TRABAJO FIN DE GRADO

CURSO ACADÉMICO 2021/2022

MEJORA DE CONEXIONES EN PROBLEMAS DE LOCALIZACIÓN: UN ALGORITMO HEURÍSTICO

UNIVERSIDAD MIGUEL HERNÁNDEZ

FACULTAD DE CIENCIAS SOCIALES Y JURÍDICAS DE ELCHE

Grado en Estadística Empresarial

Alumna: Lorena Nácher Sarrió

Tutoras: Mercedes Landete Ruiz y Marina Leal Palazón

Índice general

Capítulo 1

Introducción

Según la RAE, optimizar es "buscar la mejor manera de realizar una actividad". En otras palabras, es la capacidad de realizar una acción de manera eficiente utilizando una cantidad de recursos mínima. Esta idea se ha empleado dentro del campo de las matemáticas dando lugar al concepto de optimización matemática. La optimización matemática, también llamada investigación operativa, trata de buscar el valor que deben de tomar las variables de decisión para que la función objetivo sea óptima al mismo tiempo que se satisfacen todas las restricciones ((15), (13), (9)).

Dentro de la optimización matemática encontramos principalmente dos tipos: La optimización clásica y la heurística. El primer tipo garantiza el óptimo exacto y permite un gran número de restricciones. En el caso de los heurísticos, imitan fenómenos observados en la naturaleza y no garantizan encontrar el óptimo pero sí encuentran soluciones cercanas, pues analizan un gran número de soluciones en muy poco tiempo. Estos se utilizan para encontrar soluciones aproximadas cuando, por ejemplo, no existe un método exacto para resolver un problema, cuando sí existe un método exacto pero es computacionalmente muy costoso o, cuando se necesita incorporar al modelo condiciones muy laboriosas ((10), (11)).

Algunas de las aplicaciones de la investigación operativa son en problemas de asignación y distribución de recursos, de inventarios, de mantenimiento, reemplazamiento y fiabilidad de los equipos industriales, así como en problemas de localización (14).

Diremos que se trata de un problema de localización cuando un conjunto de clientes,

distribuidos en un área concreta, requieren de un servicio o producto. Para ello, es necesario decidir dónde se van a abrir los centros que permitirán abastecer toda la demanda de manera óptima (1).

Dentro de los problemas de localización distinguimos el problema de la p-mediana y el del p-centro. De manera general, el problema de la p-mediana trata minimizar la suma total de las distancias y, el del p-centro, de minimizar la máxima distancia entre un centro y el cliente asignado (6). También distinguimos entre problemas discretos y contínuos, siendo en este caso el espacio de ubicaciones candidatas para los centros un conjunto discreto o continuo respectivamente.

Durante la primera parte de este trabajo vamos a resolver un problema concreto de localización de p-centros discretos con mejoras mediante un modelo exacto y compararemos los resultados al resolverlo también como un problema de p-mediana.

Para la segunda parte, hemos creado un algoritmo heurístico genético para resolver el mismo problema de localización de p-centros con mejoras. Tras el diseño del algortimo hemos comprobado la efectividad y rapidez del mismo.

Para el desarrollo de este estudio han sido de gran ayuda algunas de las asignaturas impartidas en el grado de Estadística Empresarial. Las más relevantes han sido aquellas relacionadas con la optimización. Algunas de ellas han sido "Modelos de optimización"de tercer curso y "Gestión y planificación de la producción"de cuarto curso. Por otra parte, en la asignatura "Gestión de carteras e inversiones"de cuarto curso se han visto algunas pinceladas de qué es un algoritmo heurístico y de su uso. Además, el hecho de utilizar a lo largo de todo el grado el software *Rstudio* para realizar numerosas prácticas, me ha ayudado a tener mayor soltura a la hora de llevar a cabo la programación.

Por último, cabe destacar lo novedoso del estudio. A lo largo del grado sí se han visto algunos problemas de optimización pero nunca el problema de localización como tal ni el problema del p-centro que se incluye dentro de este. Del mismo modo, se dan algunas nociones del algoritmo heurístico pero nunca había procedido a su implementación de manera individual. Por todo ello, el estudio lleva un gran trabajo de investigación que me ha permitido conocer más sobre el tema y ser capaz de llevarlo a cabo.

Capítulo 2

Modelos exactos para el problema del p-centro de localización con mejoras

Como ya hemos mencionado, dentro de los problemas de localización encontramos el problema del p-centro. Este trata de minimizar la máxima distancia entre un centro y el cliente asignado. Se considera muy apropiado dentro de sistemas de emergencia (7).

El problema del p-centro fue introducido por Hakimi(1964). Fue generalizado a partir del centro absoluto creado para ubicar una comisaría u hospital de manera que la máxima distancia entre la comisaría/hospital y un conjunto de comunidades fuese mínima (4). Como bien se ha comentado, para resolver este problema se debe decidir dónde se abrirán los centros y, para ello, se definen un número finito de puntos que estarán situados dentro de un grafo. Cada punto se unirá a su centro más cercano mediante una conexión de tal manera que su distancia máxima sea mínima (3).

Por otra parte, el hecho de realizar mejoras puede incrementar la calidad del servicio. Por ejemplo, en el ámbito de los sistemas de trasporte, una mejora podría ser reparar el asfalto de la carretera que conecta un centro y un cliente. El objetivo de las mejoras es reducir el coste que una conexión pueda suponer y, para ello, se debe de tener en cuenta el presupuesto y qué conexiones elegir para ser mejoradas ((3), (5)).

Aunque existen muchas variantes del problema del p-centro, durante todo el trabajo hablaremos del problema del p-centro restringido por vértices y no ponderado. Esto permite que pueda ser formulado como un problema de programación entera. Además, consideramos que los costes son sabidos de antemano y que no cambian.

2.1. El problema de localización de p-centros discretos con conexiones mejoradas (3)

En este tipo de problema, como se mide la distancia entre cada cliente y el punto de servicio más cercano, se trata de determinar dónde se situarán los p-centros, de manera que el cliente que esté situado más lejos esté lo más cerca posible. Se minimiza la mayor de las distancias.

En primer lugar, se muestra cómo sería la resolución del poblema del p-centro discreto sin realizar actualizaciones en las conexiones. Para ello, se define:

I: Conjunto de posibles localizaciones de las instalaciones.

J: Conjunto de los puntos de demanda.

Además, se define el coste de asignar el nodo de demanda $j \in J$ al centro $i \in I$ como c_{ij}. En este caso, el coste corresponde a la distancia euclídea entre nodos. Por otro lado, se consideran las siguientes variables binarias:

$$y_i = \begin{cases} 1, & \text{si se selecciona el nodo } i \text{ para que sea un centro, } \forall i \in I, \\ 0, & \text{en caso contrario.} \end{cases}$$

$$x_{ij} = \begin{cases} 1, & \text{si el nodo de demanda } j \text{ es asignado al centro } i, \forall j \in J, \forall i \in I, \\ 0, & \text{en caso contrario.} \end{cases}$$

De tal modo, se plantea el siguiente problema:

$$(P_1) \quad Min \quad z \tag{2.1}$$

$$s.a \quad \sum_{i \in I} c_{ij} x_{ij} \leq z \qquad \forall j \in J \tag{2.2}$$

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{2.3}$$

$$x_{ij} \leq y_i \qquad \forall i \in I, \quad \forall j \in J \tag{2.4}$$

$$\sum_{i \in I} y_i \leq p \tag{2.5}$$

$$y_i \in \{0, 1\} \qquad \forall i \in I \tag{2.6}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in I, \quad \forall j \in J \tag{2.7}$$

En el modelo P_1, (2.1) es la función objetivo. Cabe destacar que para la función objetivo utilizamos la variable real z cuyo dominio son todos los reales, $z \in \mathbb{R}$. (2.2) corresponde al máximo coste; (2.3) garantiza que se cubra toda la demanda del cliente y que se asigne a un único punto de servicio; (2.4) se asegura de que solo se pueda asignar a los centros seleccionados; (2.5) indica que hay un número máximo de instalaciones a seleccionar y, (2.6) (2.7) establecen el dominio binario de las variables.

Otra manera de resolver este mismo problema es ordenar los costes de manera creciente obviando los empates. El resultado de ordenar los distintos valores del coste es $\gamma_1, \gamma_2, ..., \gamma_\kappa$. Por ello, se define $K = \{1, ..., k, ..., \kappa\}$. Como los distintos valores de γ son los distintos valores del coste y la matriz de costes no es una matriz cuadrada, sabemos que κ como mucho es $|I| \times |J|/2$.

Para $i \in I, j \in J, k \in K$ se define el parámetro a_{ijk} que será 1 si $c_{ij} \leq \gamma_k$ y 0 en caso contrario. Ahora, además de la variable y_i anteriormente mencionada, se define otro conjunto de variables binarias:

$$z_k = \begin{cases} 1, & \text{si y solo si el coste máximo de asignación es } \gamma_k, \forall k \in K, \\ 0, & \text{en caso contrario.} \end{cases}$$

De tal forma, el problema se plantea ahora de la siguiente manera:

$$(P_2) \quad Min \quad \sum_{k \in K} \gamma_k z_k \tag{2.8}$$

$$s.a \quad (2.5), \quad (2.6)$$

$$\sum_{i \in I} a_{ijk} y_i \geq z_k \qquad \forall j \in J, \quad \forall k \in K \tag{2.9}$$

$$\sum_{k \in K} z_k = 1 \tag{2.10}$$

$$z_k \in \{0, 1\} \qquad \forall k \in K \tag{2.11}$$

La función objetivo y la restricción (2.9) asegura que se seleccione la variable z_k correspondiente al coste máximo de asignación y (2.10) asegura que solo exista un único coste máximo.

