Una propuesta metaheurística para el problema del *p*-centro sobre instalaciones desagradables

Sergio Salazar[1 [0009−0007−3492−2221]], J. Manuel Colmenar[1 [0000−0001−7490−9450]], and Abraham Duarte[1 [0000−0002−4532−3124]]

Dpto. Informática y Estadística, Universidad Rey Juan Carlos, Móstoles (Madrid), España

Resumen En este artículo se aborda el problema del p-centro sobre instalaciones desagradables, en el que se deben ubicar p instalaciones en una región acotada del espacio continuo, manteniendo al mismo tiempo una distancia mínima de seguridad con respecto a los puntos de demanda. El objetivo es minimizar la distancia máxima entre cada punto de demanda y su instalación más cercana. En este trabajo se propone un algoritmo basado en la metaheurística *Iterated Greedy*, que integra las propiedades geométricas del problema mediante diagramas de Voronoi. El método combina un procedimiento constructivo que aprovecha los vértices de Voronoi viables, una fase de destrucción adaptada que elimina las instalaciones sin contribución, y una fase de reconstrucción que reintroduce instalaciones al tiempo que garantiza la factibilidad. Además, se incorpora una búsqueda local basada en la búsqueda de patrones de Hooke–Jeeves que asegura la obtención de óptimos locales. El algoritmo se evalúa sobre 300 instancias de referencia y se compara con las propuestas del estado del arte. Los resultados experimentales demuestran mejoras significativas, reduciendo la desviación media en un orden de magnitud y disminuyendo drásticamente el tiempo de cálculo a menos del 6 %. El método propuesto alcanza las mejores soluciones conocidas en la mayoría de los casos, demostrando tanto eficiencia como robustez.

Keywords: Ubicación de instalaciones desagradables · Iterated Greedy · Optimización continua.

1. Introducción

Una de las familias más importantes entre los problemas de optimización son los problemas de ubicación de instalaciones (*Facility Location Problems*, FLP). En estos problemas se busca obtener la mejor ubicación de un conjunto de instalaciones con distintos objetivos, como minimizar el coste de comunicación [1], maximizar la cobertura de los clientes [2] o minimizar la competencia entre distintas empresas [3], entre otros.

Entre la familia FLP destaca un grupo denominado problemas de ubicación de instalaciones desagradables (*Obnoxious Facility Location Problems*, OFLP), donde se atiende a la influencia negativa sobre la población que tienen algunos

tipos de instalaciones. Por ejemplo, fábricas contaminantes, vertederos malolientes o aeropuertos ruidosos. Sin embargo, estas instalaciones son necesarias en la sociedad, por lo que identificar una ubicación estratégica para las mismas es un aspecto fundamental.

Dentro del grupo de problemas OFLP existen dos perspectivas para su modelado. Por un lado se puede seguir la perspectiva desde la optimización combinatoria. En este caso, el problema se modela sobre un grafo donde algunos nodos representan núcleos de población y posibles localizaciones de instalaciones. En este escenario se encuentra el *Obnoxious p-Median Problem*, problema en el que se deben abrir un número fijo de medianas sobre un grafo con el objetivo de maximizar la suma de distancias entre cada nodo con su mediana más cercana.

Sin embargo, la mayoría de los efectos negativos producidos por estas instalaciones, como la polución o el ruido, se transmiten por el aire, por lo que una modelización desde la perspectiva de la optimización continua se acerca a una conceptualización del problema más real. En este ámbito existen múltiples problemas en la literatura como el *Multiple Obnoxious Facilities Location Problem* [4], [5], donde se deben colocar un conjunto fijo de instalaciones en el plano en un polígono restringido, intentando maximizar la distancia entre las mismas y los núcleos de población. Otro ejemplo es el *Obnoxious Facilities Planar p-Median Problem with Variable Sizes* [6], [7], donde de nuevo se colocan las medianas en el plano, en este caso buscando minimizar la distancia entre instalaciones y poblaciones, dejando siempre una distancia mínima a la que se atribuye el efecto negativo de las instalaciones.

