XX Conferencia de la Asociación Española para la Inteligencia Artificial

CAEPIA 2024

19-21 de junio de 2024

A Coruña, España

Amparo Alonso, Bertha Guijarro, Óscar Fontenla, Noelia Sánchez, Beatriz Pérez, David Camacho, Juan R. Rabuñal, Manuel Ojeda-Aciego, Jesús Medina, José Riquelme, Alicia Troncoso, Eva Onaindia, Alberto Bugarín, José Antonio Gámez, María José Del Jesús Díaz, Luis Martínez, Francisco Bellas, Sara Guerreiro, Alejandro Rodríguez, José Alberto Benítez, María del Mar Marcos

Editores

Amparo Alonso
Universidade da Coruña
A Coruña, España

Bertha Guijarro
Universidade da Coruña
A Coruña, España

Óscar Fontenla
Universidade da Coruña
A Coruña, España

Noelia Sánchez
Universidade da Coruña
A Coruña, España

Beatriz Pérez
Universidade da Coruña
A Coruña, España

David Camacho
Universidad Politécnica de Madrid
Madrid, España

Juan R. Rabuñal
Universidade da Coruña
A Coruña, España

Manuel Ojeda-Aciego
Universidad de Málaga
Málaga, España

Jesús Medina
Universidad de Cádiz
Cádiz, España

José Riquelme
Universidad de Sevilla
Sevilla, España

Alicia Troncoso
Universidad Pablo de Olavide
Sevilla, España

Eva Onaindia
Universitat Politècnica de València
Valencia, España

Alberto Bugarín
Universidade de Santiago de Compostela
A Coruña, España

José Antonio Gámez
Universidad de Castilla la Mancha
Albacete, España

María José Del Jesús Díaz
Universidad de Jaén
Jaén, España

Luis Martínez
Universidad de Jaén
Jaén, España

Francisco Bellas
Universidade da Coruña
A Coruña, España

Sara Guerreiro
Universidade da Coruña
A Coruña, España

Alejandro Rodríguez
Universidad Politécnica de Madrid
Madrid, España

José Alberto Benítez
Universidad de León
León, España

María del Mar Marcos
Universitat Jaume I
Castelló de la Plana, España

ISBN: 978-84-09-62724-0

XV Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)

Sesión General

Un algoritmo memético para la ubicación de instalaciones desagradables en el plano con tamaño variable

Sergio Salazar
Dpto. de Informática y Estadística
Universidad Rey Juan Carlos
Móstoles, España
sergio.salazar@urjc.es

Óscar Cordón
Dpto. de CC. de la Computación e IA
Universidad de Granada
Granada, España
ocordon@decsai.ugr.es

J. Manuel Colmenar
Dpto. de Informática y Estadística
Universidad Rey Juan Carlos
Móstoles, España
josemanuel.colmenar@urjc.es

***Resumen*—La ubicación de instalaciones desagradables (*obnoxious*) es un problema de optimización con un gran impacto social. En concreto, el problema de la ubicación de instalaciones desagradables en el plano con tamaño variable (OPPMVS) estudia la ubicación de instalaciones considerando que el efecto nocivo se transmite a través del aire y depende de la producción o servicio de la instalación. En este trabajo se propone un algoritmo memético donde la generación de la población inicial y los operadores genéticos se han diseñado específicamente para el problema objetivo. En esta primera aproximación sobre este problema continuo se han obtenido resultados competitivos respecto al estado del arte, identificando diferentes puntos de mejora para trabajos futuros.**

***Index Terms*—Algoritmo memético, ubicación de instalaciones desagradables, optimización continua**

I. Introducción

La familia de problemas dedicados a la localización de instalaciones, *Facility Location Problems* (FLP) en su denominación en inglés, es amplia y sus aplicaciones van desde las distintas variantes del problema de distribución [1] hasta la localización de instalaciones teniendo en cuenta la planificación de su capacidad durante pandemias [2].

Dentro de la familia FLP existe un conjunto de problemas dedicados a la localización de instalaciones desagradables cuya principal característica es su impacto negativo en la población. Algunos ejemplos son la localización de instalaciones que generan ruido, contaminación o problemas de seguridad, como pueden ser aeropuertos, vertederos o centros penitenciarios [3].