Seguidamente, se considera un número máximo de t conexiones para actualizar. Después de mejorar una conexión el coste se reduce de acuerdo a un factor f. Para resolver este problema se debe de actualizar el modelo P_1 y, para conseguirlo, se deben de añadir nuevas variables binarias.

$$m_{ij} = \begin{cases} 1, & \text{si se mejora la conexión entre el nodo } j \text{ y el centro } i, \forall i \in I, \forall j \in J, \\ 0, & \text{en caso contrario.} \end{cases}$$

El modelo P_1 actualizado queda de la siguiente manera:

$$(M_1) \quad Min \quad z \tag{2.12}$$

$$s.a \quad \sum_{i \in I} (c_{ij} x_{ij} - f c_{ij} m_{ij}) \leq z \qquad \forall j \in J \tag{2.13}$$

$$m_{ij} \leq x_{ij} \qquad \forall i \in I, \quad \forall j \in J \tag{2.14}$$

$$\sum_{i \in I} \sum_{j \in J} m_{ij} \leq t \tag{2.15}$$

$$(2.3) \quad - \quad (2.7)$$

$$m_{ij} \in \{0, 1\} \qquad \forall i \in I, \quad \forall j \in J \tag{2.16}$$

(2.13) adapta el radio máximo a la mejora y (2.14) garantiza que si se actualiza una conexión entonces se asigna.

De la misma manera se puede resolver el problema actualizando el modelo P_2. Ahora, para ordenar los costes de manera creciente ignorando empates, se deben de calcular tanto los costes con mejora, $(1-f)c_{ij}$, como los sin , c_{ij}. El resultado de ordenar todos los costes es $\hat{\gamma}_1, \hat{\gamma}_2, ..., \hat{\gamma}_\kappa$ y se define $\hat{K} = \{1, ..., \hat{\kappa}\}$.

Se deben definir los siguientes parámetros:

$$\hat{a}_{ijk} = \begin{cases} 1, & \text{si} \quad c_{ij} \leq \hat{\gamma}_k, \quad \forall i \in I, \forall j \in J, \forall k \in \hat{K}, \\ 0, & \text{en caso contrario.} \end{cases}$$

$$\hat{b}_{ijk} = \begin{cases} 1, & \text{si} \quad (1-f)c_{ij} \leq \hat{\gamma}_k, \quad \forall i \in I, \forall j \in J, \forall k \in \hat{K}, \\ 0, & \text{en caso contrario.} \end{cases}$$

Y la siguiente variable binaria:

$$s_j = \begin{cases} 1, & \text{si el nodo de demanda } j \text{ hace uso de una conexión mejorada, } \forall j \in J, \\ 0, & \text{en caso contrario.} \end{cases}$$

El modelo P_2 actualizado queda de la siguiente manera:

$$(M_2) \quad Min \quad \sum_{k\in\hat{K}} \hat{\gamma}_k z_k \tag{2.17}$$

$$s.a \quad \sum_{i\in I} \hat{a}_{ijk} y_i + \sum_{i\in I} \hat{b}_{ijk} y_i \geq z_k \qquad \forall j \in J, \quad \forall k \in \hat{K} \tag{2.18}$$

$$\sum_{i\in I} \hat{a}_{ijk} y_i + s_j \geq z_k \qquad \forall j \in J, \quad \forall k \in \hat{K} \tag{2.19}$$

$$\sum_{j\in J} s_j \leq t \tag{2.20}$$

$$\sum_{k\in\hat{K}} z_k = 1 \tag{2.21}$$

$$(2.5), \quad (2.6)$$

$$z_k \in \{0,1\} \qquad \forall k \in \hat{K} \tag{2.22}$$

$$s_j \in \{0,1\} \qquad \forall j \in J \tag{2.23}$$

(2.18) y (2.21) indica el coste máximo y (2.19) verifica las actualizaciones exactas utilizadas.

2.1.1. Ejemplos

Es importante ver gráficamente la diferencia entre los problemas de p-centro y p-mediana. Por ello, se han resuelto los modelos P_1 y M_1 tanto para el problema del p-centro como para el de la p-mediana.

El ejemplo resuelto consta de 10 nodos. Se tiene que $I = \{1, 2, ..., 10\}$ y $J = \{1, 2, ..., 10\}$ y el número de centros es $p = 3$.

En la *Figura* 2.1 podemos ver que para el problema del p-centro los centros son el 7, 8 y 2 y la máxima distancia es 3,486. Si calculamos la máxima distancia en el problema p-mediana vemos que es 4,8989. Luego, el problema del p-centro sí minimiza la mayor de las distancias.

Por otra parte, en la *Figura* 2.2 podemos ver que para el problema de la p-mediana, a diferencia del p-centro, los centros son el 4, 5 y 2 y, como el objetivo es minimizar la suma de todas las distancias, si calculamos la suma de todas las distancias en el problema p-centro esta es de 18,8431 y después de resolver el problema de la p-mediana esta suma

es de 12,306.

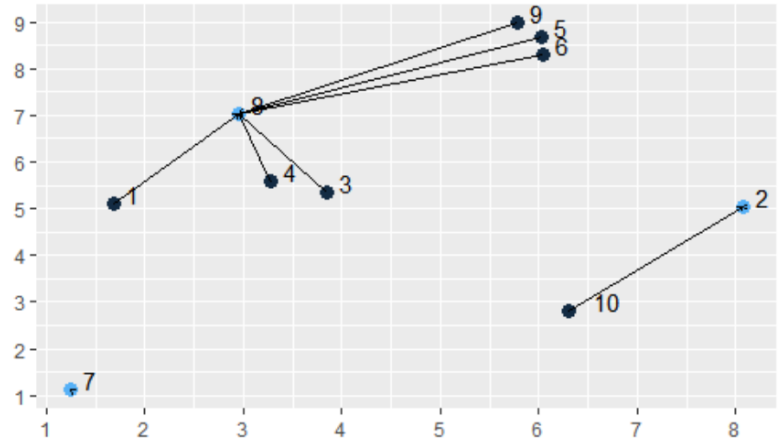

Figura 2.1: Problema p-centro del modelo P_1

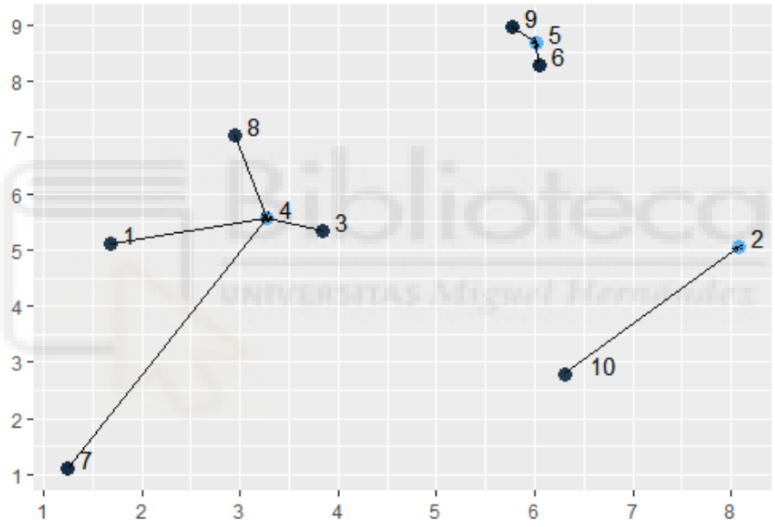

Figura 2.2: Problema p-mediana del modelo P_1

Resolvemos ahora el modelo M_1 con $t = 4$ conexiones mejoradas, con el factor de mejora $f = 0{,}25$ y con los mismos datos anteriores.

Si añadimos las conexiones mejoradas, la máxima distancia sigue siendo 4,8989 pero después de resolver el problema del p-centro con mejoras es de 2,614. Además, en la *Figura* 2.3, podemos ver que los centros son el 7, 8 y 2 y las conexiones mejoradas son la 9, 5, 6 y 10.

Aunque la suma de todas las distancias sigue siendo 18,8431, después de resolver el problema de la p-mediana con mejoras esta suma es de 9,576. En la *Figura* 2.4 podemos

ver que los centros son el 4, 5 y 2 y las conexiones mejoradas son la 8, 1, 7 y 10.