En esta perspectiva se encuadra el problema a tratar en este trabajo, el problema del p-centro sobre instalaciones desagradables (*Obnoxious p-Center problem*, OpC) [8]. El OpC es una variación del clásico problema *p-Center* [9] donde se entiende que las instalaciones a estudio crean un área de influencia negativa a su alrededor, debiendo mantener una distancia mínima respecto a las poblaciones. En el OpC se deben colocar un conjunto de p instalaciones en un polígono cerrado definido por la instancia a estudio. Cada una de las poblaciones presentes en la instancia será servida por una de las instalaciones colocadas, asumiendo que las poblaciones siempre serán asignadas a su instalación más cercana. El objetivo del problema es minimizar la máxima distancia entre los pares población-instalación. Dado que las instalaciones se consideran desagradables, se debe mantener una distancia D entre cualquier instalación y su población, lo que crea áreas circulares alrededor de las poblaciones donde no se pueden colocar las instalaciones. Como se verá más adelante, la complejidad del problema se incrementa cuando las áreas prohibidas de diferentes poblaciones se solapan.

En este trabajo se diseña una propuesta metaheurística basada en *Iterated Greedy* que es capaz de superar los resultados del estado del arte, tanto en términos de calidad de soluciones como de tiempo de ejecución. El trabajo se desarrolla de la siguiente manera: en la Sección 2 se describe formalmente el problema; en la Sección 3 se detalla la propuesta algorítmica para la resolución del problema descrito; en la Sección 4 se muestran los resultados obtenidos sobre

las instancias del estado del arte y se hace una comparativa sobre los mismos; finalmente en la Sección 5 se recogen las conclusiones y el trabajo futuro.

2. Descripción formal

Sea $\mathcal{B}$ una región acotada del plano y sean n puntos de demanda o núcleos de instalación $A_i = (a_i, b_i) \in \mathcal{B}$ con $i \in \{1, ..., n\}$. El problema OpC consiste en escoger la localización de p instalaciones con coordenadas $S_j = (x_j, y_j)$ con $j \in \{1, ..., p\}$. Sea $d(A_i, S_j)$ la distancia euclidea entre los puntos A_i y S_j, es decir, $d(A_i, S_j) = \sqrt{(a_i - x_j)^2 + (b_i - y_j)^2}$. Por último, se define una distancia $D \geq 0$, que determina la separación mínima entre instalaciones y poblaciones. El objetivo del problema consiste en minimizar la máxima distancia entre instalaciones y sus poblaciones más cercanas, sujeto a la restricción indicada por D, tal y como se describe en la Ecuación (1).

$$\begin{aligned} &\min_{S} F(S) \quad F(S) = \max_{1 \leq i \leq n} \left[\min_{1 \leq j \leq p} d(A_i, S_j) \right] \\ &\text{sujeto a} \\ &\qquad d(A_i, S_j) \geq D \qquad\qquad \forall i \in \{1, ..., n\},\ j \in \{1, ..., p\}, \\ &\qquad S_j \in \mathcal{B} \qquad\qquad \forall j \in \{1, ..., p\}. \end{aligned} \tag{1}$$

En la Figura 1 se muestra un ejemplo de instancia y solución del OpC para $p = 3$. La instancia de ejemplo muestra los puntos de demanda como triángulos y a su alrededor, en color rojo, las zonas prohibidas. Las instalaciones de la solución se representan con círculos y las asignaciones por lineas grises discontinuas. Finalmente el valor de la función objetivo se muestra resaltado por una linea morada, correspondiendo a la máxima distancia entre una población y su instalación asignada.

3. Propuesta algorítmica

El algoritmo propuesto para la resolución del OpC está basado en el uso de diagramas de Voronoi [10] para la construcción de soluciones iniciales, seguido de un proceso de mejora basado en la metaheurística *Iterated Greedy*. A continuación se describen las propuestas de este trabajo para cada uno de estos elementos.

