Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Organización Industrial

Modelización y métodos heurísticos de solución para problemas de localización con costos fijos

Autor:

Nuevo Tapioles, Pablo

Tutor:

Mata Crespo, Raquel
Departamento de Estadística e Investigación Operativa

Valladolid, Julio de 2019.

Agradecimientos

Quiero agradecer a todos aquellos que me han apoyado durante mi paso por esta carrera.

A mi familia y a toda la gente que en estos cuatro años de carrera me han aportado diferentes visiones de la vida y ganas para seguir formándome.

Agradecer a todos los profesores que me han enseñado y en concreto a mi tutora Raquel Mata y a Jesús Sáez, que me han ayudado a realizar este trabajo.

Resumen

Nos encontramos en una sociedad en el que el campo de actuación y la situación de las empresas más potentes se encuentran distribuidos, ya sea por una zona, un país o por todo el mundo. Con los modelos de localización, es posible cubrir los aspectos necesarios para determinar cuántas instalaciones abrir y donde situarlas, de modo que el coste sea el menor posible. De esta decisión depende en gran medida el éxito y la prosperidad de la empresa.

En este trabajo de Fin de Grado serán expuestos los problemas de localización sin capacidades, con capacidades y su variante de fuente única, encontrando la solución óptima exacta en Xpress-MP y desarrollando heurísticas Greedy de construcción y heurísticas de mejora de la solución basadas en búsqueda local, que sean rápidas y simples, pero presentando soluciones aceptables.

Palabras clave

Optimización; Localización; Instalaciones; Formulación; Heurísticas;

Definiciones básicas

Localizar: Determinar o señalar el emplazamiento que debe tener alguien o algo (RAE).

Heurística: Método, estrategia, truco o criterio que utilizaremos para que conseguir el resultado de un problema difícil sea más sencillo.

Relajación: Es el problema de programación lineal que obtenemos al prescindir de las condiciones de integridad de las variables. Ante un problema difícil de resolver de programación entera, usamos las relajaciones, para que con ellas el problema sea más fácil de resolver.

Algoritmo: Método, grupo de instrucciones que se llevan a cabo en caso de que se cumplan ciertas condiciones y con las que se puede obtener la respuesta de un problema al que nos enfrentemos.

Restricción: Condición que se tiene que cumplir en el problema que la introducimos.

Contenido

Índice de tablas

Índice de figuras

1 Introducción

1.1 Motivación

He decidido realizar este Trabajo de Fin de Grado por diferentes motivos que expondré a continuación.

Dando una mirada atrás a todos los temas tratados durante mi paso por el Grado de Ingeniería en Organización Industrial me he dado cuenta de que entre los temas que he tratado que considero más interesantes, se encuentran los impartidos en la asignatura de "Métodos cuantitativos en ingeniería en organización I" impartida actualmente por Raquel Mata Crespo del Departamento de Estadística e Investigación Operativa de la Universidad de Valladolid. Ella, en colaboración con Jesús Sáez Aguado me ha propuesto este tema que casualmente había llamado ampliamente mi atención en la asignatura. Además, la asignatura de "Estadística" de primer curso me proporcionó las herramientas necesarias para el análisis de datos, estadísticos y representaciones gráficas del trabajo.

En un mundo empresarial globalizado nos damos cuenta de la importancia que tiene el sector de la logística y la necesidad de reducir costes prescindibles, para poder asegurar la prosperidad de una empresa y en este trabajo, se abordará un método para conseguirlo de forma rápida y eficiente.

1.2 Contenido

Se abordará el problema de localización inicialmente sin capacidades y, posteriormente, con capacidades. Se considerará también como variante de este tipo de problemas el modelo de localización con capacidades y fuente única (SSCFL). Se proporcionará la formulación de los modelos para estos tipos de problemas con costos fijos.

En las decisiones de optimización sobre la localización se requiere de modelos de Programación Entera Mixta (MIP) y Programación Entera Pura (PEP).

Se presentarán heurísticas Greedy de construcción y se desarrollarán heurísticas de mejora en la solución basadas en búsqueda local.

Se estudiará la heurística final con intención de ser rápida y simple, proporcionando soluciones aceptables, aunque no garantiza que se alcance la solución óptima.

1.3 Objetivos

A continuación se exponen los principales objetivos marcados para este trabajo de Fin de Grado:

- Iniciar un proceso de organización y planificación de la investigación.
- Estudiar los modelos y su formulación en problemas de localización.
- Obtener conjuntos de datos de la OR Library disponible en internet.
- Resolver de forma exacta con Xpress-Mosel problemas difíciles.
- Desarrollar heurísticas Greedy de construcción para resolver los problemas de forma aproximada.
- Desarrollar métodos de mejora basados en búsqueda local con intención de acercarse más a la solución exacta.
- Obtención de resultados aproximados que sean aceptables de una forma sencilla y rápida cuando no se dispone del solver para encontrar la solución exacta.
- Análisis estadístico de los resultados obtenidos con los métodos heurísticos en cuanto a tiempo computacional y calidad de la solución con Statgraphics.

2 Problemas de localización

2.1 Planteamiento de modelos

El problema de localización de instalaciones ha sido objetivo de estudio a lo largo de los años, pero no ha sido hasta mediados de los 60, la década en el que se han empezado a crear algoritmos realistas para su solución, gracias a los avances tecnológicos de las computadoras. **Revelle y Laporte (1996)**.

Formalmente, el libro de A.Weber "Theory of the Location of Industries" de 1909 es considerado por la comunidad científica como punto de partida en el estudio de este tipo de problemas, sin embargo algunos expertos piensan que su estudio comenzó de la mano de Pierre de Fermat, Evagelistica Torricelli (un estudiante de Galileo) y Battista Cavallieri. **Farahni, SteadieSeifi y Asgari (2010)**

Generalmente, en este problema se tienen una serie de puntos donde poder localizar instalaciones y una serie de clientes cuya demanda ha de ser satisfecha, buscando optimizar cierto objetivo, dependiendo de ese objetivo es posible subdividir los problemas de localización en seis distintos tipos:

- Problemas de Cubrimiento Total (Set Covering).
- Problemas de cubrimiento Máximo.
- Problemas de Localización de Medianas.
- Problemas de Centros de Emergencia.
- Problemas de Localización de Costos Fijos.
- Problemas de Distritos.

En este trabajo se va a desarrollar el Problema de Localización con Costos Fijos, debido a sus numerosas extensiones y aplicaciones en el ámbito de la Ingeniería.

2.1.1 El problema de localización con costos fijos

Este problema tiene un gran número de aplicaciones, en todas ellas va a haber un desplazamiento para satisfacer la necesidad del cliente. Varios ejemplos de las áreas de aplicación son: apertura de nuevas plantas de empresas en crecimiento, situación de servicios públicos como un hospital, posicionamiento de almacenes para empresas de distribución, organización de horarios en aerolíneas, localización de instalaciones en planta.

Si la capacidad de producción es ilimitada, se tiene un problema de localización sin capacidades (UFLP), si es limitada, un problema de localización con capacidades (CFLP) y como variante, en caso de que las necesidades de cada cliente solo puedan ser satisfechas por una instalación, se tiene una variante del problema de localización con capacidades llamada problema de localización con capacidades y fuente única (SSCFLP). Las siglas corresponden a sus nombres en inglés

En todas las variantes del problema de localización se cuenta con una serie de factores en común.

En primer lugar se considera el conjunto de índices $J = \{j_1, j_2, \ldots, j_n\}$, que serán los puntos en los que se podrá situar una instalación.

En segundo lugar se considera el conjunto de índices $I = \{i_1, i_2, \ldots, i_m\}$, que serán un conjunto de clientes o puntos de demanda de un servicio que habrá que satisfacer.

También se tiene una serie de costes que interesa hacer mínimos, que son los costes fijos f_j ocasionados por la apertura de una instalación i en un punto j y unos costos variables c_{ij} (costo de servicio de transporte de una unidad producida en una planta j hasta el punto de demanda i).

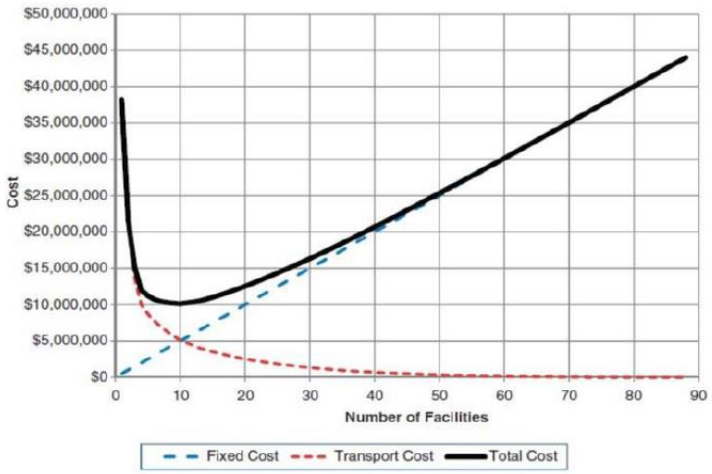

Ilustración 1 **Prototipo de costes de instalaciones, transporte y totales (Daskin, 1995)**

En la ilustración anterior se puede ver una curva de costes en la que a medida que se aumenta el número de instalaciones los costes de transporte son menores, sin embargo los costes fijos van aumentando. Esta curva pasa por un punto en el que el coste total, que consiste en la suma de los costes variables y fijos, es el mínimo posible y por lo tanto, el deseable.

Se dice de este tipo de problemas que su complejidad computacional es NP-hard, puesto que se cree que no existe un algoritmo polinómico para resolverlo, sino un algoritmo de orden exponencial. El tiempo de solución para este tipo de problemas, en el peor de los casos aumentará exponencialmente con su tamaño.

Con intención de resolver este tipo de problemas, se presentarán algunos métodos heurísticos que proporcionarán una solución aproximada aceptable en menor tiempo.

2.1.2 El problema de localización sin capacidades (UFLP)

En este problema se parte de un conjunto de puntos donde podemos situar una instalación, cuya notación será: $J = \{j_1, j_2, \ldots, j_n\}$ y otro conjunto de clientes, o puntos de demanda de un servicio cuya notación será: $I = \{i_1, i_2, \ldots, i_m\}$, los cuales, para tener un modelo de localización sin capacidades (UFLP) vamos a suponer en todo caso que una instalación puede proporcionarles cualquier número de unidades. Cada punto de demanda tiene una demanda d_i .También existe un costo fijo f_j ocasionado por la apertura de una instalación en un punto j y un coste variable c_{ij} (costo de servicio de transporte de una unidad producida en una planta j hasta el punto de demanda i). Es un costo proporcional a la distancia d_{ij} y la matriz de costos variables será nombrada como C.

La finalidad será satisfacer las necesidades de los clientes i eligiendo el lugar j en el que debemos de abrir las instalaciones de modo que el coste total sea el menor posible.

Este coste total F es calculado del siguiente modo:

$$\sum_{j=1}^{n} f_j \; x_j + \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} \; z_{ij} \qquad (1)$$

Para obtener el valor que minimice estos costes recurriendo a la programación entera mixta, se tendrá que minimizar la función (1) sujeta a las siguientes condiciones:

$$\sum_{j=1}^{n} z_{ij} = d_i \quad i \in I \qquad (2)$$

Siendo z_{ij} la demanda de un cliente i a un punto j y d_i la demanda total del cliente.

Así mismo se imponen las siguientes restricciones:

$$\sum_{i=1}^{m} z_{ij} \leq Mx_j \quad j \in J \qquad (3)$$

$$z_{ij} \geq 0 \qquad j \in J, i \in I \qquad (4)$$

$$x_j \in \{0,1\}\, j \in J \qquad (5)$$

Se dice que es un modelo de programación entera mixta (MIP) debido a que en el modelo están presentes dos tipos de variables:

1. Las binarias x_i [Restricción (5)] que pueden tomar dos valores {0,1}, los cuales corresponden a las decisiones de apertura de la planta. En el caso de que se abra la planta la variable pasará a tener el valor 1, en el caso contrario 0.
2. Las continuas z_{ij} que en este caso indican la cantidad demandada de un punto i a un punto j.

La restricción de integridad no afecta a todas las variables y la variable a la que afecta solo puede tomar valores binarios, por lo que se está hablando de un problema de Programación 0-1 (Binaria) Mixta.

En el caso de la restricción (3), siendo M un número suficientemente grande podemos garantizar a la hora de resolver el problema que si una instalación no está abierta y toma el valor 0 no pueda satisfacer ningún tipo de demanda. Es decir:

$$x_j = 0 \Rightarrow z_{i_j} = 0, i \in I$$

Este caso es conocido como la formulación de la programación débil o agregada.

Sin embargo, también existe un modelo más utilizado en programación entera mixta (MIP), en el cual se realiza un cambio de variable. De este modo se tiene la variable $y_{ij} = \frac{z_{ij}}{d_i}$ y la formulación obtenida es conocida como formulación de la programación fuerte o desagregada.

Esta nueva variable pasará a indicar qué fracción de la demanda del cliente i es cubierta desde la instalación j, por lo tanto esta variable de carácter continuo estará comprendida entre o y 1 [*véase la ecuación (9)*].

A continuación se formula de nuevo el problema de transporte sin capacidades con costos fijos (UFLP) con la nueva variable (formulación fuerte):

Coste total (Función a minimizar)

$$\sum_{j=1}^{n} f_j \; x_j + \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} \; d_i \; y_{ij} \qquad (6)$$

Sujeto a:

$$\sum_{j=1}^{n} y_{ij} = 1 \quad i \in I \qquad (7)$$

$$\sum_{i=1}^{m} y_{ij} \leq m x_j \quad j \in J \qquad (8)$$

$$0 \leq y_{ij} \leq 1 \quad j \in J, i \in I \qquad (9)$$

$$x_j \in \{0,1\} \; j \in J \qquad (10)$$

La suma de las fracciones de demanda de un cliente cubiertas por una planta es igual a 1, lo que es equivalente a la demanda total del cliente (7).

En este modelo no es necesario añadir un número M suficientemente grande al igual que en (3), ya que al haber incluido una variable cuyo máximo valor puede ser 1 (y_{ij}) la suma $\sum_{i=1}^{m} y_{ij}$ será como máximo igual al número m de plantas, y siempre igual al número de plantas abiertas (8).

Se dice que tenemos un problema de dimensión $\mathrm{n} \times \mathrm{m}$ siendo n el tamaño del conjunto J y m el tamaño del conjunto I. La matriz de costos variables C tiene el mismo tamaño que la dimensión del problema, $\mathrm{n} \times \mathrm{m}$.

La diferencia entre la formulación de la programación débil y la formulación de la programación fuerte estriba en que la solución obtenida mediante relajación lineal con la formulación fuerte está más próxima a la solución óptima entera. En el apartado de resultados esta afirmación será probada de forma numérica con las diferentes soluciones obtenidas.

2.1.3 Problema de localización con capacidades (CFLP)

En este problema, al igual que en el problema de localización sin capacidades (UFLP), se tienen una serie de puntos donde podemos situar una instalación, cuya notación será: J = {j_1, j_2, …, j_n} y un conjunto de clientes, o puntos de demanda de un servicio cuya notación será: I = {i_1, i_2, …, i_m}, pero en este caso, para tener un modelo de localización con capacidades (CFLP) las instalaciones solo van a poder proporcionar un determinado número de unidades como máximo, denominado capacidad u_j para satisfacer la demanda del cliente.

Se mantiene la demanda d_i .También el costo fijo f_j ocasionado por la apertura de una instalación i en un punto j y el coste variable c_{ij}.

La finalidad será satisfacer las necesidades de los clientes i eligiendo el lugar j en el que se debe abrir las instalaciones de modo que el coste total sea el menor posible, para lo que normalmente la utilidad de la planta, es decir, el número de unidades producidas en una planta para satisfacer la demanda del cliente, ha de estar lo más cerca posible de su capacidad, sin poder llegar a desbordarla.