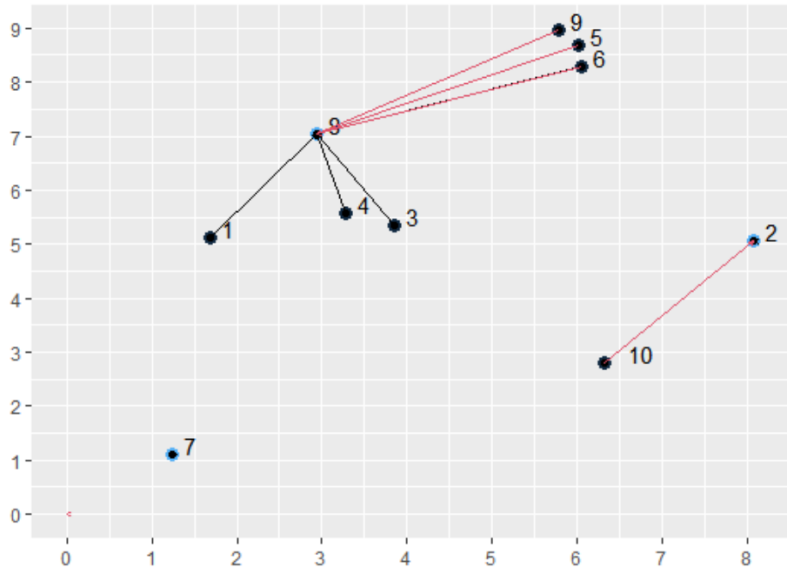

Figura 2.3: Problema p-centro del modelo M_1

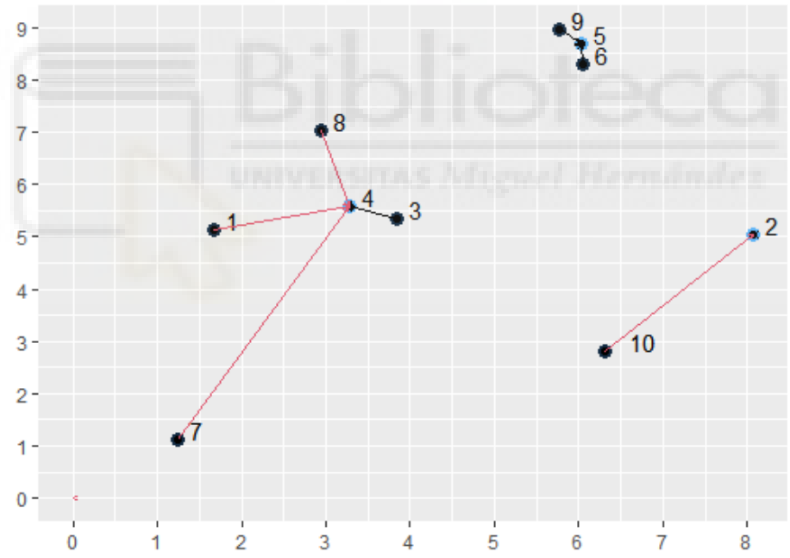

Figura 2.4: Problema p-mediana del modelo M_1

Con todo ello, comprobamos que al añadir mejoras el valor objetivo disminuye en ambos problemas dando lugar a una mejor solución.

Por otro lado, tanto la resolución del problema del p-centro, P_1, como la del problema del p-centro con mejoras, M_1, se pueden llevar a cabo de manera análoga mediante los problemas P_2 y M_2, respectivamente, y cuyo código se encuentra en la sección 4.1.2 y 4.1.4. Algo relevante en estos problemas son los distintos valores de γ que encontramos. Para

el problema P_2 existen 46 valores diferentes y para el M_2, como añadimos las conexiones mejoradas, existen 91 valores diferentes.

Capítulo 3

Algoritmo heurístico para el problema del p-centro de localización con mejoras

En optimización matemática, los algoritmos heurísticos son aquellos que no aseguran encontrar una solución óptima pero sí una solución muy cercana a una óptima y con un coste computacional reducido (16). Es difícil clasificar los métodos heurísticos que existen, pero una clasificación general puede ser la siguiente (11):

- *Métodos de Descomposición.* Hacen servir la misma estructura de los problemas para solucionar problemas de menor tamaño de manera secuencial.
- *Métodos Inductivos.* Pretenden extrapolar un problema pequeño a uno de dimensión mayor.
- *Métodos de Reducción.* Tratan de determinar propiedades que cumplen las buenas soluciones para introducirlas como restricciones del problema.
- *Métodos Constructivos.* Forman poco a poco una solución del problema y, en cada iteración, se quedan con la mejor solución.
- *Métodos de Búsqueda Local.* En este caso, identifican una solución del problema y la van mejorando hasta que no encuentran un valor mejor.

Dentro de los heurísticos situamos a los algoritmos metaheurísticos que tratan de mejorar a los algoritmos heurísticos. Estos se pueden clasificar en: *Métodos Constructivos, Métodos Evolutivos y Métodos de Búsqueda.*

En este estudio se ha creado un algoritmo genético, incluido dentro de los métodos evolutivos, para tratar de resolver el problema del p-centro de localización con mejoras. Como resultados del algoritmo, buscamos las soluciones del problema M_1 o, equivalentemente, del M_2. Un algoritmo genético es aquel que simula la evolución biológica de los seres vivos para resolver un problema de optimización (8).

A lo largo del tiempo, la selección natural, teoría de Charles Darwin (1859), ha evidenciado que solo los seres vivos más fuertes sobreviven. Para ello, se han ido produciendo cambios dentro de cada organismo provocando la adaptación a distintos ambientes y climas asegurando la supervivencia de cada especie. Los algoritmos genéticos tratan de imitar este comportamiento de tal forma que a cada individuo, que representa una solución del problema, se le asigna un valor de bondad y se van aplicando cambios, tales como cruce, mutación..., que aseguran la convergencia a una única solución del problema ((17), (2), (12)).

Este tipo de algoritmos suelen estar creados por una población con cromosomas donde cada cromosoma representa una solución. Además, los cromosomas están formados por genes y cada gen representa un valor para cada variable (12).

En nuestro caso, el agoritmo que hemos creado va a partir de una población inicial. En el momento que se ha evaluado cada uno de los individuos procedemos a realizar el cruce, la mutación y la búsqueda local, *Figura* 3.1. El número de iteraciones a realizar son 170, de las cuales el cruce se realizará en todas ellas, la mutación únicamente en el 10 % de las veces y la búsqueda local en el 50 %.

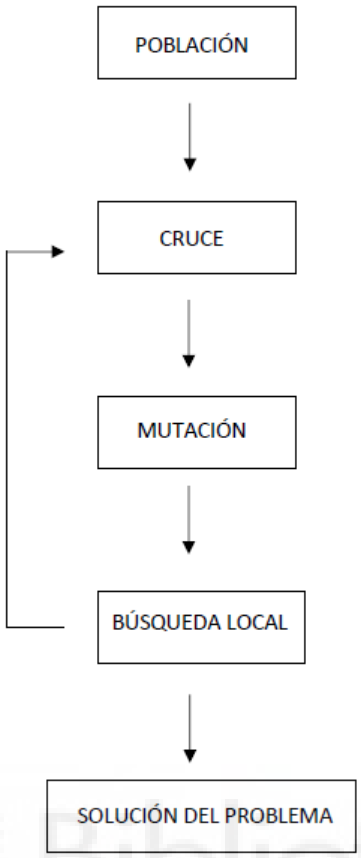

Figura 3.1: Esquema del algoritmo genético

POBLACIÓN

Para crear la problación se precisa de un gen. No existe un gen único y la finalidad de este es que sea lo más sencillo y corto posible para que así el heurístico sea rápido. El gen puede tener toda la información que el programador considere oportuna, como los centros que se abren, las conexiones que se mejoran, a qué centro va cada cliente y si usa o no una conexión mejorada. En nuestro caso, el gen tendrá una longitud de 3 e indicará los centros que se abren. Además, como la finalidad del estudio es minimizar la mayor de las distancias, necesitamos saber la distancia máxima y, para calcularla, debemos tener en cuenta las conexiones mejoradas. Para entender mejor cómo se ha realizado lo veremos con un ejemplo:

Imaginamos que nuestro gen es $(5, 8, 10)$, es decir los centros que se abren son el 5, 8 y 10. Sabemos que cada cliente va a su centro más cercano y, gracias a la matriz de distancias, conocemos a qué centro va cada cliente y su distancia. Ver *Cuadro* 3.1.

Cuadro 3.1: Plantas a las que va cada cliente

CLIENTE	PLANTAS	DISTANCIA
1	8	2,3
2	10	2,9
3	8	1,9
4	8	1,5
5	5	0
6	5	0,38
7	10	5,3
8	8	0
9	5	0,4
10	10	0

Ahora, ordenamos el *Cuadro* 3.1 de mayor a menor distancia, pues las conexiones que se vayan a actualizar serán aquellas que tengan una distancia mayor. Ver *Cuadro* 3.2.

Cuadro 3.2: Plantas a las que va cada cliente ordenadas por distancia

CLIENTE	PLANTAS	DISTANCIA
7	10	5,3
2	10	2,9
1	8	2,3
3	8	1,9
4	8	1,5
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

Si suponemos que solo se pueden mejorar 3 conexiones ($t = 3$), siempre se mejorará el valor más alto. Si el factor de mejora es $f = 0{,}25$, multiplicamos $(1 - f) \cdot distancia$ y volvemos a ordenar. El criterio de parada será que una conexión ya esté actualizada o que

se haya llegado al número máximo de conexiones a mejorar.

Es decir, en el *Cuadro* 3.2, multiplicamos $(1 - f)$ por 5,3 que es igual a 3,9. Después de ordenar los valores de nuevo, vemos que sigue siendo el valor más alto y como una conexión mejorada ya no se puede volver a mejorar, paramos. De tal modo, la distancia máxima será 3,9. Ver *Cuadro* 3.3.

Cuadro 3.3: Plantas a las que va cada cliente después de aplicar la mejora

CLIENTE	PLANTAS	DISTANCIA
7	10	3,9
2	10	2,9
1	8	2,3
3	8	1,9
4	8	1,5
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

Si ahora el factor de mejora es $f = 0{,}5$, multiplicamos la primera posición del *Cuadro* 3.2 por $(1 - f) \cdot 5{,}3$. El resultado es 2,6 y pasaría a estar en la segunda posición de la tabla como se muestra en el *Cuadro* 3.4.

Cuadro 3.4: Plantas a las que va cada cliente después de aplicar la mejora

CLIENTE	PLANTAS	DISTANCIA
2	10	2,9
7	10	2,6
1	8	2,3
3	8	1,9
4	8	1,5
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

Como el número de mejoras que se han realizado es menor a 3 y la distancia más grande aún no se ha mejorado podemos continuar. Ahora, multiplicamos $(1-f)\cdot 2{,}9 = 1{,}5$ y volvemos a ordenar.

Cuadro 3.5: Plantas a las que va cada cliente después de aplicar la mejora

CLIENTE	PLANTAS	DISTANCIA
7	10	2,6
1	8	2,3
3	8	1,9
2	10	1,5
4	8	1,5
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

Si nos fijamos en el *Cuadro* 3.5 vemos que la máxima distancia proviene de una conexión que ya está mejorada. Paramos. En este caso, la máxima distancia es de 2,6.

