

Universidad  
Rey Juan Carlos

Escuela Técnica Superior  
de Ingeniería Informática

Grado en Matemáticas

Curso 2023-2024

Trabajo Fin de Grado

**PROBLEMA DEL P-CENTRO Y HEURÍSTICAS  
ASOCIADAS**

Autor: Diego Rodrigo Pérez

Tutor: Clara Simón de Blas



# Agradecimientos

A mi madre, por su incondicional ayuda en todo lo que ha estado a su alcance, permitiéndome centrarme plenamente en este proyecto.

A mi padre, por su apoyo emocional y la tranquilidad que me ha transmitido en todo momento.

A mi tutora Clara por su guía y paciencia constante y por todas las reuniones que me ha permitido.



# Resumen

El trabajo realizado se centra en el estudio del P-Centro y aplicación de heurísticas para resolver el problema, un problema de optimización que se presenta en diversas áreas como la logística, el diseño de redes y la planificación de servicios de emergencia.

El trabajo aborda la implementación y comparación de tres heurísticas principales:

- **Heurística por Pesos Ponderados en Búsqueda Local:**
- **Heurística de Sustitución de Vértice por Búsqueda Local:**
- **Heurística de Sustitución Tabú de Vértice por Búsqueda Local:**

El análisis de estas heurísticas se realizó mediante la implementación en Python y pruebas exhaustivas con diferentes tamaños de muestra ( $n$ ) y números de centros ( $p$ ).

En conclusión, el trabajo proporciona una visión completa y comparativa de métodos heurísticos aplicados al problema del P-Centro, destacando sus fortalezas y limitaciones en diferentes escenarios prácticos.



# Índice de contenidos

Índice de tablas	IX
Índice de figuras	XI
<b>1. Introducción</b>	<b>1</b>
1.1. Contexto y alcance . . . . .	1
1.2. Objetivos Generales . . . . .	3
1.3. Metodología . . . . .	3
<b>2. Contenidos principales</b>	<b>5</b>
2.1. Conocimientos auxiliares . . . . .	5
2.1.1. Problema P versus NP . . . . .	7
2.1.2. NP Completitud . . . . .	7
2.1.3. Heurísticas y metaheurísticas . . . . .	8
2.2. Problemas de tipo P . . . . .	10
2.2.1. P-Median Problem . . . . .	12
2.2.2. P-Center Problem . . . . .	13
2.3. Planteamiento del problema P-Center . . . . .	14
2.3.1. Enunciado del problema . . . . .	15
2.4. Heurísticas . . . . .	16
2.4.1. Heurística por pesos ponderados en búsqueda local . . . . .	17
2.4.2. Heurística Sustitución de Vértice por Búsqueda Local . . . . .	21
2.4.3. Heurística Sustitución de Vértice por Búsqueda Tabú Local . . . . .	36
<b>3. Resultados</b>	<b>43</b>
3.1. Resultados y comentarios . . . . .	43
3.1.1. Heurística por pesos . . . . .	43
3.1.2. Heurística Sustitución de Vértice por Búsqueda Local . . . . .	45
3.1.3. Heurística Sustitución Tabú de Vértice por Búsqueda Local . . . . .	53
<b>4. Conclusiones y trabajos futuros</b>	<b>59</b>
4.1. Conclusiones . . . . .	59
4.2. Trabajos futuros . . . . .	60



# Índice de tablas

2.1. Evolución de variables en el paso de añadir facilidad. . . . .	23
2.2. Evolución de las variables $c_1$ y $c_2$ . . . . .	26
2.3. Variable <i>vértices</i> antes. . . . .	32
2.4. Variable <i>vértices</i> actualizada. . . . .	33
2.5. Variables $c_1$ y $c_2$ antes. . . . .	33
2.6. Variables $c_1$ y $c_2$ actualizadas. . . . .	33

# Índice de figuras

2.1. Diagrama de Euler representando estos conjuntos[1] . . . . .	8
2.2. Ejemplo 1 mapa de puntos . . . . .	11
2.3. Similitud entre problemas. . . . .	13
2.4. Resolución del mismo mapa de los problemas p-Median y p-Center con $p=1$ (solución en rojo). . . . .	15
2.5. Vértices sin marcar, tomamos un radio $R=\text{radio}$ . . . . .	17
2.6. Elegimos el de mayor peso y marcamos si $w(v)d(u, v) \leq 2r$ . . . . .	18
2.7. Repetimos el proceso con los no marcados. . . . .	18
2.8. Cuando todos los vértices están marcados, devolvemos $P$ . . . . .	18
2.9. Ejemplo con $p = 4$ . . . . .	19
2.10. Calculamos distancias ponderadas y ordenamos. . . . .	19
2.11. Con $r = 5,8$ $ P  > p$ , seguimos. . . . .	19
2.12. Con $r = 34,88$ $ P  \leq p$ , paramos. . . . .	19
2.13. $ P  < p$ , añadimos un vértice (color verde). . . . .	20
2.14. $ P  < p$ , añadimos otro vértice. . . . .	20
2.15. Posible candidato. . . . .	30
2.16. Eliminación potencial. . . . .	30
2.17. Nuevo candidato. . . . .	31
2.18. Eliminación potencial. . . . .	31
2.19. $f = 17,09$ , $in = 6$ , $out = 2$ . . . . .	32
2.20. La nueva solución a optimizar. . . . .	33
2.21. Candidato. . . . .	34
2.22. Eliminación potencial. . . . .	34
2.23. Volvemos a tener nueva solución a optimizar. . . . .	34
2.24. Candidato. . . . .	35
2.25. Eliminación potencial. . . . .	35
2.26. Salida final de nuestra heurística. . . . .	35
2.27. Paradigma anterior con $p=3$ . . . . .	36
2.28. Nueva situación con $p=3$ . . . . .	36
2.29. Sustitución en cadena con $out=2$ , $in=5$ y $cn=7$ . . . . .	37
2.30. Situación inicial. . . . .	40
2.31. Situación sin mejora explorada. . . . .	40

---

2.32. Situación con mejora explorada. . . . .	41
2.33. Situación final. . . . .	42
3.2. Resolución del problema p-Center según esta heurística. . . . .	44
3.3. La heurística no siempre da buenos resultados. . . . .	45
3.4. Heurística ejecutada sin pesos. . . . .	45
3.5. Tiempo de ejecución: 0.095s, 2 iteraciones, n=10, p=4 . . . . .	46
3.6. Tiempo de ejecución: 0.095s, 3 iteraciones, n=10, p=4 . . . . .	46
3.7. Tiempo de ejecución: 0.110s, 5 iteraciones, n=15, p=3 . . . . .	47
3.8. Tiempo de ejecución: 0.102s, 5 iteraciones, n=15, p=6 . . . . .	47
3.9. Tiempo de ejecución: 0.099s, 4 iteraciones, n=30, p=6 . . . . .	47
3.10. Tiempo de ejecución: 0.110s, 3 iteraciones, n=45, p=12 . . . . .	48
3.11. Tiempo de ejecución: 0.127s, 8 iteraciones, n=60, p=6 . . . . .	48
3.12. Tiempo de ejecución: 0.147s, 6 iteraciones, n=100, p=7 . . . . .	48
3.13. Tiempo de ejecución: 7.151s, 18 iteraciones, n=1000, p=12 . . . . .	49
3.14. Tiempo de ejecución: 43.256s, 5 iteraciones, n=10000, p=200 . . . . .	49
3.15. Tiempo de ejecución: 7.244s, 5 iteraciones, n=1000, p=2 . . . . .	50
3.16. Tiempo de ejecución: 4.916s, 4 iteraciones, n=1000, p=2 . . . . .	50
3.17. Tiempo de ejecución: 5.720s, 5 iteraciones, n=1000, p=2 . . . . .	50
3.18. Tiempo de ejecución: 3.626s, 11 iteraciones, n=1000, p=20 . . . . .	51
3.19. Tiempo de ejecución: 1.800s, 4 iteraciones, n=1000, p=20 . . . . .	51
3.20. Tiempo de ejecución: 2.509s, 4 iteraciones, n=1000, p=20 . . . . .	51
3.21. Tiempo de ejecución: 0.219s, 4 iteraciones, n=1000, p=200 . . . . .	52
3.22. Tiempo de ejecución: 0.472s, 6 iteraciones, n=1000, p=200 . . . . .	52
3.23. Tiempo de ejecución: 0.221s, 4 iteraciones, n=1000, p=200 . . . . .	52
3.24. n=10, p=4 . . . . .	54
3.25. n=10, p=4 . . . . .	54
3.26. n=10, p=4 . . . . .	54
3.27. n=10, p=4 . . . . .	55
3.28. n=100, p=4 . . . . .	55
3.29. n=100, p=40 . . . . .	56
3.30. n=1000, p=40 . . . . .	56
3.31. n=1000, p=40 . . . . .	57
3.32. n=1000 . . . . .	57
3.33. n=1000 . . . . .	57

# 1

## Introducción

### 1.1. Contexto y alcance

El mundo actual está repleto de redes y sistemas complejos de servicio que son tremendamente importantes para el desarrollo diario del mismo. Cada vez son más las empresas, gobiernos y, en general, organizaciones que muestran un creciente interés por la optimización de sus redes de servicio, ya sean de datos, de mercancía, de emergencia o de cualquier actividad que realicen. Es aquí donde surge el problema de la  $p$ -mediana, que busca minimizar la distancia media de los nodos de la red a los centros elegidos.

Este problema y sus variantes han sido muy estudiadas desde la segunda mitad del siglo XX debido a su utilidad y relevancia. Aunque la base de estos problemas suele ser la misma, cada uno de ellos tiende a estar más capacitado para modelizar y resolver situaciones del mundo real distintas.

Imaginemos que una empresa llamada “Amazin” quiere abrir un número de puntos de recogida de paquetes en el barrio de Chamberí. Amazin tiene muchos contactos y además ofrece buenas condiciones a aquellos comercios que se ofrezcan a ser un punto de recogida, por lo que puede elegir entre bastantes locales, pero solo puede tener un número  $n$  de establecimientos por su límite de presupuesto. El objetivo del departamento encargado de seleccionarlos será minimizar la distancia media que tienen que recorrer sus clientes hasta su punto de recogida más cercano, para prestar un buen servicio a la mayoría de sus clientes. Este es un ejemplo real del problema  $p$ -*median*.

Esta misma empresa tiene una red de servicios *Cloud* llamada “ASW” que ha

abierto una nueva zona de disponibilidad en España y garantiza a sus clientes la integridad y la disponibilidad de sus datos en cualquier momento. Para ello, crea copias de seguridad de los datos de los servidores y quiere que la posición física de estos *clusters* de datos copia sea lo más alejada de cualquier otro servidor posible, pero solo puede hacer que un servidor ya existente sea el *cluster* de los más cercanos. El departamento de logística se enfrenta a un problema de tipo *p-dispersion*.

Ahora, ASW quiere abrir un servicio de almacén de datos. Además, quiere poder garantizar que cualquier servidor va a poder comunicarse con el almacén con un tiempo de latencia inferior a un umbral, por lo que tiene que elegir sus servidores de correo de forma estratégica para minimizar el tiempo máximo que tarda en llegar el dato a cualquiera de los servidores. De nuevo se enfrentan a un problema de tipo  $p$ , en este caso al problema *p center*.

Como se puede observar, es fácil encontrar ejemplos en los que esta familia de problemas es útil. Hay muchos sectores en los que nos encontramos problemas similares y en los que podemos aplicar las técnicas de resolución que veremos más adelante, lo único que tenemos que definir es que es exactamente lo que queremos optimizar.

En el sector de servicios de emergencia podemos modelizar algunas situaciones como ejercicios del tipo *p-center*, *p-median* o *p-coverage* dependiendo de si tenemos que prestar un servicio mínimo a todos, buscamos la mejor solución para la mayoría o tenemos limitaciones en la distancia que se puede recorrer.

El sector de la aviación no está exento de este tipo de problemas. Si queremos mantener una red en la cual consigamos el mínimo tiempo de viaje medio nos encontramos con el problema *p-hub median* o si lo que buscamos es que el tiempo máximo de viaje sea mínimo estamos en el *p-hub center*. También podemos estar interesados en una red en la cual no todos los *hubs* estén conectados; entonces estamos en un problema de tipo *disconnected p-hub median/center*.

En el área de transporte nos encontramos con un panorama similar al de servicios de emergencia, aunque en este caso como el objetivo es maximizar el beneficio, el planteamiento de tipo *p-median* suele ser más acertado y dependiendo del tipo de empresa en la que estamos podemos tener otras restricciones o intereses que nos hagan derivar en modelos *p-coverage*, *capacitated p-median* o cualquier otro que se adapte a nuestra situación.

Si nos centramos en las redes informáticas, vemos que aquí se abre una amplia selección de situaciones y configuraciones de redes distintas. Podemos encontrar ejemplos en los que los modelos *p-median* o *p-center* son más adecuados, así como otros en los que los de tipo *hub* se aproximan mejor. En el caso de tener servidores en los que la capacidad es un problema, los del estilo *capacitated* son mejores y si encima el nivel de demanda sabemos que va a variar, podemos atacar

de problema con algún modelo que sea dinámico.

## 1.2. Objetivos Generales

El objetivo principal de este trabajo es estudiar el problema del P-Centro y las heurísticas asociadas para su resolución. Este problema de optimización se presenta en diversas áreas como la logística, el diseño de redes y la planificación de servicios de emergencia. El objetivo general se desglosa en los siguientes objetivos específicos:

- Comprender y estudiar el problema del P-Centro.
- Implementar y comparar distintas heurísticas para resolver el problema del P-Centro.
- Evaluar la efectividad y eficiencia de cada heurística en diferentes escenarios.

## 1.3. Metodología

El trabajo va a dividirse en varias etapas

- Recopilación de conocimientos auxiliares para contextualizar el problema.
- Explicación de los problemas de tipo p más relevantes.
- Análisis del problema del P-Centro .
- Implementación y estudio de heurísticas del problema del P-Centro
- Comparación y ejecución de heurísticas.



# 2

## Contenidos principales

### 2.1. Conocimientos auxiliares

En el año 1900 David Hilbert propone en el Congreso Internacional de Matemáticos celebrado en París los veintitrés problemas que él consideraba más importantes para el desarrollo de las matemáticas. Estos problemas serían conocidos como los Problemas de Hilbert y tuvieron una gran repercusión a gran escala, no solo en el ámbito científico, sino que serían ampliamente conocidos y llegaron al conocimiento de la mayoría de personas.[2]

Cien años más tarde, en el año 2000, el Instituto Clay de Matemáticas de Cambridge conmemoraba en lanzamiento de los Problemas de Hilbert con la presentación de siete problemas que tenían como objetivo concienciar a la gente de que había muchas cosas en las matemáticas sin resolver, los Problemas del Milenio [3]. Fueron elegidos los problemas que el instituto consideraba más relevantes para el desarrollo de las matemáticas, y la resolución de cualquiera de ellos está premiada con un millón de dólares. Los problemas son los siguientes:

- P versus NP: Este es uno de los problemas más famosos en ciencias de la computación teórica. Se pregunta si cada problema cuya solución puede ser rápidamente verificada por un computador (NP) también puede ser rápidamente resuelto por un computador (P) [4].
- La conjetura de Hodge: Propone que para ciertos espacios, llamados variedades algebraicas, las soluciones de ecuaciones polinómicas, se pueden

entender completamente en términos de características geométricas más sencillas conocidas como clases de cohomología de Hodge [5].

- La conjetura de Poincaré: Esta conjetura afirma que una variedad de 3 dimensiones que es topológicamente simple (es decir, cada lazo se puede ajustar a un punto sin cortar la superficie) es, de hecho, equivalente a una esfera tridimensional [6].
- La hipótesis de Riemann: Se refiere a la distribución de los ceros no triviales de la función zeta de Riemann. La hipótesis postula que todos estos ceros tienen una parte real de  $1/2$ . A pesar de su importancia fundamental en la teoría de números, aún no se ha demostrado [7].
- Existencia de Yang-Mills y del salto de masa: Se centra en la física teórica. Pregunta si las teorías de Yang-Mills, que son fundamentales para entender las fuerzas nucleares fuertes y débiles en física de partículas, existen matemáticamente y si explican el fenómeno del “salto de masa”, donde partículas sin masa adquieren masa a través de interacciones [8].
- Las ecuaciones de Navier-Stokes: Trata sobre la existencia y suavidad de soluciones para las ecuaciones de Navier-Stokes, que describen el movimiento de fluidos. El desafío es demostrar si, bajo condiciones iniciales dadas, estas ecuaciones siempre tienen soluciones que se comportan de manera suave, sin singularidades o discontinuidades [9].
- La conjetura de Birch y Swinnerton-Dyer: Se relaciona con ciertos tipos de curvas algebraicas llamadas curvas elípticas. Propone una relación profunda entre el número de soluciones racionales de la curva (su “rango”) y el comportamiento de una función asociada a la curva en un punto específico. A pesar de numerosos avances, la conjetura permanece sin demostrar en general [10].

De estos problemas solo ha sido resuelta la conjetura de Poincaré, por Grigori Perelman en 2002. Perelman consiguió finalizar y completar el trabajo de Richard Hamilton y se le ofreció el premio del millón de dólares por ello, además de la medalla Fields por su contribución a la teoría de flujo de Ricci. Perelman rechazó los dos premios debido a que consideraba que su trabajo había sido igual de valioso que el de Hamilton y que éste merecía el mismo trato que él.

Todos estos problemas tienen mucha relevancia y de cualquiera de ellos se podrían hacer múltiples TFGs, pero nosotros vamos a estudiar el problema P vs NP, ya que nos interesa conocer distintos conceptos arraigados a éste.

### 2.1.1. Problema P versus NP

El problema P versus NP es uno de los problemas más importantes del milenio. Envuelve al área de las matemáticas con la de la informática y está relacionado con el tiempo de computación de que tienen los problemas.

Imaginemos que tenemos que resolver un problema y encontramos un algoritmo que lo soluciona cuya complejidad es  $O(n^3)$ . Este problema pertenece a un conjunto denominado  $P$ , que son todos aquellos problemas cuya solución se puede computar en tiempo polinomial, es decir aquellos cuya complejidad es  $O(n^k)$ , con  $k \in \mathbb{N}$ .

Ahora vamos a ver que es el conjunto NP. Nos dan un problema cualquiera de cuál no sabemos el algoritmo de resolución, solo nos dan el resultado que ha obtenido ese algoritmo y nos piden que comprobemos que efectivamente es correcta la solución obtenida. Si existe algún algoritmo de comprobación cuya complejidad sea  $O(n^k)$  (polinomial), entonces diremos que nuestro problema pertenece a NP; es decir, NP es aquel conjunto formado por los problemas cuya solución es verificable en tiempo polinomial. Una vez visto esto, es fácil entender que  $P \subseteq NP$ , ya que para comprobar que la solución de un problema en  $P$  es correcta podemos ejecutar el propio algoritmo que la genera, que es de complejidad polinomial.

Con estos conceptos entonces se plantea la siguiente duda: Si para un problema dado podemos encontrar un algoritmo que compruebe su solución en tiempo polinomial, ¿podemos encontrar un algoritmo que la calcule en tiempo polinomial? o lo que es lo mismo, ¿es  $P = NP$ ?[4]

### 2.1.2. NP Completitud

Una vez comprendida la naturaleza del problema, vamos a seguir viendo ciertos conceptos relacionados que son relevantes para este trabajo. El objetivo principal va a ser llegar a entender la dificultad a la que nos estamos enfrentando y el por qué utilizamos las técnicas que veremos más adelante.

Se dice que un problema  $M$  es NP-completo cuando cumple las siguientes condiciones[11]:

1.  $M$  pertenece al conjunto NP.
2. Todo problema de NP es reducible a  $M$  en tiempo polinomial.

Intuitivamente esto nos está diciendo que los problemas que se clasifiquen como NP-completos van a ser los más difíciles del conjunto NP. Además, esta definición nos está diciendo que si encontramos una solución polinómica para un

problema NP-completo, todos los problemas de NP serían resolubles en tiempo polinómico, por lo que obtendríamos  $P=NP$ .

Si ahora eliminamos la primera premisa, generamos el conjunto NP-difícil, que es un conjunto más grande que contiene a todos los problemas a los cuales cualquier problema de NP es reducible. Es trivial ver que el conjunto NP-completo está contenido en el NP-difícil, pero viceversa no se cumple; existen problemas NP-difícil que no están contenidos en el conjunto NP.

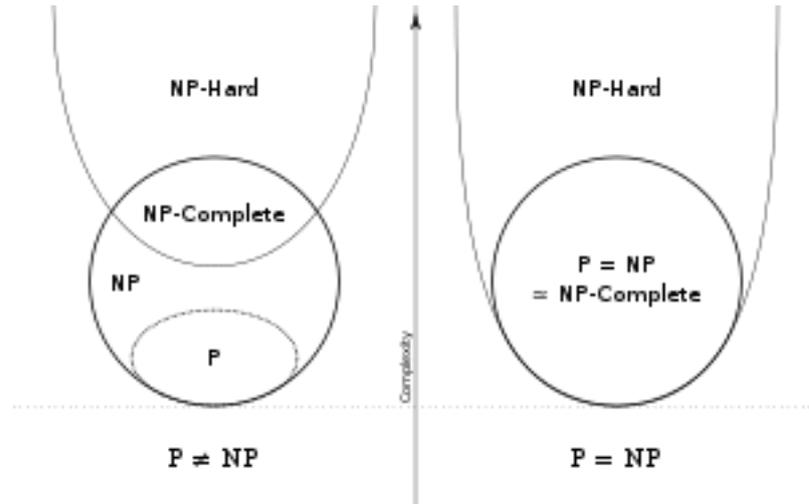


Figura 2.1: Diagrama de Euler representando estos conjuntos[1]

Aunque parezca que los problemas NP-completos son especiales, la realidad es que se conocen pocos problemas NP que no pertenezcan a NP-completo o a P. Estos problemas se denominan NP-intermedios y algunos de los pocos ejemplos que existen son el problema de factorización de enteros o el problema de isomorfismo entre grafos[12].