Para cumplir con ello, se deben de poner una serie de condiciones a la hora de formular el problema según un modelo de programación entera mixta (MIP).

Se pretende minimizar el coste total:

$$\sum_{j=1}^{n} f_j \, x_j + \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} \, d_i \, y_{ij} \qquad (11)$$

Sujeto a:

$$\sum_{j=1}^{n} y_{ij} = 1 \quad i \in I \qquad (12)$$

$$\sum_{i=1}^{m} d_i \, y_{ij} \leq u_j \, x_j \quad j \in J \qquad (13)$$

$$0 \leq y_{ij} \leq 1 \quad j \in J, i \in I \qquad (14)$$

$$x_j \in \{0,1\} \quad j \in J \qquad (15)$$

Siendo las ecuaciones (6, 7, 9, 10) iguales respectivamente a las siguientes ecuaciones (11, 12, 14, 15), añadiendo la ecuación (13) que equivale a la restricción de la capacidad, ya que la suma de toda la demanda de un cliente no puede ser mayor a la capacidad de la instalación que cubre la demanda, y asegurando que se cumpla la condición lógica:

$$x_j = 0 \Rightarrow y_{ij} = 0, \qquad i \in I$$

2.1.4 Problema de localización con capacidades y fuente única (SSCFLP)

Este modelo, conocido como modelo de localización con capacidades y fuente única, es un caso particular de los modelos citados anteriormente. Se distingue en que la demanda de un cliente solamente puede ser cubierta por una instalación.

En el caso del modelo de localización sin capacidades (UFLP) se puede obtener la solución óptima añadiendo la condición de fuente única y verificando que es la misma que la que se ha obtenido anteriormente. Esto es debido a que la solución óptima asigna a cada punto una única instalación, ya que al no tener capacidades existe una instalación con menor coste asignada a cada cliente, que le proveerá de todas las unidades necesarias. Sin embargo en el caso de problema con capacidades esto no se puede garantizar. Por ese motivo se dice que este modelo, que será formulado a continuación, es una variación del modelo de localización con capacidades.

La función objetivo a minimizar, de coste total:

$$\sum_{j=1}^{n} f_j \, x_j + \sum_{j=1}^{n} \sum_{i=1}^{m} c_{ij} \, d_i \, y_{ij} \qquad (16)$$

Sujeto a:

$$\sum_{j=1}^{n} y_{ij} = 1 \quad i \in I \qquad (17)$$

$$\sum_{i=1}^{m} d_i \, y_{ij} \leq u_j \, x_j \quad j \in J \qquad (18)$$

$$y_{ij} \in \{0,1\} \quad j \in J, i \in I \qquad (19)$$

$$x_j \in \{0,1\} \, j \in J \qquad (20)$$

Siendo las ecuaciones (16, 17, 18, 20) iguales respectivamente a las siguientes ecuaciones (11, 12, 13, 15) y haciendo que y_{ij} pase a ser una variable binaria [ecuación (19)], que cuando vale 1, significa que la demanda del cliente i es cubierta por la instalación j, es decir que el punto de demanda está asignado a una instalación candidata j, y cuando vale o significa que la demanda del cliente i no es cubierta por la instalación j, es decir que el punto de demanda no está asignado a una instalación candidata j, la cual junto con la ecuación(17) asegura que la demanda de un cliente solo sea satisfecha por una instalación.

2.2 Solución de los modelos.

2.2.1 Software

Como se ha dicho anteriormente, los modelos de localización con costos fijos, sin capacidades, con capacidades y de fuente única son modelos de Programación Entera Mixta, o más concretamente de Programación 0-1 (binaria) Mixta.

Para conseguir las soluciones óptimas, con las que compararemos la precisión de las heurísticas se resuelven varios conjuntos de problemas con el entorno de optimización Xpress-Mosel. Este programa cuenta con el método simplex para resolver problemas de Programación Lineal y con el algoritmo "Branch&Bound", actualmente conocido como "Branch and Cut", que es sinónimo en castellano de "ramificación y poda o corte" para la resolución de problemas de Programación Entera.

2.2.2 Librería de datos

Se toman como conjuntos de datos algunos procedentes de la OR Library disponible en internet. La importancia de esta librería reside en que las publicaciones científicas asociadas realizan todos los estudios con estos conjuntos de datos, por lo que existe la facilidad de trabajar con ellos de forma repetida y obtener exactamente los mismos resultados, puesto que se suele facilitar la solución exacta en caso de no disponer de un solver específico.

Este trabajo pretende resolver estos problemas, por lo que en el código programado en Xpress-Mosel han de meterse como entrada unos datos que, por comodidad, serán leídos desde un fichero con extensión ".txt".

2.2.3 Branch&Bound.

Para solucionar un problema mediante este método en primer lugar ha de plantearse el modelo de Programación Entera.

Contamos con las variables continuas v, w, etc. además de una función a optimizar Z.

En este algoritmo, en una primera instancia se procede a realizar una "relajación" del problema, es decir, se elimina la condición de integridad de las variables con dicha condición y se procede a solucionarlo mediante el método Simplex ya que sin dicha condición estamos ante un problema de Programación Lineal, también conocido como un problema "relajado" en este caso.

Se obtiene una solución tal que:

$$v = a, w = b$$

Es posible que las variables que deberían de cumplir condiciones binarias o simplemente de integridad las cumplan, por lo que la solución de este problema relajado sería la misma que la del problema inicial al cumplir las mismas condiciones. Desafortunadamente es algo que no siempre ocurrirá por lo que hay que pasar al siguiente punto, la "ramificación".

Se elige una variable X_i, que es la variable no entera con mayor valor fraccionario y se obtienen dos nuevos problemas (P1 y P2) a los que se agrega respectivamente las restricciones:

$$v \leq [a]$$

$$v \geq [a] + 1$$

Siendo [a] el valor íntegro de la solución obtenida anteriormente, es decir:

$$a = 12{,}3 \Rightarrow [a] = 12$$

Para solucionar dichos problemas se asigna a la variable v los valores de las restricciones $[a]\ y\ [a] + 1$ en P1 y P2 respectivamente y se obtiene el valor de las demás variables. En estos dos nuevos problemas la variable v, que anteriormente tenía una solución fraccional, ahora tiene una solución entera, por lo que se dice que en cada ramificación el problema tiende más a la integridad.

Si ambos problemas se pueden solucionar de modo que se satisfagan las condiciones del problema inicial a resolver, bastará con elegir el planteamiento con la solución más óptima. Si uno de ellos se puede solucionar obteniendo los valores discretos que cumplen las condiciones se dice que es un problema viable y se deja de ramificar en esa parte.

A medida que la ramificación avanza, en cada rama en caso de que se busque una maximización el valor de la función objetivo irá disminuyendo y en el caso de una minimización ocurrirá lo contrario.

En resumen, la idea del algoritmo B&B (Branch and Bound) consiste en hacer ramificaciones sucesivas de las variables que no han cumplido con las restricciones de integridad.

A continuación se ofrece una representación visual del algoritmo B&B:

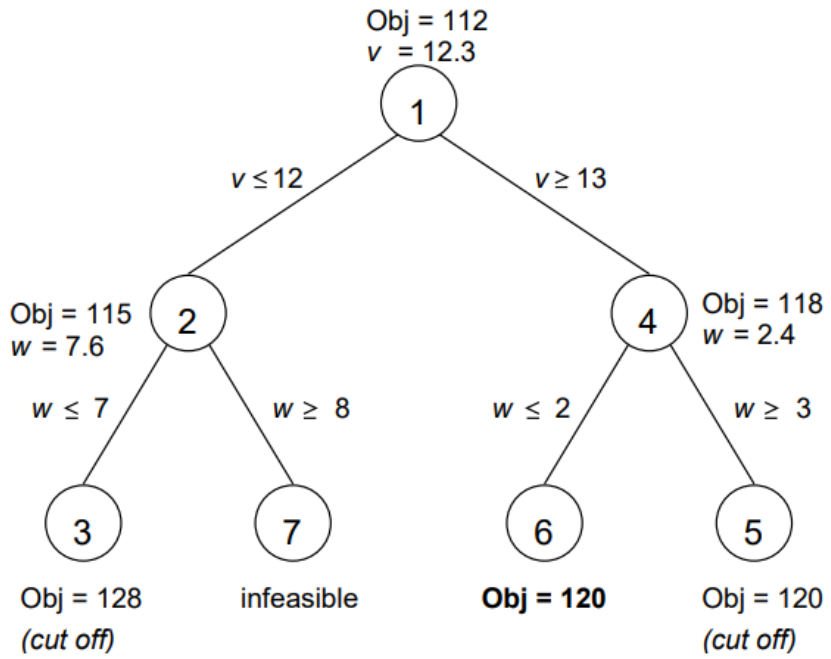

Ilustración 2 **Recuperado de: Guéret, Christelle,Prins Christian,Sevaux,Marc. Applications of optimization with Xpress-MP. 1999**

En este ejemplo se tienen 7 nodos relacionados mediante arcos que representan las ramificaciones realizadas.

Como se puede observar en el nodo 1 tenemos un valor para la solución objetivo (112) que es el valor que pretendemos que sea el mínimo posible, el cual a medida que se avanza en la ramificación está aumentando. Por otra se puede observar la variable v que toma el valor 12,3 en esa solución. Tras obtener los dos siguientes problemas se va a forzar a v a valer 12 en el caso de $v \leq 12$ y 13 en el caso de $v \geq 13$.

Añadiendo esta restricción a las anteriores se obtienen nuevos resultados (nodos 2 y 4) en los que el valor de la función objetivo es diferente y otra variable que debería de ser discreta (w) ahora tiene un valor fraccional. Por ese motivo se vuelve a ramificar, obteniendo así cuatro soluciones (nodos 3, 7, 6 y 5).

Cuando una solución es íntegra y factible se deja de explorar dicho nodo y se continúa con los demás, ya que esa solución va a ser una solución posible para el problema inicial al cumplir las condiciones de integridad. En el ejemplo superior podemos observar que el nodo 6 es factible y el valor de la función objetivo es de 120, por lo que no debemos de seguir ramificando los nodos 3 y 5 ya que, aunque aún no se haya alcanzado una solución factible, al saber que va a aumentar el valor de la función objetivo a medida que siga la cadena, no es necesario, ya que los valores de los nodos 3 y 5 son 128 y 120 respectivamente. Por otra parte tampoco se prosigue con el nodo 7 ya que no es factible su solución.

En un problema de programación entera en el que se cuente con N variables discretas, será teóricamente necesario, la generación de 2^N nodos. Esto es debido a que en cada nivel dentro del "árbol del problema" (Nombre para referirse al esquema seguido en la figura 1) una única variable obtiene un valor entero.

Se dice que es un algoritmo de orden de complejidad exponencial $O(a^n)$, puesto que para estos algoritmos no es posible acotar su complejidad de forma polinómica.

Poniéndonos en lo peor, el tiempo necesario para resolver un caso en este problema, crecerá de forma exponencial a medida que aumente el tamaño del caso.

A continuación se presenta un diagrama de flujo basado en Tapias-Isaza, C. J., Galeano-Ossa, A. A., e Hincapié-Isaza, R. A. (2011).

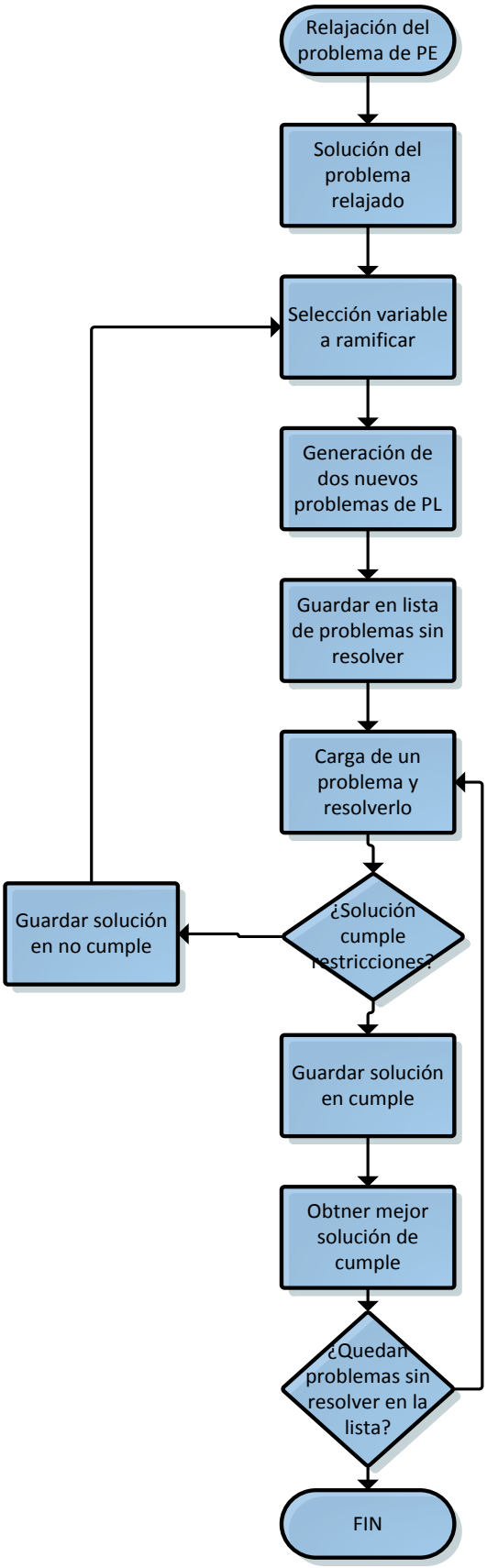

Ilustración 3 **Diagrama de flujo de Branch&Bound**

3 Métodos heurísticos.

3.1 Introducción

La palabra heurística proviene del griego (εὑρίσκειν) que significa hallar o inventar y según la RAE es una "técnica de la indagación y del descubrimiento".

Como se ha expuesto anteriormente, los problemas de localización son de un elevado orden de complejidad, por lo que los métodos de optimización, a medida que se presenta un problema más complejo aumentan el tiempo necesario para solucionarlo.

En este trabajo se presentarán heurísticas para la solución del problema de localización, en las cuales, no se buscará la obtención de un resultado exacto (no obstante, a veces se alcanzará la solución óptima) sino que se buscará la obtención de un resultado aceptable en el menor tiempo posible. A estos métodos heurísticos también se les conoce por el nombre de metaheurísticas.

Para ser apropiada, una metaheurística ha de presentar según Morales (2004), al menos una de las siguientes propiedades:

- Simplicidad.
- Precisión.
- Coherencia.
- Eficiencia.
- Efectividad.
- Robustez.

Por otra parte, Rafael Martí (2003), clasifica los métodos heurísticos de la siguiente forma:

- Métodos de descomposición:
  El problema inicial se descompone en subproblemas más sencillos de resolver, sin olvidar que ambos pertenecen al mismo problema.

- Métodos inductivos:

El objetivo de este método es la traslación de propiedades a casos más simples para el estudio, de lo que sería en el problema completo.

- Métodos de reducción:

Consiste en identificar propiedades que se cumplan en la mayoría de los casos e introducirlos como una restricción nueva en el problema, obteniendo así un problema más simple, sin embargo, se pueden dejar fuera las soluciones óptimas del problema original.

- Métodos constructivos:

Se construirá una solución paso a paso eligiendo la mejor solución posible en cada iteración.

- Métodos de mejora basados en Búsqueda local:

En este caso se tiene una solución inicial de un problema, la cual se mejora progresivamente buscando en un espacio de posibles soluciones.

