TRABAJO OPTIMIZACIÓN HEURÍSTICA

LOCALIZACIÓN DE AMBULANCIAS EN NAVARRA

Asignatura: Optimización II
Curso académico: 2024–2025
Docentes: Cildoz Esquiroz, Marta y Mallor Giménez, Fermín Francisco

Autores:
Jon Aguas
Fermín Fernández
Rubén González

Contents

1 Introducción

El acceso rápido a servicios de emergencia médica puede marcar la diferencia entre la vida y la muerte. La correcta ubicación de ambulancias en un territorio no solo mejora los tiempos de respuesta ante incidentes, sino que optimiza el uso de recursos públicos. En este contexto, el presente trabajo se enmarca dentro de una propuesta de mejora del sistema de atención urgente en la Comunidad Foral de Navarra.

La motivación principal radica en la necesidad de asesorar al Departamento de Salud del Gobierno de Navarra para diseñar un nuevo plan estratégico de despliegue de ambulancias. Para ello, se emplean técnicas de optimización heurística, capaces de generar soluciones eficientes en problemas donde las soluciones exactas resultan computacionalmente inabordables.

En este proyecto se desarrollará un algoritmo metaheurístico personalizado que, partiendo de una estructura territorial, una matriz de distancias y tiempos entre municipios, y los datos de población, buscará minimizar el coste total del sistema garantizando una cobertura adecuada a la población navarra. El modelo propuesto será evaluado bajo criterios de calidad definidos por el propio problema, priorizando la equidad territorial y la disponibilidad de ambulancias.

2 Descripción del problema

El objetivo del problema consiste en determinar la localización óptima de las sedes de ambulancias en los municipios de Navarra, así como el número de ambulancias asignadas a cada una, de forma que se minimice el coste total del sistema y se cumplan unos estándares de calidad en el servicio.

Se dispone de los siguientes datos:

- Un listado de municipios de Navarra con sus respectivas poblaciones.
- Una matriz de distancias (en km) y otra de tiempos de viaje (en minutos) entre municipios.
- Costes asociados a la apertura de una sede y al uso de ambulancias.
- Requisitos de calidad que la solución debe cumplir.
- No existe ninguna restricción en cuanto a número de sedes o ambulancias, pero se busca una solución que minimice el coste total del sistema.

La función objetivo que se desea minimizar incluye:

- El coste de apertura de sedes de ambulancias.
- El coste proporcional al número total de ambulancias utilizadas.
- El coste ponderado por población de la distancia desde cada municipio hasta la sede más cercana.

Por último, el problema está sujeto a tres restricciones clave de cobertura:

- R1: Al menos el 98% de la población debe tener una ambulancia a menos de 40 km.
- R2: Al menos el 80% de la población debe tener una ambulancia a menos de 20 km.
- R3: La probabilidad de que al menos una ambulancia esté disponible en cada sede debe ser igual o superior a 0.8.

3 Modelado matemático del problema

El problema se puede formular como un modelo de optimización con las siguientes variables, función objetivo y restricciones.

Variables de decisión

- $y_i = \begin{cases} 1 & \text{si se abre una sede en el municipio } i \\ 0 & \text{en caso contrario} \end{cases}$
- $x_{ij} = \begin{cases} 1 & \text{si el municipio } j \text{ está cubierto por la sede en } i \\ 0 & \text{en caso contrario} \end{cases}$
- $a_i \in \mathbb{Z}^+$: número de ambulancias asignadas a la sede del municipio i, en caso de existir.

Parámetros

- P_j: población del municipio j
- d_{ij}: distancia en kilómetros entre los municipios i y j
- t_{ij}: tiempo de viaje en minutos entre los municipios i y j
- D: conjunto de todos los pares (i, j) donde i puede cubrir a j dentro del umbral de distancia
- C_1: coste de abrir una sede (1.5 unidades por sede)
- C_2: coste por ambulancia (1 unidad por ambulancia)
- C_3: peso por distancia poblacional (10^{-5})

Función objetivo

Minimizar el coste total asociado a:

- Coste de apertura de sedes: $C_1 \sum_i y_i$
- Coste por ambulancia utilizada: $C_2 \sum_i a_i$
- Coste asociado a la distancia poblacional a la sede más cercana:

$$C_3 \sum_j P_j \cdot \min_{i \in I} d_{ij} \cdot x_{ij}$$

Finalmente, la función objetivo se puede expresar como:

$$\min \left(C_1 \sum_i y_i + C_2 \sum_i a_i + C_3 \sum_j P_j \cdot \min_{i \in I} d_{ij} \cdot x_{ij} \right) \tag{1}$$

Restricciones

1. **Cobertura mínima del 98% de la población a menos de 40 km:**

$$\sum_j P_j \cdot \left(\max_i \{x_{ij} \mid d_{ij} < 40\} \right) \geq 0.98 \sum_j P_j$$

2. **Cobertura mínima del 80% de la población a menos de 20 km:**

$$\sum_j P_j \cdot \left(\max_i \{x_{ij} \mid d_{ij} < 20\} \right) \geq 0.80 \sum_j P_j$$

3. **Relación entre cobertura y sedes:** un municipio sólo puede estar cubierto por una sede que esté activa:

$$x_{ij} \leq y_i \quad \forall i, j$$

4. **Probabilidad de disponibilidad:** para cada sede, la probabilidad de que al menos una ambulancia esté libre debe ser mayor o igual a 0.8:

$$1 - (1 - p_i)^{a_i} \geq 0.8 \quad \text{donde } p_i = 1 - \frac{C_i}{24}$$

   Donde C_i es la carga diaria total de trabajo de una ambulancia en i, calculada como:

$$C_i = \frac{3 \cdot P_i}{100{,}000} \cdot \left(\frac{d_{ip} + d_{ph}}{80} + 1.5 \right)$$

   (d_{ip} y d_{ph} representan la distancia sede–paciente y paciente–hospital, respectivamente)

5. **Variables enteras y binarias:**

$$y_i \in \{0, 1\}, \quad x_{ij} \in \{0, 1\}, \quad a_i \in \mathbb{Z}^+$$

Este modelo constituye la base para evaluar la calidad de las soluciones generadas por el algoritmo metaheurístico.