Se procura que este tipo de instalaciones, necesarias para la sociedad, no estén demasiado cercanas a zonas residenciales o núcleos de población, de manera que su efecto negativo se reduzca. Para modelar estos escenarios se han propuesto diferentes problemas donde se definen de manera más concreta las restricciones que deben cumplir las instalaciones con respecto a las poblaciones, así como las instalaciones entre sí. En el ámbito de la optimización combinatoria se encuentra el problema de maximización de la suma de mínimas distancias entre las localizaciones, también denominado *Obnoxious p-Median* (OpM) [4] o el problema de maximización de la distancia mínima [5]. En estos casos las instalaciones se ubican en posiciones candidatas definidas en las instancias, cuya localización optimal se realiza mediante optimización combinatoria.

Sin embargo, la realidad es que los efectos nocivos como el ruido o los olores se transmiten por el aire desde las instalaciones a las poblaciones. En consecuencia, el modelado de estos escenarios desde el ámbito continuo parece ser una alternativa más realista. Por ejemplo, el *Multiple Obnoxious Facility Location Problem* (MOFLP) [6] propone maximizar las distancias en un polígono cerrado, manteniendo una separación mínima entre las instalaciones. También se ha definido un modelo cooperativo donde se asignan pesos de manera que se intenta "proteger" a la población más afectada [7].

Dentro de este último conjunto de problemas, destaca el problema de la ubicación de instalaciones desagradables en el plano con tamaño variable, *Obnoxious Facilities Planar p-Median Problem with Variable Sizes* (OPPMVS) [8]. En este problema el objetivo consiste en minimizar la suma de distancias entre el conjunto de instalaciones y el de poblaciones considerando que las instalaciones con las que se trabaja son desagradables y por lo tanto generarán un área perjudicial a su alrededor proporcional al trabajo que estén produciendo. En consecuencia, la restricción indica que no podrá haber ninguna población en el interior de esta área. Por otro lado, además de la restricción de distancia entre poblaciones e instalaciones, se incluye una segunda restricción que implica que las instalaciones deben tener un cierto grado de separación, evitando así soluciones donde se puedan colocar dos instalaciones en el mismo punto del plano.

Los algoritmos meméticos se han utilizado previamente para resolver problemas de ubicación de instalaciones [9], [10]. Sin embargo, esos trabajos se han hecho sobre posiciones dadas, no sobre el plano. En este trabajo se propone una primera versión de un algoritmo memético para el OPPMVS, por lo que nuestra propuesta se presenta como novedad en la literatura. El hecho de escoger un algoritmo memético como técnica metaheurística para resolver este problema es debido a que presenta una serie de restricciones complicadas que reducen la región factible y el buen equilibrio entre diversificación e intensificación de estos algoritmos puede ayudar a resolverlo de forma adecuada.

Como punto de partida, se ha definido un procedimiento de generación de la población inicial donde se aplican dos métodos diferentes para construir las soluciones: uno aleatorio y otro basado en la metodología *Greedy Random Adaptative Search Procedure* (GRASP) [11]. También se ha diseñado una búsqueda local que permite desplazar los puntos de la solución en el plano. Finalmente, se proponen operadores genéticos de cruce, mutación y reemplazo específicos para el problema.

En lo referente a la validación experimental, se ha trabajado con un total de 21 instancias provenientes de [8]. Se muestran los resultados de dos experimentos diferentes del algoritmo. El primer experimento transcurre durante 200 generaciones y obtiene una desviación del $0{,}33\,\%$ utilizando un tercio del tiempo que la propuesta del trabajo previo. El segundo experimento se ejecuta durante 1000 generaciones logrando reducir la desviación al $0{,}088\,\%$, a costa de duplicar el tiempo de ejecución del método del estado del arte.

El resto del artículo se organiza de la siguiente manera: en la Sección II se define formalmente el problema, en la Sección III se describe la propuesta algorítmica, en la Sección IV se muestran los resultados experimentales y, finalmente, en la Sección V se exponen las conclusiones y el trabajo futuro.

II. Definición formal del problema

Sea $C = \{c_1, ...c_n\}$ un conjunto de puntos en el plano donde se localizan las comunidades (poblaciones) definidas por la instancia del problema, $S = \{a_1, ..., a_p\}$ un conjunto solución y $d(c_i, a_j) = d_{ij}$ la distancia euclídea entre la localización c_i de una comunidad y la instalación a_j. Se define w_i como el servicio necesitado por la comunidad c_i, de modo que W sea la matriz de asignación de dimensión $n \times p$ donde el valor w_{ij} corresponde con el servicio que aporta la instalación a_j a la comunidad c_i [8].