### 2.1.3. Heurísticas y metaheurísticas

En la resolución de problemas complejos, los algoritmos juegan un papel fundamental al ofrecer procedimientos paso a paso para alcanzar soluciones óptimas. Sin embargo, cuando nos enfrentamos a problemas NP-completos, donde la verificación de una solución es factible pero encontrarla es exponencialmente difícil, los algoritmos exactos a menudo se vuelven impracticables debido a su elevado tiempo de cómputo. Es aquí donde entran en juego las heurísticas y metaheurísticas, técnicas diseñadas para encontrar soluciones buenas en tiempos razonables, sacrificando potencialmente la optimalidad por la eficiencia.

En este apartado vamos a dar los conocimientos básicos necesarios para comprender los conceptos de heurística y metaheurística, analizando su importancia

y aplicación en problemas NP-completos, haciendo énfasis en cómo estas técnicas nos permiten abordar desafíos computacionales que de otro modo serían inaccesibles.

### Concepto de heurística

Las heurísticas representan métodos o estrategias diseñadas para solucionar problemas de forma eficiente cuando los enfoques tradicionales, como los algoritmos exactos, resultan impracticables por su alta demanda de tiempo y recursos. A diferencia de estos últimos, que garantizan encontrar la solución óptima, las heurísticas buscan soluciones buenas o “suficientemente buenas” en un tiempo razonable, especialmente útiles en problemas NP-completos donde la complejidad computacional hace inviable el uso de métodos exactos. Las heurísticas se basan en la experiencia, intuición, o reglas prácticas, facilitando la toma de decisiones en procesos complejos de búsqueda o en problemas de optimización [13].

### Ejemplo de heurística - Suma de subconjuntos

Para entender mejor la diferencia entre un algoritmo exacto y una heurística veamos el siguiente ejemplo; el problema de la suma de subconjuntos.

Este problema consiste en determinar si existe un subconjunto de números en un conjunto dado que sume un valor específico. Por ejemplo, dado el conjunto  $S = \{3, 34, 4, 12, 5, 2\}$  y el valor objetivo  $T = 9$  queremos saber si hay algún subconjunto de  $S$  cuya suma sea exactamente  $T$ .

Solución con Algoritmo Exacto Método: Fuerza bruta

Descripción: El algoritmo de fuerza bruta explora todos los posibles subconjuntos de  $S$  para verificar si alguno suma  $T$ . Esto se puede hacer generando todas las combinaciones posibles de los números en  $S$  y sumándolas para comprobar si alguna iguala a  $T$ .

Generamos todos los subconjuntos posibles:  $\emptyset, \{3\}, \{34\}, \{4\}, \dots, \{3, 34, 4, 12, 5, 2\}$ . Comprobamos cada subconjunto hasta encontrar uno cuya suma sea 9, como  $\{4, 5\}$ . Este método garantiza encontrar la solución si existe, pues examina todas las posibilidades. Sin embargo, su tiempo de ejecución crece exponencialmente con el tamaño de  $S$ , volviéndose impracticable para conjuntos grandes.

Solución con Heurística Método: Voraz

Descripción: La heurística voraz para este problema podría empezar seleccionando el número más grande de  $S$  que no exceda  $T$  y luego repetir el proceso con el resto hasta alcanzar  $T$  o determinar que no es posible.

Ejemplo:

Seleccionamos el primer número que no exceda  $T=9$ , que es 5. Restamos 5 de  $T$ , dejándonos con un nuevo objetivo de 4. Seleccionamos 4 del conjunto restante. Hemos alcanzado el objetivo  $T=9$  con el subconjunto  $\{5, 4\}$ .

Este método es mucho más rápido, pero no garantiza la solución óptima o incluso una solución en todos los casos. Por ejemplo, podría fallar si el conjunto fuera  $= \{2, 34, 4, 12, 5, 10\}$  y  $T=15$ , ya que podría elegir 10 primero y luego no encontrar una suma que complemente hasta 15 con los números restantes, aun cuando existe una combinación válida como  $\{5, 10\}$ .

### Concepto de metaheurística

Una metaheurística es un método de alto nivel que guía el proceso de búsqueda de soluciones en problemas de optimización, diseñado para explorar el espacio de soluciones de manera eficiente y efectiva, con el objetivo de encontrar soluciones óptimas o casi óptimas en un tiempo razonable. Estos métodos se basan en principios que permiten superar las limitaciones de las heurísticas tradicionales, tales como quedarse atrapados en óptimos locales o no ser capaces de explorar adecuadamente el espacio de búsqueda.

Una característica distintiva de las metaheurísticas es su capacidad para balancear la exploración del espacio de búsqueda (buscando soluciones en áreas no exploradas previamente) con la explotación de las soluciones encontradas (mejorando soluciones ya descubiertas). Esto las hace particularmente efectivas para tratar con problemas complejos y de gran escala, donde los métodos exactos son ineficaces o inviables debido a su complejidad computacional.

En este caso no vamos a dar ejemplos de metaheurísticas ya que necesitaríamos explicar más conceptos que no van a ser relevantes para este trabajo debido a que nosotros trabajaremos solo con heurísticas.

## 2.2. Problemas de tipo P

Como hemos adelantado en la introducción, los problemas de tipo  $p$  tienen que ver con la localización y asignación de nodos de servicio y son problemas NP completos. En esta sección vamos a hablar de los problemas P principales, tratando su planteamiento y sus principales estrategias de resolución.

En todos nuestros ejercicios vamos a tratar de estudiar un mapa similar al expresado en la figura 2.4.

En nuestros problemas vamos a tener que elegir una serie de puntos en base

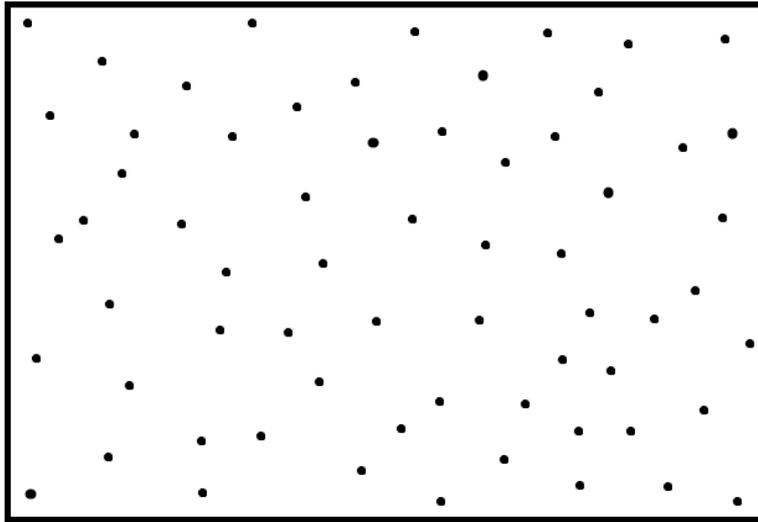


Figura 2.2: Ejemplo 1 mapa de puntos

a las condiciones que impongamos y éstas serán las que marquen las diferencias entre un problema y otro, pero va a haber unas hipótesis generales que todos los problemas seguirán:

- Solo podemos elegir puntos ya establecidos en el mapa.
- Cualquier punto puede ser elegido.
- El valor que utilizaremos para determinar la distancia sera la distancia euclídea.
- Trabajamos con demanda igual en todos nuestros puntos.
- Asumimos que cualquier facilidad tiene infinita capacidad de abastecimiento.
- Cualquier punto no elegido tendrá una única arista que se conectará siempre con uno elegido.

Fijamos estas condiciones para reducir la variabilidad del estudio, ya que estos problemas tienen muchas pequeñas variaciones que, aunque son interesantes, son muy similares y nos harían divagar demasiado. Veamos ahora la notación que vamos a emplear en las siguientes secciones teniendo en mente el mapa anterior:

Los puntos especiales seleccionados se llamarán facilidades, centros o nodos y todos los puntos van a tener un natural asociado, por lo que nuestro conjunto de puntos va a ser representado por  $I = \{1, 2, 3, \dots, i\}$ . Las aristas entre puntos serán representadas por el conjunto  $A = \{(i, j) \in I \times I\}$  y  $p$  será el número de instalaciones a utilizar. Además tendremos las siguientes variables de decisión binarias:

- $x_{i,j} = 1$  si el punto  $i$  es abastecido desde el nodo  $j$ , 0 en otro caso.
- $y_i = 1$  si el punto  $i$  es un nodo de servicio, 0 en otro caso.

Denominaremos a la distancia que hay desde el punto  $i$  al  $j$  como  $d_{i,j}$  y, como la demanda es igual en todos nuestros puntos, asumiremos que es 1 para todos.

Con estos términos ya definidos, podemos empezar a estudiar los distintos problemas que surgen al cambiar el objetivo y las condiciones.

### 2.2.1. P-Median Problem

El “P Median Problem” fue el primero de su familia en surgir, como hemos comentado al principio de este trabajo. En el sector privado tiene mucha importancia porque, por norma general, se utiliza para optimizar beneficios. El objetivo de este problema es minimizar la distancia media que hay de cualquier punto cliente a su nodo servicio más cercano y es formalmente denominado como *El problema de asignación incapacitado de  $p$  facilidades*.

#### Planteamiento

Vamos a plantear el problema con notación matemática[14]. Teniendo en mente las definiciones anteriores, tenemos:

Objetivo:

$$\text{Minimizar}(Z)$$

Condiciones:

$$Z = \sum_{j \in I} \sum_{i \in I} d_{ij} x_{ij}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I$$

$$\sum_{i \in I} y_i = p$$

$$x_{ij} - y_j \leq 0 \quad \forall i, j \in I$$

La primera condición nos da la función que queremos minimizar. Buscamos que el sumatorio de distancias sea mínimo, así que solo tenemos que sumar todas las distancias que están siendo utilizadas. Para controlar que son usadas, multiplicamos la distancia por la variable binaria  $x_{ij}$  que es 1 si el punto  $i$  se abastece desde  $j$ .

La segunda condición controla que un punto solo se pueda abastecer de un mismo nodo, aunque hay que matizar un detalle, para que esto se cumpla, estamos

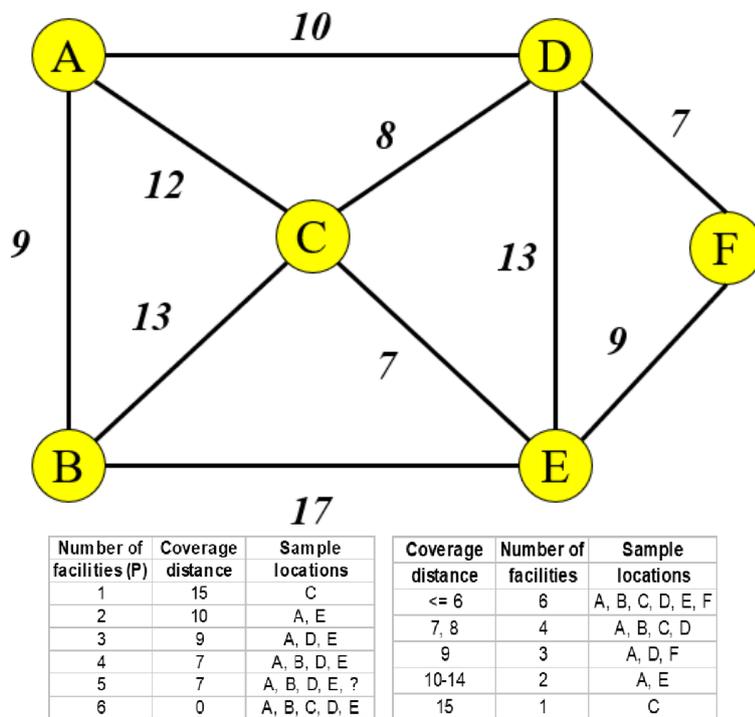
suponiendo que  $\forall i \in I, x_{ii} = y_i$ , es decir, que un nodo es un punto que se abastece a sí mismo.

La tercera restricción la utilizamos para controlar el número de localidades que permitimos abrir, siendo  $p$  dicho número. Algunos autores sustituyen la notación de  $y_i$  por  $x_{ii}$ , nosotros nos mantendremos con la dada por simplicidad.

La última de las restricciones fuerza que los nodos de servicio no lleguen a puntos a los que no tienen arista conectada.

### 2.2.2. P-Center Problem

El problema P-Center está más arraigado al sector público ya que, a diferencia del P-Median que busca minimizar el total, el P-Center busca minimizar la distancia máxima, por lo que se puede garantizar que cualquier usuario de la red va a tener un buen servicio[15]. Además, el problema del conjunto de cubrimiento está muy ligado a este como se puede ver a continuación en el siguiente ejemplo 2.3.



(a) P-Center      (b) Conjunto cubrimiento

Figura 2.3: Similitud entre problemas.

**Planteamiento**

Objetivo:

$$\text{Minimizar}(Z)$$

Condiciones:

$$Z \geq \sum_{j \in I} d_{ij} x_{ij}$$

$$\sum_{j \in I} x_{ij} = 1 \quad \forall i \in I$$

$$\sum_{i \in I} y_i = p$$

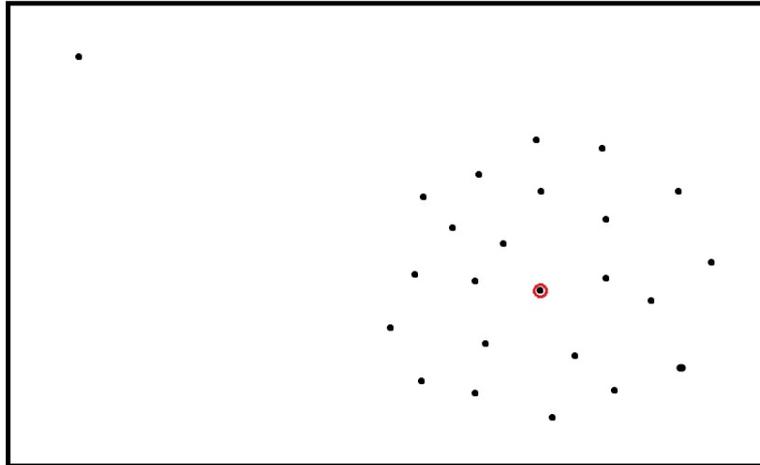
$$x_{ij} - y_j \leq 0 \quad \forall i, j \in I$$

## 2.3. Planteamiento del problema P-Center

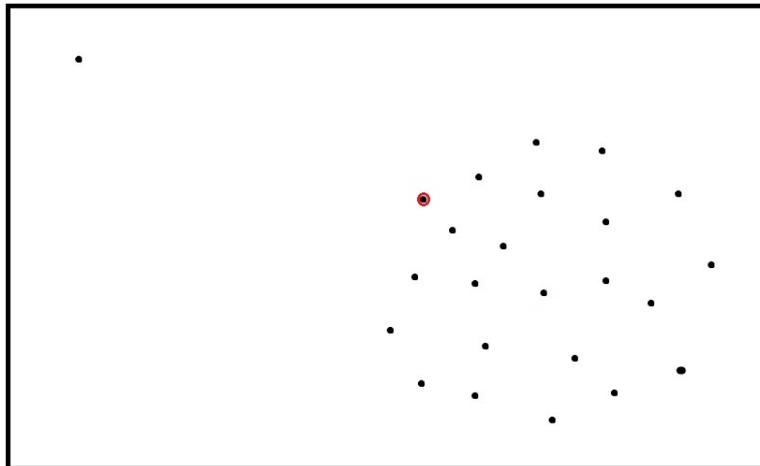
Vamos a realizar un estudio con mayor profundidad del problema p-Center. Se ha elegido esta variante por distintos motivos que han hecho más atractiva su investigación:

- Múltiples aplicaciones, orientadas a garantizar un servicio a cualquier usuario.
- Amplio historial de estudios sobre este problema. Es un problema muy presente en su ámbito.
- Gran relevancia tanto dentro del mundo matemático como fuera.
- Simplicidad y profundidad. Estamos ante un problema muy sencillo de enunciar, pero a la hora de resolverlo nos encontramos con un paradigma mucho más complejo e interesante.
- Relaciones con otros problemas también relevantes.

Como se ha explicado anteriormente, los p-problemas se plantean sobre un mapa de localizaciones entre las que tenemos que elegir un número  $p$  de ellas como nexos para cumplir cierta condición. En nuestro caso, p-Center, la condición a cumplir es que el coste de abastecimiento de la localización que mayor coste tenga sea mínimo, suponiendo que cada localización es abastecida por el nexo más cercano. La figura 2.4 muestra un ejemplo ilustrativo que permite visualizar la diferencia.



(a) Minimizar distancia media(p-Median).



(b) Minimizar distancia máxima(p-Center).

Figura 2.4: Resolución del mismo mapa de los problemas p-Median y p-Center con  $p=1$  (solución en rojo).

### 2.3.1. Enunciado del problema

Aunque es necesario tener una noción general sencilla de entender del problema, para empezar a resolverlo vamos a necesitar más rigor en su planteamiento. Para ello vamos a modelizar nuestro mapa de puntos como un grafo conexo basandonos en [16].

Sea  $U = \{u_0, u_1, u_2, \dots, u_{n-1}\}$  un conjunto de vértices o puntos en el plano y sea  $d(u_i, u_j) = d_{i,j}$  función que dados dos vértices nos da la distancia o coste asociado que hay entre ellos. Podemos representar nuestro problema como un grafo completo ponderado  $G = (V, E)$ , donde el valor de la arista  $e_{i,j} = d_{i,j}$ . Además, podemos asociar cierta demanda a cada uno de nuestros vértices,  $w_i$ ,

que añadimos porque no nos va a suponer ningún tipo de complicación añadida debido a un detalle que veremos más adelante.

Con esta literatura previa, ya podemos definir el problema a encontrar un conjunto de puntos  $P \subseteq U$ , con  $|P| = p$  que consiga que la función

$$r(P) = \max_{u_i \in U} \left\{ w_i \min_{v_j \in P} d(u_i, v_j) \right\}$$

sea mínima.

Ahora bien, podemos simplificar la función a minimizar fijándonos en un detalle. Desde el principio hemos considerado que  $d_{i,j}$  podía ser la distancia euclídea o podía ser un coste asociado, es decir, no podemos utilizar propiedades geométricas como la desigualdad triangular para nuestro método de resolución ya que el algoritmo que usemos tiene que ser independiente de la condición de ser distancia euclídea o no. Por ello, vamos a reescribir nuestra función a minimizar como

$$r(P) = \max_{u_i \in U} \left\{ \min_{v_j \in P} d'(u_i, v_j) \right\}$$

donde estaríamos asumiendo que

$$d'(u_i, v_j) = w_i \min_{v_j \in P} d(u_i, v_j)$$

Llamaremos a  $r(P)$  el radio de nuestra solución. Si al radio mínimo posible de nuestro problema lo denominamos  $r^*$ , vemos que una solución exacta significaría que  $r(P) = r^*(P)$ , que como sabemos que este problema es NP-Completo implicaría que el algoritmo que lo pueda resolver no sería de complejidad polinomial. Por ello, dependiendo de cuanto queramos garantizar que nos estamos aproximando al radio real tendremos heurísticas con mayor o menor complejidad, es decir,

$$\alpha = \frac{r}{r^*}$$

definirá la complejidad de nuestro algoritmo (aunque en muchos casos no podamos calcularlo).

## 2.4. Heurísticas

Vamos a proponer distintas soluciones a nuestro problema. Como hemos explicado antes, dependiendo del grado de exactitud que pidamos, encontraremos una mayor o menor complejidad.

### 2.4.1. Heurística por pesos ponderados en búsqueda local

La heurística que se va a proponer a continuación es presentada en [16] y asume que tenemos una matriz con todas las aristas y sus valores asociados por cada par de vértices. Como tenemos un grafo completo, calcular esta matriz nos supondría una complejidad de  $O(n^3)$  si tuviésemos que mirar por cada par de vértices su arista asociada en una lista. En nuestro caso, vamos a utilizar la distancia euclídea, que al ser una operación hace que calcular nuestra matriz sea  $O(n^2)$ . Como veremos más adelante, esta heurística tiene una complejidad de  $O(n^2 \log(n))$ .

#### Procedimiento RADIO(vértices,distancias,radio,P)

**Paso 0** Todos los vértices comienzan sin marcar,  $P = \emptyset$

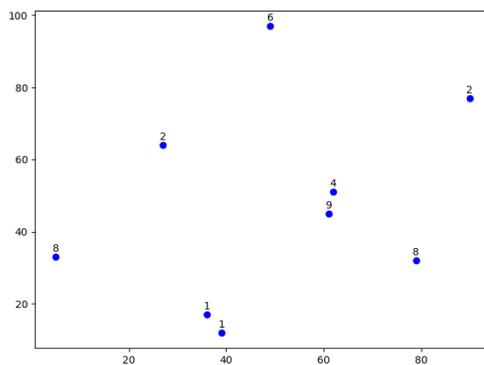


Figura 2.5: Vértices sin marcar, tomamos un radio  $R = \text{radio}$

**Paso 1** Si todos los vértices están marcados, devolver  $P$ . Sino, elegir un vértice no marcado  $u$  con el mayor peso y marcarlo. Añadir al conjunto  $P$  el vértice  $u$  y todos aquellos vértices no marcados  $v$  que cumplan que  $w(v)d(u, v) \leq 2r$ ; volver al paso 1.