4 Estrategia heurística

4.1 Generación de la solución inicial

Para iniciar el proceso de resolución del problema, optamos por construir una primera solución factible aplicando una lógica heurística de tipo greedy en lugar de utilizar una

solución aleatoria. Esta decisión se tomó por varias razones tanto prácticas como estratégicas:

En primer lugar, en problemas de optimización complejos como este, contar con una buena solución inicial puede tener un impacto significativo en la calidad y eficiencia de las soluciones finales obtenidas mediante algoritmos heurísticos o metaheurísticos (como búsqueda local, GRASP, VNS...). Una solución inicial aleatoria podría ser muy alejada del óptimo, dificultando o ralentizando la convergencia del algoritmo posterior.

Por tanto, decidimos utilizar una estrategia constructiva con una heurística voraz (greedy). Esta consiste en ir añadiendo sedes de manera iterativa eligiendo en cada paso el municipio que proporcione la mayor cobertura de población dentro del radio permitido. De esta forma, cada decisión local busca maximizar el beneficio global (número de habitantes cubiertos), construyendo paso a paso una solución que cumpla las restricciones del problema.

Dicho procedimiento se basa en los siguientes pasos:

- **Cobertura a 20 km (R2)**: Inicialmente, se van seleccionando sedes hasta que al menos el 80% de la población esté cubierta a menos de 20 km, priorizando la cercanía inmediata (respuesta rápida).
- **Cobertura a 40 km (R1)**: A continuación, se continúa el proceso hasta cubrir al menos el 98% de la población en un radio de 40 km, extendiendo así la cobertura general del sistema de emergencias.
- **Asignación final y validación (R3)**: Se asignan sedes a todos los municipios restantes y se comprueba que la carga de trabajo (población cubierta por cada sede) esté dentro de los límites permitidos. Si no es así, se ajusta el número de ambulancias en las sedes afectadas.

Esta lógica garantiza una solución inicial factible y razonablemente eficiente, sin necesidad de exploración aleatoria ni generación masiva de soluciones. Sin embargo, cabe destacar que antes de iterar entre los diferentes municipios disponibles donde asignar la sede, hacemos un suffle para barajar el orden de los municipios. Esto permite evitar sesgos en la selección inicial y asegura que la solución inicial no sea siempre la misma, proporcionandonos las siguientes ventajas:

- Factibilidad garantizada desde el inicio, cumpliendo las tres restricciones del problema.
- Cobertura poblacional maximizada localmente, lo que suele traducirse en soluciones globales de buena calidad.
- Mejor punto de partida para algoritmos posteriores, ya que reduce la necesidad de grandes modificaciones.

En resumen, hemos determinado que partir de una buena solución nos proporciona una base sólida sobre la que aplicar técnicas heurísticas más complejas en fases posteriores del trabajo.

4.2 Marco teórico VNS

Variable Neighborhood Search (VNS) es una metaheurística propuesta por P. Hansen y N. Mladenovic en 1997 basada en la idea de que explorar diferentes vecindarios puede evitar quedarse atrapado en óptimos locales y aumentar la probabilidad de encontrar la solución óptima global.

El algoritmo parte del principio de que una misma solución puede tener múltiples vecindarios (definidos de formas distintas) y que alternar entre ellos permitiendo una búsqueda más amplia y efectiva.

Estructura general del algoritmo VNS:

1. **Inicialización:** Se parte de una solución inicial factible.
2. **Selección del vecindario:** Se define una serie de vecindarios $N_1(x), N_2(x), ..., N_k(x)$.
3. **Búsqueda local:** En cada iteración, se aplica una búsqueda local dentro del vecindario actual.
4. **Cambio de vecindario:**
   - Si se mejora la solución, se vuelve al vecindario inicial (N_1).
   - Si no hay mejora, se prueba con el siguiente vecindario (N_{i+1}).
5. **Finalización:** El algoritmo termina cuando se cumplen ciertos criterios de parada (tiempo, número de iteraciones sin mejora, etc.).

Algorithm 1 VND (Variable Neighborhood Descent)

```
function VND(x, k_max)
    k ← 1
    repeat
        x' ← arg min_{y∈N_k(x)} f(y)
        (x, k) ← NeighborhoodChange(x, x', k)
    until k = k_max
    return x
end function
```

Este enfoque se apoya en las siguientes propiedades clave sobre mínimos locales y globales:

- **Propiedad 1**: Un mínimo local respecto a un determinado vecindario no es necesariamente un mínimo local respecto a otro vecindario diferente.
- **Propiedad 2**: Un mínimo global es un mínimo local respecto a todos los posibles vecindarios.
- **Propiedad 3**: Para muchos problemas, los mínimos locales respecto a uno o varios vecindarios N_k están relativamente cerca unos de otros.

Estas propiedades justifican la eficacia del VNS, pues al cambiar dinámicamente entre vecindarios, logra escapar de mínimos locales, manteniendo un balance entre intensificación (explotar la zona alrededor de una buena solución) y diversificación (explorar nuevas regiones del espacio de soluciones).

4.3 Justificación del uso de VNS

En primer lugar, trataremos de definir qué es una solución concretamente, y cuál es la estructura que sigue. Hemos tomado como solución la propuesta realizada en el Aulario: una matriz que contiene 4 columnas: Código, Sede, N_ambulancias y Código_sede.

Por tanto, una solución será una instancia de dicha matriz, en la que se especifica para cada municipio si es sede o no, el número de ambulancias asignadas y qué sede le abastece.

Dada esta solución, comenzamos a pensar en posibles algoritmos que nos permitiesen hacer frente al problema. Uno de los aspectos clave a tener en cuenta es que debemos tener la opción de explorar el espacio completo de soluciones; no pueden quedar posibles soluciones sin posibilidad de ser evaluadas.

Por ello, nos hemos fijado en que la solución puede variar en dos aspectos que no tienen que ver entre sí: desplazar la sede de una ubicación a otra, y aumentar o disminuir el número de sedes. Mediante estas dos posibles variaciones, somos capaces de explorar el espacio completo, ya que el número de ambulancias siempre se asignará en el mínimo posible para cumplir la restricción.