Bajo este escenario, la función objetivo es la distancia mínima de separación entre instalaciones, definida en la Ecuación (1), donde el parámetro $\widehat{D}$ es el factor de separación entre instalaciones y el parámetro D es el factor de separación entre comunidades e instalaciones por trabajo asignado.

$$
\begin{aligned}
\text{mín} \quad & \sum_{i=1}^{n}\sum_{j=1}^{p} w_{ij} d(c_i, a_j) \\
\text{s.t.} \quad & \\
& d(a_j, a_k) \geq \widehat{D} && \forall a_j, a_k \in S,\ j \neq k \\
& d^2(c_i, a_j) \geq \frac{D^2 p}{\sum_{i=1}^{n} w_i} \sum_{k=1}^{n} w_{kj} && \forall c_i \in C,\ \forall a_j \in S \\
& \sum_{j=1}^{p} w_{ij} = w_i && \forall c_i \in C \\
& w_{ij} \geq 0 && \forall c_i \in C,\ \forall a_j \in S
\end{aligned}
\tag{1}
$$

Véase que la formulación del problema permite dividir el trabajo requerido por una población en distintas instalaciones, de forma que se comparta el peso w_i entre las mismas, con la correspondiente propagación de la distancia requerida entre población e instalación.

Siguiendo el trabajo previo se puede observar que, dada una ubicación de las p instalaciones en el plano, asumiendo que todas ellas cumplen la restricción $d(a_j, a_k) \geq \widehat{D}$, la selección de los mejores valores de w_i se transforma en un problema lineal, que puede ser resuelto a través de un *solver* matemático.

Al mismo tiempo, en [8] se demuestra que la restricción anterior no es suficiente para que la localización dada sea factible y que la condición de la Ecuación (2) es necesaria y suficiente para que el problema tenga una asignación factible:

$$
\frac{1}{p}\sum_{j=1}^{p} \min_{1 \leq i \leq n} d(c_i, a_j) \geq D^2 \tag{2}
$$

El problema lineal y la Ecuación (2) proporcionan métodos para obtener una asignación óptima y comprobar la factibilidad de las soluciones, que se utilizarán en la propuesta algorítmica.

III. Propuesta algorítmica

Siguiendo el enfoque del trabajo previo, se partirá de un conjunto de puntos candidatos a formar parte de las soluciones provenientes de la discretización del espacio de búsqueda. Esta discretización se realiza identificando los puntos de Voronoi del diagrama conformado por las poblaciones de la instancia. Estos puntos son buenos candidatos a elementos de la solución ya que son equidistantes a al menos tres poblaciones [8].

A continuación se detalla el algoritmo propuesto, así como los diferentes elementos que lo componen.

III-A. Algoritmo memético

Para diseñar la propuesta algorítmica memética, la implementación realizada (Algoritmo 1) se ha basado en [12]. Como se puede apreciar, el algoritmo recibe 12 parámetros: C, conjunto de comunidades de la instancia; $maxGen$, número de generaciones; $pobR$, número de individuos generados aleatoriamente; $pobG$, número de individuos generados con el método GRASP; α, θ y k, que se utilizan en el método constructivo GRASP y se explicarán más adelante; λ_C, probabilidad de cruce; λ_M, probabilidad de mutación; μ, número de elementos que mutarán; λ_{LS}, probabilidad de aplicar la búsqueda local; y β, tamaño de paso (profundidad) de la búsqueda local. El algoritmo comienza obteniendo el conjunto de puntos candidatos V a partir de la discretización de Voronoi [8]. A continuación, se construyen dos conjuntos de soluciones iniciales, P_0 y P_1, a partir de dos métodos constructivos diferentes: uno aleatorio, `constructivoRandom`, y otro basado en la metodología GRASP, `constructivoGRASP`, que se describirán más adelante. Así, la unión de ambos conjuntos forma la población inicial P, generada en el paso 4. Seguidamente, comienza el bucle principal del algoritmo con la generación de los individuos resultantes del cruce, que se almacenan en el conjunto O (paso 7). A estos individuos, se le añaden los resultantes de la mutación (paso 8) y los resultantes de la búsqueda local (paso 9). La nueva población P se genera a partir de la elección de $pobRnd + pobGRASP$ individuos de los conjuntos O y P, correspondientes a la población de descendientes y a la población de progenitores, en el paso

10. Las funciones `Cruce`, `Mutacion`, `BusquedaLocal` y `Reemplazo` que implementan las operaciones homónimas, se explicarán más adelante. El último paso del bucle incrementa el valor de la iteración, correspondiente a la generación actual, de manera que el bucle iterará hasta alcanzar $maxGen$ generaciones. Finalmente el algoritmo termina devolviendo el individuo que mejor función objetivo alcanza, que se obtiene a través de la función `Elite`.