Por último y volviendo al *Cuadro* 3.2, si ahora el factor de mejora es de 0,8, multiplicamos $(1-f)\cdot 5{,}3 = 1{,}06$ y ordenamos. Ver *Cuadro* 3.6.

Cuadro 3.6: Plantas a las que va cada cliente después de aplicar la mejora

CLIENTE	PLANTAS	DISTANCIA
2	10	2,9
1	8	2,3
3	8	1,9
4	8	1,5
7	10	1,06
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

Como aun no se ha cumplido algún criterio de parada, seguimos multiplicando $(1-f)\cdot 2{,}9 = 0{,}58$ y ordenamos. Como podemos seguir mejorando realizamos la operación $(1-f)\cdot 2{,}3 = 0{,}46$ y, después de ordenar, vemos que ya se han mejorado las 3 conexiones. Paramos y la máxima distancia es 1,9. Observamos el resultado en el *Cuadro* 3.7.

Cuadro 3.7: Plantas a las que va cada cliente después de aplicar la mejora

CLIENTE	PLANTAS	DISTANCIA
3	8	1,9
4	8	1,5
7	10	1,06
2	10	0,58
1	8	0,46
9	5	0,4
6	5	0,38
5	5	0
8	8	0
10	10	0

CRUCE

Para realizar el cruce se han escogido dos genes al azar y se han unido. De este nuevo vector de dimensión $2p$ se escogen al azar p, pero debemos de tener en cuenta que puede tener plantas repetidas y estas tendrán mayor probabilidad de ser seleccionadas. Una vez se tiene el gen hijo, se calcula la máxima distancia. Finalmente, si la máxima distancia del hijo es menor que el peor de la población, entonces el hijo pasa a formar parte de dicha población reemplazando al peor.

Por ejemplo, elegimos al azar los genes $(8, 2, 10)$ y $(8, 2, 7)$. Los unimos quedando el vector $(8, 2, 10, 8, 2, 7)$ y, como hay valores repetidos, tendremos que calcular la probabilidad de cada centro de ser elegido. Ver *Cuadro* 3.8.

Cuadro 3.8: Probabilidad de cada centro de ser elegido

CENTROS	PROBABILIDAD DE SER ELEGIDO
8	33,33 %
2	33,33 %
10	16,67 %
7	16,67 %

Una vez sabemos la probabilidad de cada centro de ser elegido, escogemos 6 centros al azar formando 2 hijos. Un hijo podría ser $(7, 2, 8)$ y otro $(10, 7, 2)$. Por último, se calcula la máxima distancia de cada uno y, si dicha distancia es menor que el peor de la población, el gen hijo se incorpora a ella reemplazando al peor.

Por otro lado, si los genes elegidos para realizar el cruce son $(5, 6, 8)$ y $(4, 7, 10)$, al unirlos, $(5, 6, 8, 4, 7, 10)$, como no se repite ningún elemento, no es necesario calcular la probabilidad de aparición y se escogen 6 centros al azar formando 2 hijos. Un hijo podría ser $(6, 4, 10)$ y otro $(7, 8, 6)$. Y, del mismo modo, si la distancia de cada hijo es menor que el peor de la población , el gen hijo se incorpora a ella reemplazando al peor.

MUTACIÓN

Se realiza la mutación el 10 % de las veces. En general, en la mutación se trata de escoger un gen al azar y cambiar un elemento, también al azar, por otro y calcular la máxima distancia. En este caso, no importa si el resultado es mejor o peor, pues el gen

mutado se incorporará de todos modos a la población reemplazando al peor de esta.

En cuanto queremos cambiar un elemento del gen escogido por otro valor, debemos de tener en cuenta que no puede ser un valor repetido. Por ejemplo, el gen escogido es el $(1, 4, 5)$ y, al azar, escogemos cambiar el elemento 2 (el valor 4). Se le asigna otro valor, también al azar, pero no puede ser ni el valor 1 ni el 5. Una vez tenemos el gen mutado se calcula la máxima distancia y se incorpora a la población reemplazando al peor de esta.

BÚSQUEDA LOCAL

La búsqueda local se realiza el 50 % de las veces. En este caso, se trata de elegir un gen al azar y un elemento de dicho gen también al azar. Una vez ya se sabe qué elemento cambiar, se crean tantos aleatorios en un vector como cambios se quieren realizar y se calcula la máxima distancia de cada uno. Después de todos los cambios, el que tenga la menor distancia máxima se incorporará a la población reemplazando al peor.

Por ejemplo, el gen al azar es el $(8, 7, 1)$ y la posición elegida, también al azar, es la 3. Es decir, el centro 1 lo vamos a cambiar por 5 aleatorios distintos que no sea ni el 8 ni el 7. En el caso de salir uno de estos dos valores, no se calculará la máxima distancia y pasaremos al siguiente aleatorio. El vector de aleatorios es $(5, 1, 4, 2, 8)$. Ahora calculamos la máxima distancia del $(8, 7, 5)$, $(8, 7, 1)$, $(8, 7, 4)$ y $(8, 7, 2)$.

Cuadro 3.9: Cálculo de la máxima distancia de cada gen

GEN MUTADO	DISTANCIA
$(8, 7, 5)$	4,001834
$(8, 7, 1)$	4,12534
$(8, 7, 4)$	3,619621
$(8, 7, 2)$	2,614565

En el *Cuadro* 3.9 vemos que el gen mutado con menor máxima distancia es el $(8, 7, 2)$, luego este se incorporará a la población y reemplazará al peor de esta.

3.1. Parámetros del heurístico

Existen parámetros dentro del heurístico que se pueden modificar y analizar si el resultado mejora al realizar variaciones. En el *Cuadro* 3.10 se muestra el valor que hemos dado a cada uno de estos parámetros para realizar nuestro algoritmo. Cabe destacar que tanto el número de clientes como el factor de mejora son datos del problema.

Cuadro 3.10: Parámetros del heurístico

PARÁMETROS	VALOR
Tamaño de la población	10
Iteraciones	170
Probabilidad de mutación	0,1
Probabilidad de búsqueda local	0,5
Número de cambios en la búsqueda local	5

3.2. Estudio computacional

Después de resolver el problema tanto de manera exacta como mediante un heurístico, comparamos resultados modificando la semilla utilizada y el tamaño de la población. De tal forma, para el estudio se han considerado 4 tamaños de población y 20 semillas diferentes.

Para poder realizar las comparaciones se ha utilizado para todos los casos el mismo número de iteraciones (170).

En el *Cuadro* 3.11, la primera columna indica el tamaño de la población. Las dos columnas siguientes muestran tanto el valor óptimo como el tiempo de ejecución para el modelo exacto. La cuarta y la quinta columna muestran el valor óptimo y el tiempo de ejecución para el heurístico y, la última columna hace referencia a la diferencia entre la solución del heurístico y la del modelo exacto dividido entre el óptimo del modelo exacto y todo ello multiplicado por 100.

Cuadro 3.11: Resolución mediante el problema exacto y el heurístico

TAMAÑO	SOLUCIÓN ÓPTIMA - M. EXACTO		SOLUCIÓN ÓPTIMA - HEURÍSTICO		
	VALOR	TIEMPO	VALOR	TIEMPO	GAP
10	2,9606	0,2822 seg	2,9606	0,7474 seg	0
10	3,2053	0,2994 seg	3,2053	0,8282 seg	0
10	2,6146	0,3369 seg	2,6146	0,884 seg	0
10	3,0305	0,2939 seg	3,0305	0,8953 seg	0
10	2,7344	0,3239 seg	2,7344	0,7736 seg	0
20	3,4513	3,7456 seg	3,4513	0,978 seg	0
20	3,3439	1,3019 seg	3,3439	1,0238 seg	0
20	3,0691	3,3954 seg	3,0691	0,9441 seg	0
20	3,0106	7,1992 seg	3,0106	0,8683 seg	0
20	2,4732	17,1441 seg	2,4732	0,8577 seg	0
40	3,3554	6,4746 min	3,3554	0,95 seg	0
40	3,1947	2,8658 min	3,2673	1,151 seg	2,2725
40	3,285	2,0305 min	3,285	0,9824 seg	0
40	3,0307	1,6399 min	3,2368	1,1048 seg	6,8004
40	3,0126	41,8873 seg	3,0126	0,9547 seg	0
43	3,4502	11,0282 min	3,4502	0,9163 seg	0
43	3,5536	12,1817 min	3,5787	1,037 seg	0,7063
43	-	+15 min	3,9058	1,0231 seg	-
43	-	+15 min	3,3079	1,0225 seg	-
43	-	+15 min	3,909	1,1014 seg	-

Después de realizar todas las comprobaciones nos damos cuenta de que el modelo exacto no es igual de efectivo si aumentamos el tamaño de la población que si empleamos el heurístico. Esto se debe a que dentro del modelo exacto, el tiempo computacional y el tamaño de la población están relacionados de manera directa, es decir, cuando el tamaño aumenta el tiempo también lo hace. En nuestro caso, hemos decidido que el tiempo de ejecución para la resolución del modelo exacto no exceda de los 15 min de compilación, por lo que el tamaño máximo con el que hemos podido resolver el problema es de 43. En

cambio, si resolvemos el problema mediante el heurístico vemos que el tiempo computacional siempre oscila entorno a 1 segundo independientemente del tamaño de la población y de la semilla utilizada.

Bajo estas evidencias podemos decir que el heurístico sí es efectivo, pues llega al óptimo en 14 de las 17 situaciones resueltas con el exacto, es decir, en el 82,35 % de las veces y, el coste computacional que consume es muy escaso.