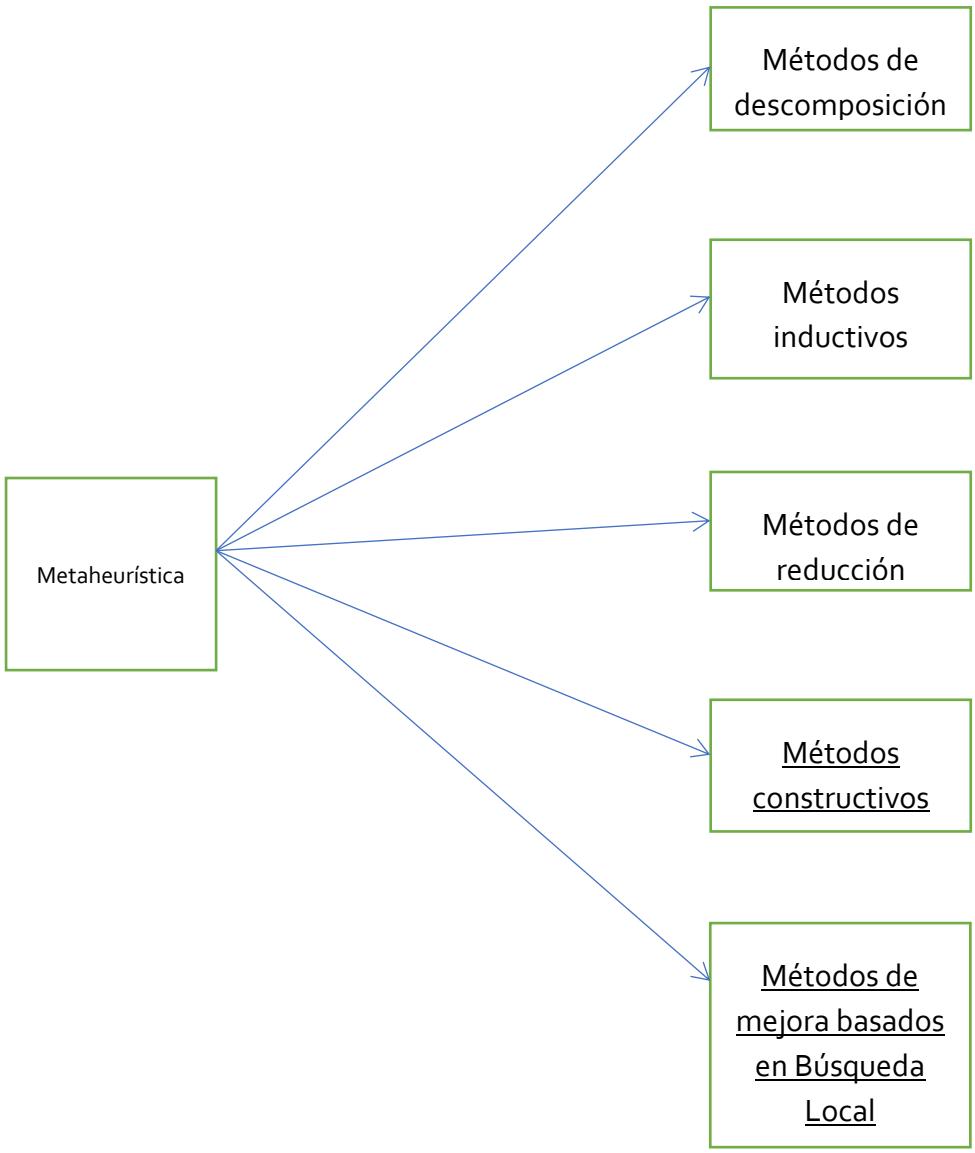

Ilustración 4 **Esquema heurísticas**

Concretamente, en este trabajo se desarrollarán heurísticas Greedy de construcción y posteriormente se desarrollarán heurísticas de mejora en la solución, basándose en estrategias de búsqueda local.

3.2 El algoritmo voraz o “Greedy”

3.2.1 Introducción

Este algoritmo está diseñado con el propósito de encontrar una solución aceptable, pero de una forma rápida. Normalmente esta solución obtenida se utilizará como solución inicial en heurísticas de mejora.

Sigue un proceso iterativo, en cada iteración se toma una decisión. Esa decisión es la asignación de uno de los posibles candidatos, el cual, es un óptimo local. Esto produce que se le asigne un valor a una variable que hará que el valor de la función objetivo, en el caso de una minimización sea mínimo y viceversa.

Esta decisión es irrevocable y una vez ha sido asignado ese valor no se le quitará, por este motivo también son conocidos como “algoritmos miopes”, haciendo referencia a que “no ven más allá” de la solución actual. Acto seguido se vuelve a realizar el mismo proceso hasta haber asignado un valor a cada variable, lo que produce que el valor de la función objetivo se obtenga de forma rápida, pero que este no sea óptimo. En casos en los que hay un elevado número de variables su proximidad con la solución óptima va siendo menor a medida que el número de variables aumenta. En resumidas cuentas, este algoritmo asigna las variables que son mejores en cada momento determinado y no globalmente.

Este método, aunque no obtenga un óptimo global es utilizado porque tiene una implementación fácil y es muy potente.

A continuación se presenta el pseudocódigo de un método Greedy genérico, siendo C el espacio en el que están comprendidas las soluciones y el conjunto de posibles soluciones $S \subseteq C$.

Heurística GREDDY

S:= Ø;

While(S no sea una solución y C ≠ Ø)

x = selección (C)

C = C – {x}

If (Su {x} es factible)

S = Su {x}

If (S es una solución)

Return S;

El elemento x depende de la función criterio elegida, la cual es dependiente de cada tipo de problema que se quiera tratar. Para elegir ese elemento hay que encontrar un índice que minimice o maximice el criterio.

Es posible que dentro de una matriz haya varios valores que optimicen el criterio y el método no discriminará ninguno. Por ese motivo es posible que un mismo método de soluciones diferentes.

Un ejemplo sencillo sería el siguiente: En la función f(x)=x-2 con una matriz de posibles valores de x :

5	45	51	22	1	24	3
1	3	2	1	52	80	0
34	52	80	24	73	45	2

Los valores situados en las coordenadas (2,6) y (3,2) harían que la función f alcanzase un valor máximo, por lo que en la iteración actual cualquiera de los dos sería un valor posible. Sin embargo, en la siguiente iteración se elegirá un nuevo valor, condicionado por la elección en la iteración anterior, es decir, si en esta iteración se ha elegido el valor situado en (2,6) en la siguiente iteración este valor no podrá ser elegido.

En la mayoría de los lenguajes de programación no es posible encontrar directamente el argumento máximo o mínimo de una matriz, por lo que en cada iteración es necesario recorrer un bucle como el siguiente, el código equivalente en Xpress-Mosel sería el siguiente:

```
cmin := infinito
forall (j in C) do
    If (criterio(j) < cmin) then
        cmin := criterio (j)
        jmin := j
    end-if
end-do
```

Aquí habremos obtenido jmin, que sería el índice de la variable o elemento que alcanza el mínimo, por lo tanto :

x (jmin) = selección (C)

3.2.2 Métodos greedy en Problema de Localización sin Capacidades (UFLP)

Para cada tipo de problema de optimización existe un método Greedy específico, acomodado a él. En el caso del problema de localización presentamos dos métodos, que son las heurísticas Greedy ADD y DROP.

Recordamos la situación de la curva de costes totales (ilustración 5):

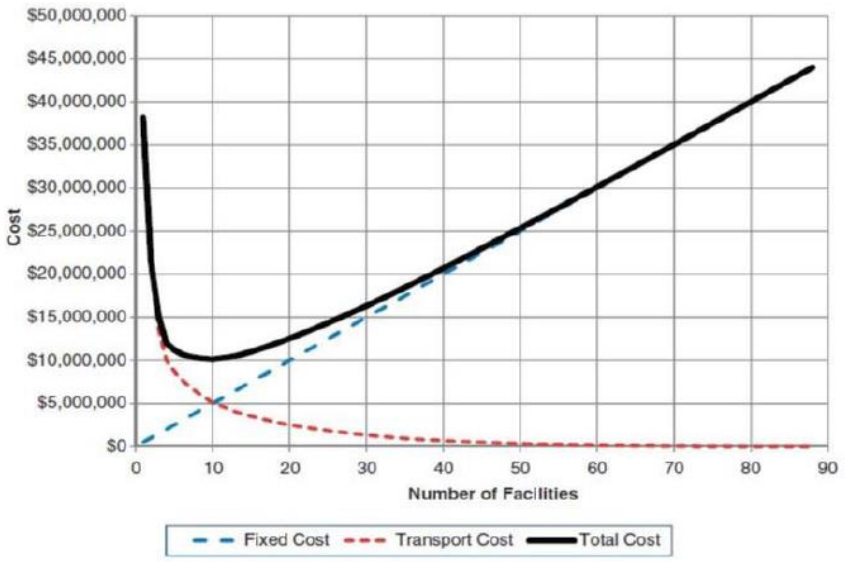

Ilustración 5 **Prototipo de costes de instalaciones, transporte y totales (Daskin, 1995)**

Empezando por la izquierda se situaría el algoritmo ADD, el cual empieza con cero instalaciones abiertas y por la derecha DROP, que empieza con todas las posibles instalaciones abiertas. Con un número mayor de instalaciones abiertas el coste variable es menor, sin embargo el coste fijo es mayor. Estas dos curvas se cruzan en un punto en el cual se obtiene el valor mínimo posible del coste total.

3.2.2.1 Heurística ADD

El algoritmo Greedy ADD para problemas de localización sin capacidades comienza teniendo todas las instalaciones cerradas, entonces la heurística va añadiendo a S instalaciones tales que $j \notin S$ cuyo valor incremental $\rho_j(S) = z(S \cup \{j\}) - z(S)$ sea el valor positivo y mayor posible hasta el momento en el que estas instalaciones dejen de existir, por lo que se acabaría con el conjunto S de instalaciones que tuviéramos abierto.

Haciendo referencia a la figura 8, que aparece más adelante, el algoritmo ADD empezará por el lado izquierdo de la curva de costes. De este modo irá añadiendo instalaciones a la solución de tal modo que el coste total vaya disminuyendo hasta el momento en el que añadir una nueva instalación no implique una reducción del coste total.

Al ser un algoritmo Greedy, en cada iteración se añadirá una solución que haga que se reduzca al máximo posible el coste total, manteniendo aquellas soluciones que previamente se hayan añadido.

Partiendo del trabajo realizado anteriormente por Rubén Sierra como TFG (*Problemas de Localización de Servicios Públicos y Privados. Métodos Heurísticos Basados en Relajación Lagrangiana*) he obtenido el diagrama de flujo para representar este método.

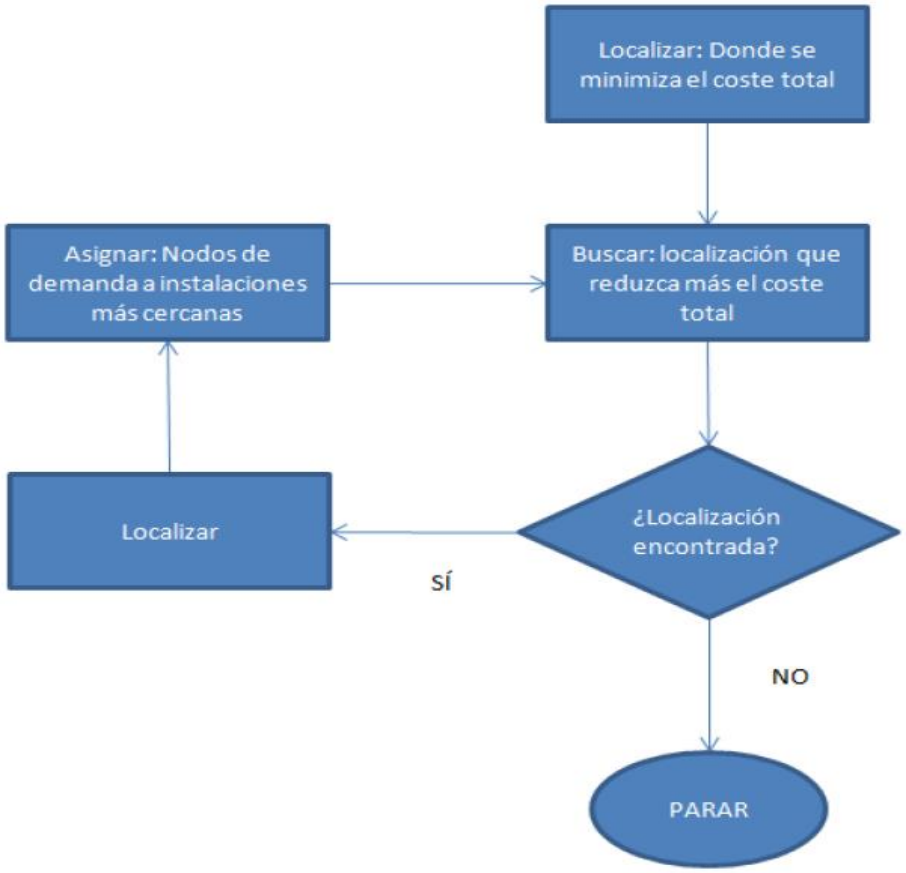

Ilustración 6 **Diagrama de flujo del algoritmo ADD para problemas de localización sin capacidades**

A continuación se expone el pseudocódigo de la heurística ADD, pues será la heurística trabajada que será llevada a cabo a la hora de obtener resultados numéricos mediante la programación en Xpress-Mosel, aunque puede ser implementada en cualquier otro lenguaje de programación.

Esta heurística consta de dos etapas:

- Etapa inicial:

Para cada $j = 1, \dots, n$ sea V_j el costo de abrir una única instalación en j, es decir $V_j = f_i + \sum_{i=1}^{m} d_i c_i$. Entonces si $V_k = mín_{j \in N} V_j$, se abre la instalación en k, y se toma: $S = \{k\}, z^h = V_k, y_i = k,\ i = 1, \dots, m$; siendo z^h el valor actual de la solución.

- Etapa general:

Dada la solución actual, con S, $y = (y_i)$, i = 1,...,m y z^h, se calcula para cada $j \notin S$, el criterio greedy, que es el valor objetivo si se añadiese j a la solución actual, es decir:

$$V_j = f_j + \sum_{k \in S} f_k + \sum_{i=d_i}^{m} d_i \, mín(c_{ij}, c_{iy_i})$$

Sea $k \in S$ con $V_k = mín_{k \in S} V_j$.

Entonces:

- Si $V_k > z^h$ el método termina con la solución actual, pues ha comenzado a incrementarse el costo total.

- Si $V_k < z^h$ se toma $S = S \cup \{k\}$, y se actualiza el array de asignaciones: si $c_{ik} < c_{iy_i}$la nueva asignación es $y_i = k$;en caso contrario, se mantiene la anterior asignación. El nuevo valor heurístico es $z^h = \sum_{j \in S} f_j + \sum_{i=1}^{m} d_i c_{i,y_i}$. Se vuelve a repetir la etapa general.

3.2.2.2 Heurística DROP

A diferencia del caso anterior, en el algoritmo DROP se comienza desde la derecha de la curva de costes (ilustración 5), lo que significa que se comienza con una instalación abierta en cada localización candidata, de tal modo que se busca una reducción máxima del coste total en cada iteración, en la que se elimina una localización hasta un punto en el que una reducción del número de localizaciones no implique una mejora del coste total.

Partiendo del trabajo realizado anteriormente por Rubén Sierra como TFG (*Problemas de Localización de Servicios Públicos y Privados. Métodos Heurísticos Basados en*

Relajación Lagrangiana) he obtenido el diagrama de flujo para representarlo de un modo gráfico.