Algorithm 1: $Memetico(C, maxGen, pobR, pobG, \alpha, \theta, k, \lambda_C, \lambda_M, \mu, \lambda_{LS}, \beta)$

```
1:  V ← Voronoi(C)
2:  P_0 ← constructivoRandom(V, pobR)
3:  P_1 ← constructivoGRASP(V, pobG)
4:  P ← P_0 ∪ P_1
5:  iters ← 0
6:  while iters < maxGen do
7:      O ← Cruce(P, λ_C)
8:      O ← O ∪ Mutacion(P, Vλ_M, μ)
9:      O ← O ∪ BusquedaLocal(P, O, λ_LS, β)
10:     P ← Reemplazo(P, O, pobR + pobG)
11:     iters ← iters + 1
12: end while
13: return Elite(P)
```

III-B. Construcción de la población inicial

En esta sección se describe la discretización inicial del espacio de búsqueda, así como las dos metodologías para la generación de la población inicial vistas en el Algoritmo 1.

La discretización del espacio factible tiene como objetivo obtener buenos puntos candidatos que posteriormente formarán parte de las soluciones de la población inicial. El método `Voronoi` del Algoritmo 1 genera este conjunto de puntos a partir del conjunto de comunidades dado.

En el problema propuesto, la codificación de los individuos de la población consiste en definir un conjunto de p puntos del plano, que serán las localizaciones de las instalaciones. Sin embargo, la construcción de una solución partirá del conjunto de candidatos obtenido con `Voronoi`. En este proceso de selección, se asegurará la factibilidad de cada solución comprobando que las soluciones generadas verifican la restricción de distancia indicada en la Ecuación (2).

En este trabajo se propone la utilización de dos métodos diferentes para la construcción de individuos de la población inicial. El primer método, denominado `constructivoRandom` en el Algoritmo 1, selecciona aleatoriamente p elementos del conjunto de puntos de Voronoi. Si estos puntos no forman una solución factible, ésta se descarta.

El segundo método, denominado `constructivoGRASP` en el Algoritmo 1, sigue la metodología GRASP donde la función voraz a evaluar se ha diseñado de modo que considere la distancia acotada a las poblaciones más cercanas utilizando dos parámetros de entrada: θ y k. Para ilustrar el cálculo de la función para un punto x dado, se siguen los pasos indicados en la Figura 1. Suponiendo seis puntos c_1 a c_6 correspondientes a comunidades, se calcula la distancia del punto x a cada una de ellas, teniendo seis distancias. De esas distancias se descartan aquellas cuyo valor sea inferior a θ, que en el ejemplo es 4. El objetivo de esta poda es obtener puntos cercanos hasta el límite definido por θ. De las distancias resultantes, se realiza la suma de las k menores, $k = 2$ en el ejemplo. Obteniendo así, el valor de $f(x, \theta, k)$.

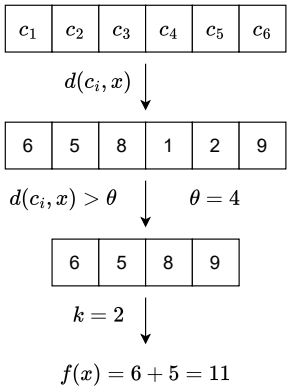

Figura 1. Ejemplo de cálculo de la función $f(x, \theta, k)$ para 6 comunidades.

El motivo por el que la función tiene este diseño es doble. Por un lado, la función objetivo debería mantener las instalaciones lo más cerca posible al conjunto de comunidades a las que aporta servicio. Sin embargo la restricción de distancia que impone la Ecuación (1) muestra como, en el caso de que una de ellas esté demasiado cerca de su población más cercana, no será posible localizarse fuera del área de prohibición variable, por lo que no se podrán asignar suficientes comunidades a esta instalación. Los parámetros θ y k, así como el modo en que se ha diseñado f, pretenden buscar un equilibrio entre ambas características.

Para cada uno de los puntos candidatos que recibe `constructivoGRASP`, se calculará el valor de $f(x, \theta, k)$. A continuación, siguiendo la metodología GRASP se obtendrá un valor umbral th en cada una de las iteraciones

$$th = (f_{min} + \alpha \cdot (f_{max} - f_{min}))$$

donde f_{min} y f_{max} representan los valores mínimo y máximo de las evaluaciones obtenidas al aplicar la función f al conjunto de puntos de cada solución. De esta manera, cuanto más cercano sea el valor de α a 0, mas voraz será la selección, y viceversa acercándose al valor 1.