Capítulo 4

Código R

En esta sección incluimos todo el código utilizado para resolver el problema inicial con 10 instalaciones únicamente. De manera que, $I = \{1, 2, ..., 10\}$ y $J = \{1, 2, ..., 10\}$. El número de centros es $p = 3$, el número de conexiones a actualizar es $t = 4$ y el factor de mejora es $f = 0{,}25$.

4.1. Modelo exacto

Cabe resaltar que el código empleado para la resolución de los problemas P_1 y P_2 está codificado para el problema específico de 10 puntos. En cambio, el código empleado para la resolución de los problemas M_1 y M_2 está codificado de manera general para cualquier número de puntos.

Por otra parte, los problemas P_1 y P_2 no incluyen mejoras, por lo que no existe el parámetro t.

4.1.1. Código para la resolución del problema P_1

```
#LAS LIBRERIAS UTILIZADAS SON
library(ggplot2)
library(philentropy)
library(lpSolve)
```

```
library(tidyverse)

# DEFINIMOS LOS 10 NODOS ALEATORIAMENTE Y LOS SITUAMOS EN UN
#GRAFICO
set.seed(3)
x<- runif(10, min=0, max=10);x
y<- runif(10, min=0, max=10);y


matriz<-matrix(c(x,y), byrow=FALSE,ncol=2)
datos<-data.frame(matriz)

graf<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(colour = 'blue', size = 1) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,10,1)) +
  scale_x_continuous(breaks = seq(0,10,1))

graf + geom_text(vjust = -0.005, hjust=-1, size=3)

# MATRIZ DE COSTES
coste<-  distance(matriz, method = "euclidean")

# MODELO
filas<- 121
columnas<- 111
p<-3
inst<-10

#vbles binarias
bin<-c(rep(0,100),rep(0,10));
for(i in 1:110) bin[i]<-1+i;


A<-matrix(0, nrow=filas, ncol=columnas)
```

```
signos<-c(rep('<=', filas ))
b<-c(rep(0,filas))

# Funci n objetivo
fun<-c(1,rep(0,100),rep(0,10))
#     z ,    x ,       y


#CODIGO PARA LA PRIMERA RESTRICCION (2.2)

k<-0
for (f in 1:10){
  for (c in 1:10){
    A[f,1+k*inst+c]<- coste[f,c]

  }
  k<-k+1
}


# -z
A[1:10,1]<-1


#CODIGO PARA LA SEGUNDA RESTRICCION (2.3)

k<-0
for (f in 1:10){
  for (c in 1:10){

      A[f+10,c+k*inst+1]<-1
      b[f+10]<-1
      signos[f+10]<-"="
```

```
  }
  k<-k+1
}


#CODIGO PARA LA TERCERA RESTRICCION (2.4)

w<-1
x<-0
z<-2
for (i in 1:10){
v<-0
for (f in 1:10){
  for (c in 1:10){

    A[f+20+x, z+v]<-1
    A[f+20+x,101+w]<- -1

  }
  v<-v+10
}
  x<-x+10
  z<-z+1
  w<-w+1
}


#CODIGO PARA LA CUARTA RESTRICCION (2.5)

for (c in 1:10){

  A[121,c+101]<- 1
}
```

```
b[121]<-p

#RESOLVEMOS EL PROBLEMA
solucion<-lp('min', fun, A, signos, b, binary.vec=bin)

solucion$objval

#DIBUJAMOS LOS CENTROS QUE SE ABREN

datos$solucion<-solucion$solution[102:111]

graf2<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(mapping=aes(x=X1, y=X2, color=solucion),
show.legend = FALSE,size = 3) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,10,1)) +
  scale_x_continuous(breaks = seq(0,10,1))

graf2 + geom_text(vjust = -0.005, hjust=-1, size=3)


#DIBUJAMOS LAS CONEXIONES
i_s<-c(1:10)
j_s<-c(1:10)

sol_x<-matrix(c(rep(i_s,10),sort(rep(j_s,10))), byrow=FALSE,
ncol=2)
sol_x<-data.frame(sol_x)
sol_x$solucion<-solucion$solution[2:101]



d<-filter(sol_x, solucion==1)

mk<-rep(0,10)
```

```
for (i in 1:nrow(d)){
  mk[i]<-d[i,1]
}

x2<-rep(0,10)
y2<-rep(0,10)
for (j in 1:length(mk)){
  x2[j]<-datos$X1[mk[j]]
  y2[j]<-datos$X2[mk[j]]
}

df<- data.frame(x1=round(c(datos$X1),2), y1=round(c(datos$X2),2),
                x2,
                y2)
graf2 + geom_text(vjust = -0.005, hjust=-1, size=4)+
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),data = df,
               arrow = arrow(length = unit(0.1,"cm")))
```

4.1.2. Código para la resolución del problema P_2

```
#LAS LIBRERIAS UTILIZADAS SON
library(ggplot2)
library(philentropy)
library(lpSolve)
library(tidyverse)


# DEFINIMOS LOS 10 NODOS ALEATORIAMENTE Y LOS SITUAMOS EN UN
#GRAFICO
set.seed(3)
x<- runif(10, min=0, max=10);x
y<- runif(10, min=0, max=10);y
```

```
matriz<-matrix(c(x,y), byrow=FALSE, ncol=2)
datos<-data.frame(matriz)

graf<-ggplot(data=datos, aes(x=X1, y=X2, label = rownames(datos)))+
geom_point(colour = 'blue', size = 1) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,10,1)) +
  scale_x_continuous(breaks = seq(0,10,1))

graf + geom_text(vjust = -0.005, hjust=-1, size=3)

# MATRIZ DE COSTES
coste<-  distance(matriz, method = "euclidean")


coste_2<- matrix(rep(0,100), byrow = TRUE, ncol = 10)

for (f in 1:10){
  for (c in 1:10){
    if (f<c){
      coste_2[f,c]<- coste[f,c]

    }
  }
}

coste_2<-as.vector(coste_2)
coste_2<-sort(coste_2) #ordenar de menor a mayor

coste_2<-unique(coste_2) #valores de gamma
length(coste_2)


# aijk
inst<-10
```

```
a<-array(0,dim = c(inst,inst,length(coste_2)))


for (k in 1:length(coste_2)){
  for (f in 1:10){
    for (c in 1:10){

      if (coste[f,c]<=coste_2[k]){

        a[f,c,k]<-1

      } else {
        a[f,c,k]<-0
      }
    }
  }
}

# MODELO
filas<- 2+inst*length(coste_2)
columnas<- length(coste_2)+10
p<-3

#vbles binarias
bin<-c(rep(0,length(coste_2)),rep(0,10));
for(i in 1:c(length(coste_2)+10)) bin[i]<-i;


A<-matrix(0, nrow=filas, ncol=columnas)
signos<-c(rep('>=', filas ))
b<-c(rep(0,filas))

#Funcion objetivo
```

```
fun<-c(coste_2,rep(0,10))

# Restricciones

#CODIGO PARA LA CUARTA RESTRICCION (2.5)

for (c in 1:10){

  A[1,c+length(coste_2)]<- 1
}

b[1]<-p
signos[1]<-"<="

#CODIGO PARA LA SEGUNDA RESTRICCION (2.9)

x<-0
for (k in 1:length(coste_2)){
  for (f in 1:10) {
    for (c in 1:10){
      A[1+f+x,length(coste_2)+c]<-a[f,c,k]
      A[1+f+x,k]<--1
    }
  }
 x<-x+10
}


#CODIGO PARA LA TERCERA RESTRICCION (2.10)

for (c in 1: length(coste_2)){

  A[inst*length(coste_2)+2,c]<-1
```

```
}

b[inst*length(coste_2)+2]<-1
signos[inst*length(coste_2)+2]<-"="



#RESOLVEMOS EL PROBLEMA
solucion<-lp('min', fun, A, signos, b, binary.vec=bin)

solucion$objval

#DIBUJAMOS LOS CENTROS QUE SE ABREN
datos$solucion<-solucion$solution[47:56]

graf2<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(mapping=aes(x=X1, y=X2, color=solucion),
show.legend = FALSE,size = 3) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,10,1)) +
  scale_x_continuous(breaks = seq(0,10,1))

graf2 + geom_text(vjust = -0.005, hjust=-1, size=3)
```