Así pues, como esos dos cambios no parecen tener mucho en común, pensamos que sería buena idea la implementación de dos vecindarios distintos, y por tanto aplicar algoritmos del estilo VNS, que permiten explorar esos dos vecindarios de forma simultánea. Ya que en otros algoritmos como GRASP o SA solamente se define un vecindario, no consideramos apropiado mezclar en la misma familia de soluciones vecinas movimientos de sede con aumentos o disminuciones.

Concretamente, definimos los dos vecindarios como:

- **Vecindario 1** (N_1): Todas las posibles soluciones que se obtienen de mover una sede de un municipio a otro de los que está abasteciendo dicha sede. Por tanto, si hay n municipios y m sedes, el número de soluciones vecinas es $n - m$.
- **Vecindario 2** (N_2): Todas las soluciones que se pueden obtener añadiendo o eliminando una sede. Por tanto, si hay n municipios en total, el número de soluciones vecinas es n.

La presencia de estos dos vecindarios bien definidos permite aplicar de forma efectiva la estrategia de VNS, ya que permite abordar tanto la optimización de la distribución geográfica de las sedes como el dimensionamiento de su número, proporcionando un marco flexible y eficaz para la mejora progresiva de la solución inicial.

4.4 Primera propuesta: BVNS

Como primera implementación dentro del marco de búsqueda en entornos variables, optamos por el **BVNS (Basic Variable Neighborhood Search)**. Esta elección se fundamenta en varias razones que la convierten en una estrategia robusta, eficiente y especialmente adecuada para nuestro problema.

En términos generales, BVNS sigue la estructura clásica del algoritmo VNS, pero con un enfoque que alterna entre fases de exploración global y mejora local. Su esquema se basa en los siguientes pasos:

1. **Inicialización**: Partimos de una solución inicial factible (generada con una heurística greedy, como se explicó previamente).
2. **Búsqueda de vecindario (Shaking)**: Se perturba la solución actual seleccionando aleatoriamente una solución vecina dentro del vecindario N_k.
3. **Búsqueda local**: Desde la solución perturbada, se realiza una búsqueda local intensiva en su vecindario.
4. **Actualización**: Si se encuentra una mejor solución, se acepta y se vuelve al vecindario inicial; en caso contrario, se incrementa k para explorar vecindarios más amplios.

Algorithm 2 NeighborhoodChange

```
function NeighborhoodChange(x, x', k)
    if f(x') < f(x) then
        x ← x'
        k ← 1
    else
        k ← k + 1
    end if
    return (x, k)
end function
```

Algorithm 3 BVNS (Basic Variable Neighborhood Search)

```
function BVNS(x, k_max, t_max)
    t ← 0
    while t < t_max do
        k ← 1
        repeat
            x' ← Shake(x, k)
            if k = 1 then
                x'' ← BestImprovement1(x')
            else if k = 2 then
                x'' ← BestImprovement2(x')
            end if
            x, k ← NeighborhoodChange(x, x'', k)
        until k = k_max
        t ← CpuTime()
    end while
    return x
end function
```

Por lo tanto, viendo el funcionamiento de dicho algoritmo, nos proporciona una solución robusta y eficaz gracias a su simplicidad, capacidad para escapar de óptimos locales y adaptabilidad a nuestros vecindarios definidos. Al combinar fases de exploración global y mejora local, el algoritmo logra un equilibrio entre calidad de solución y tiempo de cómputo, permitiendo mejorar progresivamente la solución inicial sin necesidad de configuraciones complejas, y sirviendo además como base sólida para futuras extensiones o mejoras.

Una vez implementado el algoritmo y haber hecho varias pruebas, llegamos a la conclusión de que la realización de busquedas exhautivas en la parte de busqueda local no llevaba a largos tiempos hasta que encontraba la mejor solución de mejora, por lo que decidimos implementar una segunda version del algoritmo, la cual se basa en el mismo principio pero con un enfoque diferente.

4.5 Segunda propuesta: Exploración parcial de vecindarios

Tras observar que la versión básica de BVNS puede llegar a tener un coste computacional elevado debido a la exploración exhaustiva de todos los vecinos posibles en cada iteración, planteamos una segunda propuesta orientada a mejorar la eficiencia del algoritmo. La idea principal consiste en reducir la carga computacional limitando la búsqueda local a una **muestra aleatoria del vecindario**, en lugar de examinar todas las soluciones vecinas posibles.

Concretamente, en lugar de considerar todos los posibles movimientos de sedes (ya sea en su ubicación o en su número), seleccionamos aleatoriamente un subconjunto de sedes y analizamos únicamente las posibles reubicaciones dentro de ese grupo. Esta estrategia permite reducir de forma significativa el número de soluciones vecinas evaluadas por iteración, manteniendo al mismo tiempo la posibilidad de encontrar mejoras relevantes.

Para implementar este enfoque, basta con modificar las funciones encargadas de la búsqueda local en los vecindarios:

- `best_improvement_1`: encargada de explorar los movimientos de sede dentro del vecindario N_1 (reubicación de sedes existentes). Hemos determinado que el subconjunto de sedes a explorar sea de tamaño entre 1 o 3 sedes, elegido aleatoriamente, para meter cierta aleatoriedad y facilitar la fluidez del algoritmo.

- `best_improvement_2`: encargada de explorar los cambios en el número de sedes dentro del vecindario N_2 (adición o eliminación de sedes). En este caso, en vez de analizar todos los municipios que no son sede, se escoge un municipio aleatorio entre todos ellos y se evalúa la posibilidad de añadirlo como sede. De esta forma, se evita la necesidad de evaluar todos los municipios en cada iteración, lo que hemos comprobado que es muy costoso computacionalmente. De manera similar sucede a la hora de eliminar sede, se elije un municipio con sede aleatorio entre todos ellos y se evalua.

Este enfoque tiene dos ventajas principales.

- **Reducción considerable en el tiempo de cómputo por iteración**, lo que permite ejecutar más iteraciones en el mismo tiempo.