Seguidamente, se construye la lista de candidatos restringida con todos aquellos puntos cuya evaluación sea inferior a th. De este conjunto se escoge un elemento aleatorio que se añade a la posible solución. Una vez la solución se compone de p elementos, se verifica su factibilidad.

III-C. Cruce

El operador de cruce se ha diseñado siguiendo una estrategia de votación. Sean dos soluciones $S = \{a_1, ..., a_p\}$ y

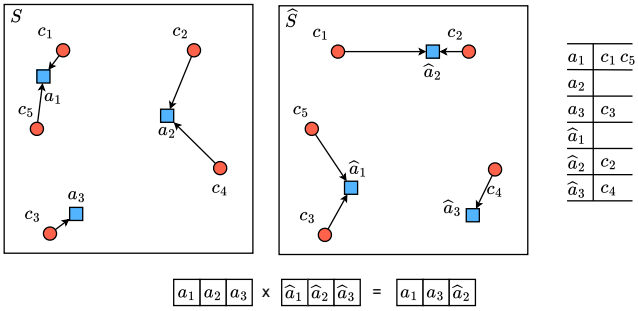

Figura 2. Ejemplo de aplicación de cruce.

$\widehat{S} = \{\widehat{a}_1, ..., \widehat{a}_p\}$, la Figura 2 muestra un ejemplo de cruce entre ellas, donde las comunidades se representan con círculos rojos y las instalaciones con cuadrados azules. El proceso itera por las distintas comunidades, de modo que cada una de ellas otorga un voto a la instalación más cercana asignada entre las dos soluciones. En el caso de la comunidad c_1, debe elegir si votar a a_1, que es la instalación asignada en S, o a $\widehat{a}_2$, que es la asignada en $\widehat{S}$. En este ejemplo, dado que $d(c_1, a_1) < d(c_1, \widehat{a}_2)$, el voto se asociará a a_1. En caso de que una instalación esté asignada parcialmente a más de una población, se tendrá en cuenta la asignación de más peso a la hora de medir las distancias. A la derecha de la figura se muestran los votos que ha recibido cada instalación. El resultado del cruce, será una solución formada por el conjunto de p instalaciones más votadas, entre las $2p$ posibles, como se muestra en la parte inferior de la figura. En el caso de empates entre número de votos, la selección será aleatoria entre las instalaciones implicadas. En el ejemplo de la figura, tres instalaciones reciben un voto, por lo que se elige aleatoriamente dos de ellas.

Por otro lado, puede suceder que dos instalaciones provenientes de distintas soluciones tengan ambas un número elevado de votos pero estén demasiado cerca entre sí, haciendo infactible el resultado del cruce. Para controlar esta situación se impone una distancia mínima entre las instalaciones escogidas en el cruce igual al parámetro $\widehat{D}$ de la instancia, de tal manera que, al intentar incluir la siguiente instalación, se detectaría el conflicto y ésta sería ignorada.

III-D. Mutación

El operador de mutación seleccionará de forma aleatoria una fracción μ de la solución que será eliminada. A continuación, se añadirán esa misma cantidad de puntos a partir de la discretización inicial de Voronoi, manteniendo la factibilidad.

III-E. Búsqueda Local

El objetivo de la búsqueda local propuesta es trasladar la ubicación de las instalaciones, moviéndolas de modo que se mejore su aportación a la función objetivo.

Sea a_i la instalación a mejorar por la búsqueda local y $C(a_i) = \{\zeta_1, ..., \zeta_k\} \subset C$ el conjunto de comunidades de la instancia que están asignadas a a_i. Entre cada una de las comunidades ζ_j y la instalación existe un vector unitario que indica la dirección de movimiento en la que se acercaría a esa comunidad. Se denota ese vector como $v_j = \overrightarrow{(a_i, \zeta_j)}$.

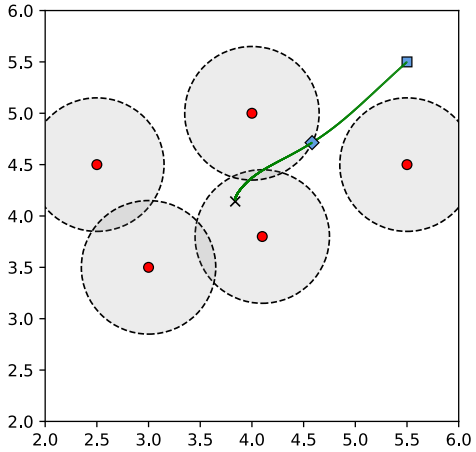

Figura 3. Ejemplo de trayectoria de instalación debido a búsqueda local.