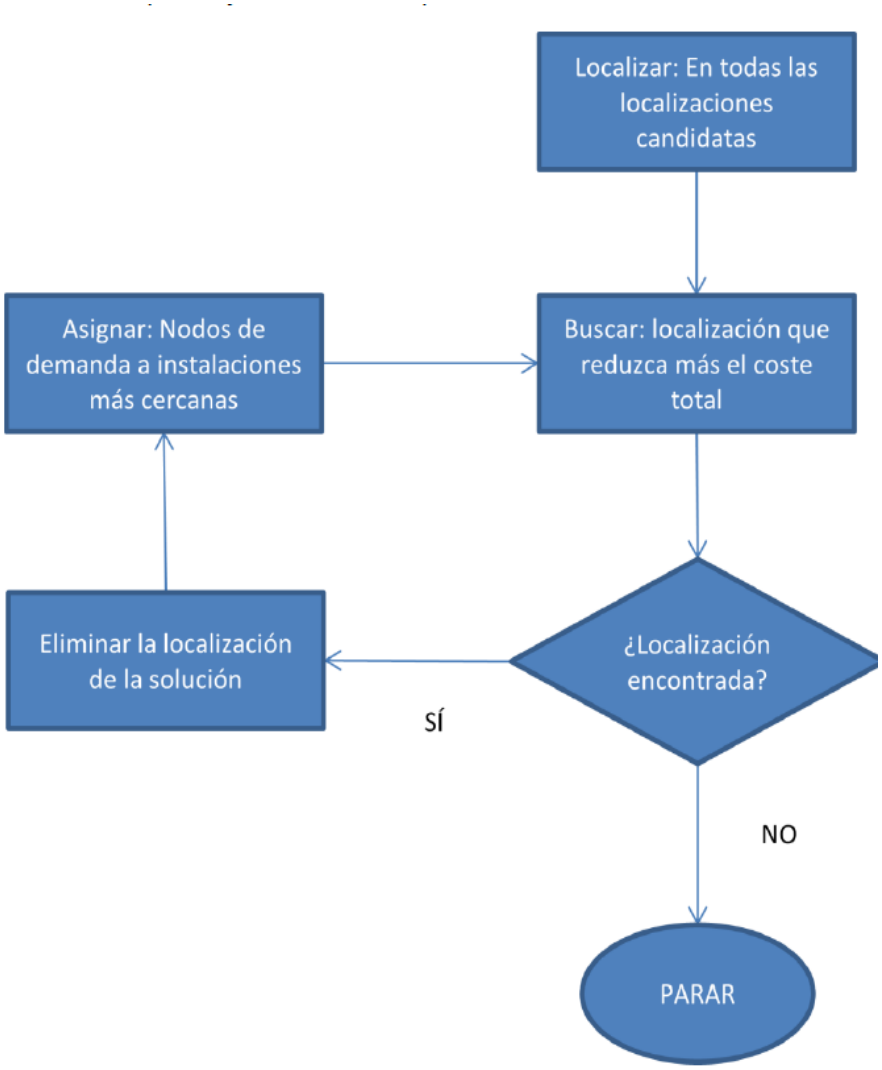

Ilustración 7 **Diagrama de flujo del algoritmo DROP para problema de localización sin capacidades**

Las soluciones de este trabajo serán obtenidas con el algortimo Greedy ADD debido a su mayor efectividad.

3.3 Búsqueda local.

3.3.2 Introducción

La búsqueda local es un método que parte de una solución dada (en este trabajo se partirá de la solución obtenida por el algoritmo de construcción Greedy), y tras una serie de iteraciones trata de mejorarla. Por ese motivo se dice que los métodos basados en búsqueda local son algoritmos de mejora de la solución.

Para poder realizar su propósito actúa sobre las vecindades de la solución (x), que es un espacio N en el que se encuentran asociadas un conjunto de soluciones $N\ (x) \subseteq X$, denominadas entorno de x y siendo X el conjunto de soluciones posibles para el problema. En otras palabras, el entorno de x son todas las posibilidades a considerar como soluciones en cada punto. Para moverse por ese espacio realiza un movimiento, que es una operación en la que partiendo de $x \in X$ se obtiene $x^* \in N(X)$, es decir, una nueva solución de su vecindad. Esta solución solo sustituirá a la solución actual en caso de que mejore la función objetivo.

Es un proceso iterativo que finaliza en el momento en el que tras una búsqueda no se encuentra ninguna solución mejor entre sus vecindades.

Para llevar estos métodos a cabo los pasos realizados son los siguientes:

1. Se escoge un $x \in X$ y se inicia el proceso.
2. Se busca un $x^* \in N(X)$ tal que f$(x^*) < f(x)$.
3. Si existe se sustituye $x \in X$ por $x^* \in X$.
4. Si no existe hemos alcanzado un óptimo local.

Los métodos para buscar la vecindad y seleccionar un vecino para reemplazar la solución se conocen como regla de pivoteo, que en general son:

Escalada

a. **Simple:** Se elige cualquier solución en cada iteración que mejore la solución actual.

b. **Máxima pendiente:** En vez de seleccionar el primer movimiento que produzca una mejora, se selecciona el mejor movimiento que suponga una mejora.

También son conocidas como (*fisrt-improvement rule)* y *(best-improvement rule)* respectivamente.

Esto hace que sea un método que no garantice un óptimo global, debido a que en el momento en el que alcanza un óptimo local no puede salir de él. Sin embargo es un método que proporciona soluciones aceptables rápidamente.

En la figura 8 se puede observar una simplificación de la diferencia entre un óptimo global y uno local. En el momento en el que alcanza una nueva solución $x * \in X^0$ el método queda atrapado en un óptimo local.

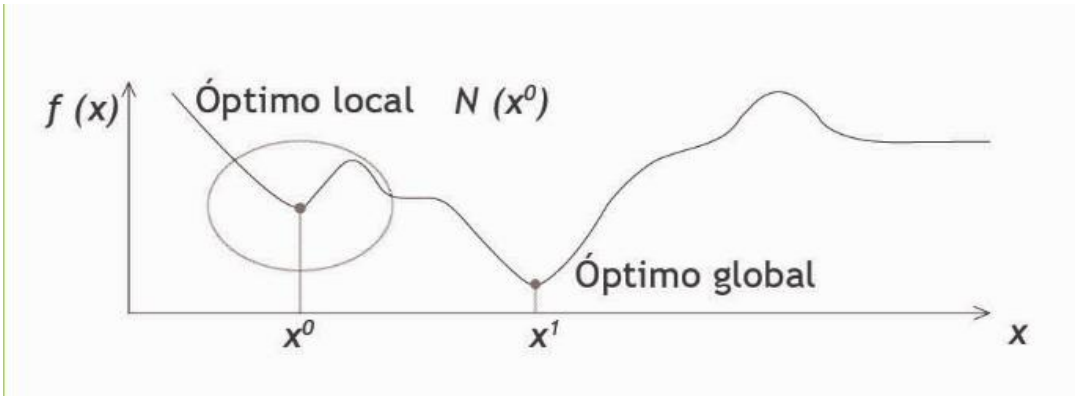

Ilustración 8 **Recuperado de: http://revistas.uaem.mx/images/Inventio%2023_imagen%20p26.jpg Óptimo local y global**

En este método siempre que sea dada una misma entrada la salida será idéntica (el mínimo local), lo que es considerado como un "pozo de atracción".

Así mismo, existen variantes de la búsqueda local con un solo entorno como la "*Fast Local Search*", la "Random Restart y Multistart Methods" y la "Guided Local Search" que pasamos a considerar:

- Fast Local Search:

A mayor número de vecinos a considerar el tiempo de búsqueda aumenta. Este caso consiste en ignorar vecinos que probablemente no mejoran la función objetivo. Para ello la vecindad se divide en sub-vecinos y a estos se les da un valor de activación, siendo solo explorados aquellos que tienen dicho valor.

- Random Restart y Multistart Methods:

  El proceso de búsqueda se reinicia cada vez partiendo de un punto de búsqueda aleatorio (de ahí su nombre "*random restart"*). Con esto se busca salir de mínimos locales por el hecho de volver a comenzar en una nueva zona.

- Guided Local Search:

  Es una alternativa de la búsqueda local en la que la función objetivo es aumentada con penalizaciones.

En este trabajo no profundizaremos más en los métodos de búsqueda local con un solo entorno. La efectividad de estos métodos depende de la estructura de su entorno, es decir, el diseño de su vecindad; ya que una u otra estructura proporcionará diferentes resultados. **Mladenović y Hansen (1997)**.

3.3.3 Búsqueda por Entornos Variables

Existe una variante de la búsqueda local conocida como Búsqueda por entornos variables, *Variable neighboord search (VNS).* En esta variante es posible definir varios entornos $N_1(x), \dots, N_p(x)$ y tiene dos características principales:

1. El mínimo local del entorno $N_1(x)$ no es el mismo que el de $N_p(x)$
2. El mínimo global es mínimo local para todas las posibles estructuras de entornos.

El método de operación de *VNS* está resumido en el siguiente pseudocódigo:

begin

solucion_inicial= x

k=1

for k=1 to k=p

if $F(x_k) < F(x)$

x_k=x

$k = 1$

else

k=k+1

end

Este método suele ser utilizado con $p = 2$ o $p = 3$ entornos.

Existen otras variantes de la VNS como son:

.

1. VNS Reducida:

2. VNS Básica:
3. VNS General
4. VNS Anidada

3.3.4 Algoritmo de mejora basado en búsqueda local

Al igual que otros algoritmos, como por ejemplo el Greedy, tienen variaciones dependiendo del problema que se quiera tratar, en el caso de la búsqueda local ocurre lo mismo.

En este trabajo se presentará una estructura de entornos compuesta por tres movimientos (Add, Swap y Drop) .

1. *Swap moves:*

Son movimientos de intercambio. En este tipo de movimientos el número de instalaciones que permanecen abiertas en la solución es constante, pues consiste en cerrar una instalación abierta y abrir una instalación cerrada en ese instante. El objetivo de este movimiento es encontrar una buena solución con un cierto número de instalaciones abiertas.

En cada iteración se busca el mejor movimiento de intercambio para realizar, de tal modo que el valor de la función objetivo sea el menor posible.

Al valor de la función objetivo habrá que sumarle el coste fijo de la apertura de la instalación j (f_j) y el coste variable de transporte (c_{ij}) y restarle el correspondiente a la instalación cerrada $i-1$, es decir:

$f_j - f_{j-1} + c_{ij}$ - $c_{i-1,j-1}$

Por lo tanto:

$$\sum_{j=1}^{n} x_j = cte$$

2. *Add moves:*

Este tipo de movimientos consisten en una adición, es decir:

$$x_j = 0 \rightarrow x_j = 1$$

De tal modo que el coste total ($\sum_{j=1}^{n} f_j x_j$) disminuya lo máximo posible.

3. Drop moves:

Este tipo de movimiento consiste en el cierre de una instalación ya abierta de tal modo que el coste total disminuya lo máximo posible con el cierre de dicha instalación, por lo tanto:

$$x_j = 1 \rightarrow x_j = 0$$

De tal modo que el coste total ($\sum_{j=1}^{n} f_j x_j$) disminuya lo máximo posible.

En el presente trabajo para obtener los resultados numéricos se ha obtenido la solución inicial mediante la heurística Greedy Add y posteriormente se ha realizado una mejora de la solución basada en estos tres movimientos Add, Drop y Swap. Alternándolos de tal modo que se consiga una solución mejor a la inicial y lo más cercana posible a la óptima.

4 Resultados en UFLP

4.1 Introducción

Para poder elaborar las tablas que serán presentadas a continuación se ha partido de dos fuentes:

1. Datos de OR LIBRARY mencionados en el punto 2.2.2 Librería de datos.
2. Datos euclídeos de elaboración propia generados conn Xpress-Mosel.

Estos datos han sido separados en seis conjuntos dependiendo de su tamaño y procedencia

Conjunto I

Datos euclídeos generados en Xpress con un tamaño de 60x60.

Conjunto II

Datos euclídeos generados en Xpress con un tamaño de 100x100.

Conjunto III

Datos obtenidos de OR-LIBRARY de tamaño 50x16, inicialmente con capacidades pero en nuestro caso serán leídos desde el fichero sin tener en cuenta las capacidades.

Conjunto IV

Datos obtenidos de OR-LIBRARY de tamaño 50x25, inicialmente con capacidades pero en nuestro caso serán leídos desde el fichero sin tener en cuenta las capacidades.

Conjunto V

Datos obtenidos de OR-LIBRARY de tamaño 50x50, inicialmente con capacidades pero en nuestro caso serán leídos desde el fichero sin tener en cuenta las capacidades.

Conjunto VI

Datos obtenidos de OR-LIBRARY de tamaño 1000x10, inicialmente con capacidades pero en nuestro caso serán leídos desde el fichero sin tener en cuenta las capacidades.

4.2 Resultados computacionales

Para la obtención de los resultados computacionales se ha utilizado el Solver de Optimización Xpress-Mosel [Véase (2.2.1)]. Cabe destacar que solo ha sido posible su implementación desde la versión profesional, instalada en un ordenador a disposición de los alumnos con licencia de la UVa, debido a la complejidad computacional de los problemas puesto que las versiones facilitadas para estudiantes no soportan las altas dimensiones de estos problemas.

En la primera columna de las tablas aparecen las situaciones consideradas y en las dos siguientes columnas se recogen las soluciones obtenidas mediante la relajación lineal [Véase (2.2.3)] con formulaciones fuerte y débil [Véase (2.1.2)] respectivamente.

La tercera columna recoge la solución óptima exacta obtenida con Xpress-Mosel.

Por otra parte se recoge la solución obtenida mediante la heurística de construcción Greddy ADD [Véase(3.2.2.1)], la cual es el punto de partida del algoritmo de mejora (VNS) cuya solución es expuesta en la última columna [Véase (3.3.4)].