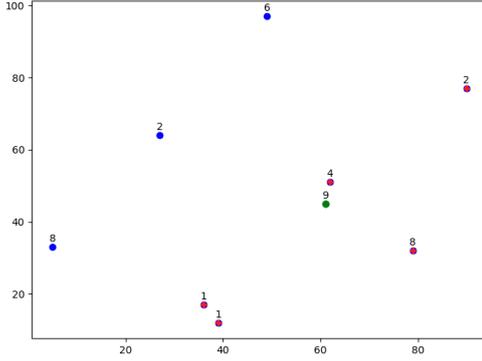


Figura 2.6: Elegimos el de mayor peso y marcamos si  $w(v)d(u, v) \leq 2r$ .

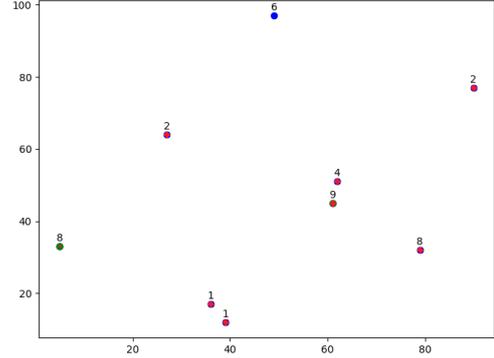


Figura 2.7: Repetimos el proceso con los no marcados.

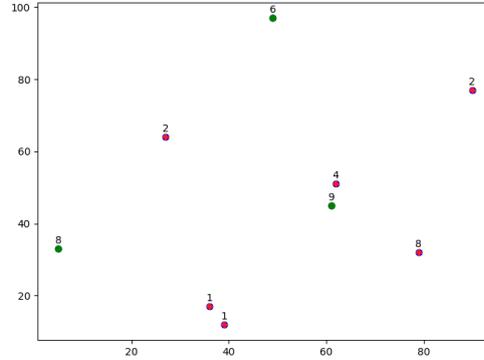


Figura 2.8: Cuando todos los vértices están marcados, devolvemos P.

**Teorema 1.**  $\forall r^* > 0$  si existe un conjunto  $P^* \subseteq V$  con  $r(P^*) \leq r^*$ , entonces el procedimiento RADIO encuentra un conjunto  $P \subseteq \text{con}|P| \leq p$  y  $r(P) \leq 2r^*$ .

*Demostración.* Sea  $P^* = v_1, \dots, v_p$  y sean  $P_1, \dots, P_p$  los conjuntos de vértices que cumplen  $w(v)d(v_i, v) \leq r^*$ . Según el algoritmo tenemos que  $w(v)d(P, v) \leq 2r^*$ , por lo que  $r(P) \leq 2r^*$ . Para demostrar que  $|P| \leq p$  veremos que como mucho solo un vértice de  $P_i$  pertenece a  $P$ . Veamos una ejecución del paso uno, con  $u$  el vértice elegido y  $P_i$  su conjunto. Ahora tenemos que por cada vértice  $v$  no marcado de  $P_i$ ,  $w(v) \leq w(u)$  que por la desigualdad triangular

$$w(v)d(u, v) \leq w(u)(d(u, v_i) + d(v_i, v)) \leq w(u)d(v_i, u) + w(v)d(v_i, v) \leq 2r^*.$$

Por lo que se tienen que etiquetar todos los vértices no etiquetados de  $P_i$ .  $\square$

Este procedimiento se encarga de devolver un numero de localidades que, dado un radio o un radio ponderado si nuestras localizaciones tienen demanda, satisfacen la condición de abastecer a cualquier otra localización con un coste por

lo menos menor o igual que  $2r$ . El problema es que nosotros recibimos un número de localizaciones, no un radio, por lo que esta función por si sola no nos va a servir.

La complejidad de esta función viene dada por tener que comprobar en cada paso todas las distancias con el resto, que en el peor de los casos es  $O(n * n) = O(n^2)$ . Aunque haya que ordenar los pesos para encontrar el máximo, solo hay que hacerlo una vez  $O(n \log(n))$ , por lo que va a delimitar nuestro algoritmo en complejidad es  $O(n^2)$ .

### Heurística

**Paso 1** Calcular todas las distancias ponderadas  $w(v)d(u, v)$  posibles, con  $u, v \in V$ , y almacenarlas en un conjunto en orden estrictamente creciente, eliminando duplicados si los hubiera.

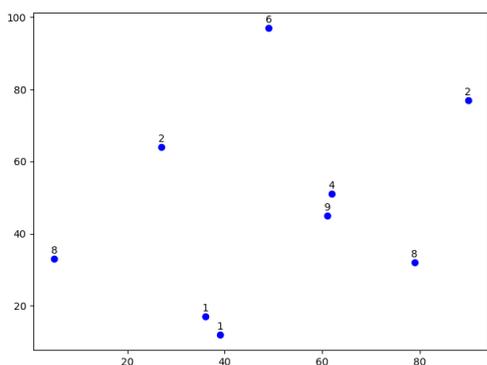


Figura 2.9: Ejemplo con  $p = 4$ .

5.8	53.36	85.58	203.96
34.88	54.74	86.37	239.09
37.53	74.67	91.23	364.32
39.66	76.02	92.64	480.29
39.96	76.41	128.65	488.45
42.80	77.89	161.44	515.44
44.72	79.32	177.62	572.71
45.27	81.04	191.20	592.05
47.85	82.61	191.42	621.32

Figura 2.10: Calculamos distancias ponderadas y ordenamos.

**Paso 2** Encontrar el menor valor de  $w(v)d(u, v)$  para el cual el procedimiento RADIO devuelve un conjunto  $P : |P| \leq p$ .

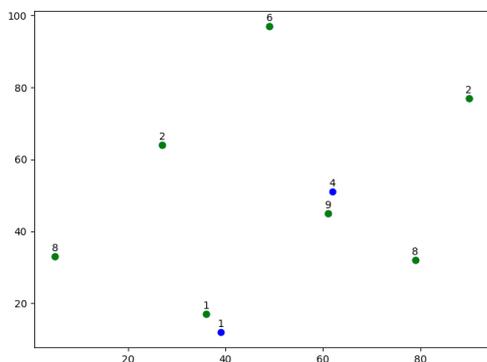


Figura 2.11: Con  $r = 5,8$   $|P| > p$ , seguimos.

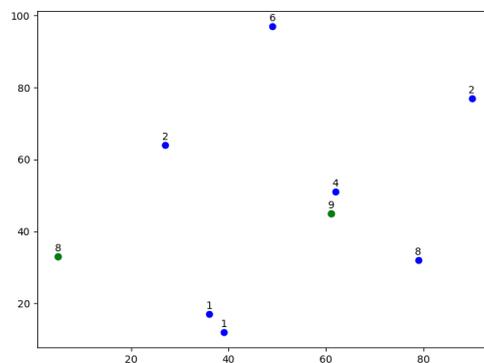


Figura 2.12: Con  $r = 34,88$   $|P| \leq p$ , paramos.

**Paso 3** Añadir al conjunto  $P$  vértices que no pertenezcan ya a  $P$  eligiendo los de más peso hasta que  $|P| = p$ .

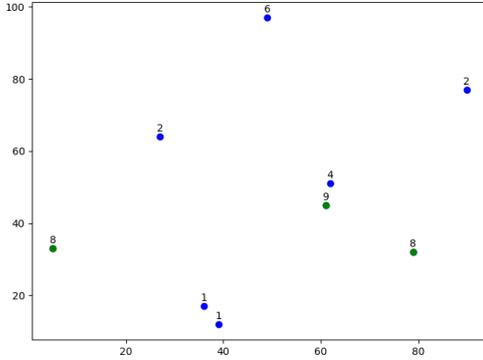


Figura 2.13:  $|P| < p$ , añadimos un vértice (color verde).

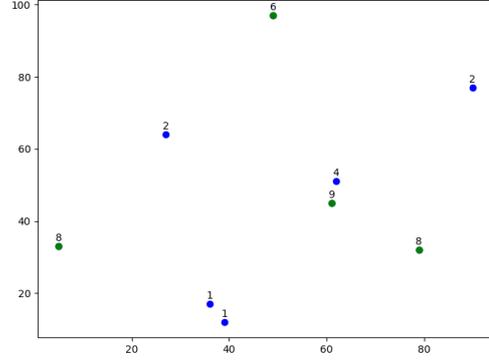


Figura 2.14:  $|P| < p$ , añadimos otro vértice.

Esta sencilla heurística se basa en el procedimiento anterior. Lo que hacemos es buscar la menor distancia ponderada posible para la cual RADIO da una salida con  $|P| \leq p$ . En el primer paso, tenemos que calcular la distancia para cada par de vértices, en total hay  $n^2$  distancias que, al ser ordenadas nos da una complejidad de  $O(n^2 \log(n^2)) = O(2n^2 \log(n)) = O(n^2 \log(n))$ .

La complejidad del segundo paso depende de el numero de veces que ejecutemos la función rango. La intuición nos sugiere que empecemos con el valor más bajo y vayamos ascendiendo de uno en uno hasta dar con el correcto. El problema es que en el peor de los casos tendríamos que recorrer toda la lista ordenada de  $n^2$  elementos, por lo que la complejidad de este paso sería de  $O(n^2 n^2) = O(n^4)$ . Sin embargo, si reflexionamos un poco, vemos que al ser un conjunto ordenado podemos aplicar búsqueda binaria hasta dar con el valor adecuado, por lo que el número de veces que se ejecuta RADIO sería  $\log(n^2) = 2 \log(n)$  que multiplicando por la complejidad de RADIO quedaría que el paso dos tiene una complejidad total de  $O(2 \log(n) n^2) = O(n^2 \log(n))$ .

El paso 3 es trivialmente  $O(n)$ , por lo que la complejidad total de la heurística quedaría:

$$O(n^2 \log(n)) + O(n^2 \log(n)) + O(n) = O(n^2 \log(n)).$$

El método anterior es interesante por su rapidez y aplicación al problema cuando los vértices tienen pesos, pero está limitado al tratar el problema sin pesos. Por ello, vamos a abordar distintas heurísticas que consideramos más aplicables por su naturaleza de trabajar sin pesos.

### 2.4.2. Heurística Sustitución de Vértice por Búsqueda Local

La heurística que se va a plantear ahora está basada en la búsqueda y optimización local y aborda el problema sin pesos. La heurística toma una solución ya existente del problema y comienza a hacer sustituciones para cada uno de los vértices que están en la solución hasta que se encuentra con un mínimo local, presentada en [17].

Antes de dar la heurística completa, veremos dos procedimientos auxiliares en los que se basa nuestro algoritmo. Además, necesitaremos ciertas estructuras que nos facilitarán la implementación:

$c_1(i)$ : Facilidad (vértice perteneciente a la solución) más cercana al vértice  $i$ .

$c_2(i)$ : Segunda facilidad más cercana al vértice  $i$ .

Los datos  $c_1$  y  $c_2$  son fácilmente calculables iterando en cada uno de los  $n$  vértices y comprobando todas sus conexiones con las  $p$  facilidades, por lo que tendríamos  $O(np)$  que en el peor caso sería  $O(n^2)$ . Además, nuestro conjunto de vértices lo denominaremos  $x_{cur}$  y lo ordenaremos de tal forma que  $x_{cur}(i)$  con  $i \in 1, \dots, p$  sean las facilidades o vértices solución y el resto sean los usuarios o vértices que no están en la solución. Además, adelantamos que vamos a hacer un abuso de notación, ya que en vez de referirnos a la facilidad  $x_{cur}(i)$  directamente la llamaremos la facilidad  $i$ .

Necesitaremos de forma auxiliar una matriz de distancias que dados dos vértices nos de su distancia en  $O(1)$  mediante la función  $d(i, j)$  que devuelve la distancia euclídea que hay entre los vértices con índice  $i$  y  $j$ . Como hemos explicado anteriormente, esta estructura se calcula en  $O(n^2)$ .

#### Procedimiento MOVER

Este procedimiento toma un vértice (in) y lo añade a la solución, eliminando el que más beneficie al problema. Utiliza dos estructuras auxiliares que iremos actualizando conforme avanza el algoritmo:

$r(i)$ : Es la distancia máxima de la facilidad  $i$  con cualquiera de sus usuarios.

$z(i)$ : Es la distancia máxima de los usuarios asociados a la facilidad  $i$  si la eliminamos.

**Data:**  $c_1, c_2, \text{vertices}, in, p$

**Result:**  $f, out$

**Inicialización:**

$f \leftarrow 0$   $r(i) \leftarrow 0, z(i) \leftarrow 0, \forall i = 1, \dots, p$

**Añadir facilidad:**

**for**  $i, i > p$  **do**

**if**  $d(i, in) < d(i, c_1(i))$  **then**

$f = \max(f, d(i, in))$

**else**

$r(c_1(i)) = \max(r(c_1(i)), d(i, c_1(i)))$

$z(c_1(i)) = \max(z(c_1(i)), \min(d(i, in), d(i, c_2(i))))$

**end**

**end**

**Mejor eliminación:**

$g_1 = \max\{r(l) | l = 1, \dots, p\}$ , guardar en  $l^*$  el índice correspondiente

$g_2 = \max_{l \neq l^*} \{r(l) | l = \{1, \dots, p\}\}$

Sea

$$g(l) = \begin{cases} \max(f, z(l), g_1) & \text{si } l \neq l^*, \\ \max(f, z(l), g_2) & \text{si } l = l^* \end{cases}$$

devolveremos  $f = \min\{g(l) | l = 1, \dots, p\}$  y  $out$  su índice correspondiente.

Algoritmo 1: Procedimiento MOVER

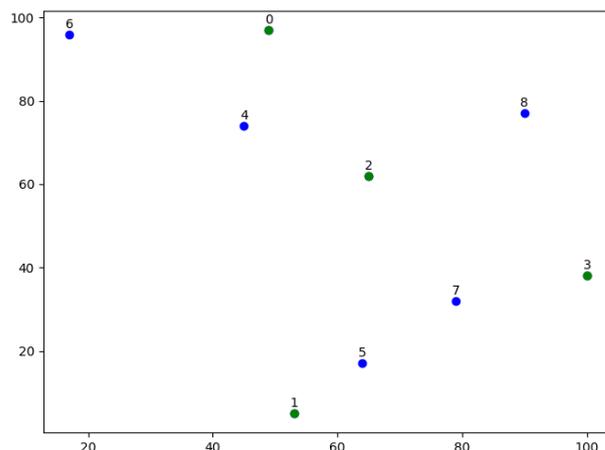
Como el paso 1 tiene complejidad  $O(1)$ , el paso 2  $O(n)$  y el paso 3  $O(n)$  la complejidad global del procedimiento MOVER se queda en  $O(n)$ . De forma simplificada este algoritmo hace lo siguiente:

**Inicialización** Creamos nuestras variables inicializadas a 0  $f = 0, r(i) = 0, z(i) = 0, \forall i = 1, \dots, p$

**Añadir facilidad** Para cada usuario  $i$  que no sea una facilidad haremos lo siguiente: Si la distancia de ese usuario al vértice añadido ( $in$ ) es menor que a su facilidad más cercana actualizamos  $f$ . Si no, actualizamos  $r$  y  $z$ .

**Mejor eliminación** Buscamos la mayor distancia de un usuario a una facilidad de la solución original, o sea, el máximo valor de  $r(i)$  y también el segundo mayor, que denominaremos  $g_1$  y  $g_2$ . Ahora utilizando  $g_1$  y  $g_2$  buscamos cual es la facilidad que menos perjudica a la solución gracias a la información de  $z(i)$ .

Comprobemos como se comporta este procedimiento con un ejemplo sencillo. En este caso supondremos que la facilidad que entra ( $in$ ) es la 5 y que previamente hemos calculado  $c_1$  y  $c_2$ .



Vemos como el paso de añadir facilidad calcula los datos necesarios  $f$ ,  $r$  y  $z$  teniendo en cuenta la nueva facilidad que entra a la solución ( $in$ ).

i	Acción	f	r	z
4	r[2] y z[2]	0	0, 0, 23.32, 0	0, 0, 23.35, 0
5	f	0	0, 0, 23.32, 0	0, 0, 23.35, 0
6	r[0] y z[0]	0	32.02, 0, 23.32, 0	58.82, 0, 23.35, 0
7	f	21.21	32.02, 0, 23.32, 0	58.82, 0, 23.35, 0
8	r[2] y z[2]	21.21	32.02, 0, 29.15, 0	58.82, 0, 40.26, 0

Tabla 2.1: Evolución de variables en el paso de añadir facilidad.

Una vez tenemos las variables auxiliares necesarias, ya podemos completar el verdadero objetivo del algoritmo, elegir que vértice debe ser borrado de la solución para beneficiarla al máximo. Calculamos las variables  $l^*$ ,  $g1$  y  $g2$

$l^*$	$g1$	$g2$
0	32.02	29.15

que representan la mayor y la segunda mayor distancia de una de las facilidades que ya estaban en la solución a su usuario más lejano. Calculamos ahora  $g(l)$

$l$	0	1	2	3
$g(l)$	58.82	32.02	40.26	32.02

que representa el valor que tendría nuestra función objetivo si eliminásemos la facilidad  $l$ . Ahora simplemente queda elegir el mínimo y su facilidad asociada:

$f$	$out$
32,02	1

Una vez comprendido el procedimiento mover, veamos el siguiente algoritmo auxiliar necesario antes de ver la heurística completa. La función de éste será la de actualizar las estructuras necesarias para cada iteración de la heurística y, aunque sea mas largo que el anterior, es a su vez más sencillo.

### Procedimiento ACTUALIZAR

Este procedimiento se encarga de actualizar las estructuras  $c_1$  y  $c_2$  una vez los vértices de nuestra solución han sido alterados por MOVER. Funciona iterando sobre todos los usuarios y comprobando los casos posibles que puedan afectar a cada uno.

Necesitaremos para este algoritmo una función auxiliar  $BuscarSegundaFacilidad(v)$ , que toma un vértice y devuelve la segunda facilidad más cercana. Ésta puede ser implementada de forma sencilla iterando sobre los  $n-1$  vértices restantes, por lo que su complejidad es  $O(n)$ .

**Data:** vertices, in, out, p, c1, c2

**Result:** c1, c2

```

for  $i$ ,  $i > p$  do
  if  $c_1(i) = out$  then
    if  $d(i, in) \leq d(i, c_2(i))$  then
       $c_1(i) = out$ 
    else
       $c_1(i) = c_2(i)$   $c_2(i) = BuscarSegundaFacilidad(i)$ 
    end
  else
    if  $d(i, c_1(i)) > d(i, in)$  then
       $c_2(i) = c_1(i)$   $c_1(i) = in$ 
    else
      if  $d(i, in) < d(i, c_2(i))$  then
         $c_2(i) = in$ 
      else
        if  $c_2(i) = out$  then
           $c_2(i) = BuscarSegundaFacilidad(i)$ 
        end
      end
    end
  end
end

```

Algoritmo 2: Procedimiento ACTUALIZAR

Este procedimiento es fácil ver que tiene complejidad  $O(n^2)$ , ya que en el peor de los casos ejecutamos la función *BuscarSegundaFacilidad* ( $O(n)$ )  $n$  veces. Su funcionamiento es sencillo de entender ya que simplemente comprueba los posibles casos ante un cambio de facilidad:

- **Caso A:** La facilidad eliminada es la más cercana a  $i$ .
  - **Caso A1:** La facilidad entrante está más cerca de  $i$  que su segunda facilidad más cercana anterior.
  - **Caso A2:** La facilidad entrante no está más cerca de  $i$  que su segunda facilidad más cercana anterior.
- **Caso B:** La facilidad eliminada no es la más cercana a  $i$ .
  - **Caso B1:** La facilidad entrante está más cerca de  $i$  que su facilidad más cercana anterior.
  - **Caso B2:** La facilidad entrante no está mas cerca de  $i$  que su facilidad más cercana anterior.
    - **Caso B2a:** La facilidad entrante está más cerca de  $i$  que su segunda facilidad más cercana anterior.
    - **Caso B2b:** La facilidad entrante no está mas cerca de  $i$  que su segunda facilidad más cercana anterior.
      - ◇ **Caso i:** La facilidad eliminada es la segunda más cercana a  $i$ .
      - ◇ **Caso ii:** La facilidad eliminada no es la segunda más cercana a  $i$ .