La composición de todos los vectores v_j con $\zeta_j \in C(a_i)$ genera un nuevo vector que indica una dirección de movimiento que reduce la suma de distancias a las comunidades asignadas a a_i. Este vector se denota como $\overrightarrow{r}$.

En cada iteración de la búsqueda local se desplazará la instalación a_i una fracción β del vector $\overrightarrow{r}$. Cabe destacar que en cada paso de la iteración el punto a_i se modifica y por tanto también lo hacen v_j y $\overrightarrow{r}$, por lo que este debe ser recalculado.

Por otro lado, la elección del paso β es significativa para el algoritmo. Un valor demasiado alto reduce la precisión de la búsqueda local e incluso puede acarrear problemas de convergencia al no poder acercarse correctamente al objetivo. Sin embargo, un paso demasiado bajo ralentiza el algoritmo, aunque ajuste mejor la curva de desplazamiento que sigue la instalación. En la práctica, el coste computacional de la búsqueda local es significativamente bajo en comparación con el resto de operadores. Esto permite utilizar valores pequeños para β, como se verá en la sección de experimentos.

Cabe destacar que la trayectoria se puede ver interrumpida por las zonas restringidas alrededor de cada comunidad. En caso de alcanzar una de esas zonas, la búsqueda local se detiene, dejando la instalación ubicada en la intersección entre la trayectoria y la zona restringida.

La Figura 3 ilustra el funcionamiento de la búsqueda local. La instalación representada en el cuadrado azul se desplazaría por la trayectoria indicada por la línea verde. Sin embargo, se detiene en la intersección con la zona restringida gris, devolviendo el punto denotado con un rombo azul.

III-F. Reemplazo

El reemplazo de los individuos para la nueva población se realiza a través de torneo binario [13]. Los individuos de la

población anterior P y sus descendientes O (ver paso 10 del Algoritmo 1) se añaden de manera aleatoria a una lista. El proceso de reemplazo enfrenta el individuo i e $i+1$ de la lista, de modo que aquel cuya función objetivo tenga un valor menor, será seleccionado para formar parte de la siguiente generación. Es en este punto, por tanto, cuando se realiza la evaluación completa de los individuos de P y O. El otro individuo no se deshecha, sino que se vuelve a añadir al final de la lista, ya que la suma de la cardinalidad de P y de O puede ser inferior al doble del tamaño de población requerido.

Es importante destacar que el reemplazo garantiza que el mejor individuo sobreviva, y aporta diversidad gracias a la selección aleatoria de los individuos participantes en el torneo.

IV. Resultados experimentales

En esta sección se describen los resultados experimentales obtenidos al aplicar la propuesta algoritmica al problema OPPMVS. Los experimentos se han desarrollado en un ordenador portátil con una CPU `Intel(R) Core(TM) i5-10210U, 1.60 GHz`, 16GB RAM ejecutando Windows 10. El código se ha desarrollado en `Java 21.0.1` y se ha utilizado `Gurobi` para la resolución del problema lineal mencionado en la Sección II.

Durante la experimentación se ha utilizado el conjunto de instancias presentadas en el trabajo previo [8]. En este conjunto de instancias se usan 3 valores para el parámetro $n \in \{100, 200, 500\}$ y 7 valores para el parámetro $p \in \{2, 3, 4, 5, 10, 15, 20\}$. En total, el algoritmo se ejecuta en 21 instancias donde las localizaciones de las comunidades se obtienen a través del procedimiento aleatorio descrito en [8].

IV-A. Análisis de convergencia del algoritmo

Durante los experimentos preliminares de este trabajo se observó una convergencia prematura del algoritmo. Para identificar este fenómeno se estudió la calidad de la mejor solución de cada generación con respecto a la calidad promedio de toda la población. La convergencia prematura lleva al algoritmo a reducir las posibilidades de mejorar la solución élite identificada. Para contrarrestar este efecto la estrategia utilizada consiste en regenerar la población actual cuando el valor promedio de las soluciones de las ultimas t generaciones se diferencie en menos de un ϵ por ciento del mejor valor obtenido hasta el momento. En la regeneración se crea una nueva población aplicando la fase constructiva del algoritmo, compuesta por los dos métodos descritos en la Sección III, añadiendo siempre la mejor solución encontrada hasta el momento. En Figura 4 se puede observar una comparativa de la calidad de la mejor solución de cada generación con respecto a la calidad promedio de toda la población para la instancia $n = 100, p = 15$. La linea continua azul corresponde con el valor obtenido para la mejor solución encontrada en cada generación, mientras que la linea discontinua amarilla corresponde con el valor promedio de la población. La gráfica superior muestra la evolución sin regeneración, mientras que la parte inferior muestra el mismo experimento pero con regeneración para $\epsilon = 5\,\%$ y $t = 30$. Como se puede apreciar, el valor de la función objetivo de la mejor solución se reduce tanto al inicio de la ejecución como en generaciones posteriores, obteniendo mejores resultados finales.