- Introducción de un componente estocástico que favorece la **diversificación** del proceso de búsqueda, pudiendo ayudar a escapar de óptimos locales de forma más efectiva.

Además, con el objetivo de intensificar la búsqueda y evitar que el algoritmo quede estancado en óptimos locales poco prometedores, hemos incorporado un mecanismo de **shake forzado**. Este consiste en aplicar un movimiento de mayor intensidad sobre la solución actual cada vez que se acumulan varias iteraciones sin mejora (en nuestro caso, 10). Para ello, se incrementa el radio utilizado en la función de `shake`, lo que permite generar soluciones significativamente distintas y explorar regiones más alejadas del espacio de búsqueda. Este enfoque tiene un doble propósito: por un lado, romper ciclos de estancamiento, y por otro, favorecer la diversificación sin abandonar por completo la estructura del algoritmo BVNS original. En la implementación, esto se gestiona a través de un contador (`no_mejora`) y una condición que activa el *shake forzado* con un mensaje de seguimiento en consola cuando se detecta la situación.

Algorithm 4 NeighborhoodChange

```
1: function NeighborhoodChange(x, x', k)
2:     if f(x') < f(x) then
3:         x ← x'
4:         k ← 1
5:         mejora ← true
6:     else
7:         k ← k + 1
8:         mejora ← false
9:     end if
10:    return (x, k, mejora)
11: end function
```

Algorithm 5 BVNS con *shake* forzado

```
1: function BVNS(x, k_max, t_max)
2:     t ← 0
3:     no_mejora ← 0
4:     while t < t_max do
5:         k ← 1
6:         repeat
7:             if no_mejora ≥ 10 then
8:                 x' ← ShakeForzado(x, k)
9:                 no_mejora ← 0
10:            else
11:                x' ← Shake(x, k)
12:            end if
13:            if k = 1 then
14:                x'' ← BestImprovement1(x')
15:            else if k = 2 then
16:                x'' ← BestImprovement2(x')
17:            end if
18:            (x, k, mejora) ← NeighborhoodChange(x, x'', k)
19:            if ¬mejora then
20:                no_mejora ← no_mejora + 1
21:            end if
22:        until k = k_max
23:        t ← CpuTime()
24:    end while
25:    return x
26: end function
```

En resumen, esta propuesta busca mantener una buena calidad de solución al tiempo que mejora notablemente la eficiencia del algoritmo, adaptándose mejor a escenarios en los que el tamaño del vecindario completo es demasiado grande para ser explorado en cada iteración.

5 Parámetros y componentes del algoritmo

5.1 Función fitness

En nuestro caso, la **función fitness coincide con la función objetivo**, ya que evaluar dicha expresión no supone un coste computacional elevado debido a cómo está estructurado el algoritmo final.

Esta función tiene como objetivo minimizar el coste total asociado a la solución, considerando tres componentes:

$$f(x) = 1{,}5 \cdot \text{NumSedes} + \text{NumAmbulancias} + 10^{-5} \sum_j P_j \cdot d_j$$

donde:

- P_j representa la población del municipio j,
- d_j es la distancia desde el municipio j hasta su sede asignada más cercana.

5.2 Componentes del algoritmo final

5.2.1 Función Shake

La función **Shake** tiene como objetivo perturbar la solución actual desplazando una sede existente a un municipio cercano elegido aleatoriamente. Este movimiento está restringido por un **radio de actuación** r, lo que implica que una sede únicamente puede ser trasladada a otro municipio que se encuentre dentro de dicho radio.

Tras realizar el cambio de ubicación, es necesario actualizar las asignaciones de los municipios a las nuevas sedes más cercanas. Esta operación nos permite escapar de óptimos locales y explorar otras regiones del espacio de soluciones, ampliando así la capacidad de búsqueda del algoritmo.

Algorithm 6 Shake

function SHAKE(x, M_d, k)
    $x' \leftarrow x$
    $S \leftarrow$ sedes activas en x'
    Escoger aleatoriamente $s_o \in S$
    $V \leftarrow \{v \in \text{Municipios} \mid M_d[s_o, v] \leq k \wedge v \neq s_o\}$
    if $V = \emptyset$ **then**
        return x'
    end if
    Escoger aleatoriamente $s_d \in V$
    Trasladar sede de s_o a s_d en x'
    Reasignar cada municipio m a su sede más cercana según M_d
    return x'
end function

5.2.2 Función BestImprovement1

A diferencia de la función **BestImprovement1** empleada en el primer algoritmo, en esta versión se realiza un muestreo parcial del vecindario, en lugar de explorarlo por completo. Esta estrategia permite reducir el coste computacional del algoritmo.

Se selecciona aleatoriamente un número entero entre 1 y 3, el cual determina la cantidad de sedes que serán desplazadas a municipios dentro de su radio de cobertura. Al igual que en el procedimiento **Shake**, después de cada movimiento se realiza una reasignación de los municipios previamente cubiertos por la sede original.

Una vez aplicadas las transformaciones, se elige el mejor movimiento entre los generados, siempre que suponga una mejora en el valor de la función objetivo.

Algorithm 7 BestImprovement1 (BVNS Alternativo)

function BestImprovement1($x, M_h, M_d, t, t_{\max}$)
  $x^* \leftarrow x, \quad z^* \leftarrow f(x)$
  Seleccionar aleatoriamente $S' \subseteq S(x)$ con $|S'| \in \{1, 2, 3\}$
  for all $s \in S'$ **do**
    $C_s \leftarrow$ municipios cubiertos por s
    for all $m \in C_s$ **do**
      $x' \leftarrow x$ con sede s trasladada a m
      Reasignar $i \in C_s$ a $\arg\min_{s' \in S(x')} M_d[i, s']$
      $z' \leftarrow f(x')$
      if $z' < z^*$ **then**
        $x^* \leftarrow x', \quad z^* \leftarrow z'$
      end if
    end for
  end for
  return (x^*, z^*)
end function

5.2.3 Función BestImprovement2

De nuevo, seguimos una lógica aleatoria, aunque planteada de forma diferente. Esta función alterna entre dos tipos de operaciones:

- Añadir una sede en un municipio no cubierto.
- Eliminar una sede activa.