Conjuntol	60x60(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución óptima	Greedy ADD	Mejora (VNS)
	e60_1	146,68	1909,92	2353,07	2401,72	2353,07
	e60_2	150,27	2496,4	2496,4	2522,51	2496,4
	e60_3	154,33	2539,1	2539,1	2686,16	2539,1
	e60_4	151,52	2422,06	2422,06	2539,19	2425,23
	e60_5	156,22	2611,38	2611,38	2679,28	2611,38

Tabla 1-Resultados numéricos del conjunto I

Conjunto II	100x100(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución óptima	Greedy ADD	Mejora (VNS)
	e100_1	150,34	3305,47	3305,47	3591,66	3323,01
	e100_2	151,98	3423,14	3423,14	3481,75	3423,14
	e100_3	154,59	3531,05	3531,05	3655,19	3531,05
	e100_4	150,48	3315,51	3315,51	3448,36	3315,51
	e100_5	152,13	3474,17	3474,17	3676,32	3484,07

Tabla 2-Resultados numéricos del conjunto II

Conjunto III	50x16(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución óptima	Greedy ADD	Mejora (VNS)
	cap71	844807,59	932615,75	932615,75	932615,75	932615,75
	cap72	849169,07	977799,4	977799,4	981538,85	977799,4
	cap73	853434,98	1010641,45	1010641,45	1012476,98	1012476,98
	cap74	859463,45	1034976,98	1034976,98	1034976,98	1034976,98

Tabla 3 Resultados numéricos del conjunto III

Conjunto IV	50x25(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución óptima	Greedy ADD	Mejora (VNS)
	cap101	659341,15	796648,44	796648,44	797508,72	797508,72
	cap102	664015,9	854704,2	854704,2	855971,75	854704,2
	cap103	668503,01	893782,11	893782,11	895027,19	895027,19
	cap104	674734,59	928941,75	928941,75	928941,75	928941,75

Tabla 4 Resultados numéricos del conjunto IV

Conjunto V	50x50(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución entera	Greedy ADD	Mejora (VNS)
	cap131	631421,45	793439,56	793439,56	794299,85	794299,85
	cap132	636321,45	851495,32	851495,32	851495,32	851495,32
	cap133	893076,71	893076,71	893076,71	894095,76	894095,76
	cap134	648426,3	928941,75	928941,75	928941,75	928941,75

Tabla 5 Resultados numéricos del conjunto V

Conjunto VI	1000x10(Resultados)					
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Solución óptima	Greedy ADD	Mejora (VNS)
	capa	4801483,12	17156454,48	17156454,48	17902353,24	17156454,48
	capc	3300869,23	11500104,96	11505594,33	11947717,76	11551802,86

Tabla 6 Resultados numéricos del conjunto VI

4.3 Tiempos computacionales

A continuación serán mostradas las tablas que recogen los tiempos computacionales que requieren la obtención de las soluciones:

	60x60(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
ConjuntoI	e60_1	0,281	0,14	0,01	0,296
	e60_2	0,265	0,141	0,015	0,171
	e60_3	0,218	0,141	0,01	0,422
	e60_4	0,281	0,141	0,01	0,265
	e60_5	0,296	0,141	0,01	0,343
	Promedio	0,2682	0,1408	0,011	0,2994

Tabla 7 Tiempo computacional relativo al conjunto I

	100x100(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
Conjunto II	e100_1	0,858	0,375	0,031	4,805
	e100_2	0,873	0,375	0,032	1,482
	e100_3	0,827	0,375	0,015	3,369
	e100_4	0,905	0,375	0,031	2,808
	e100_5	0,889	0,359	0,016	3,057
	Promedio	0,8704	0,3718	0,025	3,1042

Tabla 8 Tiempo computacional relativo al conjunto II

	50x16(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
Conjunto III	cap71	0,047	0,047	0,01	0,016
	cap72	0,047	0,031	0,01	0,016
	cap73	0,047	0,031	0,01	0,016
	cap74	0,046	0,032	0,015	0,016
	Promedio	0,04675	0,03525	0,01125	0,016

Tabla 9 Tiempo computacional relativo al conjunto III

Conjunto IV	50x25(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	cap101	0,046	0,063	0,01	0,016
	cap102	0,063	0,046	0,01	0,016
	cap103	0,078	0,062	0,01	0,016
	cap104	0,062	0,063	0,01	0,016
	Promedio	0,06225	0,0585	0,01	0,016

Tabla 10 Tiempo computacional relativo al conjunto IV

Conjunto V	50x50(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	cap131	0,11	0,109	0,01	0,093
	cap132	0,14	0,109	0,01	0,125
	cap133	0,156	0,11	0,01	0,046
	cap134	0,172	0,11	0,01	0,031
	Promedio	0,1445	0,1095	0,01	0,07375

Tabla 11 Tiempo computacional relativo al conjunto V

Conjunto VI	1000x10(Tiempo computacional)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	capa	31,48	4,68	0,062	4,165
	capc	31,3	7,19	0,109	23,26
	Promedio	31,39	5,935	0,0855	13,7125

Tabla 12 Tiempo computacional relativo al conjunto VI

4.4 Calidad de las soluciones

A continuación serán mostradas las tablas en las que figura un ratio que hemos considerado como porcentaje indicador de la calidad de la solución, definido para la relajación como:

$$\text{Calidad}=\frac{Solución\ óptima-Solución\ relajación}{Solución\ óptima}x100$$

Y para las heurísticas como:

$$\text{Calidad}=\frac{Solución\ heurística-Solución\ óptima}{Solución\ óptima}x100$$

Por lo tanto el resultado obtenido será el porcentaje en el que varía la solución obtenida mediante la heurística con la solución óptima siendo un 0% una coincidencia exacta.

	60x60(Calidad)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
Conjuntol	e60_1	93,77%	18,83%	2,07%	0,00%
	e60_2	93,98%	0,01%	1,05%	0,00%
	e60_3	93,92%	0,01%	5,79%	0,00%
	e60_4	93,74%	0,01%	4,84%	0,13%
	e60_5	94,02%	0,01%	2,60%	0,00%
	Promedio	94%	4%	3%	0%

Tabla 13 Calidad de las soluciones del conjunto I

	100x100(Calidad)				
	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
Conjunto II	e100_1	95,45%	0,01%	8,66%	0,53%
	e100_2	95,56%	0,01%	1,71%	0,00%
	e100_3	95,62%	0,01%	3,52%	0,00%
	e100_4	95,46%	0,01%	4,01%	0,00%
	e100_5	95,62%	0,01%	5,82%	0,28%
	Promedio	96%	0%	5%	0%

Tabla 14 Calidad de las soluciones del conjunto II

	50x16(Calidad)				
Conjunto III	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	cap71	9,42%	0,01%	0,00%	0,00%
	cap72	13,16%	0,01%	0,38%	0,00%
	cap73	15,56%	0,01%	0,18%	0,18%
	cap74	16,96%	0,01%	0,00%	0,00%
	Promedio	14%	0%	0%	0%

Tabla 15 Calidad de las soluciones del conjunto III

	50x25(Calidad)				
Conjunto IV	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	cap101	17,24%	0,01%	0,11%	0,11%
	cap102	22,31%	0,01%	0,15%	0,00%
	cap103	25,21%	0,01%	0,14%	0,14%
	cap104	27,37%	0,01%	0,00%	0,00%
	Promedio	23%	0%	0%	0%

Tabla 16 Calidad de las soluciones del conjunto IV

	50x50(Calidad)				
Conjunto V	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	cap131	20,42%	0,01%	0,11%	0,11%
	cap132	25,27%	0,01%	0,00%	0,00%
	cap133	0,00%	0,01%	0,11%	0,11%
	cap134	30,20%	0,01%	0,00%	0,00%
	Promedio	19%	0%	0%	0%

Tabla 17 Calidad de las soluciones del conjunto V

	1000x10(Calidad)				
Conjunto VI	Problema	Relajación lineal (formulación débil)	Relajación lineal (formulación fuerte)	Greedy ADD	Mejora (VNS)
	capa	72,01%	0,01%	4,35%	0,00%
	capc	71,31%	0,05%	3,84%	0,40%
	Promedio	72%	0%	4%	0%

Tabla 18 Calidad de las soluciones del conjunto VI

Promedio total	52,81%	0,64%	2,07%	0,09%

Tabla 19 Promedio calidad

4.5 Soluciones gráficas

En el siguiente apartado serán mostradas de forma gráfica las coordenadas en las que para obtener la solución óptima, de los conjuntos I y II, se deberían de abrir instalaciones y por otra parte, a qué puntos habría que transportar desde las instalaciones abiertas:

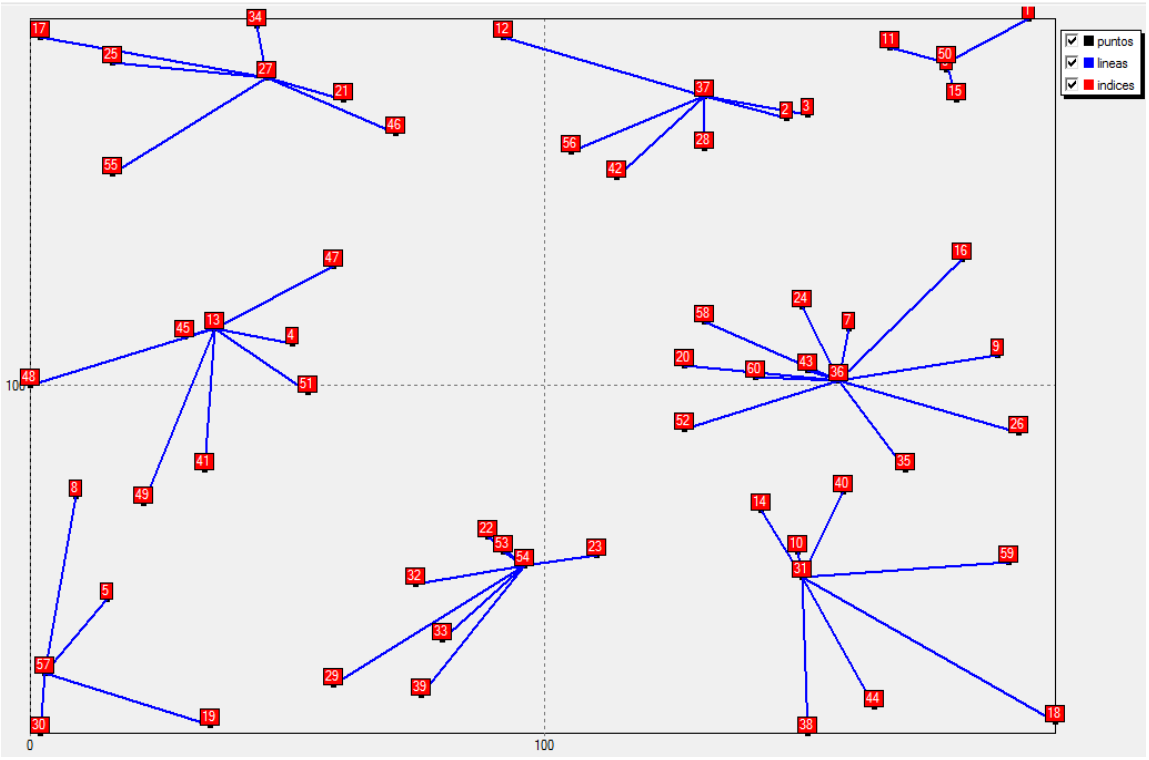

Ilustración 9 **Solución gráfica asociada a la situación de coordenadas e_60_1**

En este problema se han abierto ocho instalaciones en los puntos 13, 27, 31, 36, 37, 50, 54 y 57, realizando servicios de transporte a los puntos más cercanos a ellos , lo que produce que el coste sea el menor posible. Esto se repetirá en los siguientes ejemplos.

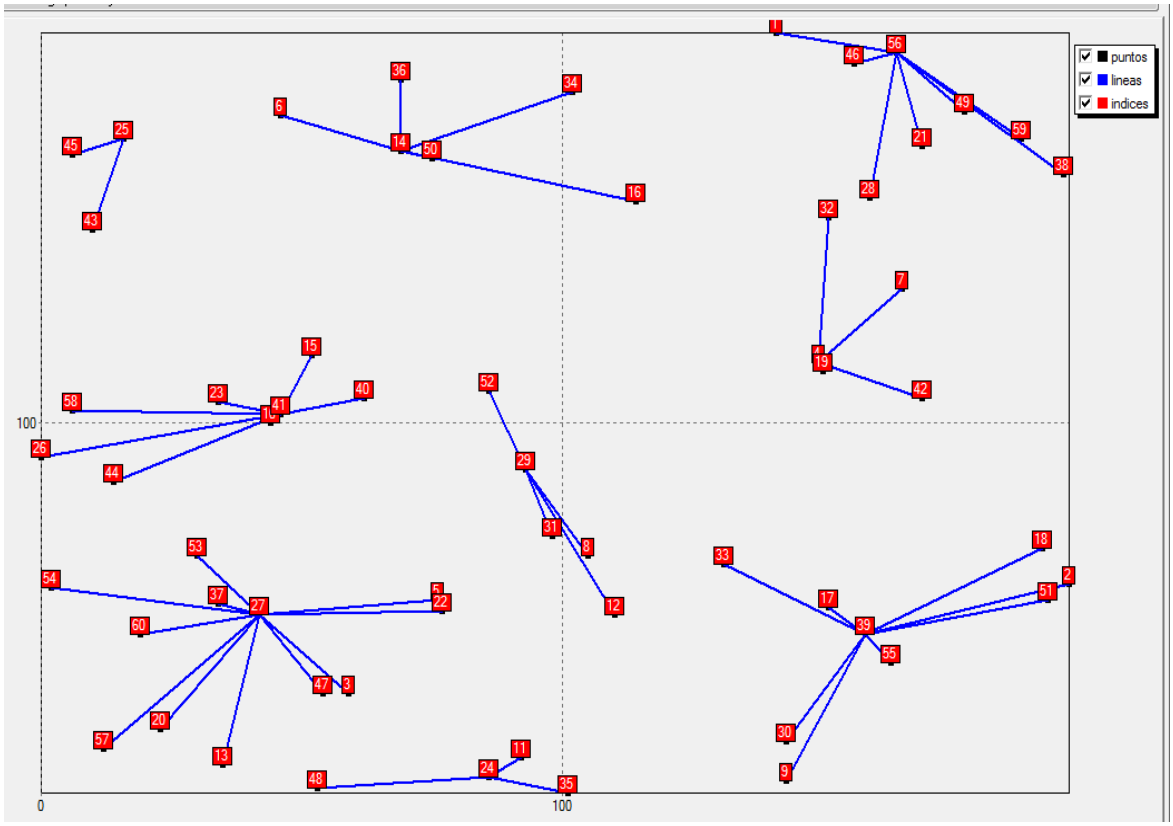

Ilustración 10 **Solución gráfica asociada a la situación de coordenadas e_60_2**

En este ejemplo se han abierto nueve instalaciones , que son la 4, 14, 24, 25, 27, 29, 39, 41 y 56 .

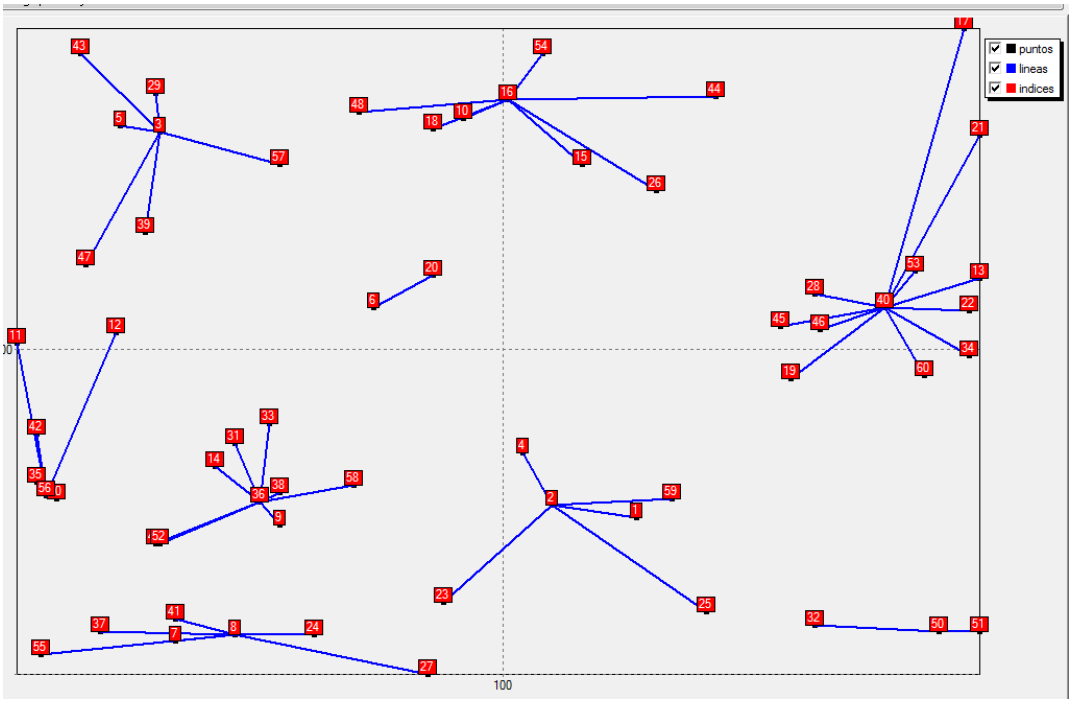

Ilustración 11 Solución gráfica asociada a la situación de coordenadas e_60_3

En el problema e_60_3 se han abierto nueve instalaciones que son los puntos 2, 3, 8, 16 , 20, 36, 40, 50 y 56.