3.1. Algoritmo constructivo

En los problemas de localización de instalaciones desagradables, es ampliamente conocido el uso de diagramas de Voronoi sobre el conjunto de poblaciones [4]. Los vértices de estos polígonos, denominados puntos de Voronoi, son aquellos puntos que equidistan a sus 3 poblaciones más cercanas, encontrando un equilibrio en la distancia entre ellas.

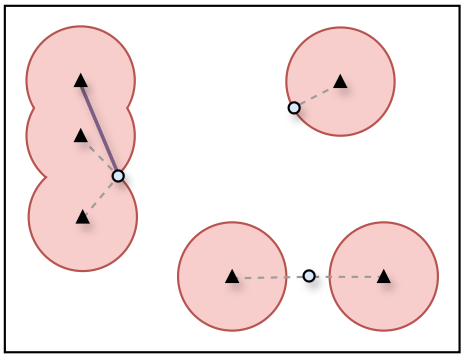


Figura 1. Ejemplo de instancia (poblaciones como triángulos y área restringida en rojo) y solución (círculos blancos) para el OpC.

Aprovechando esta característica, se propone un algoritmo constructivo que realizará una selección aleatoria del conjunto factible de puntos del diagrama de Voronoi (V^*) como semilla para la solución inicial. El Algoritmo 1 muestra el pseudocódigo del proceso constructivo propuesto. Inicialmente, en el paso 1, se construye el diagrama de Voronoi sobre el conjunto de poblaciones de la instancia, identificando todos los vértices del mismo en el conjunto V. Posteriormente se itera sobre los mismos, eliminando aquellos que están demasiado cerca de una población, es decir, filtrando los puntos infactibles, en los pasos 2 y 3. Finalmente, se escogen p puntos aleatorios que constituirán la solución de partida para el algoritmo y se devuelven junto con el conjunto de puntos de Voronoi factible, V^*, que se utilizará más adelante.

Algoritmo 1 `Constructivo`()

1: $V \leftarrow$ `Voronoi`(A)
2: $U \leftarrow \{z \in V : \min_{1 \leq i \leq n} d(A_i, z) < D\}$
3: $V^* \leftarrow V \setminus U$
4: $S \leftarrow$ `Random`(V^*, p)
5: **return** S, V^*

3.2. Algoritmo de mejora

El algoritmo de mejora está basado en la metaheurística *Iterated Greedy* (IG) [11]. En esta metaheurística se incluye un proceso de destrucción y otro de reconstrucción de la solución, de manera que permite explorar distintas regiones del espacio de búsqueda. Tras ello, dado que el proceso de reconstrucción puede no asegurar identificar un óptimo local de la instancia, se propone ejecutar un proceso de búsqueda local donde se identifica una nueva solución. Este proceso se realiza de forma iterativa siempre almacenando y reiniciando el proceso con la mejor solución encontrada hasta el momento, hasta que el algoritmo converge.

Fase destructiva La fase destructiva implementada tiene dos objetivos principales: por un lado, diversificar las soluciones y, por otro lado, identificar el conjunto de puntos de la solución que no tienen un aporte a la misma, para su posible eliminación.

Se considera que un punto de la solución no tiene aporte a la misma si, al eliminarlo, la función objetivo de la solución no cambia. El primer caso de puntos sin aporte que se consideran en esta propuesta lo forman todos aquellos puntos que, una vez la solución ha sido completamente construida, no tienen ninguna población asignada, es decir, para toda población existe una instalación que está mas cerca que el punto considerado. La Figura 2 ilustra este ejemplo, mostrando con un circulo azul un punto sin aporte.

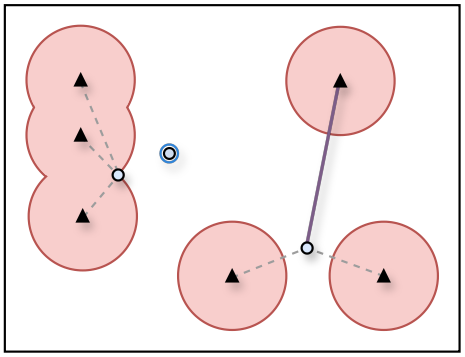


Figura 2. Ejemplo de un punto que no tiene demanda asignada.