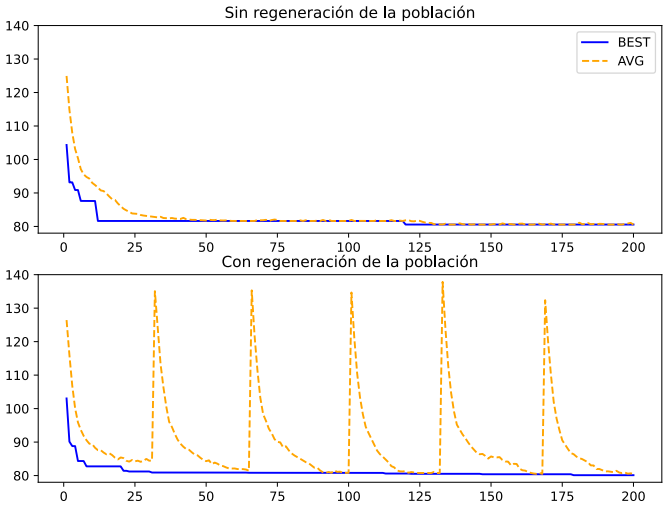

Figura 4. Comparativa de la evolución de los valores de la población sin regeneración y con regeneración para la instancia $n = 100, p = 15$.

Parámetro	*pobR*	*pobG*	α	θ	k	λ_C
Valor	25	25	0,33	$0{,}75 \cdot D$	n	60 %
Parámetro	λ_M	μ	λ_{LS}	β	ϵ	t
Valor	10 %	25 %	6,25 %	0,01	5 %	30

Tabla I
Valores de los parámetros para los experimentos

IV-B. Comparativa con el estado del arte

En esta sección se comparan los resultados obtenidos por la propuesta memética con regeneración frente al estado del arte, que se denotará como Kalczynski [8].

La ejecución del algoritmo depende de 13 parámetros cuyos valores se han establecido tras un conjunto de experimentos preliminares, teniendo en cuenta que los valores de λ_C, λ_M y λ_{LS} han sido establecidos siguiendo los trabajos [14], [15]. La Tabla I muestra los valores para el conjunto de parámetros.

Para realizar la comparativa, se han ejecutado dos experimentos, diferenciados únicamente en el número de generaciones ejecutadas por cada uno: 200 y 1000. Para cada experimento mostrará el valor promedio de la función objetivo (*F.O.*), el tiempo de ejecución en segundos (*T. CPU(s)*), la desviación promedio sobre el valor de la mejor solución encontrada (*Desv.*) y el número de instancias en que se ha obtenido la mejor solución de entre las 21 estudiadas (*#Mejores*). Para evitar el sesgo aleatorio, el algoritmo memético se ha ejecutado 30 veces.

En la Tabla II se muestra el resultado del algoritmo para 200 generaciones. El promedio de valores de la función objetivo obtenido queda un 0,33 % por encima del estado del arte, que también obtiene el mejor resultado para desviación y número

	F.O.	*T. CPU (s)*	*Desv. (%)*	*#Mejores*
Kalczynski [8]	**446,516**	231,2	**0,03**	**19**
Memético (200)	449,874	**74,0**	0,33	2

Tabla II
RESULTADOS DEL ALGORITMO MEMÉTICO EN 200 GENERACIONES

	F.O.	*T. CPU (s)*	*Desv. (%)*	*#Mejores*
Kalczynski [8]	446,516	**231,2**	**0,059**	**14**
Memético (1000)	**446,439**	596,1	0,088	7

Tabla III
RESULTADOS DEL ALGORITMO MEMÉTICO EN 1000 GENERACIONES

de instancias con mejor valor. Sin embargo, el algoritmo memético completa la ejecución de las 200 generaciones en un tiempo promedio de 74 segundos, 3 veces más rápido que el previo. Dado que la regeneración de soluciones permite una mejora a lo largo de toda la ejecución, se ha incrementado el número de generaciones como primera medida para intentar mejorar los resultados.

