

Los resultados obtenidos en el conjunto OR-Library demuestran la eficacia y la eficiencia de Gurobi, que es capaz de alcanzar bastantes mejores soluciones

con una desviación algo superior al 2 %, ya que existen instancias en las que solo puede proporcionar una solución sin garantizar la optimalidad. GRASP surge como un método eficiente, capaz de alcanzar aproximadamente el mismo número de mejores soluciones requiriendo menos de 6 segundos de media, lo que lo hace aproximadamente 200 veces más rápido que Gurobi.

Para comprobar si las diferencias observadas en las tablas de resultados son estadísticamente significativas, se ha llevado a cabo un análisis estadístico sobre los valores de la función objetivo obtenidos por ambos métodos. Se ha optado por utilizar un test de rangos con signo de Wilcoxon, estableciendo un nivel de significancia del 5 % ($\alpha = 0{,}05$). La hipótesis nula ($H_0$) de este test postula que no existe diferencia significativa entre las soluciones encontradas por ambos métodos.

El test devolvió un p-valor de 0,18047. Puesto que el p-valor obtenido es mayor que el nivel de significancia, no se dispone de evidencia estadística suficiente para rechazar la hipótesis nula. Por tanto, se concluye que no hay evidencia de una diferencia estadísticamente significativa en la calidad de las soluciones aportadas por Gurobi y las obtenidas por el algoritmo GRASP en este conjunto de instancias.

Ahora es necesario evaluar los límites de ambos algoritmos al considerar un subconjunto más desafiante. Con ese fin, se ha realizado la misma comparación, pero esta vez considerando el nuevo conjunto de 165 instancias con tamaños significativamente superiores. Este conjunto de instancias se ha generado con distintas densidades: alta, media y baja. En este caso, la densidad es el número de subconjuntos que cubre cada elemento. En las instancias de densidad baja, cada elemento está cubierto por un número de subconjuntos en el rango del 0,02 % al 0,2 % del total de subconjuntos; en las de densidad media el rango es del 0,2 % al 2 %; y en las de denisdad alta es del 2 % al 20 %. La Tabla 6 muestra los resultados obtenidos en este experimento.

| **Cobertura** | **Algoritmo** | **F.O.** | **T(s)** | **Desv.(%)** | **#Mejores** |
|---|---|---|---|---|---|
| *90 %* | **GRASP** | **614,67** | **1142,00** | **0,00** | **165** |
| | Gurobi | 1514,58 | 1800,74 | 246,33 | 0 |
| *95 %* | **GRASP** | **792,90** | **827,28** | **0,00** | **165** |
| | Gurobi | 1544,47 | 1800,98 | 162,92 | 0 |

Tabla 6: Comparación de GRASP($\alpha = 0$) y Gurobi 13 (límite de tiempo de 1800 segundos) con el nuevo conjunto de 165 instancias más desafiantes.

En este caso, la necesidad de algoritmos heurísticos para el PSCP resulta aún más evidente. Debido al consumo excesivo de memoria, que impedía a Gurobi devolver alguna solución en ciertas instancias, el *solver* se restringió a un único hilo, lo que impactó negativamente en su rendimiento. El modelo exacto no es capaz de garantizar la optimalidad en ninguna de las instancias, alcanzando el

límite de tiempo de 1800 segundos en todas ellas. El valor 0 en la columna #Mejores para Gurobi indica que, en todas las instancias, la solución proporcionada por el *solver* fue peor que la obtenida por GRASP. Por el contrario, GRASP es capaz de alcanzar la mejor solución en todas las nuevas instancias en tiempos de cómputo razonables. Adicionalmente, el valor medio de la función objetivo obtenido por GRASP y la desviación de Gurobi muestran que la calidad de las soluciones proporcionadas por Gurobi está lejos del óptimo.

Sobre este conjunto de instancias también se ha realizado un test de rangos con signo de Wilcoxon estableciendo un nivel de significancia del 5 %. En este caso, el p-valor obtenido es de $7{,}5213 \times 10^{-56}$. Al ser un valor prácticamente nulo, podemos rechazar la hipótesis nula y confirmar que existe una diferencia estadísticamente significativa en la calidad de las soluciones a favor del algoritmo GRASP.

Estos resultados muestran la relevancia de los algoritmos aproximados al abordar problemas difíciles de optimización combinatoria, surgiendo GRASP como un algoritmo competitivo para tratar instancias más complejas del PSCP.