El segundo conjunto de puntos sin aporte que se consideran se compone de aquellos puntos de la solución que, para todas sus poblaciones asignadas, su segunda instalación más cercana está a una distancia menor que el valor de la función objetivo. En ese caso, al eliminar esa instalación, el conjunto de poblaciones asignadas cambiarían su asignación a su segunda más cercana, que sigue teniendo una distancia menor a la de la función objetivo, por lo que esta no cambiaría. La Figura 3 ilustra este caso. El punto marcado con un circulo azul puede ser eliminado sin que el valor de la función objetivo cambie. Al hacerlo, la demanda de la instalación superior izquierda cambiaría a la instalación marcada por la flecha azul, dado que esta distancia no supera el valor de la función objetivo, esta no cambiaría.

El método destructivo propuesto identifica ambos conjuntos sin aporte y los elimina de la solución. Dado que el método busca generar diversificación, en caso de que se hayan eliminado pocos puntos, menos de un 10 %, se seguirán eliminando puntos de manera aleatoria hasta llegar a ese umbral.

El Algoritmo 2 muestra el pseudocódigo de la fase destructiva del método IG. En el primer paso, se identifican en el conjunto R aquellos puntos sin aporte para la función objetivo, como se ha descrito más arriba. Estos puntos se eliminan

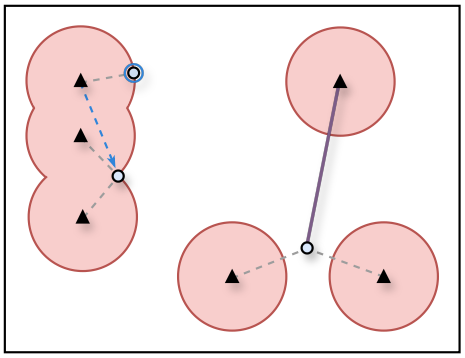


Figura 3. Ejemplo de un punto con demanda que no tiene aporte a la solución, resaltado con un círculo azul.

de la solución en el paso 2. Finalmente, entre los pasos 3 y 7 se eliminan puntos de manera aleatoria hasta alcanzar un 10 % de la solución. Como resultado, la solución parcial S' se devuelve en el paso final.

Algoritmo 2 `Destructivo`(S)

1: $R \leftarrow \{z \in S : F(S) = F(S \setminus \{z\})\}$
2: $S' \leftarrow S \setminus R$
3: $k \leftarrow \lfloor 0{,}1 \cdot |S| \rfloor$
4: **while** $|S| - |S'| < k$ **do**
5: $\quad z \leftarrow$ `Random(S,1)`
6: $\quad S' \leftarrow S' \setminus \{z\}$
7: **end while**
8: **return** S'

Fase reconstructiva En la fase reconstructiva se incluyen nuevas instalaciones asegurando que se encuentran en una zona factible.

El proceso de reconstrucción comienza identificando aquella comunidad que define la función objetivo, es decir, aquella cuya distancia a su instalación más cercana es máxima. En este punto se asigna una nueva instalación. El resto de instalaciones se seleccionan aleatoriamente sobre el conjunto de puntos de Voronoi factibles que se ha calculado en la fase constructiva (ver Sección 3.1).

Se puede observar que, dado que se ha colocado una instalación sobre una comunidad, a no ser que la distancia D fuera nula, se ha creado una solución infactible. Para hacer factible la solución se inicia un proceso de reconstrucción donde se aplican desplazamientos con dirección aleatoria y distancia $D/2$ sobre aquellas instalaciones que se encuentran en espacios infactibles. El proceso finaliza cuando todas las instalaciones están en un área factible de la instancia. La Figura 4 ilustra este proceso. Partiendo de la situación mostrada en la Figura 3, el nuevo punto se introduciría sobre la población superior derecha. Dado que se

encuentra en un espacio infactible, el proceso de reparación la desplaza en dos movimientos, hasta llegar a una zona factible. En este momento, el nuevo valor máximo para la función se reduce, como se ilustra en la Figura 4.