En la Tabla III se muestran los resultados la ejecución del algoritmo durante 1000 generaciones. En este caso se obtiene una mejora del $0{,}03\,\%$ en el promedio de la función objetivo, con una desviación del $0{,}088\,\%$. Además se obtiene el mejor resultado en 7 de las 21 instancias bajo estudio. Sin embargo el algoritmo emplea 596 segundos en promedio para completar su ejecución y el algoritmo del estado del arte sigue obteniendo el mejor resultado en 14 instancias, lo que lleva a la necesidad de mejorar estos resultados preliminares.

V. CONCLUSIONES Y TRABAJO FUTURO

El problema OPPMVS es un problema de localización de instalaciones desagradables continuo que destaca por la variabilidad del espacio de búsqueda sujeta a las restricciones del problema. En este trabajo se propone una primera versión de un algoritmo memético, que si bien no mejora los resultados del estado del arte, obtiene resultados muy prometedores.

Como trabajos futuros se plantea avanzar en la selección de los valores de los parámetros del algoritmo. Para su funcionamiento es necesario establecer valores para un total de 13 parámetros, lo que implica una combinatoria demasiado alta para ser comprobada manualmente. Las técnicas automáticas y estadísticas de ajuste de parámetros como `irace` pueden mejorar el desempeño del algoritmo.

Por otro lado, pese a implementar una técnica de regeneración de la población, se sigue observando una cierta rapidez en la convergencia del algoritmo. Un análisis más exhaustivo sobre los métodos de mutación y regeneración de la población pueden aportar mejoras en los resultados obtenidos.

La unión de estas propuestas, junto con la revisión de la complejidad algorítmica de la implementación realizada, conducirá, sin duda, a una mejora en los resultados.

REFERENCIAS

[1] M. Davoodi y J. Rezaei, "Bi-sided facility location problems: an efficient algorithm for k-centre, k-median, and travelling salesman problems", *International Journal of Systems Science: Operations & Logistics*, vol. 10, n.º 1, pág. 2 235 814, 2023.

[2] K. Liu, C. Liu, X. Xiang y Z. Tian, "Testing facility location and dynamic capacity planning for pandemics with demand uncertainty", *European journal of operational research*, vol. 304, n.º 1, págs. 150-168, 2023.

[3] R. L. Church y Z. Drezner, "Review of obnoxious facilities location problems", *Computers & Operations Research*, vol. 138, pág. 105 468, 2022.

[4] J. M. Colmenar, P. Greistorfer, R. Martí y A. Duarte, "Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem", *European Journal of Operational Research*, vol. 252, n.º 2, págs. 432-442, 2016.

[5] O. Berman, Z. Drezner, J. Wang y G. O. Wesolowsky, "The minimax and maximin location problems on a network with uniform distributed weights", *IIE Transactions*, vol. 35, n.º 11, págs. 1017-1025, 2003.

[6] P. Kalczynski y Z. Drezner, "Extremely non-convex optimization problems: The case of the multiple obnoxious facilities location", *Optimization Letters*, págs. 1-14, 2022.

[7] T. Drezner, Z. Drezner y P. Kalczynski, "Multiple obnoxious facilities location: A cooperative model", *IISE Transactions*, vol. 52, n.º 12, págs. 1403-1412, 2020.

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

[9] M. Marić, Z. Stanimirović, A. Djenić y P. Stanojević, "Memetic algorithm for solving the multilevel uncapacitated facility location problem", *Informatica*, vol. 25, n.º 3, págs. 439-466, 2014.

[10] J. M. Colmenar, R. Martí y A. Duarte, "Multi-objective memetic optimization for the bi-objective obnoxious p-median problem", *Knowledge-based systems*, vol. 144, págs. 88-101, 2018.

[11] T. A. Feo y M. G. Resende, "Greedy randomized adaptive search procedures", *Journal of global optimization*, vol. 6, págs. 109-133, 1995.

[12] F. Neri, C. Cotta y P. Moscato, *Handbook of memetic algorithms*. Springer, 2011, vol. 379.

[13] B. L. Miller, D. E. Goldberg et al., "Genetic algorithms, tournament selection, and the effects of noise", *Complex systems*, vol. 9, n.º 3, págs. 193-212, 1995.

[14] J. Santamaría, O. Cordón, S. Damas, J. García-Torres y A. Quirin, "Performance evaluation of memetic approaches in 3D reconstruction of forensic objects", en, *Soft Computing*, vol. 13, n.º 8-9, págs. 883-904, jul. de 2009.

[15] M. Chica, Ó. Cordón, S. Damas y J. Bautista, "Multiobjective memetic algorithms for time and space assembly line balancing", en, *Engineering Applications of Artificial Intelligence*, vol. 25, n.º 2, págs. 254-273, mar. de 2012.