Una vez identificados el caso en el que estamos para cierto vértice  $i$ , hacemos las modificaciones necesarias para actualizar sus facilidades cercanas. Las posibles acciones a realizar son bastante intuitivas, pero vamos a comentarlas para mayor claridad:

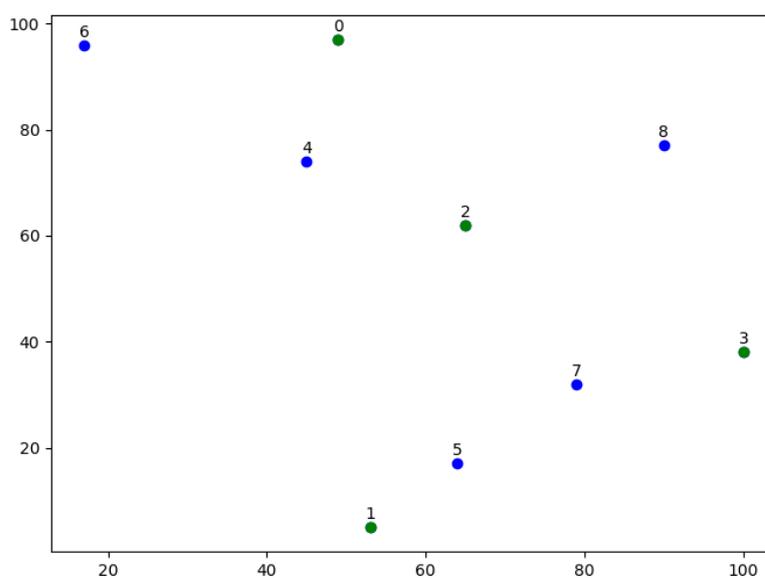
- **Caso A1:** Como hemos eliminado  $c_1(i)$  y la facilidad entrante es la más cercana, simplemente asignamos la nueva facilidad a  $c_1(i)$ .
- **Caso A2:** Hemos eliminado  $c_1(i)$  pero la facilidad entrante no es la más cercana, por lo que la antigua segunda facilidad más cercana pasa a ser la primera y hacemos una búsqueda para encontrar la segunda facilidad más cercana.
- **Caso B1:** Ahora no hemos eliminado  $c_1(i)$  y la facilidad entrante está más cerca que  $c_1(i)$ , asignamos  $c_1(i)$  a  $c_2(i)$  y la entrante a  $c_1(i)$ .
- **Caso B2a:** No hemos eliminado  $c_1(i)$  y la facilidad entrante está más cerca que la antigua segunda facilidad más cercana ( $c_2(i)$ ) y mas lejos que  $c_1(i)$ , por lo que asignamos a  $c_2(i)$  la nueva facilidad.

- **Caso i:** Hemos eliminado la segunda facilidad más cercana y la entrante está más lejos que ésta, por lo que hacemos una búsqueda para encontrar  $c_2(i)$ .
- **Caso ii:** La facilidad eliminada no es ni  $c_1(i)$  ni  $c_2(i)$  y la entrante está más lejos que  $c_2(i)$ , por lo que no hacemos nada.

Veamos como se actualizaría nuestra estructura en el caso anterior caso, en el que la facilidad entrante es la 5 y la saliente la 1.

Vértice	Caso	c1	c2
0	B2bii	0, 1, 2, 3, 2, 1, 0, 3, 2	2, 3, 0, 2, 0, 3, 2, 2, 3
1	A1	0, 5, 2, 3, 2, 1, 0, 3, 2	2, 3, 0, 2, 0, 3, 2, 2, 3
2	B2bii	0, 5, 2, 3, 2, 1, 0, 3, 2	2, 3, 0, 2, 0, 3, 2, 2, 3
3	B2a	0, 5, 2, 3, 2, 1, 0, 3, 2	2, 3, 0, 5, 0, 3, 2, 2, 3
4	B2bii	0, 5, 2, 3, 2, 1, 0, 3, 2	2, 3, 0, 5, 0, 3, 2, 2, 3
5	A1	0, 5, 2, 3, 2, 5, 0, 3, 2	2, 3, 0, 5, 0, 3, 2, 2, 3
6	B2bii	0, 5, 2, 3, 2, 5, 0, 3, 2	2, 3, 0, 5, 0, 3, 2, 2, 3
7	B1	0, 5, 2, 3, 2, 5, 0, 5, 2	2, 3, 0, 5, 0, 3, 2, 3, 3
8	B2bii	0, 5, 2, 3, 2, 5, 0, 5, 2	2, 3, 0, 5, 0, 3, 2, 3, 3

Tabla 2.2: Evolución de las variables c1 y c2.



## Heurística

Asimilados estos algoritmos de apoyo, ya podemos comenzar a explicar la heurística que da nombre a esta sección. Como hemos observado, estos procedimientos operan de manera local, así que nuestra heurística va a intentar aplicarlos a toda la solución original.

**Data:** vértices, distancias,  $p$

**Result:** vértices

**Inicialización:**

Tomar cualquier permutación de vértices y asumir que los  $p$  primeros son las facilidades.

Calcular  $c_1$  y  $c_2$ .

Calcular  $f_{opt}$  e  $i^*$  su índice asociado.

**Paso iterativo:**

$f^* = \infty$

**for**  $in = p, in < n$  **do**

**if**  $d(i^*, in) < d(i^*, c_1(i^*))$  **then**

$f, out = \text{MOVER}(c_1, c_2, \text{distancias}, in, \text{vértices}, p)$

**if**  $f < f^*$  **then**

$f^* = f$

$in^* = in$

$out^* = out$

**end**

**end**

**end**

**Condición de parada:**

**if**  $f^* \geq f_{opt}$  **then**

        | Devolver vértices y finalizar.

**end**

**Paso de actualización.**

$f_{opt} = f^*$ .

    Calcular nuevo  $i^*$ .

    Actualizar vértices intercambiando  $in^*$  por  $out^*$  (el vértice asociado a  $in^*$  pasa a estar asociado a  $out$  y viceversa).

$c_1, c_2 = \text{ACTUALIZAR}(in^*, out^*, \text{vértices}, p, c_1, c_2)$

    Volver a Paso iterativo.

Algoritmo 3: Heurística de sustitución de vértice local

Vamos a explicar ahora detalladamente cada una de las partes de nuestra heurística.

**Inicialización:**

Nuestra heurística se basa en la mejora y optimización local, por lo que necesitamos una solución inicial que trabajar que elegiremos de forma aleatoria. Decidimos seleccionarla tomando una permutación cualquiera y eligiendo los primeros  $p$  vértices como facilidades, que desde el punto de vista teórico es igual que tomar  $p$  vértices de forma aleatoria pero a efectos prácticos nos aligera sutilmente los algoritmos.

Calcular  $c_1$  y  $c_2$  es sencillo. Como hemos comentado anteriormente, iteramos sobre nuestros vértices y comprobamos las distancias con sus facilidades guardándonos las dos menores. (ver algoritmo 4).

Para conseguir los datos  $f_{opt}$  e  $i^*$  debemos calcular  $f_{opt} = \max(d(j, c_1(j)))$  con  $j \in (0, \dots, n-1)$  e  $i^*$  el  $j$  asociado, que obtenemos simplemente iterando sobre  $c_1$ .

```

for  $i = 0$  to  $n - 1$  do
   $c1[i] \leftarrow -1$ 
   $c2[i] \leftarrow -1$ 
   $min1 \leftarrow \infty$ 
   $min2 \leftarrow \infty$ 

  for  $j = 0$  to  $p - 1$  do
    if  $d[i][j] < min1$  then
       $min2 \leftarrow min1$ 
       $c2[i] \leftarrow c1[i]$ 
       $min1 \leftarrow d[i][j]$ 
       $c1[i] \leftarrow j$ 
    end
    else if  $d[i][j] < min2$  then
       $min2 \leftarrow d[i][j]$ 
       $c2[i] \leftarrow j$ 
    end
  end
end

```

Algoritmo 4: Encontrar  $c_1$  y  $c_2$ **Paso iterativo:**

Este paso recorre todos los vértices que no pertenecen a la solución y comprueba para cada vértice  $i$  si al tomarlo como facilidad podría mejorar localmente la situación, es decir, si pasaría a ser la facilidad más cercana de  $i^*$ . En tal caso, ejecuta nuestro procedimiento MOVER que hemos explicado anteriormente tomando como facilidad entrante  $i$  y devolviendo el posible nuevo valor de  $f$  y la facilidad saliente.

La parte de la asignación  $f^* = \infty$  junto con la comprobación que hay después de

MOVER se utilizan para guardar cuál es el vértice añadido y eliminado que ha dado un mejor resultado para intentar mejorar la solución.

**Condición de parada:**

Cuando no hemos conseguido mejorar nuestra función objetivo paramos, sino la heurística no dejaría de ejecutarse.

**Paso de actualización:**

Si hemos llegado a este punto, es que hemos conseguido mejorar nuestra función objetivo, por lo que ahora tenemos que actualizar las estructuras y variables necesarias para seguir optimizando la solución.

En primer lugar tomaremos como nuestra nueva condición de parada el valor de  $f^*$  que sabemos que es menor que la anterior y actualizaremos  $i^*$  de igual forma que en la inicialización.

Ahora modificamos nuestra solución, que en nuestro caso la estábamos almacenando en los primeros  $p$  vértices de la solución, por lo que para mantener nuestra estructura debemos intercambiar el vértice  $in^*$  con  $out^*$ .

Como el entorno ha cambiado y seguimos necesitando hacer uso de  $c_1$  y  $c_2$ , llega el momento de utilizar nuestro algoritmo auxiliar ACTUALIZAR. Éste nos va a devolver los  $c_1$  y  $c_2$  acordes con nuestra nueva solución.

Dado que todavía no hemos encontrado un máximo local, volvemos al paso iterativo con las estructuras y valores actualizados.

Veamos cual es la complejidad de nuestra heurística analizando cada una de las secciones.

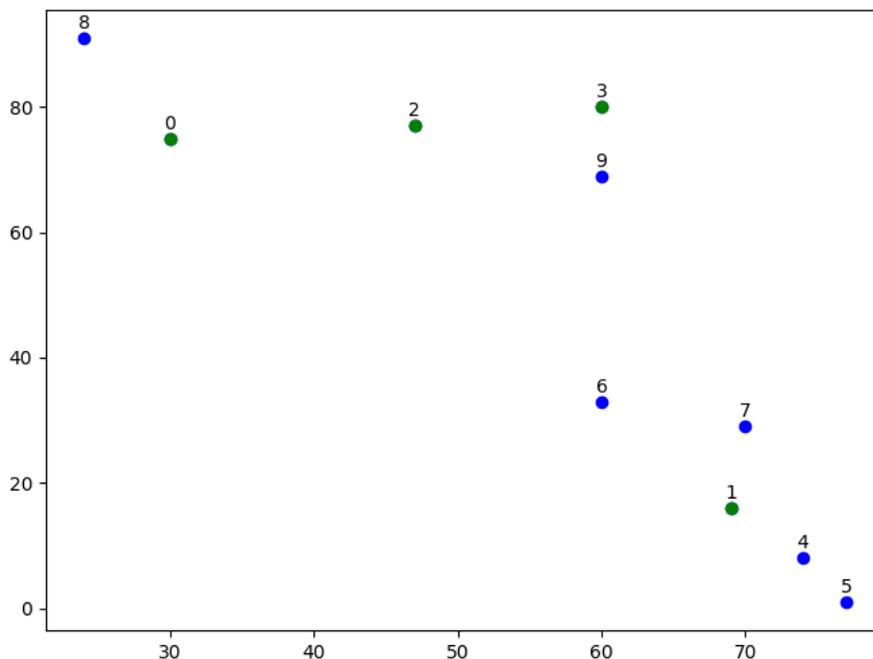
Para el paso de inicialización sabemos que la complejidad de calcular  $c_1$  y  $c_2$  es  $O(np)$  y que la complejidad de calcular  $f_{opt}$  es  $O(n)$  por lo que la complejidad de Inicialización es  $O(np)$ .

El paso iterativo ejecuta  $n - p$  veces el algoritmo MOVER, cuya complejidad previamente hemos deducido que es  $O(n)$ , por lo que la complejidad de este paso es  $O((n - p)n) = O(n^2)$ .

Condición de parada es trivialmente constante por lo que vamos directamente con el paso de actualización. Calcular el nuevo  $i^*$  es, al igual que en la inicialización,  $O(n)$  e intercambiar los vértices es  $O(1)$ , por lo que va a marcar la complejidad de este paso es ACTUALIZAR, con  $O(n^2)$ .

Una vez tenemos la complejidad de cada una de las partes, debemos contar cuantas veces se van a ejecutar, ya que recordemos que esta heurística se sigue ejecutando hasta cumplir la condición de parada. Las partes relevantes que se repiten son el paso iterativo y el de actualización, que ambos son  $O(n^2)$  por lo que nuestra complejidad global va a ser  $O(k * n^2)$  donde  $k$  es el número de iteraciones máximas hasta llegar a una parada.

Analicemos un ejemplo de funcionamiento para comprender mejor como se comporta esta heurística. Empezaremos con una permutación de un grupo de puntos aleatoria y tomaremos  $n = 10$  y  $p = 4$ :



Calculamos su función objetivo asociada  $f_{opt} = 19,24$  y su  $i^* = 6$  vinculado.

Ahora el algoritmo va comenzar a iterar sobre todos los vértices que no son facilidades hasta que encuentre un posible candidato para hacer la sustitución, que son aquellos que al convertirse en facilidad mejoran la situación del vértice  $i^*$ . Nuestra heurística encuentra este primer candidato y ejecuta el algoritmo MOVER para encontrar cuál es la mejor eliminación posible.

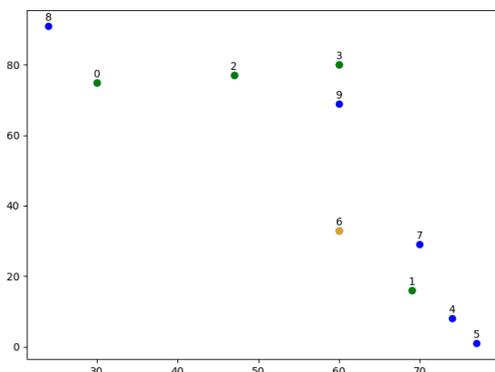


Figura 2.15: Posible candidato.

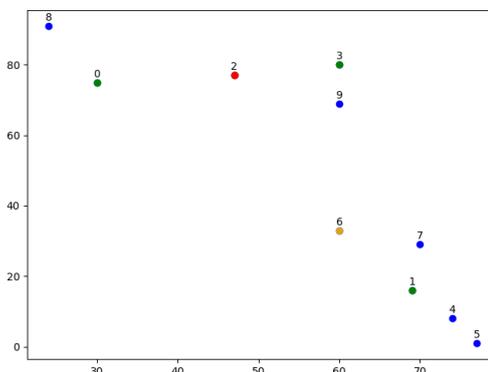


Figura 2.16: Eliminación potencial.

Además, guardamos el valor  $f = 17,09$  de nuestra nueva posible solución para

más adelante decidir si estamos mejorando el problema o no. El algoritmo sigue explorando el resto de vértices, pero ahora en caso de encontrar otro candidato, solo se va a quedar con él si supera al anterior, es decir, si consigue un valor para  $f$  menor que el otro candidato.

La heurística encuentra otro aspirante que mejora la situación de  $i^*$

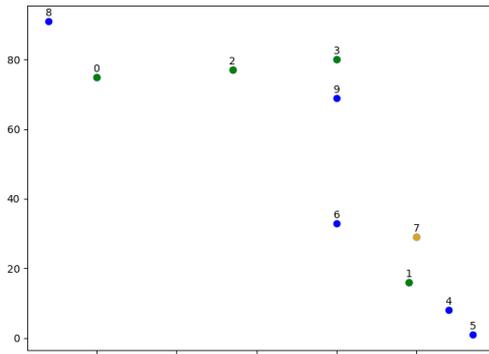


Figura 2.17: Nuevo candidato.

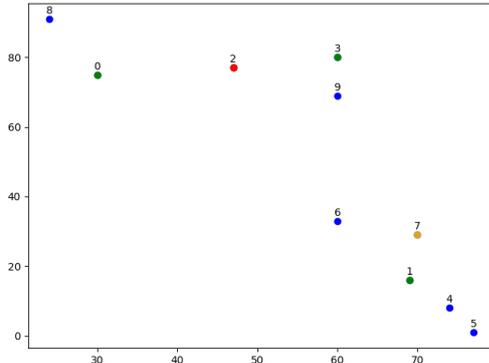


Figura 2.18: Eliminación potencial.

pero resulta que su función objetivo asociada ( $f = 17,09$ ) es igual que la del anterior candidato, por lo que al no mejorar descartamos este aspirante y nos quedamos con el primero.

La heurística termina de recorrer todos los vértices restantes sin encontrar ninguno que supere al primer candidato, por lo que termina el paso iterativo con la situación que se muestra en la figura 2.19.

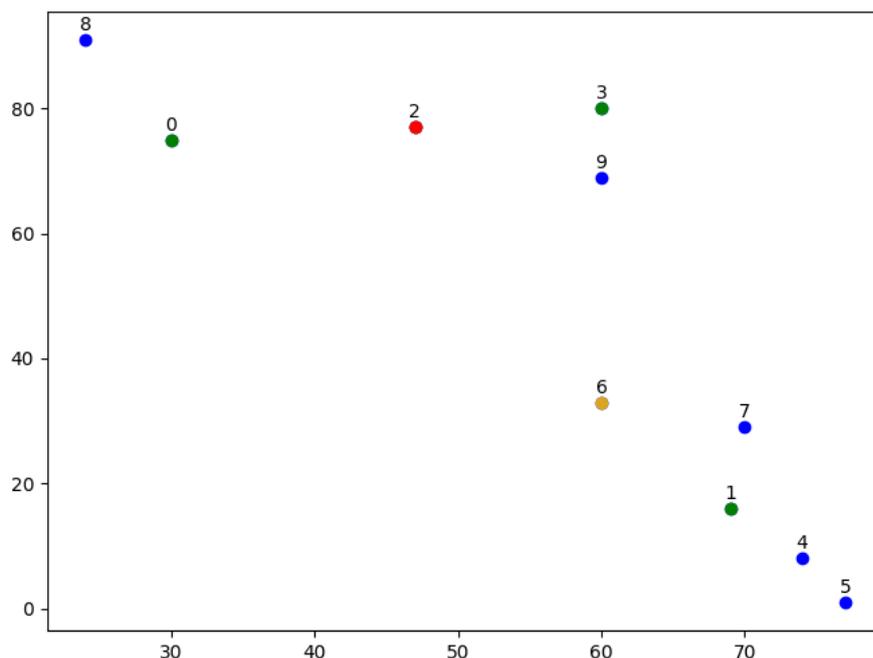


Figura 2.19:  $f = 17,09$ ,  $in = 6$ ,  $out = 2$ .

Ahora debemos hacer la comprobación más importante, ¿hemos mejorado la solución original? Para eso existe nuestra condición de parada, que va a comprobar si la nueva  $f$  calculada es menor que la de la anterior solución,  $f_{opt}$ . En este caso tenemos que  $f = 17,09$  y que  $f_{opt} = 19,24$ , por lo que hemos mejorado y continuamos ejecutando la heurística.

De forma intuitiva, lo que queremos hacer a continuación es volver a ejecutar lo que hemos aplicado hasta ahora pero con nuestra nueva situación. Antes de ello debemos actualizar las estructuras y los datos pertinentes a nuestro nuevo paradigma:

La heurística guarda  $f_{opt} = f = 17,09$  y calcula el nuevo  $i^* = 8$  asociado, el vértice más "problemático". Ahora intercambiamos las posiciones de los vértices  $in$  y  $out$  para que el algoritmo mantenga en las cuatro primeras posiciones de nuestros vértices las facilidades (en la gráfica no cambia el número asociado para sea más fácil seguir el proceso, pero a nivel interno si que han cambiado).

Indice	0	1	2	3	4	5	6	7	8	9
vértices	0	1	2	3	4	5	6	7	8	9

Tabla 2.3: Variable *vértices* antes.

Índice	0	1	2	3	4	5	6	7	8	9
vértices	0	1	6	3	4	5	2	7	8	9

Tabla 2.4: Variable *vértices* actualizada.

Finalmente  $c_1$  y  $c_2$  son actualizadas utilizando el procedimiento ACTUALIZAR:

Índice	0	1	2	3	4	5	6	7	8	9
$c_1$	0	1	2	3	1	1	1	1	0	3
$c_2$	2	3	3	2	3	3	2	3	2	2

Tabla 2.5: Variables  $c_1$  y  $c_2$  antes.

Índice	0	1	2	3	4	5	6	7	8	9
$c_1$	0	1	3	3	1	1	6	6	0	3
$c_2$	6	6	0	6	6	6	1	1	3	0

Tabla 2.6: Variables  $c_1$  y  $c_2$  actualizadas.

Nuestro paradigma con el que vamos a seguir trabajando es el siguiente:

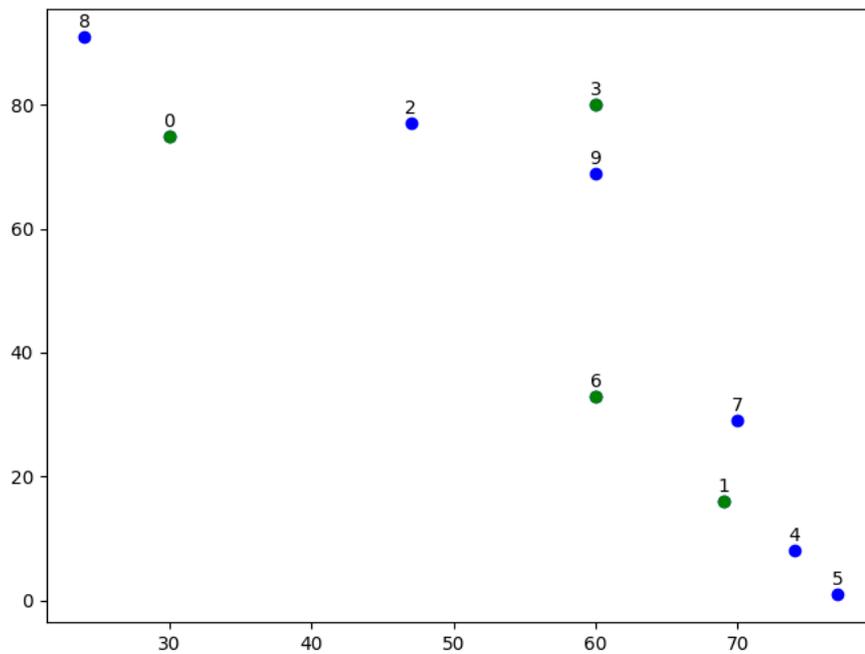


Figura 2.20: La nueva solución a optimizar.