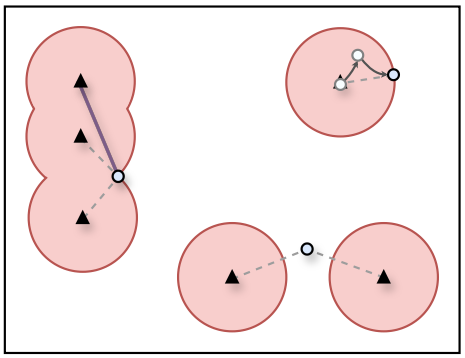


Figura 4. Ejemplo de reconstrucción de la solución

El Algoritmo 3 describe la fase reconstructiva dentro del esquema IG. En primer lugar, se identifica la población cuya distancia a su instalación más cercana es máxima. Esta población i^* es la que determina el valor de la función objetivo. A continuación, en el paso 2, se selecciona dicha población como nueva localización candidata para una instalación, construyendo el punto z^*. Este nuevo punto se añade a la solución parcial, obteniendo una solución intermedia S'. Posteriormente, en el paso 4, la solución se completa hasta alcanzar la cardinalidad p añadiendo instalaciones seleccionadas aleatoriamente del conjunto de puntos de Voronoi factibles V^*. Una vez construida la solución, se procede a garantizar su factibilidad. Para ello, en el paso 5, se recorre cada instalación de la solución. Para aquellas ubicadas en el área de infactibilidad se aplica un proceso de desplazamiento. En concreto, en el paso 7 se genera una dirección aleatoria uniforme en la circunferencia unidad, y en el paso 8 se desplaza la instalación una distancia $D/2$ en dicha dirección. Este proceso se repite hasta que la instalación se sitúa en una región factible. Finalmente, en el paso 11, se devuelve la solución reconstruida S', que cumple las condiciones de factibilidad del problema.

Búsqueda local La búsqueda local implementada está basada en el procedimiento de búsqueda de patrones de Hooke-Jeeves [12]. El proceso inicia con una distancia inicial τ recibida como parámetro e intenta mejorar la localización de la peor instalación posible, aquella cuya distancia es máxima. El proceso busca en las 8 direcciones cardinales a una distancia τ comprobando si modificar la localización mejora la función objetivo. En caso de mejora el punto se desplaza en esa dirección y se continua el proceso. Si no hay mejora, la distancia τ

Algoritmo 3 `Reconstructivo`(S, V^*)

1: $i^* \leftarrow \arg \max_{1 \leq i \leq n} d(A_i, S)$
2: $z^* \leftarrow A_{i^*}$
3: $S' \leftarrow S \cup \{z^*\}$
4: $S' \leftarrow S' \cup \texttt{Random}(V^*, p - |S'|)$
5: **for all** $z \in S'$ **do**
6: **while** $\min_{1 \leq i \leq n} d(A_i, z) < D$ **do**
7: $\vec{u} \leftarrow \texttt{RandomVector}(\mathbb{S}^2)$
8: $z \leftarrow z + \frac{D}{2}\vec{u}$
9: **end while**
10: **end for**
11: **return** S'

se modifica usando el parámetro γ. El proceso finaliza cuando τ se acerca lo suficientemente a cero, según el parámetro ϵ.

Durante el proceso de búsqueda, es posible que el punto de distancia máxima, el que debe ser desplazado para modificar la función objetivo, cambie. En ese caso el proceso simplemente itera manteniendo los mismos valores de τ con el nuevo punto. De hecho, podría iterar entre distintas instalaciones de la solución si el proceso lo requiriese.

Por último es necesario mencionar que el proceso de mejora es susceptible de buscar en áreas infactibles debido a la restricción del problema. En ese caso la evaluación de la función objetivo se asume como infinito, impidiendo que este sea un paso posible en la búsqueda.