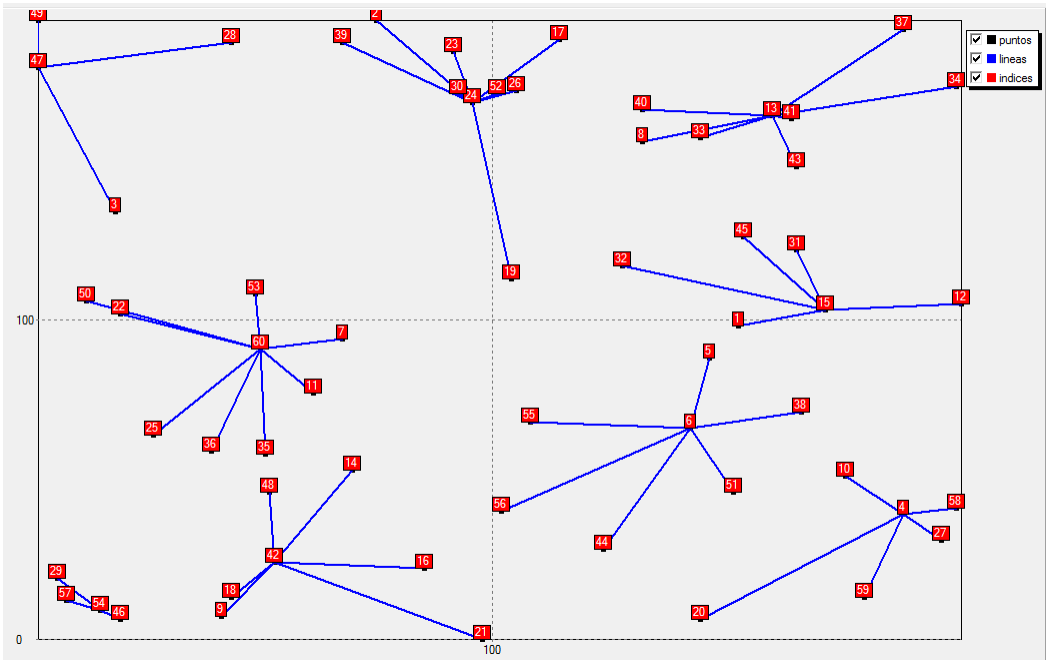

Ilustración 12 Solución gráfica asociada a la situación de coordenadas e_60_4

En el problemae_60_4 las aperturas son: 4, 6, 13, 15, 24, 42, 47, 54 y 60.

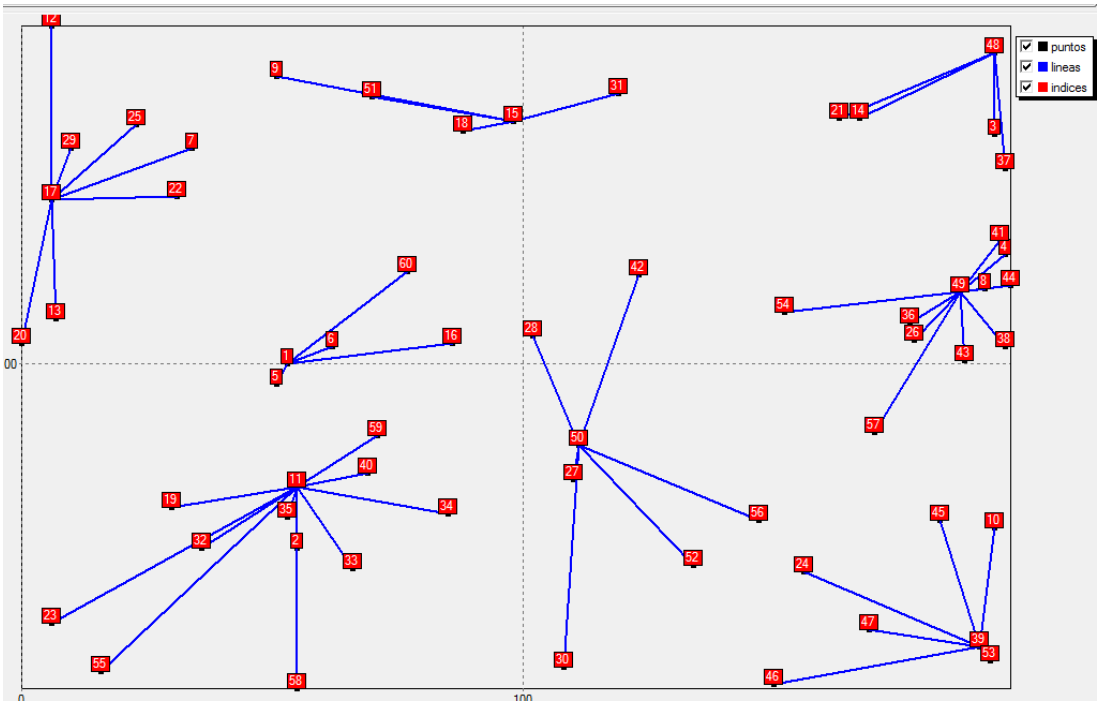

Ilustración 13 **Solución gráfica asociada a la situación de coordenadas e_60_5**

Los puntos de servicio abiertos son:

1, 11, 15, 17, 39, 48, 49 y 50.

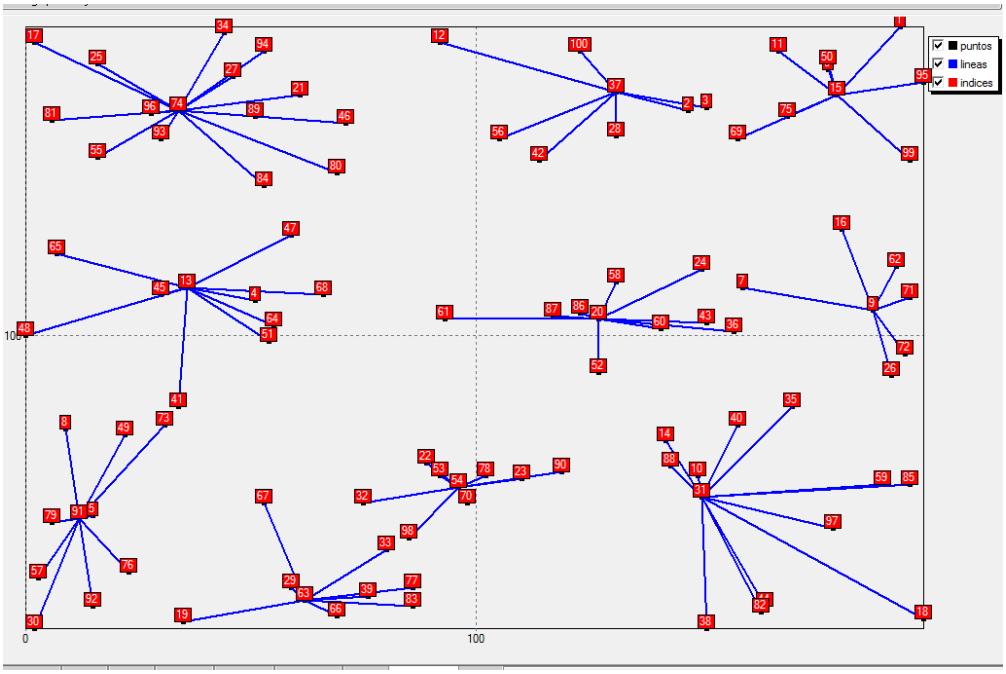

Ilustración 14 **Solución gráfica asociada a la situación de coordenadas e_100_1**

Puntos de servicio abiertos:

9, 13, 15, 20, 31, 37, 54, 63, 74 y 91 .

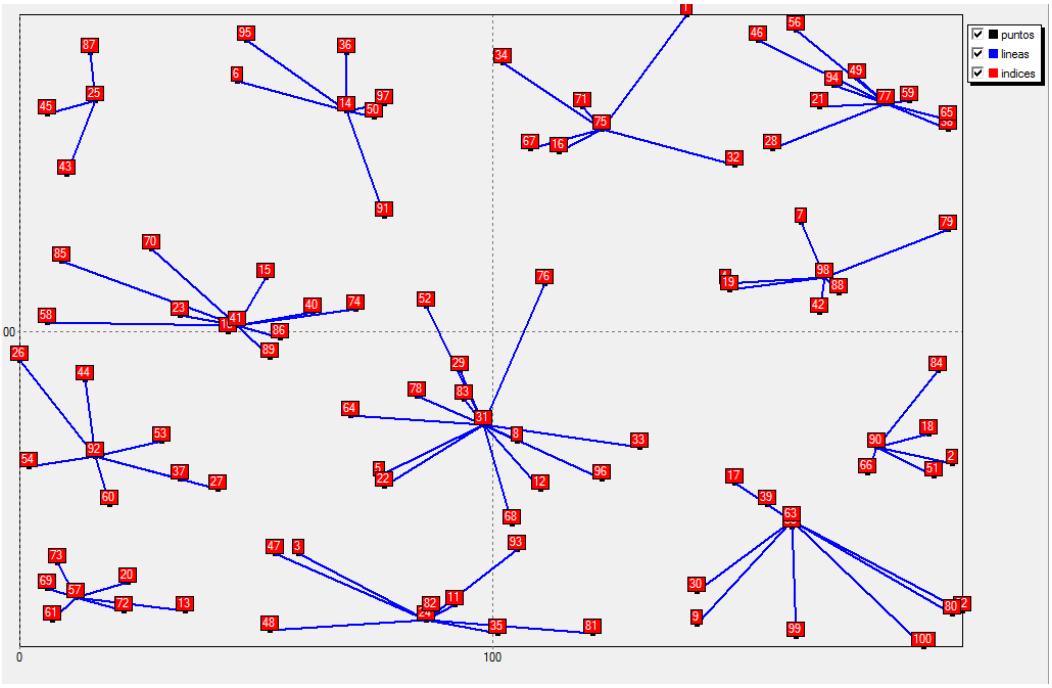

Ilustración 15 **Solución gráfica asociada a la situación de coordenadas e_100_2**

Puntos de servicio abiertos:

14, 24, 25, 31, 41, 57, 63, 75, 77, 90, 92 y 98.

.

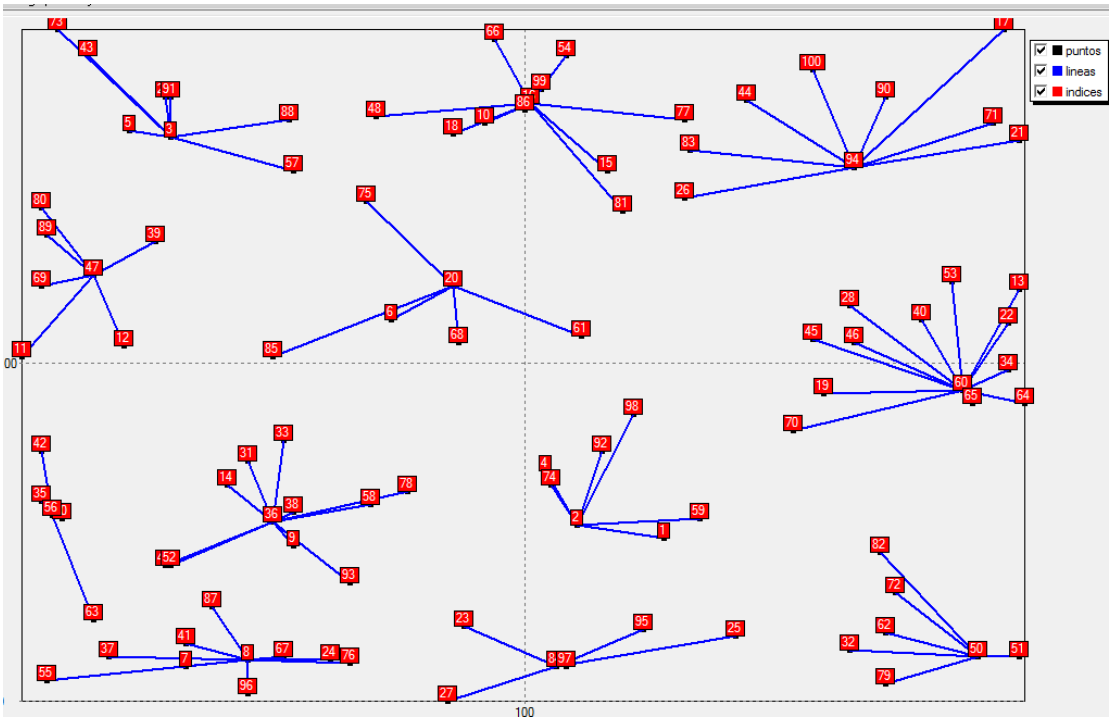

Ilustración 16 **Solución gráfica asociada a la situación de coordenadas e_100_3**

Puntos de servicio abiertos:

2, 3, 8, 16, 20, 36, 47, 50, 56, 60, 84 y 94.

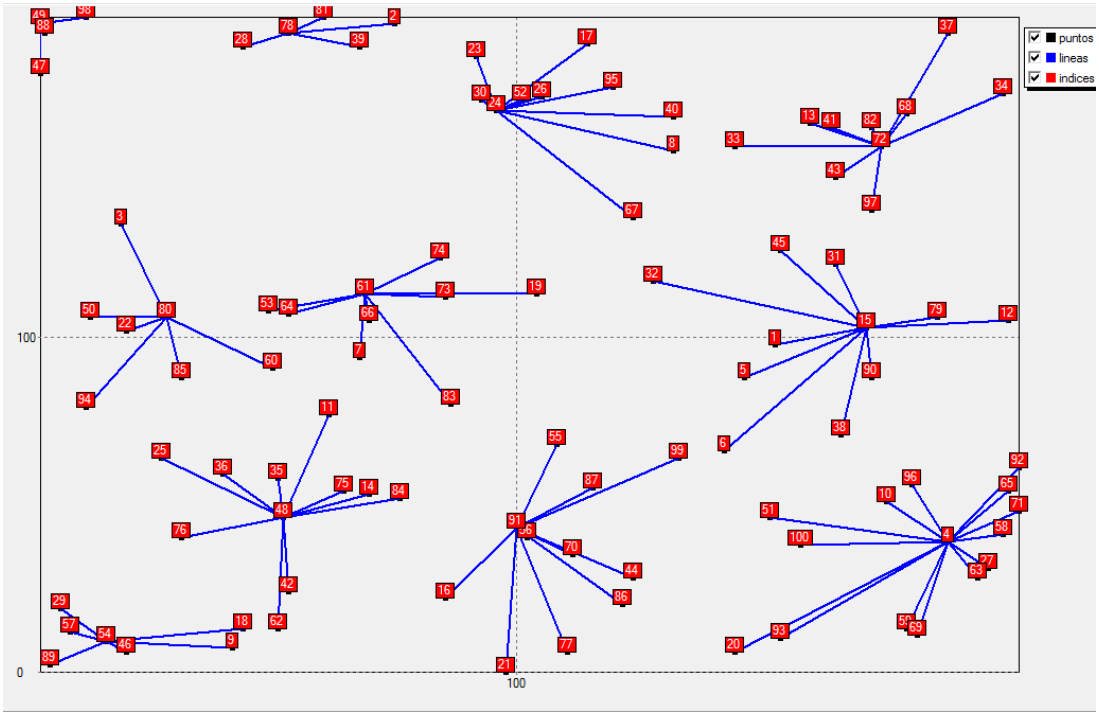

Ilustración 17 **Solución gráfica asociada a la situación de coordenadas e_100_4**

Puntos de servicio abiertos:

4, 15, 24, 48, 49, 54, 61, 72, 78, 80 y 91.