## 4. Conclusiones

Esta investigación propone un algoritmo metaheurístico basado en GRASP para resolver el Problema de Cobertura Parcial de Conjuntos. Este problema surge como una generalización del clásico Problema de Cobertura de Conjuntos que permite modelar de forma más realista escenarios prácticos. La complejidad del problema hace que los enfoques exactos, como la Programación Lineal Entera (ILP), no resulten adecuados para abordarlo, tal y como puede deducirse de los resultados experimentales obtenidos en las instancias más complejas. En este contexto, GRASP surge como una metaheurística simple pero potente para proporcionar soluciones de alta calidad para el PSCP. Los resultados obtenidos respaldan esta hipótesis y abren nuevas líneas de investigación para este problema y para nuevas variantes del mismo.

Como líneas de trabajo futuro, se plantean diversas extensiones orientadas tanto a mejorar el rendimiento del algoritmo propuesto como a ampliar su aplicabilidad. En primer lugar, resulta prometedora la implementación de un esquema GRASP reactivo que ajuste dinámicamente el parámetro $\alpha$ según el rendimiento observado, o incluso incorporar *Variable Neighborhood Search* (VNS) para mejorar la capacidad de escapar de óptimos locales mediante múltiples estructuras de vecindario (p. ej., intercambios 1-1, 2-1, etc.). Otra mejora significativa sería la intensificación mediante *path-relinking*, manteniendo un conjunto de soluciones élite para explorar combinaciones entre ellas. Además, dado el carácter independiente de las iteraciones de GRASP, una implementación paralela multihilo permitiría reducir los tiempos de cómputo sin sacrificar calidad. Asimismo, resulta de interés realizar una comparativa entre el GRASP propuesto y otras aproximaciones metaheurísticas de la literatura. Por otro lado, resulta de interés inmediato extender la propuesta a la variante ponderada del problema (PSCP con costes), donde cada subconjunto posee un coste diferenciado, lo que exigirá

rediseñar la función voraz y adaptar los criterios de la búsqueda local. Estas extensiones permitirían no solo mejorar el método propuesto, sino también ampliar su campo de aplicación a escenarios más generales de la logística y la optimización de recursos.

**Acknowledgments.** Esta investigación ha sido posible gracias al apoyo de la Comunidad Autónoma de Madrid (ref. TEC-2024/COM-404), del Ministerio de Economía y Competitividad (ref. PID2024-160226OB-C22) y del Ministerio para la Transformación Digital y de la Función Pública (Cátedra ENIA AI4DDS, ref. TSI-100930-2023-3).

## Referencias

1. Bilal, N., Galinier, P., Guibault, F.: An iterated-tabu-search heuristic for a variant of the partial set covering problem. Journal of Heuristics **20**(2), 143–164 (2014)
2. Casado, A., Pérez-Peló, S., Sánchez-Oro, J., Duarte, A.: A grasp algorithm with tabu search improvement for solving the maximum intersection of k-subsets problem. Journal of Heuristics **28**(1), 121–146 (2022)
3. Daskin, M.S., Owen, S.H.: Two new location covering problems: The partial p-center problem and the partial set covering problem. Geographical Analysis **31**(3), 217–235 (1999)
4. Duarte, A., Laguna, M., Martí, R., Sánchez-Oro, J.: Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem. Computers & operations research **51**, 123–129 (2014)
5. Feo, T.A., Resende, M.G.: A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters **8**(2), 67–71 (1989)
6. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. Journal of global optimization **6**(2), 109–133 (1995)
7. Hartmanis, J.: Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson). Siam Review **24**(1), 90 (1982)
8. Inamdar, T., Varadarajan, K.: On partial covering for geometric set systems. arXiv preprint arXiv:1711.04882 (2017)
9. Laguna, M., Martí, R., Martínez-Gavara, A., Pérez-Peló, S., Resende, M.G.: Greedy randomized adaptive search procedures with path relinking. an analytical review of designs and implementations. European Journal of Operational Research (2025)
10. Liu, X., Zhou, Y., Yin, M., Lv, S.: An argentine ant system algorithm for partial set covering problem. Data Technologies and Applications **56**(5), 762–781 (2022)
11. Sánchez-Oro, J., López-Sánchez, A.D., Hernández-Díaz, A.G., Duarte, A.: Grasp with strategic oscillation for the $\alpha$-neighbor p-center problem. European Journal of Operational Research **303**(1), 143–158 (2022)
12. Zhang, Y., He, J., Zhou, Y., Hu, S., Cai, D., Tian, N., Yin, M.: An effective population-based approach for the partial set covering problem. Journal of Heuristics **31**(1), 17 (2025)