El Algoritmo 4 muestra el pseudocódigo de la búsqueda local. En primer lugar se inicializa el tamaño de paso τ a partir del parámetro de entrada. A continuación, el algoritmo entra en un bucle (paso 2) que finaliza cuando el tamaño de paso es suficientemente pequeño, controlado por un umbral ϵ. En cada iteración, se identifica la instalación de distancia máxima de la solución actual, z^*, (paso 3). Este punto será el candidato a ser modificado. En el paso 5, se exploran las 8 direcciones cardinales $\mathcal{D}_8$ (horizontal, vertical y diagonales). Para cada dirección, se genera un punto candidato z' desplazando la instalación actual una distancia τ (paso 6) y se evalúa si este movimiento produce una mejora en la función objetivo (paso 7). Si se encuentra una mejora, se actualiza la solución y se interrumpe la exploración de direcciones para comenzar una nueva iteración (paso 9). Si no se encuentra ninguna mejora en ninguna de las direcciones (paso 12), se reduce el tamaño de τ multiplicándolo por un factor γ que se recibe como parámetro (paso 13), lo que permite refinar la búsqueda en un entorno más cercano. Finalmente, cuando el tamaño de paso es suficientemente pequeño, el algoritmo finaliza y devuelve la solución mejorada S.

Algoritmo 4 `BusquedaLocal`$(S, \tau_0, \epsilon, \gamma)$

1: $\tau \leftarrow \tau_0$
2: **while** $\tau > \epsilon$ **do**
3: $z^* \leftarrow \arg \max_{z \in S} d(z, A)$
4: $mejora \leftarrow$ `false`
5: **for all** $\vec{u} \in \mathcal{D}_8$ **do**
6: $z' \leftarrow z^* + \tau \cdot \vec{u}$
7: **if** $F(S \setminus \{z^*\} \cup \{z'\}) < F(S)$ **then**
8: $S \leftarrow S \setminus \{z^*\} \cup \{z'\}$
9: $mejora \leftarrow$ `true` , **break**
10: **end if**
11: **end for**
12: **if** $!mejora$ **then**
13: $\tau \leftarrow \tau \cdot \gamma$
14: **end if**
15: **end while**
16: **return** S

3.3. Propuesta final

El algoritmo IG propuesto en este trabajo se compone de los métodos que han sido descritos en las anteriores secciones, añadiendo como criterio de finalización el número de iteraciones sin mejora.

El Algoritmo 5 muestra el pseudocódigo de la propuesta. En primer lugar, se inicializa el contador de iteraciones sin mejora, que actuará como criterio de parada del algoritmo. A continuación, se genera una solución inicial S^* mediante el procedimiento constructivo. El algoritmo entra entonces en un proceso iterativo (paso 3) que se mantiene mientras no se alcance un número máximo $maxIters$ de iteraciones consecutivas sin mejora. En cada iteración, se aplica en primer lugar la fase destructiva sobre la mejor solución conocida, obteniendo una solución parcial (paso 4). Seguidamente, en el paso 5, dicha solución es reconstruida y, en el paso 6, se aplica la búsqueda local para obtener un óptimo local. Una vez generada la nueva solución S, en el paso 7 se evalúa si mejora la mejor solución conocida. En ese caso, en el paso 8 se actualiza S^* y se reinicia el contador de iteraciones sin mejora. En caso contrario, en el paso 11, se incrementa dicho contador. Finalmente, cuando se alcanza el límite de iteraciones sin mejora, el algoritmo finaliza y devuelve la mejor solución encontrada S^*.

4. Resultados experimentales

En esta sección se describen los experimentos llevados a cabo sobre el conjunto de instancias del estado del arte, comparando los resultados obtenidos con los métodos de la literatura.