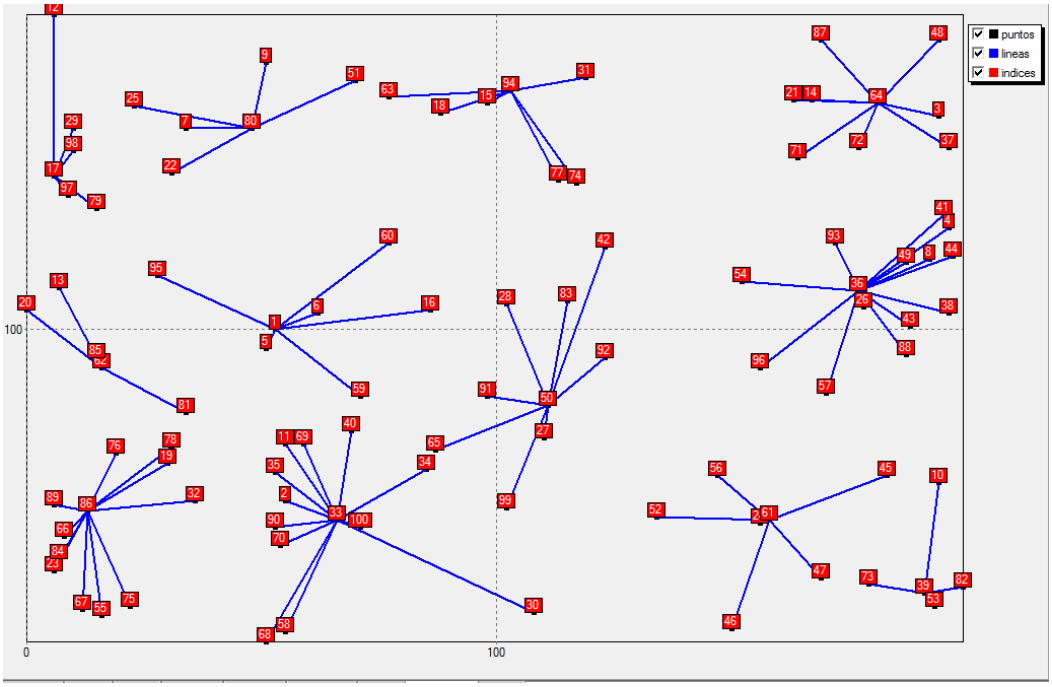

Ilustración 18 **Solución gráfica asociada a la situación de coordenadas e_100_5**

Puntos de servicio abiertos:

1, 17, 33, 36, 39, 50, 61, 62, 64, 80, 86 y 94.

4.6 Conclusiones

La relajación lineal obtenida con la formulación fuerte, considerada como una cota inferior de la solución, en la mayor parte de los conjuntos de problemas considerados llega directamente a la solución óptima en un tiempo prácticamente inmediato. A diferencia de la relajación lineal obtenida con la formulación débil que es una cota inferior mucho más alejada de la solución óptima que con la formulación fuerte. En los casos de tamaño más grande se observa que el tiempo computacional que tarda en resolver el óptimo a partir de la relajación lineal es mayor con la formulación débil que con la formulación fuerte.

En las tablas de las secciones 4.2, 4.3 y 4.4 se mostraba la media muestral (promedio de los datos) como descripción numérica de las variables cuantitativas a considerar X= Tiempo computacional (en segundos) e Y= Calidad de la solución (en porcentaje). La importancia de esta medida de posición o localización es que es la zona o punto central en torno al cual se aglutinan los valores de la variable.

El interés en este estudio se centra en comparar los métodos heurísticos propuestos en cuanto a tiempo computacional requerido y calidad de la solución proporcionada. En cuanto a la estructura de entornos propuestos en este trabajo se destaca en general la mejora el movimiento swap, obteniendo una solución ligeramente por encima, a veces el movimiento add y podría ser en algún caso el movimiento drop, pero en menor proporción de los tres.

Para el análisis de los resultados obtenidos se ha manejado Statgraphics, introduciendo los datos de los estadísticos obtenidos para cada conjunto de situaciones en dos columnas (Tiempo y Calidad) con variables de tipo numérico y una tercera columna (Método) de tipo carácter que distingue las dos heurísticas estudiadas en este trabajo.

Se representa gráficamente la variable cuantitativa bidimensional (X,Y) con un gráfico de burbujas para cada conjunto de situaciones, representando en color rojo el método Greedy Add y en color azul la mejora VNS, indicando el tamaño de la burbuja el conjunto de situaciones considerado, desde el conjunto I que equivale al tamaño pequeño hasta el conjunto VI que equivale al tamaño grande.

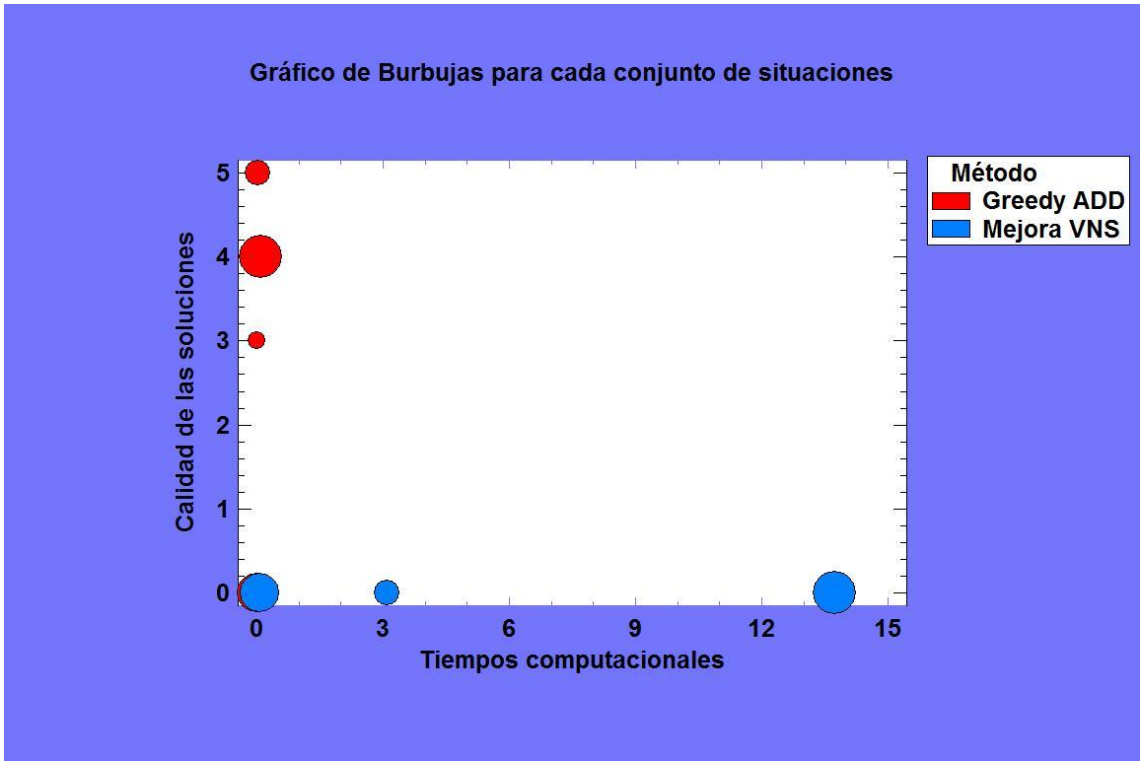

Ilustración 19 **Gráfico de Burbujas para cada conjunto de situaciones**

Se demuestra que en cada conjunto de situaciones el método del descenso con búsqueda local (Mejora VNS) proporciona mejor solución en un tiempo razonable. A la vista del gráfico (teniendo en cuenta el tamaño de la burbuja), en concreto para el conjunto de situaciones II y VI el tiempo computacional requerido para obtener las soluciones con el método de mejora es superior al método Greedy. En estos casos, donde los tamaños considerados son algo elevados, sería necesario valorar la calidad de la solución que es deseable que el valor esté lo más cercano posible a cero, puesto que se considera el ratio indicado anteriormente en porcentaje respecto a la solución óptima, de hecho se consigue el objetivo que se perseguía: mejorar la solución. Sin embargo para muchos casos considerados en el estudio, se comprueba que el método Greedy es suficientemente eficiente, puesto que se alcanza directamente la solución óptima y entonces la heurística ya no mejora. Teniendo en cuenta la rapidez en la obtención de las soluciones, se comprueba la validez de estas heurísticas en la resolución de problemas de localización con costos fijos y sin capacidades, considerados en este trabajo.

5 Líneas Futuras

5.1 Futuras extensiones

Existen otros métodos heurísticos de mejora basados en búsqueda local, que ayudan a salir de óptimos locales mediante su código para obtener una solución bastante más aceptable y próxima a la solución óptima. En estos algoritmos, como en el expuesto a lo largo del desarrollo del trabajo, un valor obtenido previamente como solución al problema con objetivo de mejorarlo de tal modo que se obtenga una solución más óptima.

Existen varios tipos de mejoras, de las cuales dos usan memoria, la Búsqueda Tabú (TS) y la Búsqueda Local Completa con Memoria (CLM). En este texto se tratará la Búsqueda Local Completa con Memoria y se definirá la Búsqueda Tabú para una posible extensión futura.

5.2 Búsqueda tabú

La Búsqueda Tabú Es un método metaheurístico comúnmente utilizado para resolver problemas de optimización combinatoria que combina la búsqueda local con una heurística que evita quedar atrapado en óptimos locales de tal modo que no entre en un ciclo **(Glover, 1986) .**

5.2.1 Espacio de soluciones

En primer lugar se cuenta con un espacio que contiene a todos los candidatos que pueden formar parte de la solución cuya notación será : J = {j_1, j_2, …, j_n}, que como se escribió en la definición del problema de localización, serán todos los lugares posibles en los cuales se puede situar una instalación.

Cada uno de estos puntos tomará un valor $x_j \in \{0,1\}$ que tomará el valor 1 en caso de que sea la solución explorada que devuelve un valor más óptimo en la función objetivo F y o en el caso de que esto no se cumpla. En caso de haber dos J que reporten el mismo valor cualquiera de los dos será el seleccionado.

5.2.2 Movimientos realizados

En cada iteración que se encuentra una solución mejor a la actual, esta pasará a ser la nueva solución del problema y la solución actual pasará a formar parte de una lista tabú. De una solución a otra se puede llegar únicamente mediante un movimiento, ya que se exploran los vecinos de dicha solución.

5.2.3 Memoria

Para llevar a cabo su propósito, utiliza dos memorias, de las cuales depende un cambio continuo que se realiza en la vecindad de la solución que haya en el momento actual. Gracias a esto, la búsqueda puede salir de los óptimos locales.

Esta memoria se divide en dos partes:

- Una memoria de búsquedas recientes o a corto plazo.
- Una memoria basada en la frecuencia.

La memoria de corto plazo guarda los últimos candidatos durante un tiempo fijo o variable conocido como "*tabu tenure*" previamente determinado. El hecho de que aparezca en esta memoria desalienta al algoritmo de incorporar este candidato en la iteración actual. Es conocida por el nombre de "lista tabú". Los candidatos que aparezcan en ella no podrán formar parte de las iteraciones siguientes salvo que cumplan un criterio de aspiración que será explicado a continuación.

Por otra parte, la memoria basada en frecuencia es una memoria de largo plazo que desalienta al algoritmo de realizar movimientos que hayan sido realizados con una frecuencia mayor. En esta memoria aparece el número de veces que una posible solución ha sido tenida en cuenta en el total de movimientos realizados. Por lo tanto aquellos valores que tengan un valor más alto serán aquellos que hayan sido tenidos en cuenta más veces y por lo tanto serán penalizados. Esto desembocará en una menor preferencia de estos valores a la hora de realizar una nueva iteración.

5.2.4 Criterio de aspiración

El concepto de criterio de aspiración es utilizado en Búsqueda Tabú para aquellos candidatos que mejoran la solución de tal modo que su probabilidad de ser los óptimos sea elevada, de este modo no serán olvidados e ignorados en una "lista tabú". Una iteración cumple este criterio en caso de que el valor de la función objetivo obtenido sea el más óptimo que se haya encontrado hasta ahora, por ese motivo es importante que no caiga en la lista tabú.

El hecho de guardar estos valores para volver a utilizarlos en caso de que no se encuentre una solución mejor, permite que se pueda salir de los óptimos locales, ya que, permite libertad de seguir explorando en el espacio de soluciones aunque se obtengan dos soluciones vecinas que vayan empeorando el resultado de la función.

A continuación se planteará un pseudocódigo de la heurística. Cuya explicación será expuesta posteriormente.

5.2.5 Pseudocódigo Búsqueda Tabú

Entrada: I, J, F, C y una solución inicial S_o

Salida: Una solución al problema.

Código:

Begin (1)

Mejor_solución:=S_o; (2)

for *i* =1 to *n* (3)

$freq_i$;=0; (4)

for iteración:=1 to maxiter do(5)

begin (6)

disminuir la *"tabu tenure"* de todos los elementos de la lista tabú en 1; (7)

eliminar todas las soluciones de la lista tabú cuya *"tabu tenure"* sea -1; (8)

S_{nt}:=*arg min*{*F(S)* : un movimiento de S_o a *S* no entra en lista tabú}; (9)

S_t:=*arg min*{$F_p(S,S_o)$: un movimiento de S a S_o entra en lista tabú}; (10)

if $F_p(S_t,S_o)<F(S_{nt})$ (11)

begin (12)

if $F(S_t) < F$(mejor_solución) (13)

begin (14)

mejor_solución:= S_t; (15)

for i ϵ ($S_t \setminus S_o$) ∪ ($S_o \setminus S_t$) do (16)

begin (17)

$freq_i$:= $freq_i$ +1; (18)

tabu tenure para solución i := número aleatorio entre 1 y *maxtabu*; (19)

end; (20)

$S_o = S_t$; (21)

end; (22)

else (23)

begin (24)

if $F(S_{nt}) < F$(mejor_solución); (25)

mejor_solucion=S_{nt}; (26)

for i ϵ ($S_{nt} \setminus S_o$)∪ ($S_o \setminus S_{nt}$) do (27)

begin (28)

$freq_i$:= $freq_i$ + 1; (29)

tabu tenure de solución i := número aleatorio entre 1 y *maxtabu* ; (30)

end; (31)

S_o := S_{nt}; (32)

end; (33)

end; (34)

end; (35)

return mejor_solución; (36)

end. (37)

En primer lugar tenemos los datos de entrada I, J, F, C y una solución inicial S_o, siendo I = {i_1, i_2, …, i_m} los puntos de demanda de un servicio proporcionado por una instalación, que se puede situar en una serie de puntos J = {j_1, j_2, …, j_n}. También tenemos la matriz de costos variables C, que depende de la distancia entre el punto de demanda y la instalación que satisface dicha demanda y por último la función objetivo a minimizar F, que en el caso del problema de instalación será el coste causado por la solución que obtengamos.

En segundo lugar tenemos el código, cuya finalidad es obtener una solución al problema que definamos.

Se empieza dando un valor a la variable mejor_solución que es en una primera instancia el valor de S_o (2).

En (3) abrimos un bucle for con una serie de n iteraciones que es igual al número de puntos en los que se puede situar la instalación y por lo tanto igual al número de soluciones posibles en un punto dado. Dentro de este bucle:

Damos a todas las soluciones el valor cero dentro de la memoria de frecuencia ya que aún no hemos realizado ninguna iteración (4) y empezamos un nuevo bucle cuyo número de iteraciones lo elegiremos dando un valor a la variable *maxiter*, variable que indica el número máximo de iteraciones que se van a realizar.

En este bucle: Se disminuye la "tabu tenure" de todos los elementos ya que esta depende del número de iteraciones (7) y aquellas que hayan llegado a -1 (8) serán eliminados de la lista tabú, ya que ya han estado en ella el número de iteraciones definidas en la línea (19).

S_{nt} obtiene el valor de S que hace que la función *F(S)* tenga el mínimo valor posible y en caso de sustituir la solución actual, esta entrará a formar parte de la lista tabú.

En el caso de que una solución en la lista tabú (S_t) o la solución S_0 mejoren el valor obtenido por la solución no tabú se realiza lo siguiente: (11)

Si la iteración actual da un valor mejor que el obtenido en la anterior iteración realizada (la solución actual) la variable mejor_solución pasará a valer el valor del candidato que se está explorando actualmente (15).