La heurística ahora vuelve al paso iterativo y repite el procedimiento anterior de buscar al candidato más óptimo. El primero que encuentra es el siguiente:

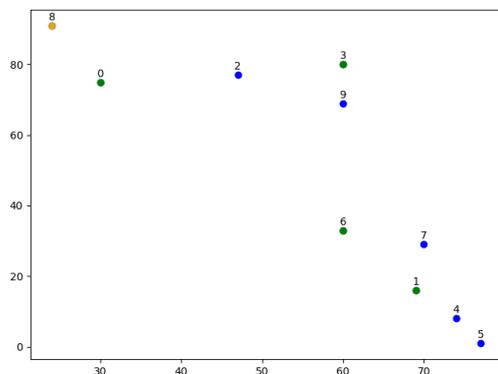


Figura 2.21: Candidato.

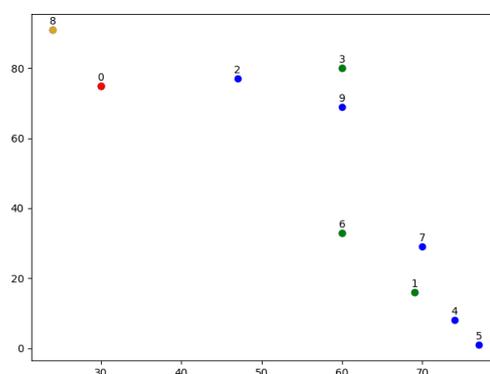


Figura 2.22: Eliminación potencial.

que a su vez es el que acaba eligiendo como candidato final ya que no encuentra ninguno mejor que él. Procede a comprobar si la nueva posible solución mejora la función objetivo. Como  $f = 17 < f_{opt} = 17,09$ , la heurística procede a actualizar las estructuras y valores necesarios para finalmente encontrarnos en este paradigma:

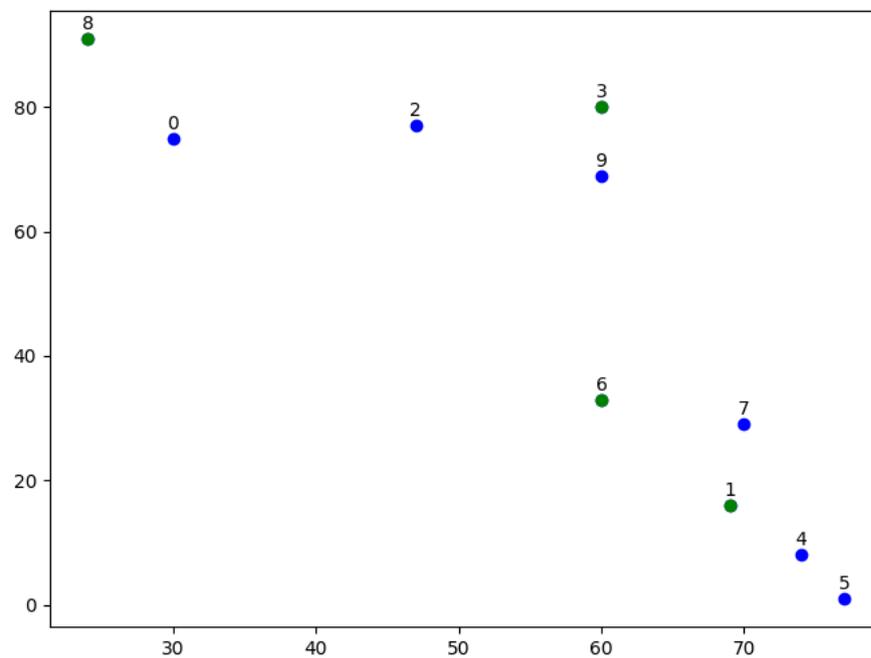


Figura 2.23: Volvemos a tener nueva solución a optimizar.

Tras volver al paso iterativo, busca de nuevo al candidato más óptimo, que en este caso es:

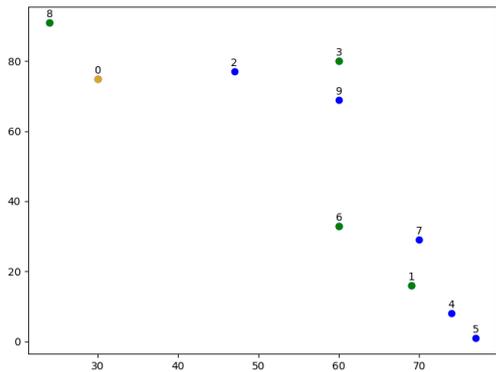


Figura 2.24: Candidato.

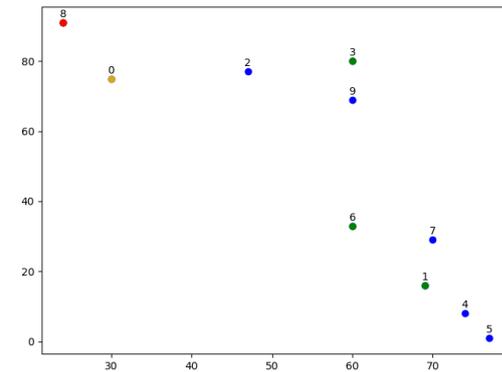


Figura 2.25: Eliminación potencial.

Ahora vuelve a hacer la comprobación de la condición de parada, donde efectivamente calcula que  $f = 17,09 > f_{opt} = 17$ . Como no ha habido mejora. termina su ejecución y deja la solución final así:

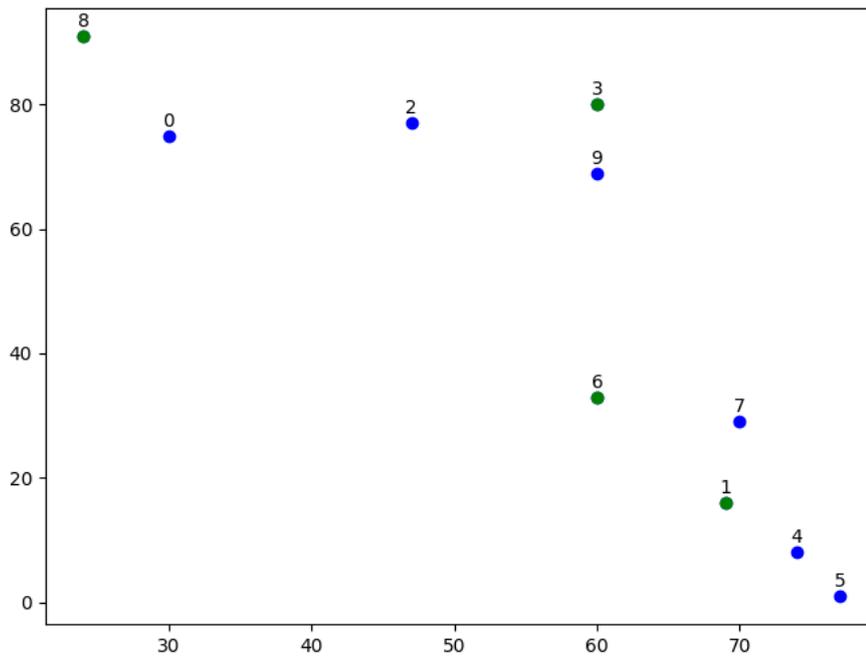


Figura 2.26: Salida final de nuestra heurística.

Debemos mencionar un detalle importante respecto al funcionamiento de esta heurística. Como podemos observar, el algoritmo utilizado necesita hacer uso de la estructura  $c_2$ , que como recordaremos almacena la segunda facilidad más cercana, necesaria sobre todo por el procedimiento ACTUALIZAR. Esto implica que para el correcto funcionamiento necesitaremos  $p \geq 2$  a no ser que realicemos algunas modificaciones pertinentes que no han sido implementadas debido a que ya hay

algoritmos exactos que abordan el problema con  $p = 1$  en tiempo polinomial, por lo que agregarlas a esta heurística sería intrascendente.

### 2.4.3. Heurística Sustitución de Vértice por Búsqueda Tabú Local

Vamos a presentar una modificación de la heurística anterior que va a tratar de evitar quedarse en máximos locales mediante el uso de una lista tabú. Debido a la alta similitud entre este algoritmo y el anterior, no vamos a explicar con tanto detalle el funcionamiento de ésta y nos vamos a centrar en entender las modificaciones.

#### Lista tabú

En primer lugar, veamos qué es una lista tabú con ayuda del ejemplo anterior. En nuestro algoritmo original, a la hora de decidir que posibles candidatos podían entrar a ser nuevas facilidades, simplemente mirábamos todos aquellos que no lo eran ya y aplicábamos un criterio para elegirlo. Como guardamos todos los vértices en una lista en los que los  $p$  primeros son las facilidades, teníamos ésta situación:

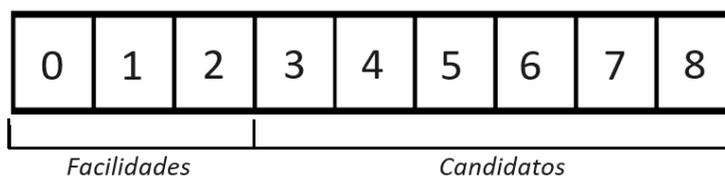


Figura 2.27: Paradigma anterior con  $p=3$ .

Ahora vamos a introducir una lista tabú (realmente va a ser una parte de la lista) en la que recogeremos a parte de los vértices que no pertenecen a la solución. Éstos vértices no podrán ser seleccionados como candidatos a entrar a la solución de forma directa, por lo que el nuevo paradigma que proponemos es el siguiente:

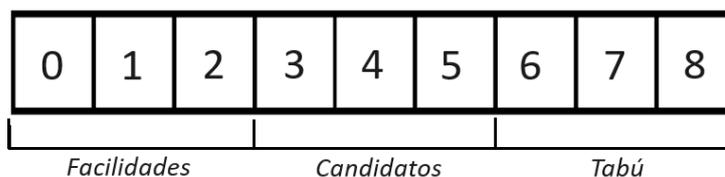


Figura 2.28: Nueva situación con  $p=3$ .

En resumen, si antes nuestra heurística tenía como posibles candidatos los vértices 3,4,5,6,7,8 ahora vamos a restringirlos a 3,4,5.

### Sustitución en cadena

La lista tabú por si sola no tiene mucho sentido ya que simplemente estaríamos restringiendo nuestro vecindario sin ningún tipo de beneficio. Por ello, necesitamos algún modo de actualizarla si queremos conseguir explorar las máximas posibilidades.

**Result:** Sustitución en cadena en la lista  $x$

**Data:**  $index, out, cn, x$

```

aux ← x[out]
x[out] ← x[index]
x[index] ← x[cn]
x[cn] ← aux
return x
    
```

Algoritmo 5: Sustitución en cadena

El algoritmo que hemos presentado realiza una sustitución secuencial entre la facilidad saliente, el candidato entrante y el elemento tabú. La idea es conseguir que todos los vértices pasen por el estado tabú en algún momento para que la heurística pueda escapar de máximos locales. En la siguiente esquema podemos ver representada la sustitución.

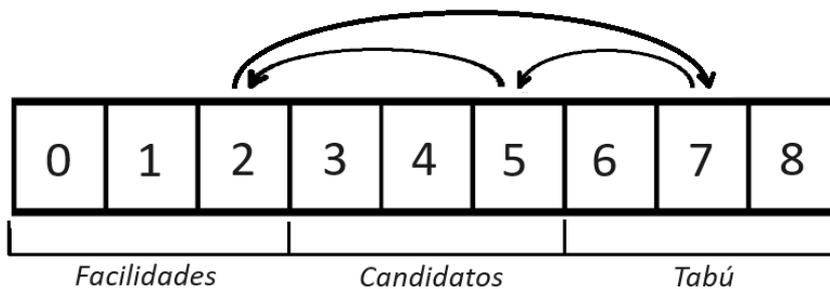


Figura 2.29: Sustitución en cadena con  $out=2$ ,  $in=5$  y  $cn=7$ .

Como podemos observar, el elemento que deja de ser tabú viene dado por el contador  $cn$ , cuando veamos la heurística completa veremos que éste será actualizado en cada iteración.

## Heurística

Asimilados estos dos conceptos de apoyo, ya podemos explicar la heurística. Este algoritmo 6 se basa en la anterior heurística pero con ciertas modificaciones para restringir su vecindario en cada iteración; lo que de forma paradójica hace que pueda salir de máximos locales.

Aunque parezca que se ha complicado mucho el algoritmo, la mayoría simplemente son modificaciones lógicas de la heurística anterior.

En el paso de inicialización, hacemos dos cambios, el primero, duplicamos las estructuras creadas. Esto tiene un motivo que podemos razonar si miramos a la siguiente línea; la heurística termina transcurrido un tiempo determinado. Por ello, debemos guardar la mejor solución encontrada hasta el momento en caso de que termine explorando peores soluciones.

El segundo es que añadimos una variable  $t$  que marca el tamaño de la lista tabú y *mejora* que se utiliza para dar a la heurística todavía más posibilidades de explorar nuevos vecindarios.

El paso iterativo comienza eligiendo los posibles candidatos a pasar a ser nueva solución, es aquí donde se utiliza nuestra variable *mejora*. Tenemos dos casos:

- *Mejora = True*: Esto implica que todavía podemos mejorar la solución en el vecindario en el que estamos, por lo que buscamos candidatos de la misma forma que en la anterior heurística excluyendo los tabú. En caso de no encontrar, saltamos al siguiente caso.
- *Mejora = False*: Si no hemos conseguido mejorar la solución en la anterior iteración, abrimos al máximo nuestro vecindario para tener mejores posibilidades de explorar otros (excluyendo siempre los elementos tabú).

La siguiente parte funciona igual que anteriormente, busca el mejor candidato a entrar y el mejor a salir. Si ha encontrado una mejor solución la guarda y lo único que cambia es que actualiza la variable *mejora* a *True* y si no la ha encontrado, *mejora = False*.

En la sección de actualizar, ahora aplicamos la sustitución en cadena en vez de un intercambio normal como hacíamos antes. Finalmente, actualizamos el contador que elige que elemento tabú debe salir.

**Data:** vértices, distancias,  $p$ , tiempo total

**Result:**  $x_{opt}$

**Inicialización:**

Tomar cualquier permutación de vértices  $x_{opt}$  y asumir que los  $p$  primeros son las facilidades.

Calcular  $c_1$  y  $c_2$ .

Calcular  $f_{opt}$  e  $i^*$  su índice asociado.

$c_{1cur}, c_{2cur} = c_1, c_2$ .

$f_{cur}, x_{cur}, i_{cur}^* = f_{opt}, x_{opt}$ .

$cn = n - 1$ .

$mejora = True$

$t = (n - p) // 2$

**Paso iterativo:**

**while** tiempo transcurrido < tiempo total **do**

    // Restringir el vecindario.

$J = \{j | d(i_{cur}^*, j) < f_{cur}\} \setminus tabu$

**if** not mejora or  $J = \emptyset$  **then**

        |  $J = [p, n - t - 1]$

**end**

    // Mejor solución del vecindario:

$f^* = \infty$

**for**  $in \in J$  **do**

        |  $out, f = \text{MOVER}(x_{cur}, c_{1cur}, c_{2cur}, \text{distancias}, in, p)$

        | **if**  $f < f^*$  **then**

            |  $f^* = f$   $in^* = in$   $out^* = out$

        | **end**

**end**

    // Mejora

**if**  $f^* < f_{opt}$  **then**

        |  $x_{aux} = x_{cur}$

        | Swap  $x_{aux}[in^*]$  with  $x_{aux}[out^*]$

        |  $f_{opt} = f^*$   $mejora = True$   $x_{opt} = x_{aux}$

**end**

**else**

        |  $mejora = False$

**end**

    // Actualizar

$x_{cur} = \text{SUSTITUCIÓN EN CADENA}(i^*, out^*, cn, x_{cur})$

$c_{1cur}, c_{2cur} = \text{ACTUALIZAR}(\text{distancias}, out^*, cn, x_{cur}, p)$

$cn = cn - 1$

**if**  $cn == (n - t - 1)$  **then**

        |  $cn = n - 1$

**end**

    Calcular  $f_{cur}$  e  $i_{cur}^*$  su índice asociado.

**end**

Algoritmo 6: Heurística Sustitución de Vértice por Búsqueda Tabú Local

Veamos ahora un ejemplo de ejecución para comprender mejor el funcionamiento. Comenzamos ejecutando el algoritmo con esta solución inicial:

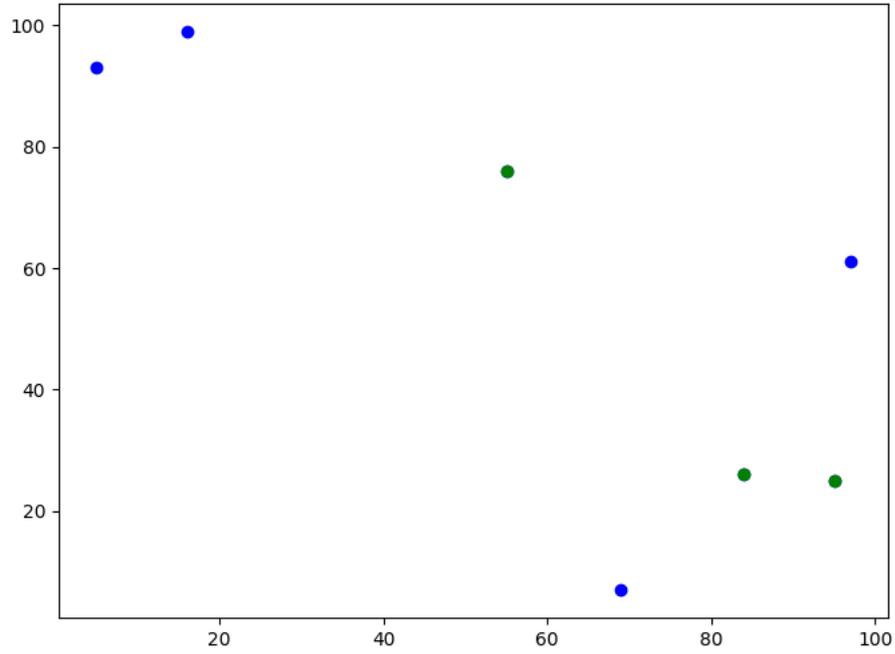


Figura 2.30: Situación inicial.

La heurística se ejecuta y explora soluciones y se queda con la mejor:

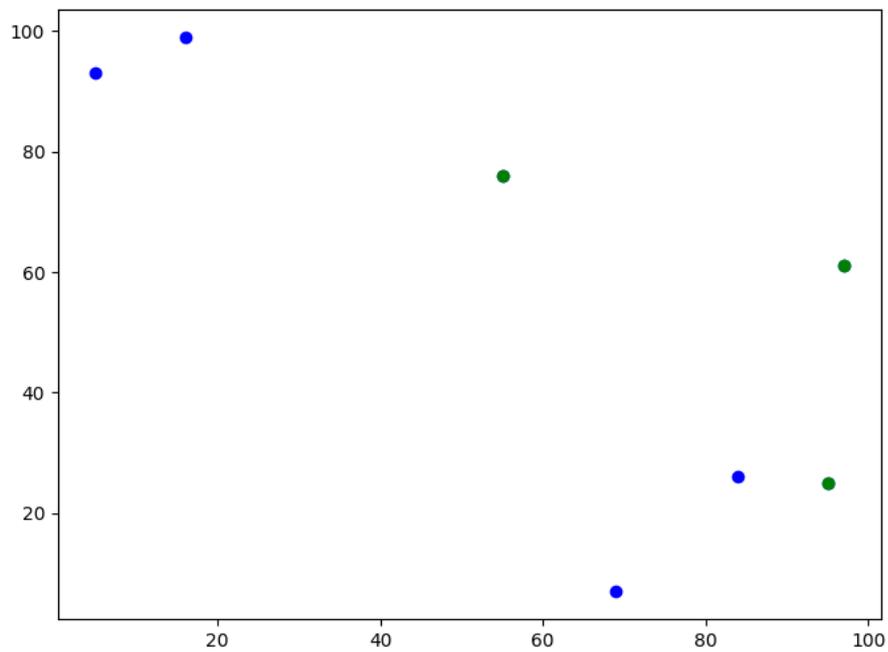


Figura 2.31: Situación sin mejora explorada.