La elección entre ambas opciones es aleatoria, y en cada llamada a la función se realiza únicamente una de ellas. Después de cada cambio, se lleva a cabo el proceso de reasignación de los municipios a sus sedes más cercanas. Finalmente, si el cambio produce una mejora en la función objetivo, se actualiza la mejor solución encontrada.

Algorithm 8 BestImprovement2 (BVNS Alternativo)

function BESTIMPROVEMENT2(x, M_h, M_d)
  $x^* \leftarrow x, \quad z^* \leftarrow f(x)$
  Escoger aleatoriamente una operación: `añadir` o `eliminar`
  if `añadir` **then**
    $U(x) \leftarrow$ municipios no cubiertos
    if $U(x) \neq \emptyset$ **then**
      Elegir aleatoriamente $m \in U(x)$
      $x' \leftarrow x$ con sede añadida en m
      Reasignar todos los municipios según M_d
      $z' \leftarrow f(x')$
      if $z' < z^*$ **then**
        $x^* \leftarrow x', \quad z^* \leftarrow z'$
      end if
    end if
  else if `eliminar` **then**
    $S(x) \leftarrow$ sedes activas
    if $S(x) \neq \emptyset$ **then**
      Elegir aleatoriamente $s \in S(x)$
      $x' \leftarrow x$ con sede eliminada en s
      Reasignar todos los municipios según M_d
      $z' \leftarrow f(x')$
      if $z' < z^*$ **then**
        $x^* \leftarrow x', \quad z^* \leftarrow z'$
      end if
    end if
  end if
  return (x^*, z^*)
end function

5.3 Parámetros del algoritmo

5.3.1 Radio de Shake

Determina el alcance de la perturbación en la función **Shake**. Un valor pequeño (por ejemplo, 5 km) mantiene la solución más estable. Sin embargo, un valor grande provoca exploraciones más agresivas, lo que permite al algoritmo escapar de posibles óptimos locales.

5.3.2 Tiempo de ejecución

En lugar de utilizar como criterio de parada el número de iteraciones, optamos por un límite de tiempo computacional, ya que proporciona mayor flexibilidad y control. Este parámetro será calibrado en un apartado posterior del informe, con el objetivo de determinar un tiempo máximo razonable a partir del cual no se observan mejoras significativas.

6 Análisis del tiempo computacional y de la calidad de la solución

En este apartado, trataremos de analizar y valorar la calidad de la solución obtenida una vez implementamos la heurística, no solo en términos de lo buena que sea la Función objetivo, sino también del tiempo que requiere para obtener dicha solución junto a la velocidad de convergencia.

Para ello, rescataremos el gráfico que hemos obtenido en primer lugar con la implementación del BVNS:

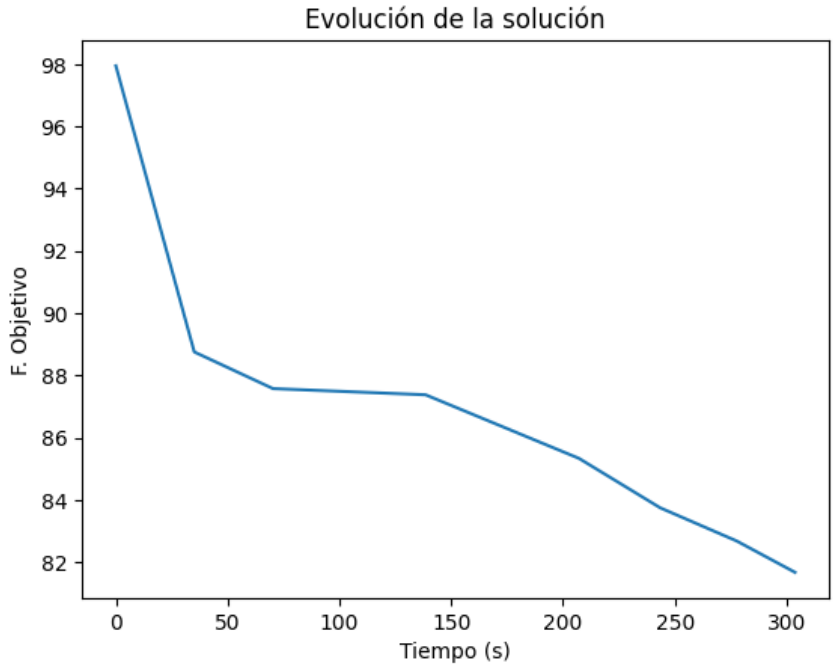

Figure 1: Evolución de la función objetivo usando BVNS tradicional

Podemos observar que la solución es razonablemente buena, ya que llega a un valor final para la función objetivo de 81.68 unidades. En cuanto al tiempo de ejecución, no es especialmente rápido, ya que ha tardado cinco minutos en llegar a dicha solución. Y, sobre todo, consideramos que tiene una convergencia lenta, ya que a excepción de los primeros cincuenta segundos en los que sí baja con rapidez, el resto del tiempo va mejorando a un ritmo bastante moderado. Y tras casi dos minutos de ejecución, aún apenas hemos llegado a soluciones ligeramente inferiores a noventa.