4.1.3. Código para la resolución del problema M_1

```
#LAS LIBRERIAS UTILIZADAS SON
library(ggplot2)
library(philentropy)
library(lpSolve)
library(tidyverse)

# DEFINIMOS LOS 10 NODOS ALEATORIAMENTE Y LOS SITUAMOS EN UN
#GRAFICO
puntos<-10
set.seed(3)
```

```
x<- runif(puntos, min=0, max=10);x
y<- runif(puntos, min=0, max=10);y

matriz<-matrix(c(x,y), byrow=FALSE,ncol=2)
datos<-data.frame(matriz)

graf<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(colour = 'blue', size = 1) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf + geom_text(vjust = -0.005, hjust=-1, size=3)

# MATRIZ DE COSTES
coste<-  distance(matriz, method = "euclidean")

# MODELO
filas<- puntos+puntos*puntos+1+puntos+puntos*puntos+1
columnas<- 1+puntos*puntos+puntos*puntos+puntos
p<-3
inst<-puntos
factor<-0.25
t<-4

#VARIABLES BINARIAS
bin<-c(rep(0,puntos*puntos),rep(0,puntos*puntos),rep(0,puntos));
for(i in 1:sum(puntos*puntos+puntos*puntos+puntos)) bin[i]<-1+i;


A<-matrix(0, nrow=filas, ncol=columnas)
signos<-c(rep('<=', filas ))
b<-c(rep(0,filas))
```

```
# FUNCION OBJETIVO
fun<-c(1,rep(0,puntos*puntos),rep(0,puntos*puntos),rep(0,puntos))
#       z ,    x ,                              m        ,            y

#CODIGO PARA LA PRIMERA RESTRICCION (2.13)
k<-0
for (f in 1:puntos){
  for (c in 1:puntos){
    A[f,1+k*inst+c]<- coste[f,c]
    A[f,puntos*puntos+1+k*inst+c]<- - factor*coste[f,c]

  }
  k<-k+1
}

# -z
A[1:puntos,1]<--1

#CODIGO PARA LA SEGUNDA RESTRICCION (2.14)
w<-1
x<-0
z<-2
for (i in 1:puntos){
  v<-0
  for (f in 1:puntos){
    for (c in 1:puntos){

      A[f+puntos+x, z+v]<- -1 #xij
      A[f+puntos+x, puntos*puntos+z+v]<- 1 #mij


    }
    v<-v+puntos
  }
```

```
  x<-x+puntos
  z<-z+1
  w<-w+1
}



#CODIGO PARA LA TERCERA RESTRICCION (2.15)
A[puntos+puntos*puntos+1,
        c(puntos*puntos+2):c(puntos*puntos+puntos*puntos+1)]<- 1

b[puntos+puntos*puntos+1]<-t



#CODIGO PARA LA CUARTA RESTRICCION (2.3)
k<-0
for (f in 1:puntos){
  for (c in 1:puntos){

    A[f+puntos+puntos*puntos+1,c+k*inst+1]<-1
    b[f+puntos+puntos*puntos+1]<-1
    signos[f+puntos+puntos*puntos+1]<-"="

  }
  k<-k+1
}



#CODIGO PARA LA QUINTA RESTRICCION (2.4)
w<-1
x<-0
z<-2
for (i in 1:puntos){
  v<-0
```

```
  for (f in 1:puntos){
    for (c in 1:puntos){

      A[f+puntos+puntos*puntos+1+puntos+x, z+v]<-1 #xij
      A[f+puntos+puntos*puntos+1+puntos+x,
                    puntos*puntos+puntos*puntos+1+w]<- -1

    }
    v<-v+puntos
  }
  x<-x+puntos
  z<-z+1
  w<-w+1
}

#CODIGO PARA LA SEXTA RESTRICCION (2.5)
for (c in 1:puntos){

  A[puntos+puntos*puntos+1+puntos+puntos*puntos+1,
            c+puntos*puntos+puntos*puntos+1]<- 1
}

b[puntos+puntos*puntos+1+puntos+puntos*puntos+1]<-p



#RESOLVEMOS EL PROBLEMA
solucion<-lp('min', fun, A, signos, b, binary.vec=bin)

round(solucion$objval,4)

#DIBUJAMOS LOS CENTROS QUE SE ABREN
datos$solucion<-solucion$solution[c(puntos*puntos+puntos*puntos+2):
c(puntos*puntos+puntos*puntos+puntos+1)]
```

```
graf2<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(mapping=aes(x=X1, y=X2, color=solucion),
show.legend = FALSE,size = 3) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf2 + geom_text(vjust = -0.005, hjust=-1, size=4)



#DIBUJAMOS LAS CONEXIONES
i_s<-c(1:puntos)
j_s<-c(1:puntos)

sol_x<-matrix(c(rep(i_s,puntos),sort(rep(j_s,puntos))),
byrow=FALSE,ncol=2)
sol_x<-data.frame(sol_x)
sol_x$solucion<-solucion$solution[2:c(puntos*puntos+1)]



d<-filter(sol_x, solucion==1)

mk<-rep(0,puntos)
for (i in 1:nrow(d)){
  mk[i]<-d[i,1]
}

x2<-rep(0,puntos)
y2<-rep(0,puntos)
for (j in 1:length(mk)){
  x2[j]<-datos$X1[mk[j]]
  y2[j]<-datos$X2[mk[j]]
}
```

```
df<- data.frame(x1=round(c(datos$X1),2), y1=round(c(datos$X2),2),
                x2,
                y2)

graf2 + geom_text(vjust = -0.005, hjust=-1, size=3)+
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),data = df,
               arrow = arrow(length = unit(0.1,"cm")))



#DIBUJAMOS LAS CONEXIONES MEJORADAS
sol_m<-matrix(c(rep(i_s,puntos),sort(rep(j_s,puntos))),
byrow=FALSE,ncol=2)
sol_m<-data.frame(sol_m)
sol_m$solucion<-solucion$solution[c(puntos*puntos+2):
c(puntos*puntos+puntos*puntos+1)]



d_m<-filter(sol_m, solucion==1)

final_m<-rep(0,nrow(d_m))
ini_m<-rep(0,nrow(d_m))

for (i in 1:nrow(d_m)){
  final_m[i]<-d_m[i,1]
  ini_m[i]<-d_m[i,2]
}

x2_m<-rep(0,puntos)
y2_m<-rep(0,puntos)

x1_m<-rep(0,puntos)
```

```
y1_m<-rep(0,puntos)
for (j in 1:length(final_m)){
  x2_m[j]<-datos$X1[final_m[j]]
  y2_m[j]<-datos$X2[final_m[j]]

  x1_m[j]<-datos$X1[ini_m[j]]
  y1_m[j]<-datos$X2[ini_m[j]]
}


df_m<- data.frame(x1_m, y1_m,
                  x2_m,
                  y2_m)

graf2 + geom_text(vjust = -0.005, hjust=-1, size=4)+
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),data = df,
               arrow = arrow(length = unit(0.1,"cm")))+

  geom_point() +
    geom_segment(aes(x = df_m$x1_m, y = df_m$y1_m, xend = df_m$x2_m,
    yend = df_m$y2_m),color=2,
                 arrow = arrow(length = unit(0.1,"cm")))
```

4.1.4. Código para la resolución del problema M_2

```
#LAS LIBRERIAS UTILIZADAS SON
library(ggplot2)
library(philentropy)
library(lpSolve)
library(tidyverse)

# DEFINIMOS LOS 10 NODOS ALEATORIAMENTE Y LOS SITUAMOS EN UN
#GRAFICO
puntos<-10
```

```
set.seed(3)
x<- runif(puntos, min=0, max=10);x
y<- runif(puntos, min=0, max=10);y


matriz<-matrix(c(x,y), byrow=FALSE,ncol=2)
datos<-data.frame(matriz)

graf<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(colour = 'blue', size = 1) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf + geom_text(vjust = -0.005, hjust=-1, size=3)

# MATRIZ DE COSTES
factor<-0.25
p<-3
t<-4
inst<-puntos

coste<-  distance(matriz, method = "euclidean")


coste_2<- matrix(rep(0,puntos*puntos), byrow = TRUE, ncol = puntos)

for (f in 1:puntos){
  for (c in 1:puntos){
    if (f<c){
      coste_2[f,c]<- coste[f,c]

    }
  }
}
```

```
coste_2<-as.vector(coste_2)
coste_2<-sort(coste_2) #ordenar de menor a mayor

coste_2<-unique(coste_2) #cij ordenados e ignorando empates


coste_3<- (1-factor)*coste
coste_3_2<- matrix(rep(0,puntos*puntos), byrow = TRUE,
ncol = puntos)


for (f in 1:puntos){
  for (c in 1:puntos){
    if (f<c){
      coste_3_2[f,c]<- coste_3[f,c]


    }
  }
}

coste_3_2<-as.vector(coste_3_2)
coste_3_2<-sort(coste_3_2) #ordenar de menor a mayor

coste_3_2<-unique(coste_3_2) #(1-f)cij ordenados e ignorando empates

coste_f<- c(coste_2,coste_3_2)
        #     cij,    (1-f)cij


coste_f_o<-unique(coste_f)
coste_f_o<-sort(coste_f_o)
```

```
length(coste_f_o)

# aijk
a<-array(0,dim = c(inst,inst,length(coste_f_o)))

for (k in 1:length(coste_f_o)){
  for (f in 1:puntos){
    for (c in 1:puntos){

      if (coste[f,c]<=coste_f_o[k]){

        a[f,c,k]<-1

      } else {
        a[f,c,k]<-0
      }
    }
  }
}

# bijk
b_ijk<-array(0,dim = c(inst,inst,length(coste_f_o)))

for (k in 1:length(coste_f_o)){
  for (f in 1:puntos){
    for (c in 1:puntos){

      if (coste_3[f,c]<=coste_f_o[k]){

        b_ijk[f,c,k]<-1

      } else {
        b_ijk[f,c,k]<-0
      }
```

```
    }
  }
}

# MODELO
filas<- puntos*length(coste_f_o)+puntos*length(coste_f_o)+1+1+1
columnas<- length(coste_f_o)+puntos+puntos


#vbles binarias
bin<-c(rep(0,length(coste_f_o)),rep(0,puntos),rep(0,puntos));
for(i in 1:c(length(coste_f_o)+puntos+puntos)) bin[i]<-i;


A<-matrix(0, nrow=filas , ncol=columnas)
signos<-c(rep('>=', filas ))
b<-c(rep(0,filas))

# Funci n objetivo
fun<-c(coste_f_o,rep(0,puntos),rep(0,puntos))
#           z         ,       y   ,       s

#CODIGO PARA LA PRIMERA RESTRICCION (2.18)
x<-0
for (k in 1:length(coste_f_o)){
  for (f in 1:puntos) {
    for (c in 1:puntos){
      A[f+x,length(coste_f_o)+c]<-a[f,c,k]+b_ijk[f,c,k] #aijk+bijk

      A[f+x,k]<-1
    }
  }
  x<-x+puntos
}
```

```
#CODIGO PARA LA SEGUNDA RESTRICCION (2.19)
x<-0
w<-1
for (k in 1:length(coste_f_o)){
  v<-0
  for (f in 1:puntos) {
    for (c in 1:puntos){
      A[puntos*length(coste_f_o)+f+x,length(coste_f_o)+c]<-a[f,c,k]
      A[puntos*length(coste_f_o)+f+x,puntos*puntos+2+v]<-1 #s_j
      A[puntos*length(coste_f_o)+f+x,k]<--1
    }
    v<-v+1
  }
  x<-x+puntos
  w<-w+1
}

#CODIGO PARA LA TERCERA RESTRICCION (2.20)
for (c in 1:puntos){
  A[puntos*length(coste_f_o)+puntos*length(coste_f_o)+1,
  length(coste_f_o)+puntos+c]<-1
}

signos[puntos*length(coste_f_o)+puntos*length(coste_f_o)+1]<-"<="
b[puntos*length(coste_f_o)+puntos*length(coste_f_o)+1]<-t


#CODIGO PARA LA CUARTA RESTRICCION (2.21)
for (c in 1: length(coste_f_o)){

  A[puntos*length(coste_f_o)+puntos*length(coste_f_o)+2,c]<-1

}
```

```
b[puntos*length(coste_f_o)+puntos*length(coste_f_o)+2]<-1
signos[puntos*length(coste_f_o)+puntos*length(coste_f_o)+2]<-"="

#CODIGO PARA LA QUINTA RESTRICCION (2.5)
for (c in 1:puntos){

  A[puntos*length(coste_f_o)+puntos*length(coste_f_o)+3,
  c+length(coste_f_o)]<- 1
}

b[puntos*length(coste_f_o)+puntos*length(coste_f_o)+3]<-p
signos[puntos*length(coste_f_o)+puntos*length(coste_f_o)+3]<-"<="


#RESOLVEMOS EL PROBLEMA
solucion<-lp('min', fun, A, signos, b, binary.vec=bin)

solucion$objval


#DIBUJAMOS LOS CENTROS QUE SE ABREN
datos$solucion<-solucion$solution[c(length(coste_f_o)+1):
c(puntos*puntos+1)]

graf2<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(mapping=aes(x=X1, y=X2, color=solucion),
show.legend = FALSE,size = 3) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf2 + geom_text(vjust = -0.005, hjust=-1, size=3)

#CAMBIAMOS EL COLOR DEL NUMERO DEPENDIENDO
```

```
#DE SI HACEN USO DE UNA CONEXION MEJORADA O NO
datos$sol_m<-solucion$solution[c(puntos*puntos+2):
c(puntos*puntos+puntos+1)]



graf2<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos),
col=sol_m))+  geom_point(mapping=aes(x=X1, y=X2, color=solucion),
show.legend = FALSE,size = 3) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf2 + geom_text(vjust = -0.005, hjust=-1, size=3)+
theme(legend.position='none')
```