Como la solución no ha mejorado la función, el algoritmo vuelve a ejecutarse con la situación de 2.30, la heurística anterior aquí hubiese dejado de ejecutarse, pero como a esta aún le queda tiempo, va a cambiar su vecindario. La solución explorada es la siguiente:

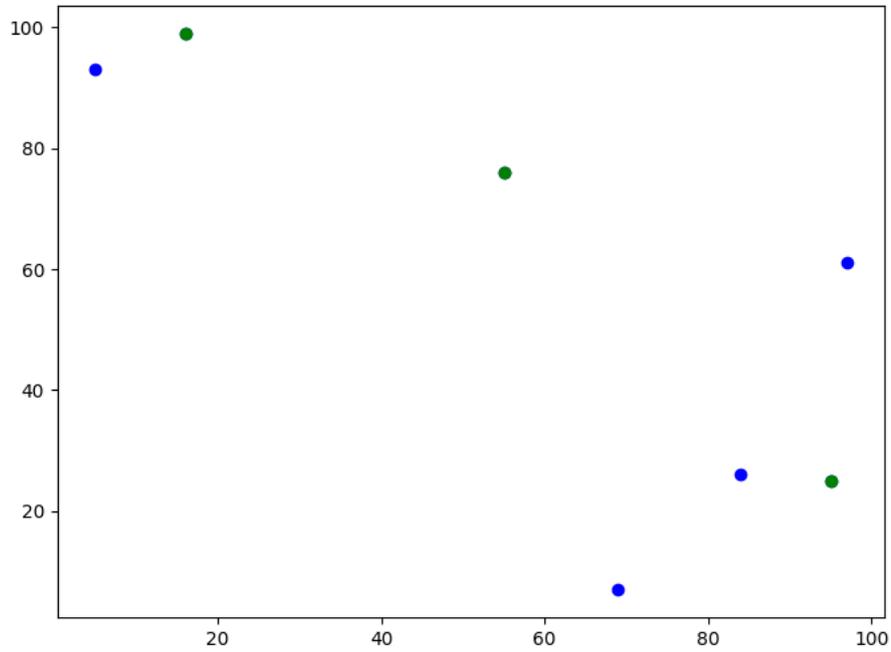


Figura 2.32: Situación con mejora explorada.

El algoritmo guarda esta solución y sigue buscando soluciones y cambiando su vecindario para encontrar mejores resultados. Finalmente llega a la solución representada en la figura 2.33.

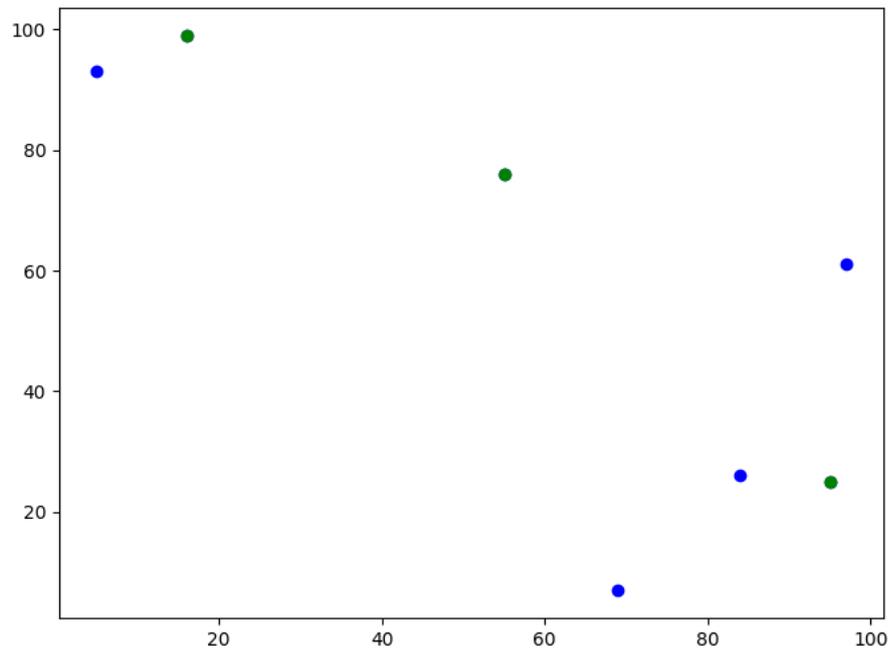


Figura 2.33: Situación final.

Ahora la heurística sigue explorando nuevos vecindarios hasta que se le acabe el tiempo pero manteniendo guardada esta solución. Como no encuentra mejora aunque explore muchos vecindarios, se queda con esta solución final.

# 3

## Resultados

### 3.1. Resultados y comentarios

#### 3.1.1. Heurística por pesos

La primera heurística propuesta es significativamente rápida,  $O(n^2 \log(n))$  es una complejidad notoriamente buena si tenemos en cuenta que estamos abordando un problema NP. En principio no sabemos si esta heurística encuentra siempre una solución ya que el procedimiento RADIO podría no encontrar una solución, pero gracias al teorema que hemos demostrado garantizamos que encontraremos alguna solución si existiese, además de darnos una idea de por que elegimos en RADIO el valor  $2r$ .

Ahora veamos distintas ejecuciones de nuestro algoritmo implementado en Python. Hemos hecho una pequeña modificación que hace que en el ultimo paso en vez de elegir vértices aleatorios se elijan los que más peso tienen. Veamos los resultados de las ejecuciones en las figuras [3.2](#).

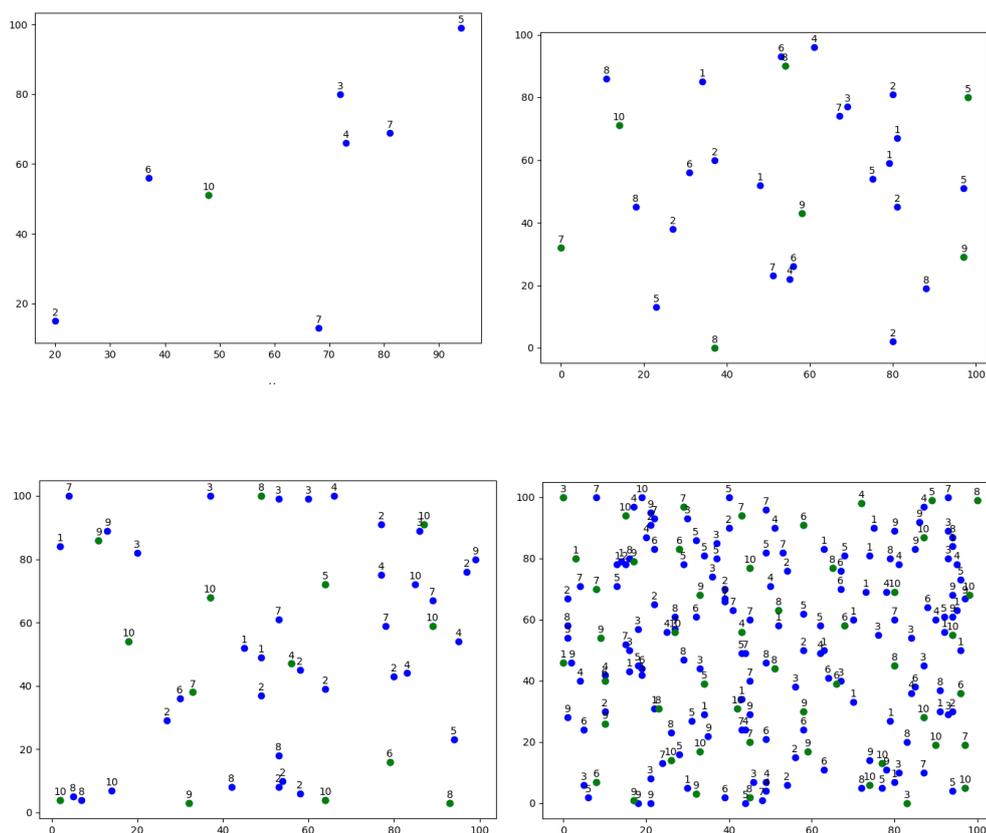


Figura 3.2: Resolución del problema p-Center según esta heurística.

Como se puede observar hemos incluido distintas ejecuciones tanto con pocos puntos y localizaciones como con muchos, y podríamos haber puesto muchos más si no fuese por la poca claridad que había en los gráficos. Esta heurística nos ha permitido realizar en tiempo razonable menor a un minuto instancias del problema de hasta  $n = 10000$  sin buscar optimizaciones de código ni de software para acelerarlo. Es sin duda un gran hito si somos conscientes de que estamos abordando un problema NP.

Ahora veamos su parte negativa. Esta heurística se basa en buscar el vértice con más peso y eliminar a los que se encuentren a cierta distancia ponderada para volver a aplicar esto mismo al resto de vértices. Es decir, el vértice más pesado siempre va a estar en nuestra solución, lo que limita bastante la exactitud del resultado porque puede haber soluciones que no lo contengan. Veamos un par de ejemplos en los que claramente la heurística no proporciona un buen resultado en las figuras 3.3.

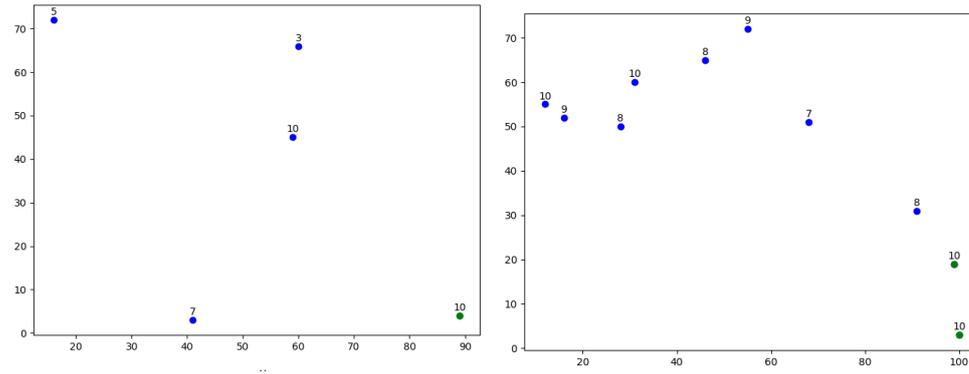


Figura 3.3: La heurística no siempre da buenos resultados.

Además, si nuestros vértices poseen todos el mismo peso, la heurística se vuelve prácticamente aleatoria:

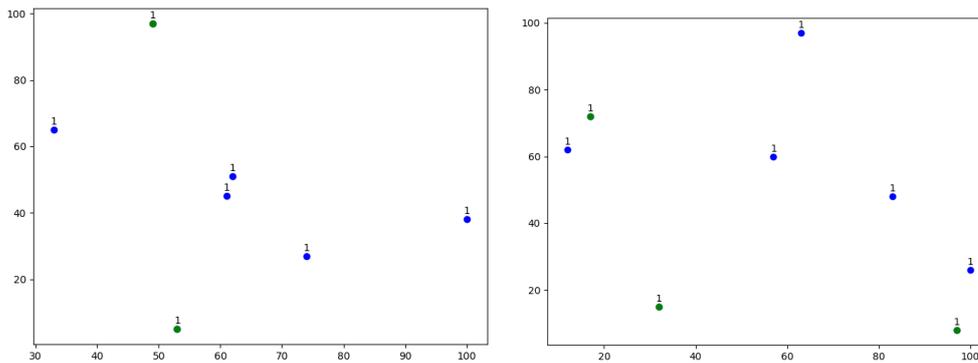


Figura 3.4: Heurística ejecutada sin pesos.

A raíz de estos resultados, podemos sacar algunas conclusiones. Estamos ante una heurística sencilla, que es capaz de realizar cálculos sobre muestras significativamente grandes sin exigir apenas recursos computacionales pero que también puede dar resultados alejados de la solución real en muestras simples. Si tuviéramos que imaginar un caso de uso para esta heurística, sería uno en el que necesitamos comprobar muchas localidades y nodos o uno en el que el peso de cada facilidad es significativamente relevante.

### 3.1.2. Heurística Sustitución de Vértice por Búsqueda Local

La segunda heurística ofrece buenas soluciones en tiempo razonable y maneja bien grandes cantidades de datos. Aunque supuestamente no es tan rápida como

la primera, cubre los problemas que tenía la anterior ya que ésta supone pesos iguales para todas.

Hemos comprobado que la heurística anterior tiene un muy buen comportamiento incluso con casos con una muestra muy grade, pero que a su vez es excesivamente dependiente de los pesos. Ahora vamos a someter a esta siguiente heurística que no es dependiente de los pesos a pruebas similares para tratar de delimitar sus puntos fuertes y débiles.

Comenzamos con distintas ejecuciones con un  $n$  pequeño y múltiples valores para  $p$ .

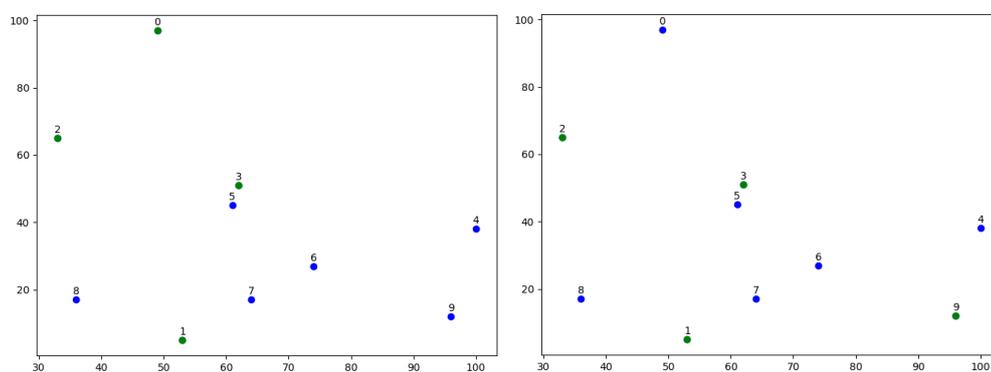


Figura 3.5: Tiempo de ejecución: 0.095s, 2 iteraciones,  $n=10$ ,  $p=4$

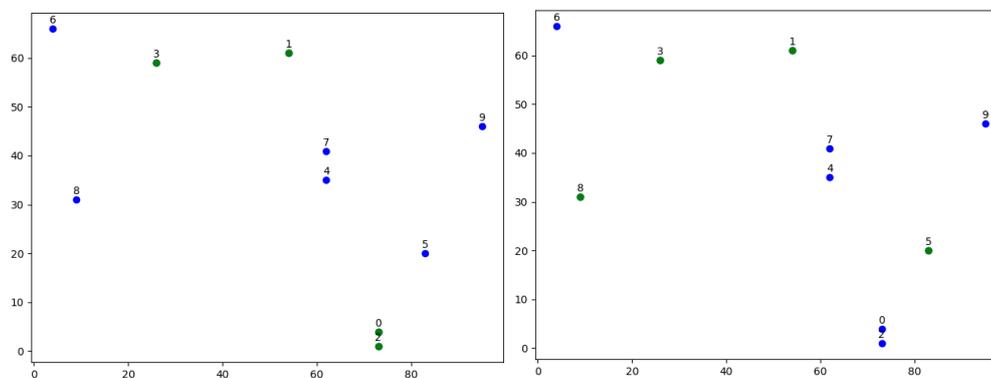


Figura 3.6: Tiempo de ejecución: 0.095s, 3 iteraciones,  $n=10$ ,  $p=4$

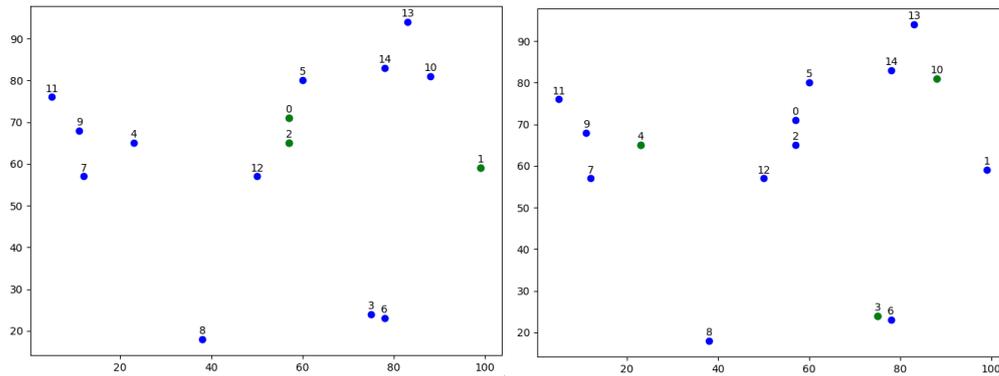


Figura 3.7: Tiempo de ejecución: 0.110s, 5 iteraciones,  $n=15$ ,  $p=3$

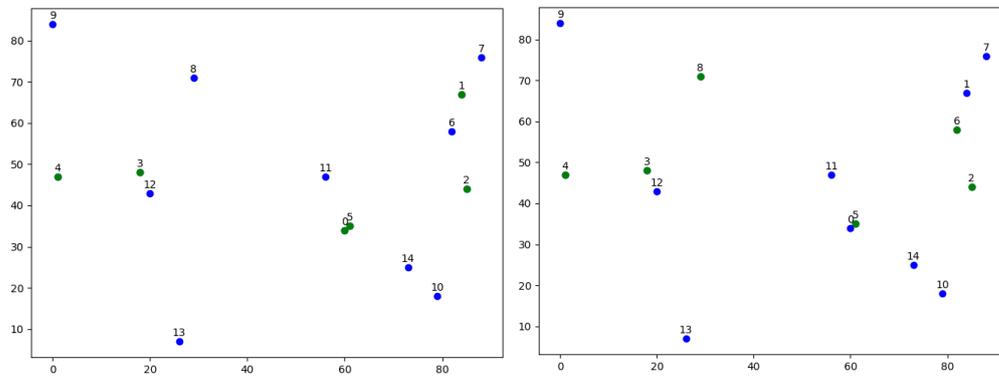


Figura 3.8: Tiempo de ejecución: 0.102s, 5 iteraciones,  $n=15$ ,  $p=6$

Como podemos observar, la heurística es bastante consistente y produce resultados esperados. A simple vista, las soluciones que ofrece tienen sentido y en varios casos se ve una clara mejora respecto al caso inicial. Vamos a ir poco a poco aumentando los tamaños de la muestra y viendo como se comporta.

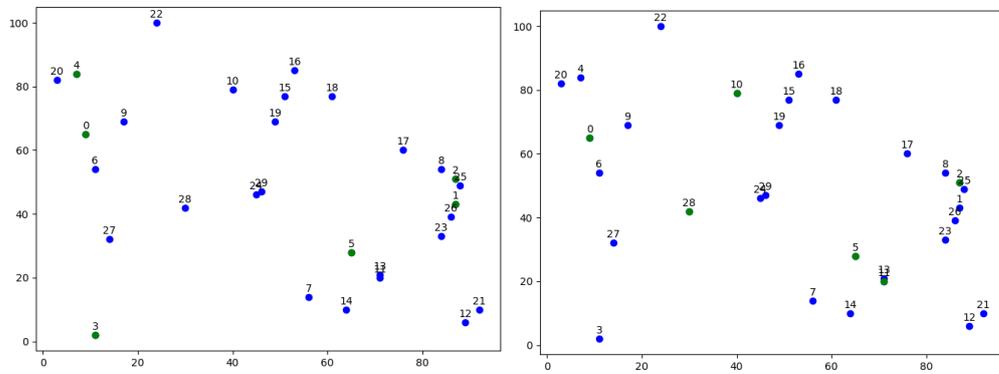


Figura 3.9: Tiempo de ejecución: 0.099s, 4 iteraciones,  $n=30$ ,  $p=6$

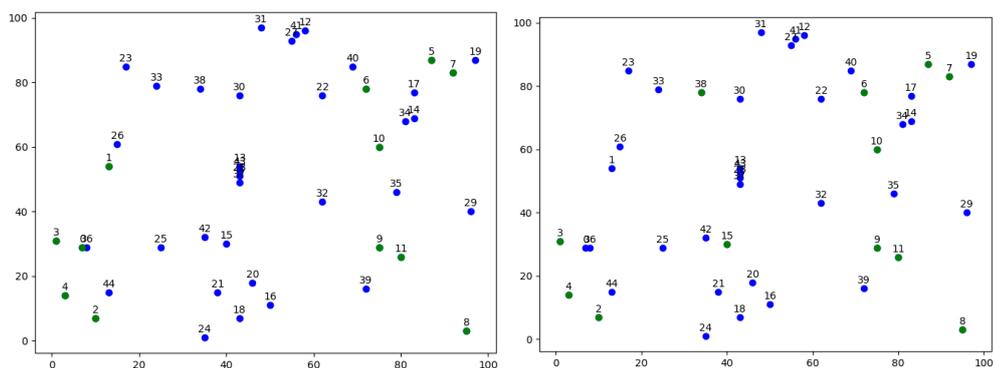


Figura 3.10: Tiempo de ejecución: 0.110s, 3 iteraciones,  $n=45$ ,  $p=12$

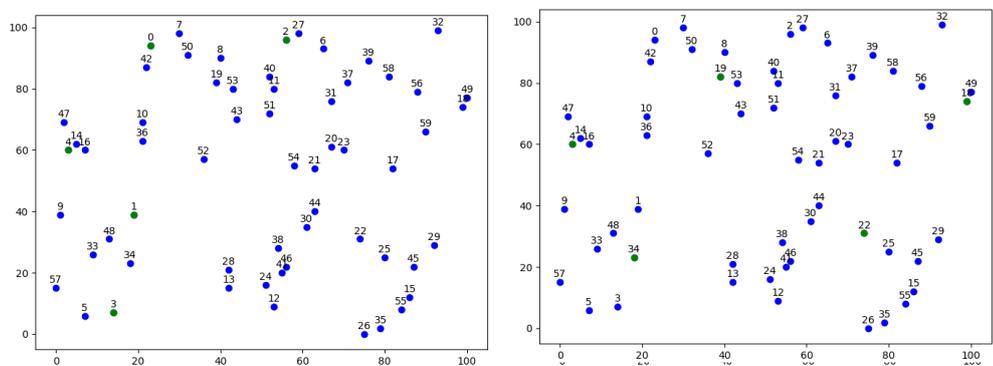


Figura 3.11: Tiempo de ejecución: 0.127s, 8 iteraciones,  $n=60$ ,  $p=6$

Como podemos observar, la heurística prácticamente ni se inmuta frente a este incremento constante de  $n$ . Ahora vamos a ir aumentando nuestro tamaño de la muestra exponencialmente, por lo que, aunque los gráficos no sean muy limpios, el tiempo va a ser significativamente más interesante.