Algoritmo 5 `IteratedGreedy`$(maxIters, \tau_0, \epsilon, \gamma)$

$iteracionesSinMejora \leftarrow 0$
$S^*, V^* \leftarrow$ `Constructivo`()
while $iteracionesSinMejora < maxIters$ **do**
  $S \leftarrow$ `Destructivo`(S^*)
  $S \leftarrow$ `Reconstructivo`(S, V^*)
  $S \leftarrow$ `BusquedaLocal`$(S, \tau_0, \epsilon, \gamma)$
  if $F(S) < F(S^*)$ **then**
    $S^* \leftarrow S$
    $iteracionesSinMejora \leftarrow 0$
  else
    $iteracionesSinMejora \leftarrow iteracionesSinMejora + 1$
  end if
end while
return S^*

4.1. Configuración e instancias

El algoritmo diseñado se ha ejecutado sobre un servidor AMD EPYC 7282 con 32 GB de RAM. El algoritmo al completo ha sido ejecutado en un solo núcleo del servidor. Los resultados previos se han tomado del estado del arte [8].

Como se ha ido describiendo a lo largo del artículo, la propuesta de este trabajo dispone de distintos parámetros que pueden ser modificados para variar el comportamiento del algoritmo, como puede ser el criterio de parada, el porcentaje mínimo de destrucción de la solución o los parámetros de la búsqueda de patrones. La configuración elegida tras una serie de experimentos preliminares es la siguiente: $maxIters = 5000$, $\tau_0 = 100$, $\epsilon = 10^{-4}$ y $\gamma = 0{,}75$.

El estado del arte presenta dos algoritmos distintos. Un primer enfoque basado en formulaciones matemáticas e implementado en Gurobi 11 que se denotará como MIQCP [13], y un segundo basado en el Sparse Nonlinear OPTimizer, que se denotará como SNOPT [14]. Solo el primer algoritmo aprovecha el uso de procesos paralelos para mejorar la velocidad del algoritmo.

Las instancias del estado del arte han sido escogidas por [15]. En concreto, son las denominadas `pr439`, `rat575`, `rat783`, `pr1002` y `rl1232` del *TSPLIB* [16]. Para cada una de ellas se establecen los 10 valores de $p = \{10, 20, ..., 90, 100\}$ y 6 valores distintos de D que corresponden a distintas porciones del espacio de búsqueda [8]. En total, se estudia un conjunto de 300 instancias.

4.2. Comparativa con el estado del arte

La Tabla 1 resume los resultados de la ejecución del algoritmo sobre las 300 instancias mencionadas. La primera columna (Algoritmo) muestra el algoritmo ejecutado, la segunda columna (F.O.) muestra el promedio de la función objetivo sobre las 300 instancias, la tercera (Desviación) muestra el promedio de la desviación del valor de la función objetivo sobre el mejor resultado conocido, la cuarta columna (Tiempo (s)) muestra el promedio de tiempo de ejecución sobre

todas las instancias y finalmente la última columna (# Bests) muestra el número de instancias donde el algoritmo alcanza la mejor solución conocida.

Tabla 1. Comparativa con el estado del arte

Algoritmo	**F.O.**	**Desviación**	**Tiempo (s)**	**# Bests**
MIQCP	1340.13	65.15 %	15791	66
SNOPT	1092.28	30.47 %	12076	119
IG	**811.30**	**3.78 %**	**716**	**179**

Los resultados obtenidos muestran la superioridad de la propuesta realizada contra las versiones del estado del arte. El algoritmo reduce a un 3,78 % los valores de desviación sobre los mejores resultados conocidos, mientras que las propuestas anteriores están un orden de magnitud por encima. Destaca además el tiempo de ejecución, que necesita únicamente 716 segundos, contra la mejor propuesta del estado del arte, SNOPT, que requiere 12076 en promedio. Por tanto, se reduce el tiempo de ejecución a menos de un 6 % respecto a los algoritmos conocidos. El algoritmo no es solo robusto, sino que además obtiene los mejores resultados en más de la mitad de las instancias, superando en 60 unidades a SNOPT. El resto de 121 instancias implican ese 3 % de desviación, por lo que, aunque no se encuentra la mejor solución, la distancia a la misma es reducida.