4.2. Algoritmo heurístico

Resolvemos el mismo problema mediante un algoritmo genético en *Rstudio*.

```
#LAS LIBRERIAS UTILIZADAS SON
library(ggplot2)
library(philentropy)
library(lpSolve)
library(tidyverse)

puntos<-10
factor<-0.25
inst<-puntos
instalaciones<-c(1:inst)
p<-3
t<-4

#FUNCION PARA CALCULAR LA MAXIMA DISTANCIA

func_distancia<- function(vect_x,vect_inst,mat_coste){
  dist<-rep(0,puntos)

  tabla<-matrix(0,vect_inst,2)

  for (j in 1:vect_inst){
    cost<-rep(0,length(vect_x))

    for (i in 1:length(vect_x)){

      cost[i]<-mat_coste[j,vect_x[i]]
    }

    dist[j]<-min(cost)
  }
```

```
  tabla[,1]<-dist
  tabla[,2]<-1

  #Ordenamos por la segunda columna
  tabla_ord <- tabla[order(tabla[,1], decreasing = TRUE), ]

  w<-0 #contador que indica cuantas veces se ha realizado el
  #procedimiento

  #Si no se ha actualizado y lo ha hecho menos de t veces
  while (tabla_ord[1,2]!=0 & w!=t){

    tabla_ord[1,1]<-tabla_ord[1,1]*(1-factor)
    tabla_ord[1,2]<-0

    tabla_ord <- tabla_ord[order(tabla_ord[,1], decreasing = TRUE), ]

    w<-w+1
  }

  max<-tabla_ord[1,1]

}

# DEFINIMOS LOS 10 NODOS ALEATORIAMENTE Y LOS SITUAMOS EN UN
#GRAFICO

set.seed(3)
x<- runif(puntos, min=0, max=10);x
y<- runif(puntos, min=0, max=10);y


matriz<-matrix(c(x,y), byrow=FALSE,ncol=2)
```

```
datos<-data.frame(matriz)

graf<-ggplot(data=datos, aes(x=X1, y=X2,label = rownames(datos)))+
geom_point(colour = 'blue', size = 1) +
  labs(x="",y="") + scale_y_continuous(breaks = seq(0,puntos,1)) +
  scale_x_continuous(breaks = seq(0,puntos,1))

graf + geom_text(vjust = -0.005, hjust=-1, size=3)


# MATRIZ DE COSTES
coste<-  distance(matriz, method = "euclidean")


fobjetivo<-10000000

#CREAMOS LA POBLACION
columnas<-p+1
filas<-10
pob<- matrix(0,filas,columnas)

for (r in 1:filas) {
  #p numeros aleatorios
  x<-sample(instalaciones,p,replace = FALSE);x
  x<-sort(x);x


  #Calculamos la maxima distancia
  max=func_distancia(x,inst,coste)

  pob[r,1:p]<-x
  pob[r,p+1]<-max
}
```

```
#Si el mejor de la poblacion es menor que la funcion
#objetivo se cambia
if (min(pob[1:filas,p+1])<=fobjetivo){
  fobjetivo<-round(min(pob[1:filas,p+1]),6)

}

cadena<-paste("fobjetivo", fobjetivo)
print(cadena)

#########################################################################
iteraciones<-170
vect_fobjetivo<-rep(0,iteraciones+1)
vect_fobjetivo[1]<-fobjetivo


veces_mutacion<-0
veces_bus_loc<-0
for (z in 1:iteraciones){


  #se hace mutacion el 10% de las veces
  prob_mut<-c(0.1,0.9)
  mutacion<-sample(c(1,0),1,prob = prob_mut) #mutacion
  #1 si se hace mutacion y 0 si no

  if (mutacion ==1){
    print("se hace mutacion")
  }else{
    print("no se hace mutacion")
  }


  #se hace busqueda local el 50% de las veces
```

```
prob_bus_loc<-c(0.5,0.5)
busqueda_loc<-sample(c(1,0),1,prob = prob_bus_loc) #busqueda_loc
#1 si se hace busc loc y 0 si no

if (busqueda_loc ==1){
  print("se hace la busqueda local")
}else{
  print("no se hace la busqueda local")
}



#####################################################################
#####################################################################
# CRUCE
#####################################################################

#elijo dos genes al hazar para que sean los padres
a_p1<-sample(1:filas, 1)
a_p2<-sample(1:filas, 1)

#Para que no saque el mismo numero
while (a_p1 == a_p2){
  a_p2<-sample(1:filas, 1)
}

p1<-pob[a_p1, 1:p]
p2<-pob[a_p2,1:p]

#vector con los dos padres unidos
p1_p2<-c(p1,p2)

#Vemos si tiene repeticiones

p1_p2_u<-unique(p1_p2)
```

```
#si no tiene valores repetidos que saque p valores al azar
#2 veces
if (length(p1_p2)==length(p1_p2_u)){
  x_hijo<-sample(p1_p2,3)
  x_hijo_2<-sample(p1_p2,3)

} else{
  #si tiene valores repetidos se calcula probabilidad de ser
  #seleccionado
  tabla_2<-table(p1_p2)
  tabla_2<-as.data.frame(tabla_2)


  prob<-rep(0,length(p1_p2_u))
  for (i in 1:nrow(tabla_2)) {

    prob[i]<- (tabla_2[i,2]*100)/length(p1_p2)

  }

  x_hijo<-sample(p1_p2_u,3,prob = prob);
  x_hijo_2<-sample(p1_p2_u,3,prob = prob);
}

##################################################
#Maxima distancia para el primer hijo
max_h<-func_distancia(x_hijo,inst,coste)

m<-max(pob[1:filas,p+1])

  #si la max. distancia del hijo es mas peque a que el
  #peor de la poblacion se cambia
```

```
if (max_h<m){
  pob[which.max(pob[1:filas,p+1]),]<-c(x_hijo,max_h)
}
##############################################
#Maxima distancia para el segundo hijo
max_h<-func_distancia(x_hijo_2,inst,coste)

m<-max(pob[1:filas,p+1])

if (max_h<m){
  pob[which.max(pob[1:filas,p+1]),]<-c(x_hijo_2,max_h)
}



#################################################################
#################################################################
# MUTACION
#################################################################


if (mutacion==1){

  #Elijo un gen al azar
  mut_1<-sample(1:filas, 1)

  #Elijo un elemento al azar
  elem<-sample(1:p, 1)



  #Ahora del gen al azar elijo un elemento al azar
  #y lo cambio por otro al azar
  pob[mut_1,elem]<-sample(1:inst, 1)

  x_mut<-pob[mut_1,1:p]
```

```
  #Para que sean 3 centros distintos
  while (length(unique(x_mut))!=length(x_mut)){
    pob[mut_1,elem]<-sample(1:inst, 1)
    x_mut<-pob[mut_1,1:p]
  }


  ##############################################
  #Max. dist
  max_h<-func_distancia(x_mut,inst,coste)

  m<-max(pob[1:filas,p+1])

  #La mutacion se cambia por el peor si o si
  pob[which.max(pob[1:filas,p+1]),]<-c(x_mut,max_h)

  veces_mutacion<-veces_mutacion+1
}


#####################################################################
#####################################################################
# BUSQUEDA LOCAL
#####################################################################

if (busqueda_loc==1){

  cambios<-5
  fbus_loc<-100000

  #Elijo un gen al azar
  bus_loc_1<-sample(1:filas, 1)
```

```
#Elemento al azar
bus_loc_elem<-sample(1:p, 1)



x_bus_loc<-pob[bus_loc_1,1:p]
x_bus_loc

#Elijo tantos aleatorios como cambios quiero hacer
ale<-sample(1:inst, cambios)


x_bus_loc_max<-rep(0,p+1)

veces_nada<-0
for (l in 1:cambios) {

  x_bus_loc[bus_loc_elem]<-ale[l]
  x_bus_loc

  #Si no hay repeticiones que calcule la max. distancia
  if (length(unique(x_bus_loc))==length(x_bus_loc)){

    ###################################
    #Max. dist
    max_h<-func_distancia(x_bus_loc,inst,coste)

    if (max_h<=fbus_loc){
      fbus_loc<-max_h
      x_bus_loc_max<-c(x_bus_loc,fbus_loc)
    }

  }else{
```

```
        veces_nada<-veces_nada+1
      }




  }
  veces_bus_loc<-veces_bus_loc+1

  #Cambiamos el mejor por el peor de la poblacion
  pob[which.max(pob[1:filas,p+1]),]<-x_bus_loc_max

  cadena<-paste("veces nada: bus_loc", veces_nada)
  print(cadena)

}



#########################################################################
#funcion objetivo

if (min(pob[1:filas,p+1])<=fobjetivo){
  fobjetivo<-round(min(pob[1:filas,p+1]),4)

}

vect_fobjetivo[z+1]<-fobjetivo
cadena<-paste("fobjetivo", fobjetivo)
print(cadena)

cadena2<-paste("veces mutacion", veces_mutacion)
print(cadena2)

cadena3<-paste("veces bus_loc", veces_bus_loc)
```

```
    print(cadena3)

}

vect_fobjetivo
```