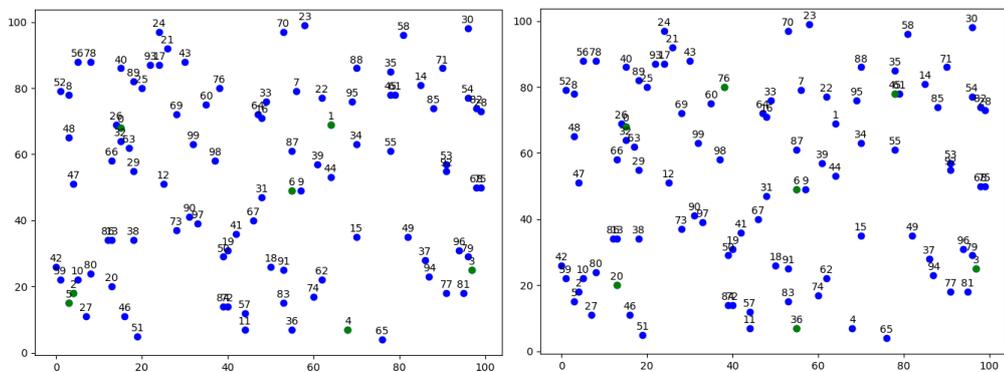


Figura 3.12: Tiempo de ejecución: 0.147s, 6 iteraciones,  $n=100$ ,  $p=7$

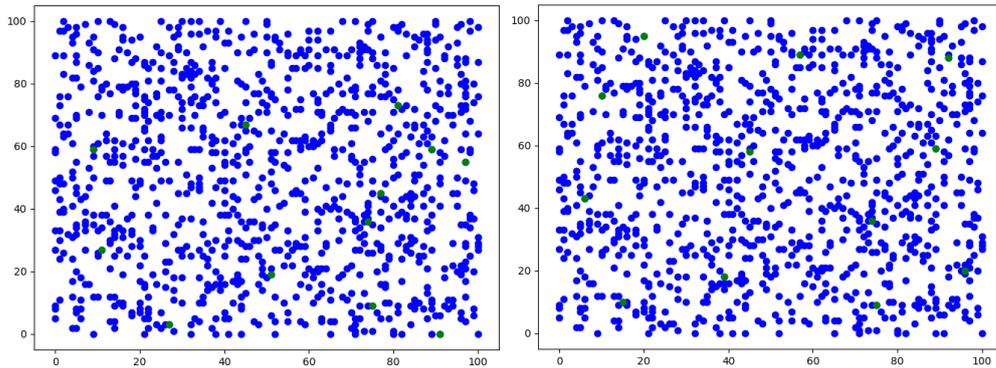


Figura 3.13: Tiempo de ejecución: 7.151s, 18 iteraciones,  $n=1000$ ,  $p=12$

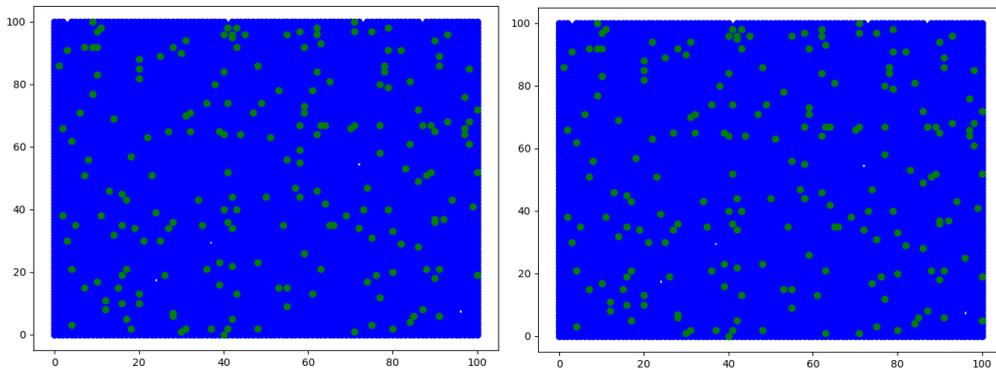


Figura 3.14: Tiempo de ejecución: 43.256s, 5 iteraciones,  $n=10000$ ,  $p=200$

Como podemos ver, el tiempo ha crecido notoriamente incluso al haber sido un caso en el que solo ha habido 5 iteraciones. El rango alrededor de  $n = 1000$  parece interesante de estudiar, ya que es lo suficientemente grande como para que otros factores externos a la heurística (acceso a memoria, organización de procesos de la CPU...) no sean relevantes, pero lo suficientemente manejable como para que el tiempo de ejecución no sea un impedimento en las pruebas. Por ello, vamos a realizar diferentes ejecuciones con  $n = 1000$  y en algunas variaremos el valor de  $p$  para ver si podemos encontrar alguna correlación entre el número de iteraciones, el tiempo de ejecución y  $p$  (hemos comprobado la correlación trivial entre  $n$  y el tiempo de ejecución).

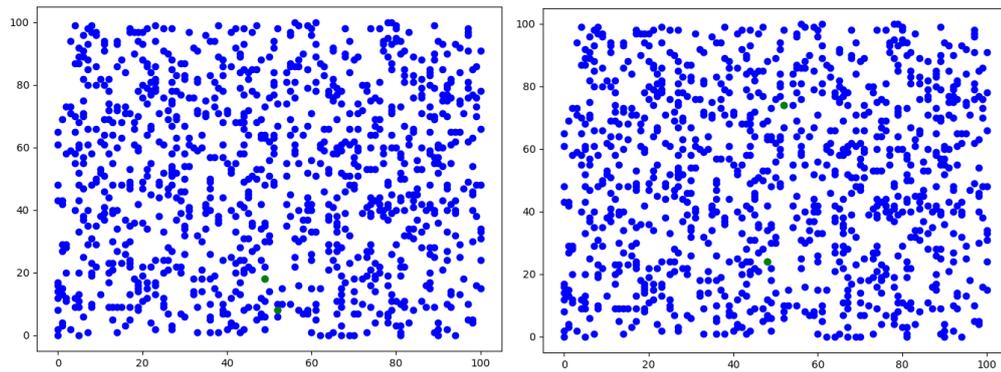


Figura 3.15: Tiempo de ejecución: 7.244s, 5 iteraciones,  $n=1000$ ,  $p=2$

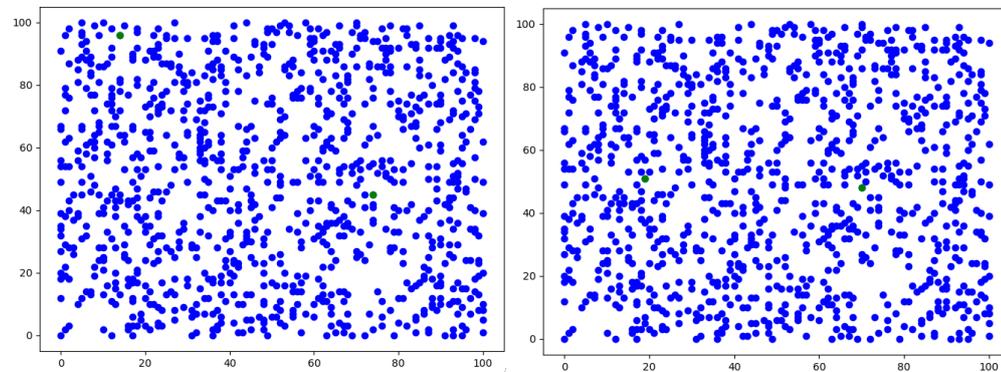


Figura 3.16: Tiempo de ejecución: 4.916s, 4 iteraciones,  $n=1000$ ,  $p=2$

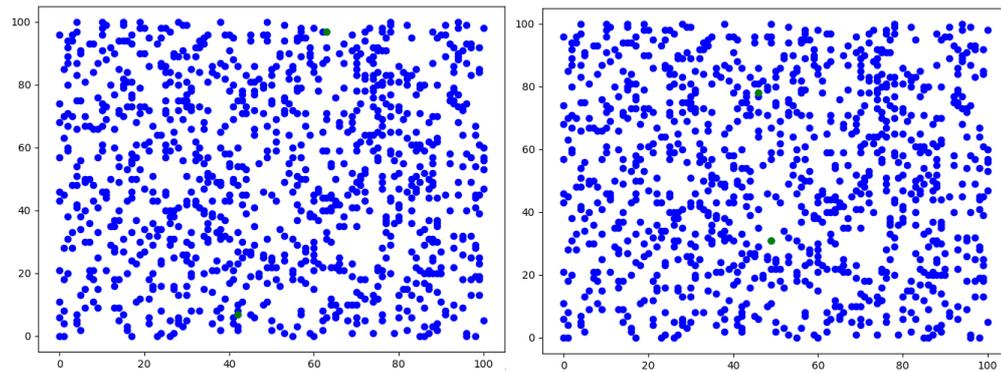


Figura 3.17: Tiempo de ejecución: 5.720s, 5 iteraciones,  $n=1000$ ,  $p=2$

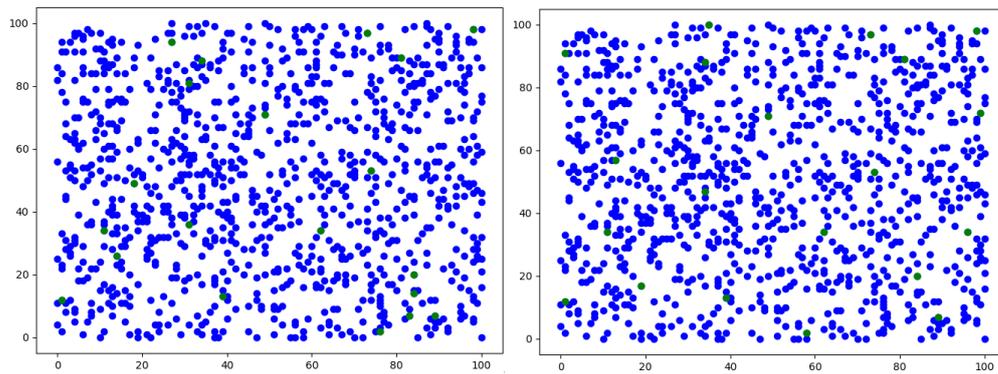


Figura 3.18: Tiempo de ejecución: 3.626s, 11 iteraciones,  $n=1000$ ,  $p=20$

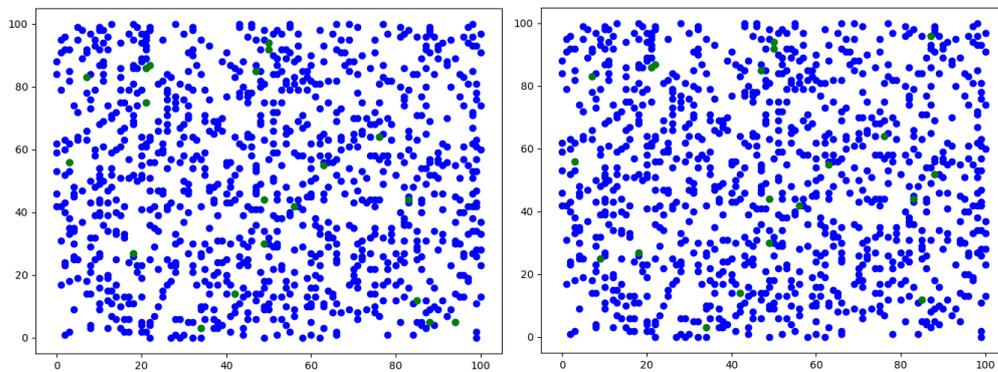


Figura 3.19: Tiempo de ejecución: 1.800s, 4 iteraciones,  $n=1000$ ,  $p=20$

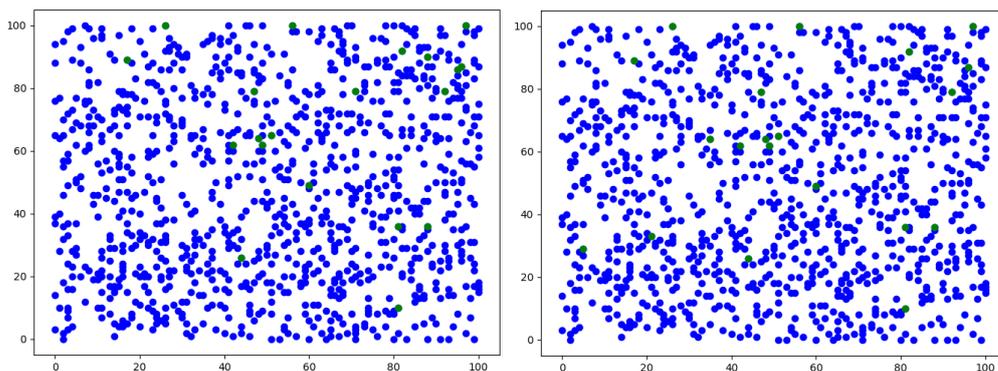
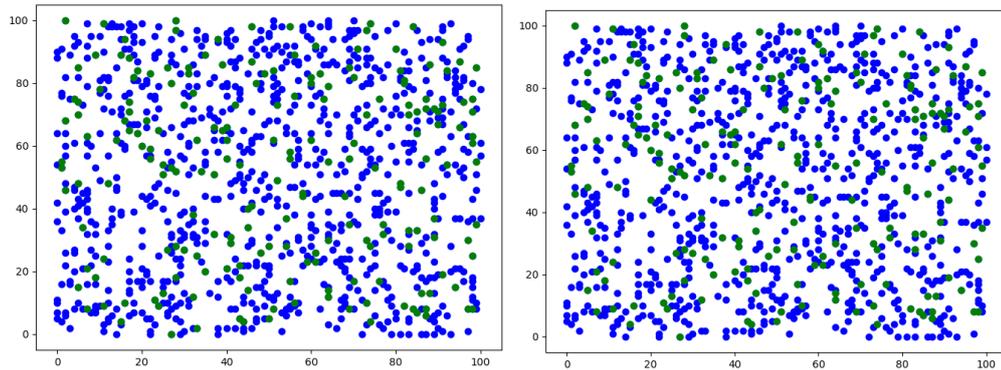
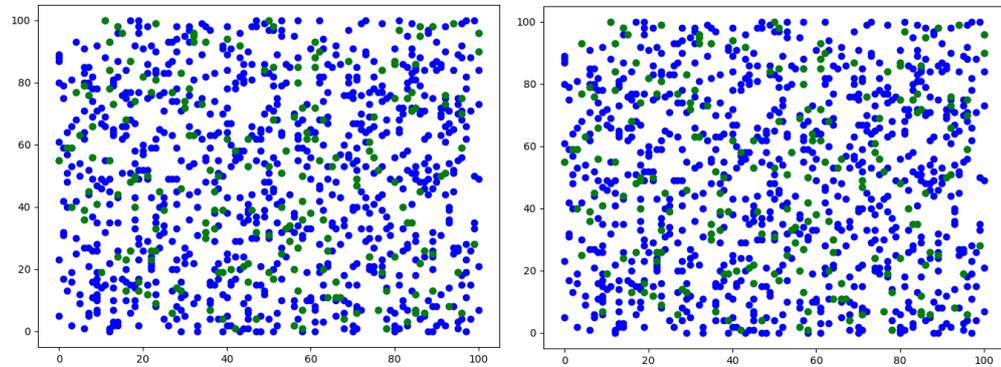
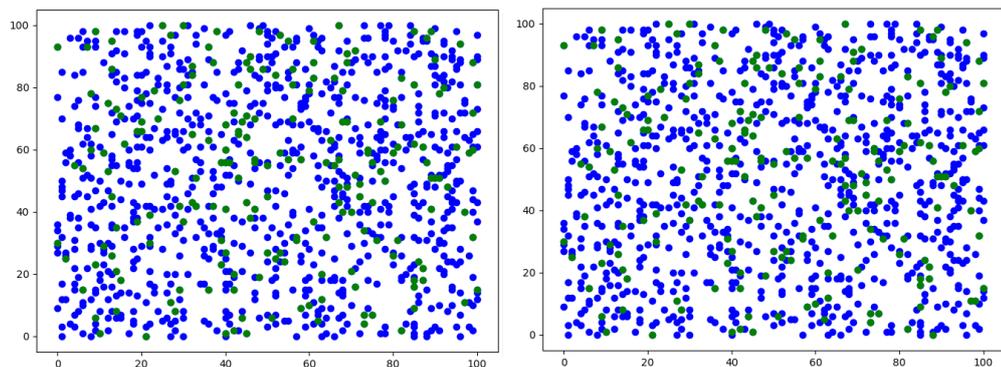


Figura 3.20: Tiempo de ejecución: 2.509s, 4 iteraciones,  $n=1000$ ,  $p=20$

Figura 3.21: Tiempo de ejecución: 0.219s, 4 iteraciones,  $n=1000$ ,  $p=200$ Figura 3.22: Tiempo de ejecución: 0.472s, 6 iteraciones,  $n=1000$ ,  $p=200$ Figura 3.23: Tiempo de ejecución: 0.221s, 4 iteraciones,  $n=1000$ ,  $p=200$ 

Tras todas estas ejecuciones distintas podemos realizar múltiples observaciones sobre esta heurística:

- La primera observación es que pese a ser una heurística más compleja y elaborada que la que hemos presentado por pesos, ha conseguido dar buenos

resultados para muestras muy grandes, consiguiendo rivalizar en tiempo con la primera.

- La segunda observación tiene que ver con la consistencia respecto a  $n$  que tiene la heurística, ya que es capaz de dar resultados coherentes y no parece que el tamaño de  $n$  afecte a esto.
- La tercera observación es la aparente aleatoriedad de las iteraciones realizadas. En muchas ocasiones parece mantenerse consistente, pero de repente en una ejecución con similares parámetros se produce una fuerte desviación; sería necesario un estudio en más profundidad para poder dar una conclusión clara.
- La cuarta está relacionada con las últimas ejecuciones que hemos realizado. Si prestamos atención, nos podemos dar cuenta que conforme aumentábamos  $p$  con un  $n$  fijo, el tiempo disminuía. Esto puede estar relacionado con el hecho de que nuestro algoritmo es de optimización local, y cuando hay muchas facilidades no encuentra mejoras claras.
- La quinta y última es similar a la anterior; conforme aumentamos el número de facilidades, la solución no parece variar mucho respecto de la original. Esta podría ser quizás la debilidad más notable de esta heurística, ya que con  $p$  alto no consigue una buena optimización.

### 3.1.3. Heurística Sustitución Tabú de Vértice por Búsqueda Local

Una vez estudiada la anterior heurística, vamos a ejecutar la tercera, que es una modificación de ésta que busca salir de máximos locales. Veamos ahora el rendimiento de la modificación de la variación tabú de la heurística anterior. Comenzamos ejecutando

Comenzamos comparando los resultados y la mejora de la solución con el mismo conjunto de puntos pero con tiempos de ejecución dados distintos.