En contraste con el anterior, analizaremos el gráfico obtenido con nuestro algoritmo BVNS modificado, el cual hemos tratado de orientarlo en obtener soluciones de forma más rápida, como hemos explicado anteriormente:

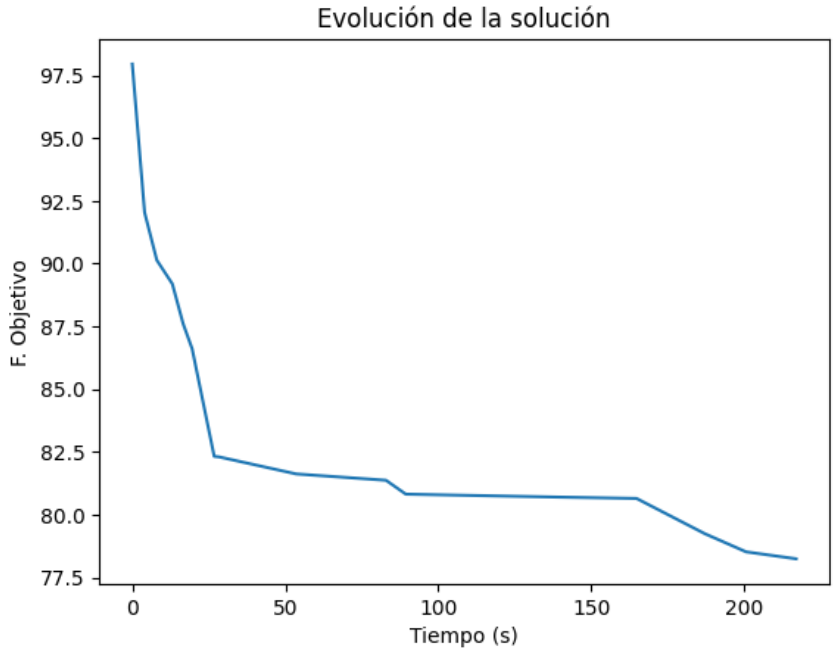

Figure 2: Evolución de la función objetivo con BVNS modificado

En este caso, podemos observar que la solución final obtenida, en cuanto a valor de la función objetivo es mejor con 78.24 unidades (frente a 81.68). Sin embargo, no es esta la diferencia más notable. Tampoco en cuanto al tiempo, ya que se ha obtenido en el segundo 220 frente al 300 anterior, hay entorno a un minuto de diferencia. Lo que sí marca la diferencia, es la velocidad de convergencia hacia buenas soluciones, ya que, en esta nueva implementación, podemos observar cómo baja mucho más rápido inicialmente, para llegar a soluciones muy buenas en apenas cien segundos.

Por tanto, como conclusión, sin duda hemos corroborado que la decisión de implementar estos cambios ha sido acertada. Ya que en situaciones en las que dispongamos de un tiempo más limitado, y necesitemos de soluciones buenas (sin necesidad de que sean perfectas), podremos aplicar este segundo algoritmo obteniendo en apenas un minuto un valor de 85, y sin embargo con el BVNS completo tardaríamos entorno a cuatro minutos en llegar a la misma solución.

7 Calibración de los parámetros del algoritmo

En este apartado, nos centraremos en tratar de encontrar los parámetros adecuados para obtener las mejores soluciones del algoritmo modificado que hemos propuesto y hemos podido ver que es más efectivo que el BVNS tradicional.

Cabe destacar que la calibración de los parámetros es específica de cada problema en particular y de cómo están distribuidos los municipios, sus distancias y demás, por lo que los resultados obtenidos aquí, no tienen por qué ser replicables en otro tipo de problema diferente.

En cuanto a los parámetros, por la definición de nuestro algoritmo nos centraremos en dos de ellos: el tiempo máximo de ejecución y el radio de cobertura de la función Shake.

Tiempo máximo de ejecución

Nuestro algoritmo funciona por tiempo de ejecución, es decir está realizando iteraciones en busca de mejores soluciones hasta que se consume el tiempo establecido. No tiene un límite de CPU o de iteraciones. Por tanto, en primer lugar, queremos averiguar cuál es el tiempo máximo con el que se obtienen las mejores soluciones, para así no tener que ejecutarlo durante más tiempo del necesario.

Además, como el otro parámetro que ajustaremos a continuación es el radio, probaremos a encontrar el tiempo máximo con tres radios distintos, representativos de radios pequeños (10 km), radios medianos (20 km) y radios grandes (50 km). Y así nos podremos fijar en si hay diferencias significativas en cuanto al tiempo requerido en función del radio.

Por tanto, hemos realizado un bucle con varias iteraciones por cada radio, y obtenemos el siguiente gráfico:

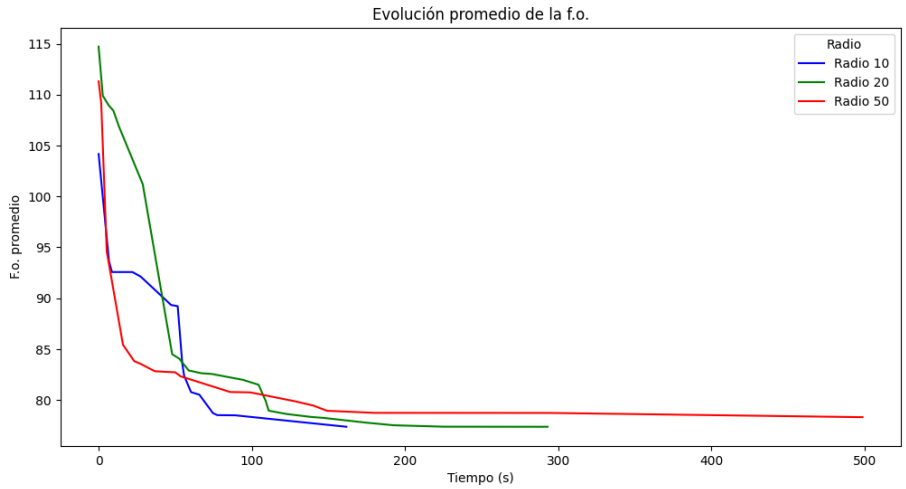

Figure 3: Evolución de la función objetivo para diferentes radios

Aquí se representa el tiempo empleado para cada radio, y como ha evolucionado la función objetivo. En primer lugar, podemos concluir que no parece haber diferencias destacadas en comportamiento para los diferentes radios. Como hemos comentado anteriormente, también se observa que en torno a cien segundos se llega a buenas soluciones. Y, por último, que el óptimo se halla en torno a los doscientos segundos de ejecución en todos los casos, por lo que no sería necesario invertir más de eso por cada iteración.

Radio del Shake

Como hemos explicado con anterioridad, el radio del Shake es el parámetro del algoritmo que determina en qué zona se puede desplazar la sede de la solución inicial. Es decir, mayores valores de radio sacuden más la solución y permiten una mayor exploración, y menores valores de radio conservan mayor parte de la solución anterior, al desplazar la sede a un municipio cercano, sin modificar en exceso la cobertura de cada sede.