Capítulo 5

Conclusiones

Después de realizar todo el estudio, cabe resaltar que el número de iteraciones elegido no ha sido al azar. En la *Figura* 5.1 se muestra cómo varía el valor óptimo en cada iteración y para cada tamaño de población. Cuando el tamaño es 10 vemos que durante unas 100 iteraciones el valor no se modifica pero a partir de, aproximadamente, la iteración 160 este valor se reduce llegando al óptimo.

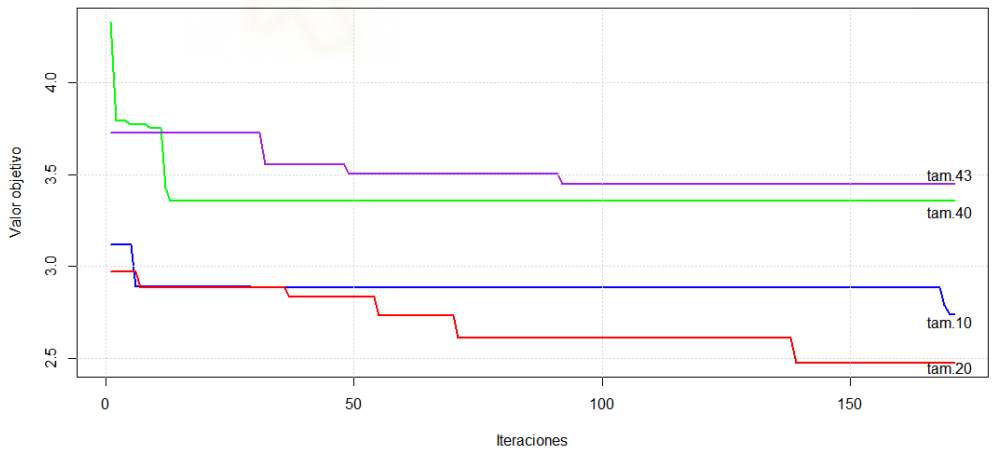

Figura 5.1: Valor objetivo para cada tamaño

En el *Cuadro* 5.1 vemos cómo varía la solución del algoritmo en función de cuántos cambios le introduzcamos. Como cabe esperar, a menor número de cambios menor es el tiempo computacional. Por ejemplo, si solo le aplicamos el cruce o solo el cruce y la

mutación el algoritmo no llega al óptimo en 170 iteraciones pero el tiempo es muy reducido. En cambio, en el momento que añadimos la búsqueda local vemos que la solución mejora considerablemente y el tiempo de ejecución aumenta notoriamente.

Entre cruce y búsqueda local y cruce, mutación y búsqueda local, nos quedamos con el último, pues nos aseguramos de que se obtengan mejores resultados a un coste computacional muy similar.

Cuadro 5.1: Algoritmo heurístico con distintos cambios

TAMAÑO	ÓPTIMO	CRUCE		CRUCE Y MUTACIÓN	
		VALOR	TIEMPO	VALOR	TIEMPO
10	2,7344	2,8907	0,3639 seg	2,7882	0,4063 seg
20	2,4732	2,7515	0,3989 seg	2,8821	0,4253 seg
40	3,3554	3,9145	0,417 seg	3,4114	0,4819 seg
43	3,4502	3,6536	0,4669 seg	3,5781	0,4643 seg

TAMAÑO	CRUCE Y BÚSQUEDA LOCAL		CRUCE, MUTACIÓN Y BÚSQUEDA LOCAL	
	VALOR	TIEMPO	VALOR	TIEMPO
10	2,7344	0,464 seg	2,7344	0,7461 seg
20	2,4732	0,6545 seg	2,4732	0,534 seg
40	3,4003	0,5779 seg	3,3554	0,6133 seg
43	3,4502	0,6143 seg	3,4502	0,6532 seg

En definitiva, todo este estudio trata de mostrar que se puede resolver el problema del p-centro de localización con mejoras, que era computacionalmente muy costoso a través de un modelo exacto, mediante un algoritmo heurístico. Al inicio del estudio, se ha resuelto el problema de manera exacta y, además, dada la importancia de distinguir los problemas p-centro de los p-mediana, se ha procedido a su resolución de ambas maneras para visualizar las diferencias. Seguidamente, mediante la programación de un heurístico, se han comparado los resultados tanto de manera exacta como de manera heurística obteniendo muy buenos resultados y, corroborando así, la efectividad de este. Por último, se verifica la importancia de los cambios introducidos dentro del heurístico pues, en función de cuántos se añadan, será más eficiente en encontrar una buena solución o menos.

Bibliografía

[1] Aguilar Gochicoa, J. S. (2004). *Diseño e implementación de un algoritmo heurístico para el problema de localización p-centro por vértices.*Tesis Licenciatura en Ingeniería Industrial. Departamento de Ingeniería Industrial y Textil, Escuela de Ingeniería, Universidad de las Américas Puebla.

[2] *Algoritmos genéticos.* (s.f). www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

[3] Anton-Sanchez, L. , Landete, M., Saldanha-da-Gama, F. (2022). *The Discrete p-Center Location Problem with Upgraded Connections.* Universidad Miguel Hernández y Universidad de Lisboa.

[4] Barcelos, G., Gaboardi, J., Wolf, L. J., Zhao, Q. (s.f). *P-Center Problem.* Pysal. https://pysal.org/spopt/notebooks/p-center.html

[5] Campbell, A. M., Lowe, T. J., Zhang, L. (2005). *Upgrading Arcs to Minimize the Maximum Travel Time in a Network.* University of Iowa, Pennsylvania State University and The Citadel.

[6] Canós Darós, M. J., Martínez Romero, M. L., Mocholí Arce, M. (s.f). *Un algoritmo para el cálculo del conjunto dominante finito del problema generalizado de la p-centdiana.* Universidad de Valencia y Florida Universitaria.

[7] Chung-Cheng Lu. (2013) *Robust weighted vertex p-center model considering uncertain data: An application to emergency management.* Department of Transportation and Logistics Management. National Chiao-Tung University.

[8] Garduño Juárez, R. (2018). *Algoritmos genéticos.* Conogasi. https://conogasi.org/articulos/algoritmos-geneticos/

[9] *Optimización matemática.* (s.f). Decide. https://decidesoluciones.es/soluciones-tecnologicas/optimizacion-matematica/

[10] Ramos, A. (s.f). *Investigación operativa y optimización*. Escuela Técnica Superior de Ingeniería. Universidad Pontificia Comillas.

[11] Riojas Cañari, A. C. (2005). *Capítulo 2. Heurística y metaheuristica*. E.A.P. de Investigación Operativa. Universidad Nacional Mayor De San Marcos.

[12] Sainz-Pardo, J. L. (2021). *Tema 7. Machine Learning aplicado a la Gestión de Carteras*. Gestión de Carteras e Inversiones. Universidad Miguel Hernández.

[13] *Significado de Optimización*. (s.f). Significados. https://www.significados.com/optimizacion/

[14] *Tema 1: Introducción a la Investigación Operativa*.(s.f). Universitat de València.

[15] Torres Contreras, S.P. (2020). *Cápsula — La Optimización Matemática – Algoritmos Metaheurísticos*. Ucuenca. https://www.ucuenca.edu.ec/component/content/article/269-espanol/investigacion/blog-de-ciencia/ano-2020/octubre-2020/1742-metaheuristicos

[16] Vidal Esmorís, A. (2013).*Algoritmos heurísticos en optimización*. Trabajo fin de máster en Técnicas Estadísticas. Universidad de Santiago de Compostela.

[17] Yañez, G., López, R. (s.f). *Nota: La Supervivencia del más apto: Computación Evolutiva y sus aplicaciones – Dirección General de Investigaciones*. https://www.uv.mx/investigacion/general/nota-la-supervivencia-del-mas-apto-computacion-evolutiva-y-sus-aplicaciones/