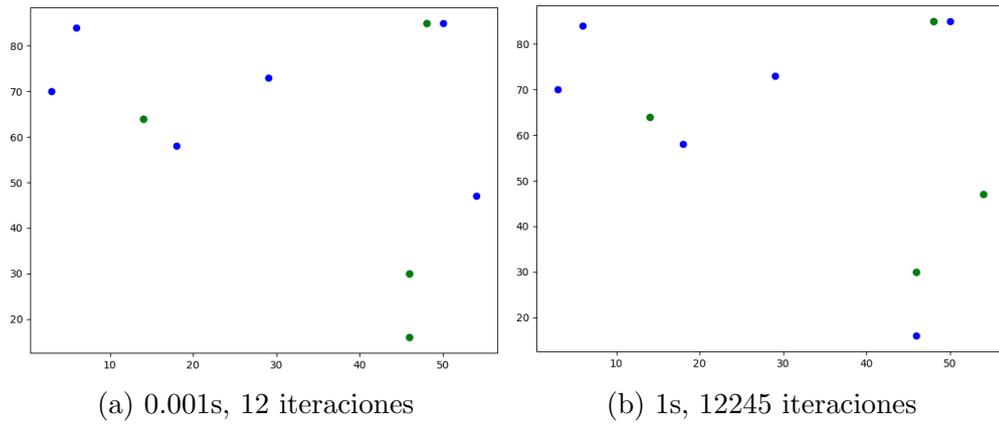


Figura 3.24:  $n=10$ ,  $p=4$

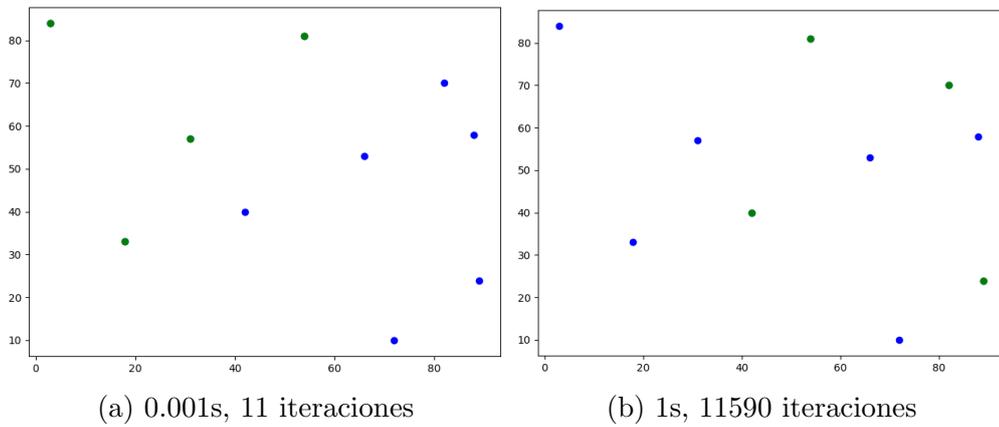


Figura 3.25:  $n=10$ ,  $p=4$

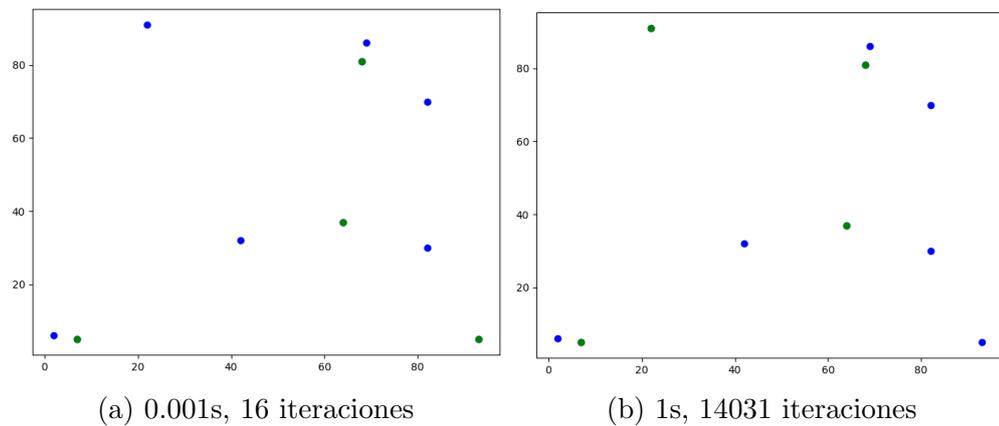


Figura 3.26:  $n=10$ ,  $p=4$

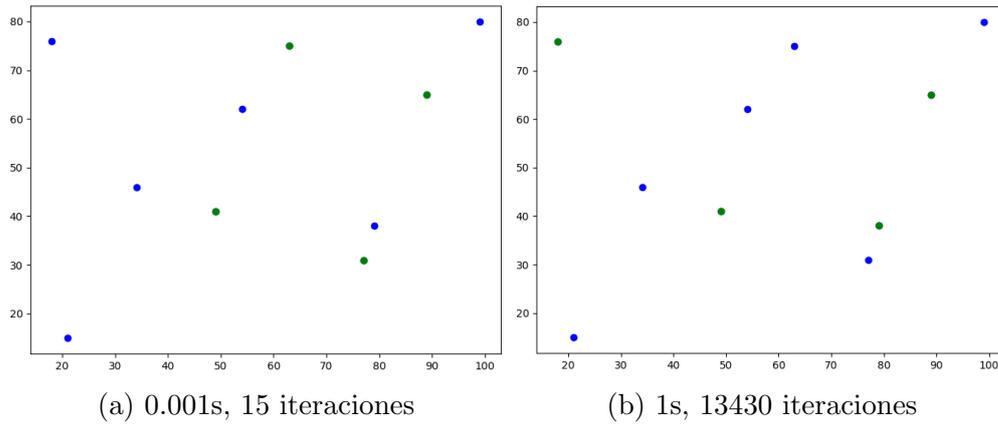


Figura 3.27:  $n=10$ ,  $p=4$

Come se puede observar, la heurística produce mejores resultados al aumentar el tiempo ya que explora más vecindarios. Vamos a incrementar y variar los valores de  $n$  y  $p$  para estudiar como evoluciona.

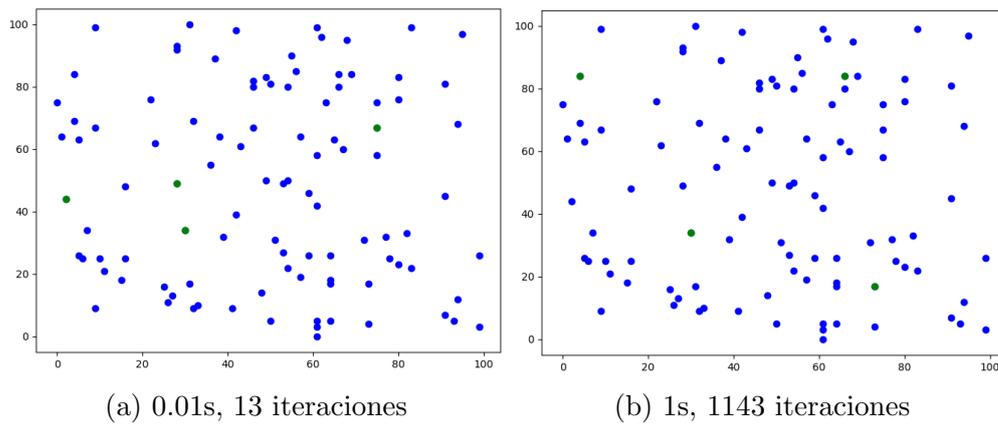
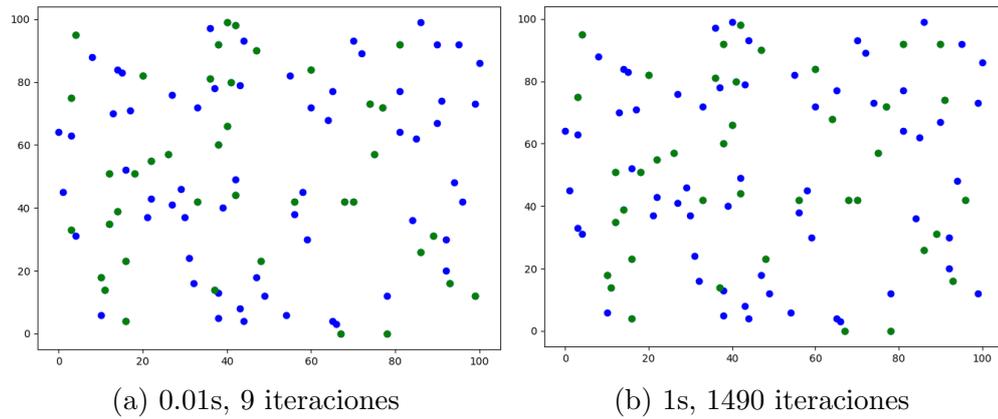
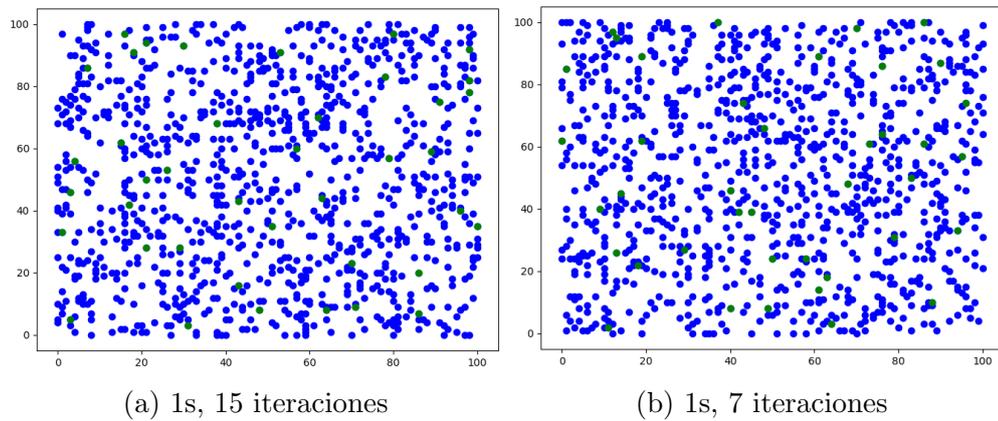


Figura 3.28:  $n=100$ ,  $p=4$

Figura 3.29:  $n=100$ ,  $p=40$ 

Vemos que las iteraciones de la heurística escalan de manera proporcional al tamaño de la muestra y al tiempo dado (fijémonos que en el primer caso hemos cambiado  $t$  a 0.01s), por lo que parece que la complejidad es lineal. Probemos más ejecuciones aumentando  $n$  para ver si se cumple esta relación:

Figura 3.30:  $n=1000$ ,  $p=40$

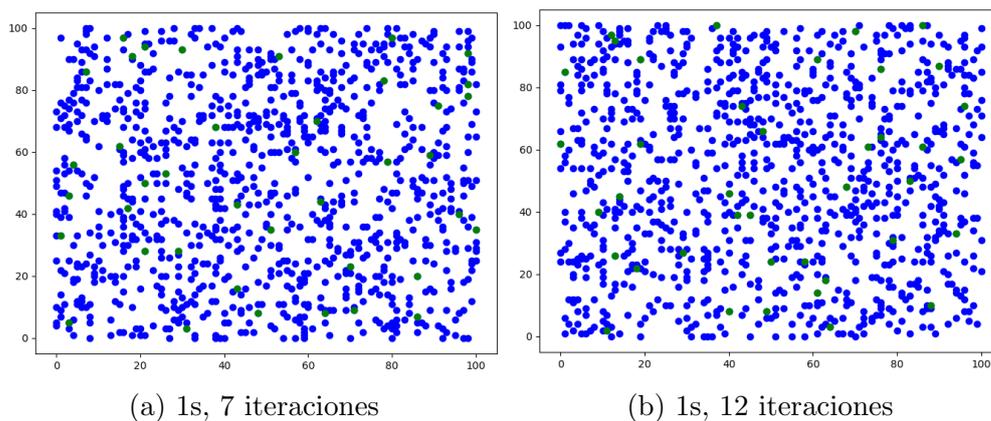


Figura 3.31:  $n=1000$ ,  $p=40$

Finalmente vamos a ver si al modificar  $p$  con un  $n$  fijo los resultados varían:

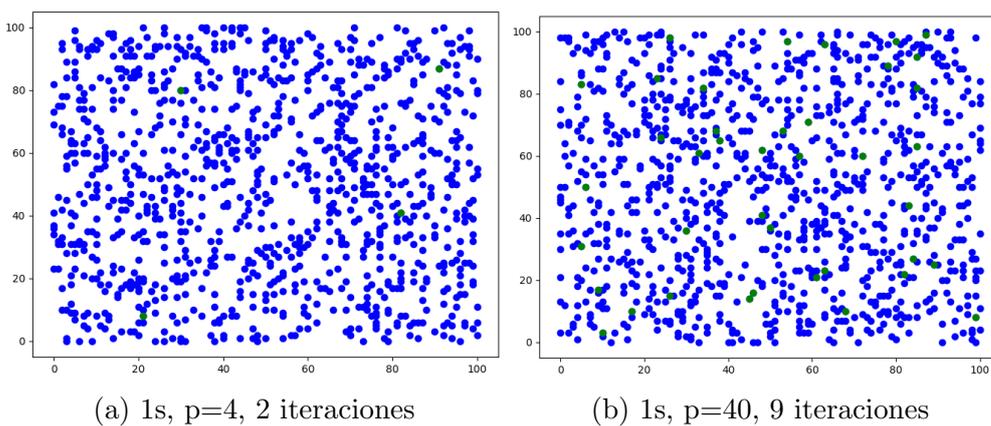


Figura 3.32:  $n=1000$

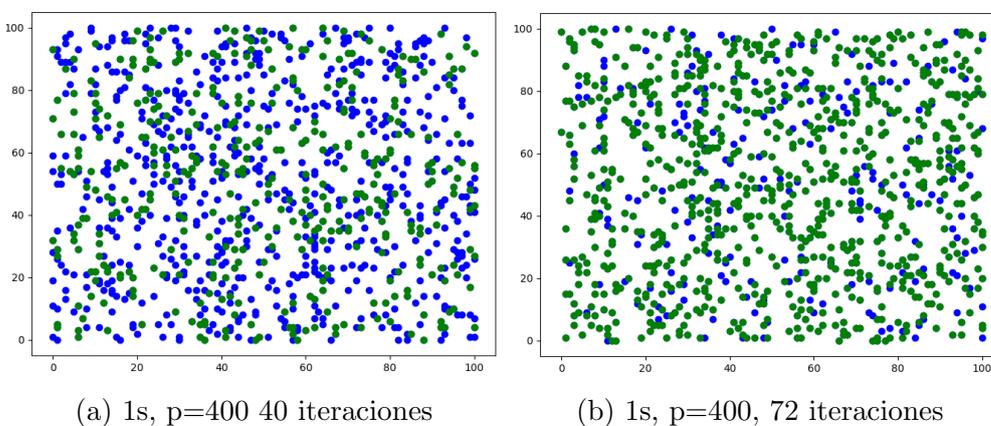


Figura 3.33:  $n=1000$

Con esto sacamos las siguiente conclusiones:

- La heurística escala de manera constante cuando dejamos un  $p$  fijo, mostrando una relación de linealidad.
- Al igual que la anterior heurística, es capaz de dar resultados coherentes para cualquier tamaño de  $n$  si se le da suficiente tiempo.
- Si hacemos variar  $p$ , vemos que para valores bajos con respecto a  $n$ , en cada iteración realiza muchas exploraciones de vecindarios, por lo que tarda más y para un tiempo fijo realiza menos iteraciones. Conforme vamos aumentando  $p$ , se vuelve más rápida y permite realizar más iteraciones, aunque los resultados sugieren que de manera no lineal.

Cabe resaltar que hay un aspecto importante que no hemos comentado ni considerado; la generación de puntos en el mapa y la selección de facilidades. Todos los test realizados han partido de un grupo de puntos generado de forma aleatoria (algoritmo pseudo-aleatorio) y las facilidades han sido seleccionadas también de forma aleatoria. Esto implica que no hemos tratado con casos triviales ni excesivamente degenerados, pero lo más importante es que no hemos comprobado si existe algún tipo de relación entre la forma de elegir los puntos o seleccionar las facilidades con el tiempo de ejecución ni las iteraciones.

# 4

## Conclusiones y trabajos futuros

### 4.1. Conclusiones

A lo largo de este trabajo se han abordado distintas metodologías para resolver el problema p-Center, un problema NP-completo de gran relevancia en el ámbito de la optimización y la investigación operativa. Se han propuesto y analizado tres heurísticas principales: la heurística basada en pesos ponderados, la heurística de sustitución de vértice por búsqueda local y una variación de esta última que emplea una lista tabú.

La heurística por pesos ponderados ha demostrado ser extremadamente eficiente en términos de tiempo de ejecución, siendo capaz de manejar instancias con un número muy elevado de puntos ( $n$ ) en un tiempo razonable. Sin embargo, su dependencia de los pesos asignados a los puntos puede limitar su aplicabilidad en casos donde los pesos no sean un factor relevante.

Por otro lado, la heurística de sustitución de vértice por búsqueda local ha mostrado una mayor consistencia en términos de calidad de la solución, independientemente de los pesos. Aunque esta heurística es más compleja y su tiempo de ejecución aumenta con el tamaño de la muestra, ha proporcionado resultados satisfactorios incluso para grandes instancias. No obstante, su rendimiento disminuye cuando el número de facilidades ( $p$ ) es alto, debido a la menor capacidad de optimización local en estos casos.

La variación de búsqueda tabú ha introducido una mejora adicional, permitiendo escapar de máximos locales y explorando un vecindario más amplio. Esta

modificación ha mostrado un potencial significativo para mejorar la calidad de las soluciones.

En resumen, cada una de las heurísticas propuestas tiene sus fortalezas y debilidades, y la elección de una u otra dependerá de las características específicas del problema a abordar, como el tamaño de la muestra, la relevancia de los pesos y el número de facilidades a ubicar.

## 4.2. Trabajos futuros

Existen varias direcciones posibles para extender y mejorar este trabajo:

- **Metaheurísticas:** Explorar el uso de metaheurísticas como algoritmos genéticos, recocido simulado o búsqueda de colonia de hormigas podría proporcionar mejoras adicionales en la calidad de las soluciones, especialmente para instancias de gran tamaño o con un alto número de facilidades.
- **Análisis de casos específicos:** Ampliar el estudio a casos específicos del problema p-Center, considerando distintas distribuciones de puntos, configuraciones de pesos y restricciones adicionales, podría ayudar a identificar patrones y optimizaciones específicas para diferentes tipos de aplicaciones.
- **Implementación en entornos reales:** Sería valioso aplicar las heurísticas desarrolladas en este trabajo a problemas reales de localización de instalaciones, como la planificación de centros de emergencia o la optimización de redes de transporte, para evaluar su rendimiento y adaptabilidad en situaciones prácticas.
- **Elaboración de herramientas interactivas:** Finalmente, se podrían elaborar herramientas interactivas que permitan personalizar de forma cómoda al usuario la ejecución de las heurísticas. Este va a ser el objeto de estudio del siguiente proyecto.

El desarrollo y perfeccionamiento de técnicas para resolver el problema p-Center sigue siendo un área de gran importancia y desafío en la investigación operativa, con numerosas aplicaciones prácticas y teóricas. La exploración de estas y otras direcciones futuras contribuirá a avanzar en la comprensión y resolución de este problema complejo.

# Bibliografía

- [1] Wikipedia. P, np, np-complete, np-hard. [Online]. Available: [https://en.m.wikipedia.org/wiki/File:P\\_np\\_np-complete\\_np-hard.svg](https://en.m.wikipedia.org/wiki/File:P_np_np-complete_np-hard.svg)
- [2] M. Ghil, “Hilbert problems for the geosciences in the 21st century,” *Nonlinear Processes in Geophysics*, vol. 8, pp. 211–222, 2001.
- [3] Wikipedia. Millennium prize problems. [Online]. Available: [https://en.wikipedia.org/wiki/Millennium\\_Prize\\_Problems](https://en.wikipedia.org/wiki/Millennium_Prize_Problems)
- [4] ——. P versus np problem. [Online]. Available: [https://en.wikipedia.org/wiki/P\\_versus\\_NP\\_problem](https://en.wikipedia.org/wiki/P_versus_NP_problem)
- [5] Mètode. La conjetura de hodge. [Online]. Available: <https://metode.es/revistas-metode/monograficos/la-conjetura-de-hodge.html>
- [6] Wikipedia. Poincaré conjecture. [Online]. Available: [https://en.wikipedia.org/wiki/Poincar%C3%A9\\_conjecture](https://en.wikipedia.org/wiki/Poincar%C3%A9_conjecture)
- [7] ——. Hipótesis de riemann. [Online]. Available: [https://es.wikipedia.org/wiki/Hip%C3%B3tesis\\_de\\_Riemann](https://es.wikipedia.org/wiki/Hip%C3%B3tesis_de_Riemann)
- [8] D. PY. Los problemas del milenio: Campo de yang-mills y el salto de masa. [Online]. Available: <https://demostracionpy.wordpress.com/2021/10/04/los-problemas-del-milenio-p1-campo-de-yang-mills-y-el-salto-de-masa/>
- [9] Wikipedia. Ecuaciones de navier-stokes. [Online]. Available: [https://es.wikipedia.org/wiki/Ecuaciones\\_de\\_Navier-Stokes](https://es.wikipedia.org/wiki/Ecuaciones_de_Navier-Stokes)
- [10] ——. Conjetura de birch y swinnerton-dyer. [Online]. Available: [https://es.wikipedia.org/wiki/Conjetura\\_de\\_Birch\\_y\\_Swinnerton-Dyer](https://es.wikipedia.org/wiki/Conjetura_de_Birch_y_Swinnerton-Dyer)
- [11] A. Schrijver, “Np-completeness,” in *Combinatorial Optimization*, ser. Algorithms and Combinatorics. Springer, Berlin, Heidelberg, 2008, vol. 21, pp. 121–133. [Online]. Available: [https://doi.org/10.1007/978-3-540-71844-4\\_15](https://doi.org/10.1007/978-3-540-71844-4_15)
- [12] Wikipedia. Np-intermediate. [Online]. Available: <https://en.wikipedia.org/wiki/NP-intermediate>
- [13] E. A. Silver, R. Victor, V. Vidal, and D. de Werra, “A tutorial on heuristic methods,” *European Journal of Operational Research*, vol. 5, no. 3, pp. 153–162, 1980. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0377221780900843>
- [14] B. University of California, *IEOR 151 – Lecture 13: P-Median Problem*, 2024, lecture notes.
- [15] S. Elloumi, M. Labbé, and Y. Pochet, “A new formulation and resolution method for the p-center problem,” *INFORMS Journal on Computing*, vol. 16, no. 1, pp. 84–94, 2004.
- [16] J. Plesník, “A heuristic for the p-center problems in graphs,” *Discrete Applied Mathematics*, vol. 17, no. 3, pp. 263–268, 1987. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0166218X87900291>

- [17] N. Mladenović, M. Labbé, and P. Hansen, “Solving the p-center problem with tabu search and variable neighborhood search,” *Networks*, vol. 42, no. 1, pp. 48–64, 2003, first published: 08 July 2003.