5. Conclusiones y trabajo futuro

En este trabajo se propone el primer algoritmo metaheurístico para el problema del p-centro sobre instalaciones desagrables en el espacio continuo. El algoritmo propuesto aprovecha las características del problema para que los diferentes componentes de la metaheurística, como el proceso de destrucción, reconstrucción y la búsqueda generen soluciones de calidad de manera eficiente. Los resultados muestran la superioridad del algoritmo propuesto frente a los existentes en el estado del arte, reduciendo en un orden de magnitud la desviación promedio respecto a la mejor solución conocida y reduciendo a un 6 % el tiempo de cómputo requerido.

Como se ha descrito, el algoritmo dispone de diversos parámetros de configuración cuyo valor ha sido escogido tras una experimentación preliminar. El uso de herramientas automáticas de configuración pueden mejorar aun más los resultados obtenidos. Por otro lado, otras estrategias basadas también en las características geométricas del problema pueden ser utilizadas como procesos de destrucción y reconstrucción en el método *Iterated Greedy*.

Agradecimientos. Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación (MCIN/AEI/10.13039/501100011033) y FEDER, *una manera de hacer Europa*, en los proyectos PID2024-156045NB-I00 y TSI-100930-2023-3 así como por la Comunidad Autónoma de Madrid, en el proyecto TEC-2024/COM-404 y la beca PIPF-2024/COM-35036.

Referencias

[1] M. S. Daskin y K. L. Maass, "The p-median problem," en *Location science*, Springer, 2015, págs. 21-45.

[2] F. Rauh, J. Matuschke y H. Yaman, "Dynamic Programming and Block-Cut Tree Decompositions for a Maximum Covering Location-Network Design Problem," en *Book of Abstracts*, 2025, pág. 10.

[3] M. R. Maia et al., "Metaheuristic techniques for the capacitated facility location problem with customer incompatibilities," *Soft Computing*, vol. 27, n.º 8, págs. 4685-4698, 2023.

[4] Z. Drezner, P. Kalczynski y S. Salhi, "The planar multiple obnoxious facilities location problem: A Voronoi based heuristic," *Omega*, vol. 87, págs. 105-116, 2019.

[5] S. Salazar, A. Duarte y J. M. Colmenar, "A fast metaheuristic approach for the planar multiple obnoxious facility location problem," *International Transactions in Operational Research*, 2025.

[6] P. Kalczynski y Z. Drezner, "The obnoxious facilities planar p-median problem with variable sizes," *Omega*, vol. 111, pág. 102 639, 2022.

[7] S. Salazar, O. Cordón y J. M. Colmenar, "Efficient heuristics for the obnoxious planar p-median problem with variable sizes," *Applied Soft Computing*, vol. 181, pág. 113 401, 2025.

[8] P. Kalczynski y Z. Drezner, "The obnoxious facilities p-center problem with forbidden regions," *Computers & Operations Research*, vol. 185, pág. 107 297, 2026.

[9] A. Suzuki y Z. Drezner, "The p-center location problem in an area," *Location science*, vol. 4, n.º 1-2, págs. 69-82, 1996.

[10] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM computing surveys (CSUR)*, vol. 23, n.º 3, págs. 345-405, 1991.

[11] R. Ruiz y T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European journal of operational research*, vol. 177, n.º 3, págs. 2033-2049, 2007.

[12] I. Moser, "Hooke-Jeeves revisited," en *2009 IEEE Congress on Evolutionary Computation*, IEEE, 2009, págs. 2670-2676.

[13] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2023.

[14] P. E. Gill, W. Murray, M. A. Saunders y E. Wong, "User's guide for SNOPT 7.7: Software for large-scale nonlinear programming," *Center for Computational Mathematics Report CCoM*, vol. 15, n.º 3, 2018.

[15] B. Callaghan, S. Salhi y J. Brimberg, "Optimal solutions for the continuous p-centre problem and related-neighbour and conditional problems: A relaxation-based algorithm," *Journal of the Operational Research Society*, vol. 70, n.º 2, págs. 192-211, 2019.

[16] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA journal on computing*, vol. 3, n.º 4, págs. 376-384, 1991.