Para tratar de encontrar el mejor valor, realizaremos tres iteraciones con diferentes radios, y esta vez fijaremos el tiempo a 250 segundos, ya que lo hemos calibrado anteriormente:

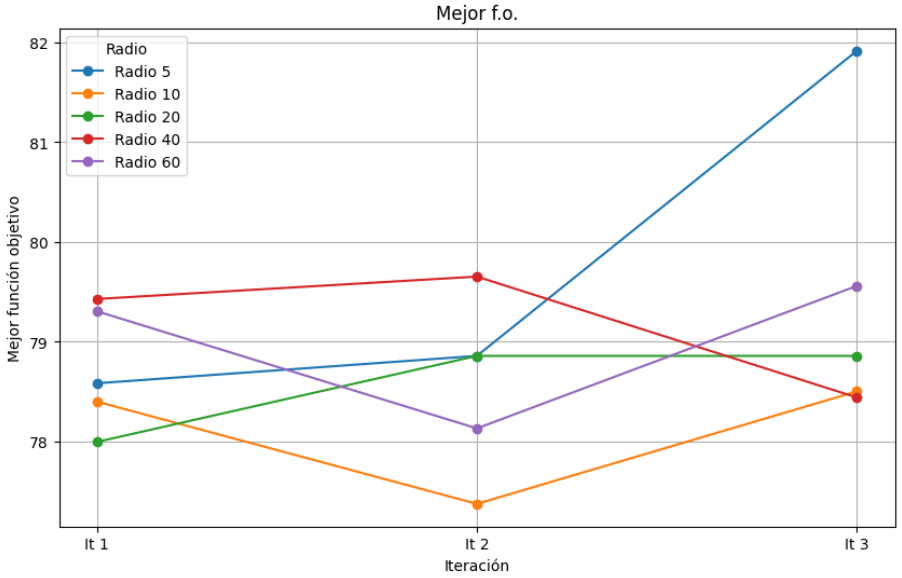

Figure 4: Comparativa del mejor resultado obtenido según el radio del Shake

Una vez ejecutado, en el gráfico anterior podemos observar el mejor valor obtenido en cada una de las 3 iteraciones, para todos los radios que hemos probado a calibrar.

Podemos observar que el mejor radio parece ser el de 10 km, ya que hemos obtenido la mejor solución de 77.38, y las otras dos iteraciones parecen consistentes, con valores no demasiado alejados. Para el radio de 20 km, observamos que en la primera iteración llegamos a una solución ligeramente mejor, pero en las otras dos parecen ser peores, por lo que tiene pinta de no ser tan consistente.

Además, el radio de 5km parece excesivamente pequeño para generar shakes adecuados, ya que las soluciones no son especialmente buenas y además una de ellas es la peor de todas. Los radios muy amplios como 40 y 60 parecen comportarse de forma más consistente, ya que al tener radios amplios exploran más region factible, sin embargo las soluciones no son las mejores en cuanto a función objetivo.

En conclusión, tomaremos el radio de 10km como nuestra elección, ya que parece combinar un equilibrio adecuado entre resultados fiables y consistentes, con resultados buenos y de bajo coste.

8 Mejor solución obtenida

En este apartado, hemos buscado encontrar la mejor solución posible para el problema, aplicando el algoritmo con los parámetros que acabamos de ajustar (tiempo = 250 s y radio = 10 km). Hemos iterado la misma ejecución 10 veces para tratar de desaleatorizar los resultados, y obtener la máxima información posible.

Resultados:

- **Función objetivo:** 76.87

- **Tiempo de ejecución:** 193.02 s
- **Función objetivo promedio:** 77.72
- **Tiempo de ejecución promedio:** 201.54 s

Observamos que los resultados tanto de tiempo como de valor son muy consistentes, por lo que parece que el algoritmo funciona correctamente, destacando que llega a valores de función objetivo inferiores a 78 en 8 de las 10 ejecuciones.

Información completa de la solución:

En cuanto a reparto de sedes, ambulancias, y conexiones entre los municipios, la solución obtenida es la siguiente:

Figure 5: Asignación de municipios por sede

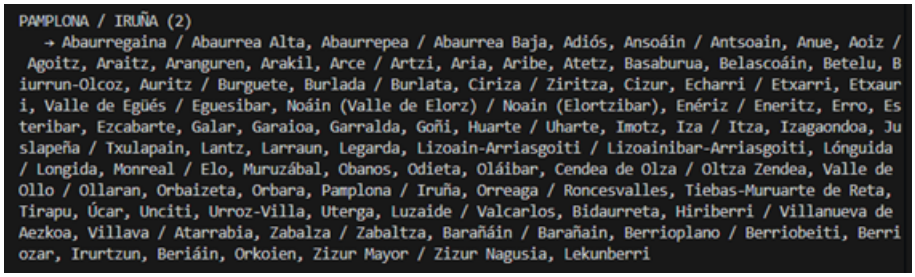

Figure 6: Asignación de municipios por sede

```
PERALTA / AZKOIEN (1)
   → Andosilla, Azagra, Caparroso, Cárcar, Carcastillo, Falces, Funes, Mélida, Milagro, Murillo el Cue
nde, Peralta / Azkoien, San Adrián, Villafranca

RIBAFORADA (1)
   → Ablitas, Buñuel, Cabanillas, Cortes, Fontellas, Fustiñana, Ribaforada

TAFALLA (1)
   → Añorbe, Artajona, Barásoain, Beire, Berbinzana, Eslava, Garínoain, Larraga, Leoz / Leotz, Lerga,
Marcilla, Mendigorria, Miranda de Arga, Murillo el Fruto, Olite / Erriberri, Olóriz / Oloritz, Orísoai
n, Pitillas, Pueyo / Puiu, San Martín de Unx, Santacara, Tafalla, Ujué / Uxue, Unzué / Untzue

TUDELA (1)
   → Arguedas, Barillas, Cascante, Monteagudo, Murchante, Tudela, Tulebras
```

Figure 7: Asignación de municipios por sede

Y podemos representarla en el mapa de la siguiente forma:

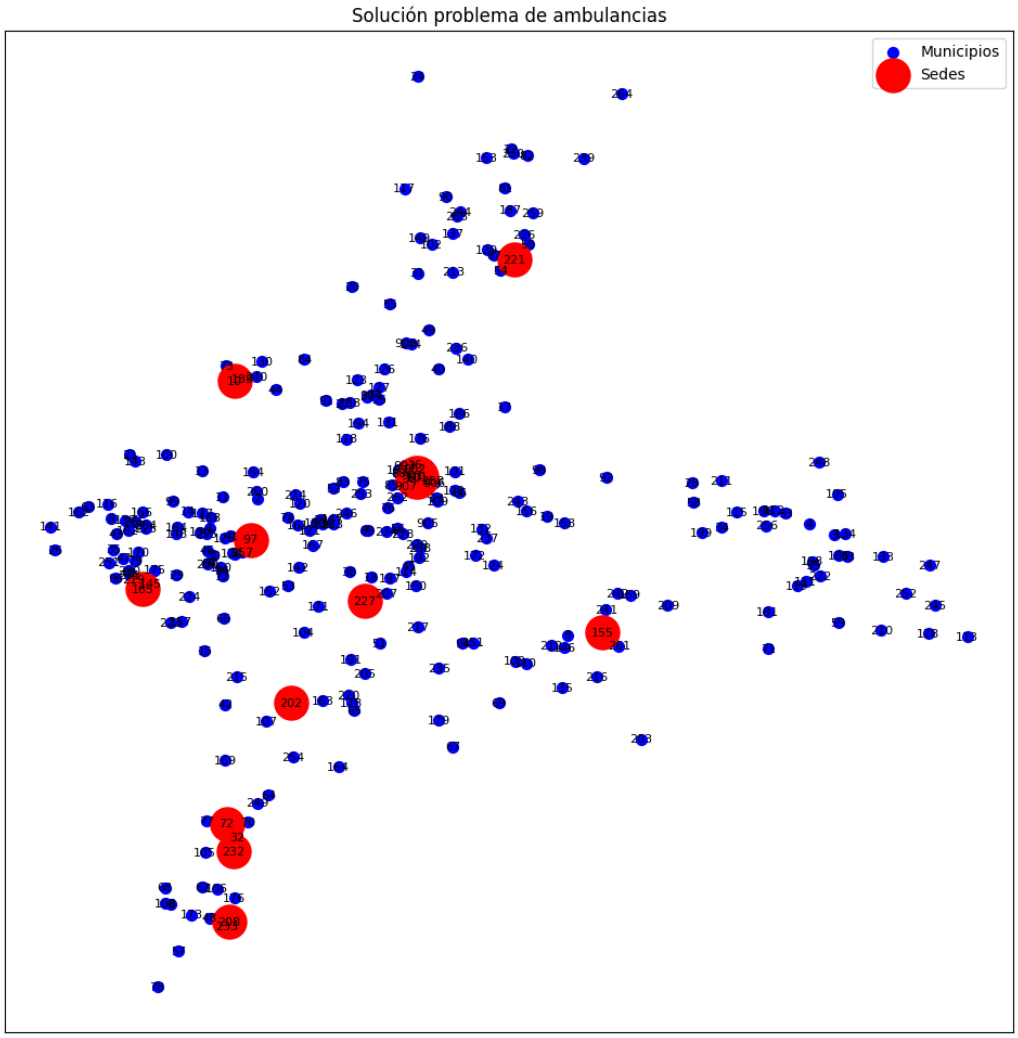

Figure 8: Visualización de la solución final

9 Discusión y conclusiones

A lo largo de este trabajo hemos desarrollado e implementado distintas versiones del algoritmo BVNS con el objetivo de optimizar la localización de ambulancias en Navarra. Tras los análisis correspondientes, hemos decidido implementar una versión modificada,

y hemos visto que la decisión ha sido acertada ya que se ha evidenciado que no solo mejora la calidad final de las soluciones, sino que lo hace más rápido, especialmente en los primeros compases de ejecución.

Además, creemos que en el caso de estos algoritmos heurísticos también debe aplicar el principio de parsimonia, y depende del contexto, probablemente no podamos justificar un algoritmo que tarda horas en ejecutarse, a costa de obtener una solución apenas un 1% mejor que otros algoritmos que llegarían en pocos minutos. En esta balanza, valoramos que hemos conseguido un algoritmo suficientemente equilibrado en cuanto a rapidez/calidad.

Los cambios propuestos se pueden resumir en cambiar las funciones que evalúan el vecindario completo y obtienen la mejor solución, por funciones que solamente toman una muestra del vecindario, y cambios en la función de Shake para otorgar más robustez ante posibles óptimos locales y mayor posibilidad de escape.

Finalmente, con la calibración de parámetros hemos logrado optimizar las posibles variables a controlar para obtener las soluciones de la mejor forma posible. Hemos controlado tanto el tiempo de ejecución como el parámetro del Shake. Esta configuración ha demostrado un rendimiento consistente y con buenas soluciones.

En conclusión, creemos que la implementación es adecuada y hemos tratado de aplicar diversos conceptos vistos a lo largo del curso. Sin embargo, hemos visto que también hay soluciones ligeramente mejores. Nos preguntamos si la elección de algún otro algoritmo podría ayudarnos a obtener aún mejores resultados, y si en nuestro caso, nos hemos visto algo limitados en la solución final por las peculiaridades de este tipo de algoritmos de la familia VNS.

En cuanto a aplicaciones futuras, podríamos tratar como acabamos de mencionar con otro tipo de familias de algoritmos, o opciones híbridas que implementen varias familias de algoritmos en una misma propuesta. Nos gustaría también poder ver las aplicaciones de esta misma solución con problemas similares, pero de distintas regiones, ya que tal vez en otro tipo de regiones geográficas, con diferente densidad de población y municipios no sería replicable, y tenemos curiosidad por saber cuánto podría afectar esto.

A Código fuente del Jupyter Notebook

Todo el código de análisis y generación de resultados está disponible en formato HTML en: code.html