Posteriormente para cada candidato que se explora su frecuencia aumentará en un punto en la memoria de frecuencia (18) y su duración en la "tabu tenure" será obtenida en (19) mediante un número aleatorio. Al final del bucle S_0 pasará a tener el valor de S_t.

En caso contrario a (13) , si el valor de la función objetivo con una solución no tabú es menor que el actual, la variable mejor solución pasará a tener el valor de la solución no tabú(26) y se abrirá un bucle cuyo número de iteraciones sea igual a (27) en el que se empezará aumentando en uno el valor de la memoria de frecuencia correspondiente y dando un valor en la "tabu tenure".

Este bucle llegará a su fin y se le asignará a S_0 el valor de S_{nt}.

Por último se devolverá el valor de la variable mejor_solución (36) en la que han quedado guardados los valores de la solución a lo largo del programa.

5.3 Búsqueda Local Completa con Memoria

5.3.1 Origen

Este algoritmo almacena datos sobre el espacio de soluciones, al igual que la Búsqueda Tabú. Este almacenamiento cumple la función de guardar diferentes soluciones encontradas durante la ejecución de la heurística, teniendo un uso completamente distinto al de la Búsqueda Tabú.

Se ha partido de Ghosh, D., & Sierksma, G. (2002). Complete local search with memory. *Journal of Heuristics*, *8*(6), 571-584) para obtener el pseudocódigo con alguna pequeña modificación.

5.3.2 Memoria

Es un método relacionado con los algoritmos de búsqueda gráfica, por lo que tiene en cuenta que para llegar de la solución inicial a la final hay varios caminos posibles. Esto ocurre gracias a la estructura de su memoria.

En la memoria se almacenan las soluciones que ha generado la heurística. De este modo se asegura de no perder tiempo buscando más de una vez entre sus vecindades y en caso de no encontrar una solución mejor, recuperará una solución explorada anteriormente. Su tamaño dependerá del número de soluciones que sean necesarias guardar.

La memoria está dividida en tres partes. Cada parte alberga un conjunto diferente de soluciones y está denominada de forma diferente:

- LIVE: Almacena las soluciones que serán exploradas en un futuro por la heurística
- DEAD: Almacena las soluciones que formaron parte de LIVE pero que ya fueron exploradas. Esto asegura que una solución no es explorada dos veces.
- NEWGEN: Es un almacenamiento temporal, en el que se guardarán las soluciones dadas por la heurística en la iteración actual.

5.3.3 Funcionamiento de la memoria

Al igual que en las demás heurísticas mencionadas, se empieza con una solución dada, a partir de la cual se llegará a la solución final.

Esta solución será almacenada en la memoria LIVE, estando la memoria DEAD y NEWGEN en un inicio vacías al no haberse realizado ninguna iteración aún.

Después se realizan las iteraciones que serán posteriormente descritas hasta que se alcance una condición previamente definida, con el objetivo de no agotar la memoria del programa.

Al realizarse la iteración se busca en la vecindad de la solución y todos aquellos vecinos que proporcionen un valor de la función objetivo mejor que un valor límite "r" serán generadas por el algoritmo, es decir, serán considerados por el algoritmo para una futura exploración.

Posteriormente se comprueba a que conjunto pertenecen esos vecinos, si a LIVE, DEAD o NEWGEN. Si es miembro de LIVE significa que fue generado anteriormente y que aún no ha sido explorado. Si es de DEAD, significa que ya fue una solución generada y que ya fue explorada, por lo que no volverá a explorarse y si es de NEWGEN es que acaba de ser una solución generada en esta iteración, es decir, forma parte de la vecindad de la solución actual y cumple la condición de mejorar el valor límite "r".

Una vez comprobado el conjunto al que pertenecen estos vecinos se puede dar el caso de que no pertenezca a ninguno aún. En caso de no formar parte de ninguno de los tres conjuntos pasará a formar parte de NEWGEN y será un candidato a la solución que será explorado en la siguiente iteración, pero hasta no finalizar esta iteración y haber explorado todo LIVE no pasará de NEWGEN a LIVE.

Este método llega a su fin en el momento en que no existe ningún dato en LIVE, pero también puede darse el caso de que se dé la condición de que la capacidad de almacenamiento de DEAD y LIVE haya llegado a su límite. En este caso se procederá a realizar una operación de pos procesado en basada en una búsqueda local general en LIVE, enviando todas las soluciones a DEAD y de entre ellas se recuperará la mejor solución. También puede programarse de tal modo que llegue a su fin en caso de que se hayan realizado un número x de iteraciones o en caso de que tras realizar varias iteraciones la solución no se haya mejorado. Estas dos últimas condiciones son más normales en casos de mejora de búsqueda local.

A continuación se presenta un diagrama de flujo que resume el método:

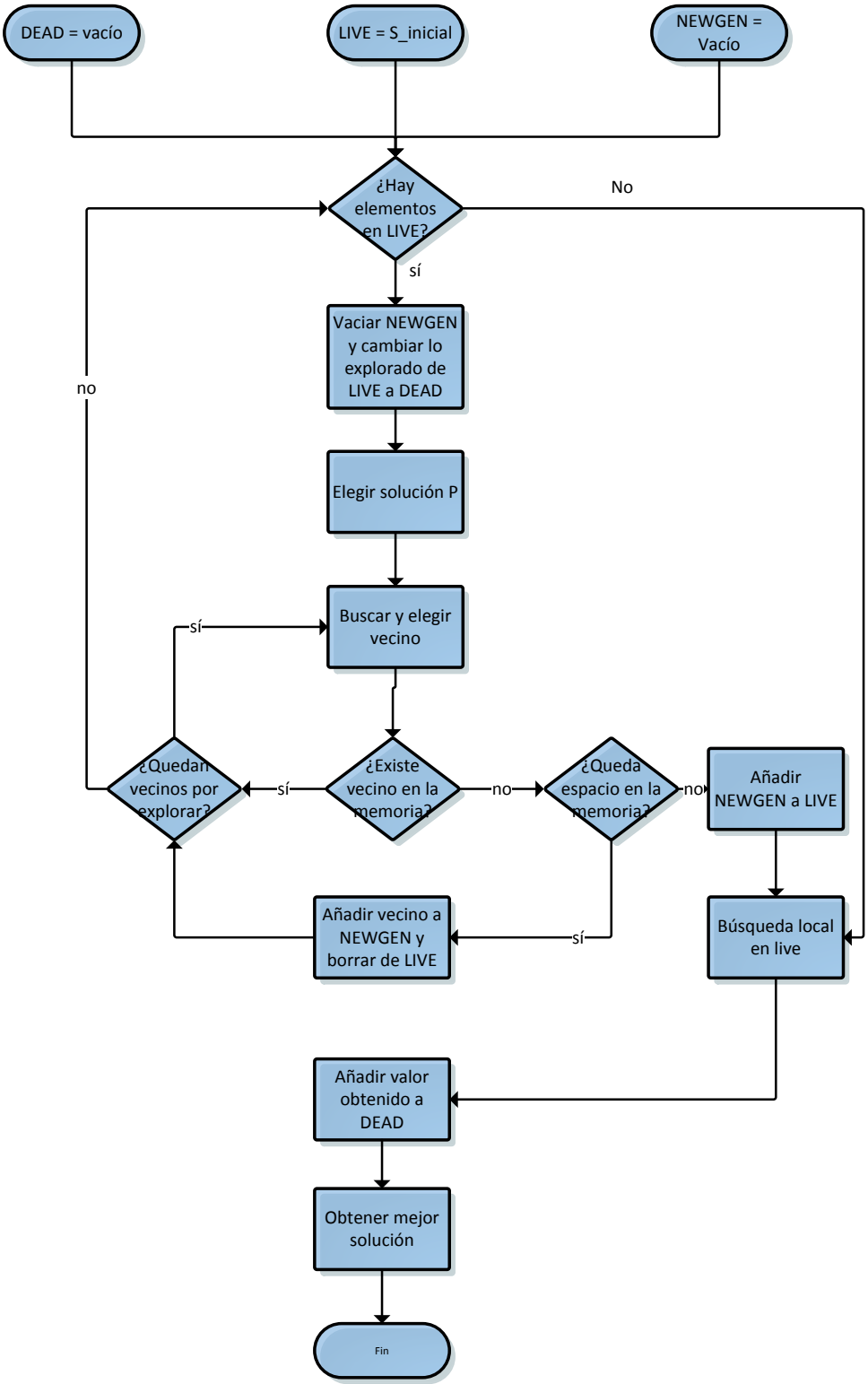

Ilustración 20 Diagrama de flujo de CLM

5.3.4 Pseudocódigo CLM

Entrada: I, J, F, C y una solución inicial S_o.

Salida: Una solución al problema.

Código:

Begin(1)

mejor_solución:= S_o;(2)

LIVE:={S_o};(3)

DEAD:= Ø;(4)

mientras LIVE ≠ Ø $hacer$(5)

begin (6)

NEWGEN = Ø;(7)

elegir una solución P de LIVE;(8)

para cada vecino P_n de P tal que F(P_n)<=F(P) hacer(9)

begin (10)

if P_n no está en LIVE,DEAD o NEWGEN (11)

begin (12)

if |LIVE| + |DEAD| > maxmemoria (13)

LIVE:= LIVE ∪ NEWGEN; (14)

ir a **posprocesado**; (15)

if |LIVE| + |DEAD| < maxmemoria (16)

NEWGEN := P_n; (17)

end; (18)

end; (19)

LIVE:= LIVE \ {P} ∪ NEWGEN; (20) "a live se le quita lo explorado y se le añade lo nuevo"

DEAD:= DEAD ∪ {P}; (21)

End; (22)

posprocesado: para cada solución s en LIVE; (23)

begin (24)

hacer búsqueda local en la solución de menor coste P en LIVE para obtener una solución óptima localmente P_{n1}; (25)

DEAD= DEAD ∪ P_{n1}; (26)

eliminar todas las soluciones viiftadas en el paso anterior de LIVE; (27)

end; (28)

obtener mejor solución de DEAD, que será llamada P_{lo}; (29)

if $F(P_{lo})$< F(mejor_solución) (30)

mejor_solución:= P_{lo}; (31)

return mejor_solución; (32)

end; (33)

En primer lugar tenemos los datos de entrada I, J, F, C y una solución inicial S_o, siendo I = {i_1, i_2, …, i_m} los puntos de demanda de un servicio proporcionado por una instalación, que se puede situar en una serie de puntos J = {j_1, j_2, …, j_n}. También tenemos la matriz de costos variables C, que depende de la distancia entre el punto de demanda y la instalación que satisface dicha demanda y por último la función objetivo a minimizar F, que en el caso del problema de instalación será el coste causado por la solución que obtengamos.

En segundo lugar tenemos el código, cuya finalidad es obtener una solución al problema que definamos.

Se empieza dando un valor a la variable *mejor_solución* que es en una primera instancia el valor de S_o (2).

Posteriormente añadimos esa solución a LIVE (3) para ser explorada y damos valor o a DEAD (4). Luego añadimos una condición de parada, que en este caso será que LIVE siga estando ocupada, ya que en caso de que no haya nada almacenado en esta memoria significará que se han explorado todas las posibilidades. También es posible añadir otros criterios de parada como un número máximo de iteraciones, por ejemplo.

En caso de que esta condición se cumpla y pueda dar lugar el inicio de la iteración (6) en primer lugar se borrará NEWGEN ya que es una memoria temporal y en la iteración anterior sus datos se han pasado a LIVE (20).

En (8) se elige una solución de P, pero para ello hay que decidir el criterio en el que este método se basará para su elección, tres de ellos podrían ser, la elección de soluciones con mejor valor en la función objetivo, la elección de soluciones que permitan mejores mejoras en la exploración o directamente, elegir una solución aleatoriamente. Una vez hecho esto se explorará la solución P, es decir, según la estructura de sus vecindades se obtendrán los valores en la función objetivo de sus vecinos y todos aquellos que mejoren la solución de la iteración actual (9), en caso de que pertenezca a LIVE, DEAD o NEWGEN no se hará nada ya que han sido objetivo de exploración o lo serán, sin embargo en caso de no estarlo (11) habrá dos posibilidades: La primera que se desborde la memoria en caso de que se añadan a LIVE (13), por lo que todos los integrantes de NEWGEN pasarán a formar parte de LIVE (14) y se pasará a la operación de pos procesado (15) y la segunda que no se desborde(16), en este caso, todos los vecinos P_n de la solución actual *P* pasarán a formar parte de NEWGEN con el objetivo de ser explorados en la próxima iteración.

No olvidar que esto es un bucle cuya duración depende del número de vecinos encontrados por lo que es posible que para el primero haya espacio en la memoria pero esta se acabe llegado un determinado número de iteraciones.

Una vez incluidos todos los vecinos en NEWGEN, estos serán añadidos a LIVE y de LIVE serán eliminadas todas las soluciones ya exploradas (20) añadiéndolas a DEAD (21). En este momento finaliza la iteración y en caso de seguir habiendo componentes en LIVE, esta seguirá teniendo lugar hasta su finalización.

Para que el pos procesado sea llevado a cabo se tiene que cumplir la condición (14). De todas las soluciones que en este momento estén guardadas en LIVE, en la de menor coste se hará una búsqueda local simple (25) en la que se obtendrá la solución, que será añadida a DEAD (26) y se vaciará LIVE (27).

Unja vez finalizado el posprocesado (28) o alcanzada la condición de parada (22) se llevará a cabo la obtención del valor (P_{lo}) que produzca un coste menor en la función objetivo *F* que haya sido guardado en DEAD (29). En caso de que sea mejor que la solución inicial que fue guardada en la variable *mejor_solución*, esta variable pasará a valer P_{lo} , valor que devolverá el algoritmo y que significa queel valor de la función objetivo ha sido mejorada.

Bibliografía

En este último apartado se enumeran las fuentes (documentos, libros, artículos, webs, etc.) consultadas para la realización de este Trabajo Fin de Grado.

Carro, R., & González Gómez, D. A. (2012). Localización de instalaciones.

Chudak, F. A., & Shmoys, D. B. (2003). Improved approximation algorithms for the uncapacitated facility location problem. *SIAM Journal on Computing*, 33(1), 1-25.

Daskin, M.S. (2010) "Service Science" John Wiley & Sons.

Daskin, M.S. (2013) "Network and Discrete Location, Models, Algorithms

and Applications" Second editionNew York: John Wiley & Sons, INC.

Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research, 26(6),* 992-1009.

Flores Garrido, L., & Oliva San Martín, C. (2016). Algoritmos para el problema de localización de plantas y centros de distribución maximizando beneficio. *Ingeniare. Revista chilena de ingeniería,* 24(3), 493-501.

García, M. S. (2000). Optimización combinatoria. *Números: Revista de didáctica de las matemáticas*, (43), 115-120.

Ghaziri, H., Diaz, A., & Glover, F. (1996). Optimización heurística y Redes Neuronales. *Madrid: Paraninfo*.

Glover, F., & Laguna, M. (1998). Tabu search. *Handbook of combinatorial optimization* (pp. 2093-2229). Springer, Boston, MA.

Guéret, C., Prins, C., & Sevaux, M. (1999*). Applications of optimization with Xpress-MP*. contract, 00034.

Hansen, P., Mladenovic, N., & Pérez, J. A. M. (2003). Búsqueda de entorno variable. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 7(19), 0

Martı, R. (2003). Procedimientos metaheurısticos en optimización combinatoria. *Matemátiques, Universidad de Valencia*, 1(1), 3-62.

Marulanda, M. V., Leguizamón, G. I., & Mora, K. Y. N. (2017). Solución al problema de localización (cflp) a través de búsqueda tabú y relajación lagrangeana, caso de estudio: industria de productos alimentarios. *Puente,* 4(2), 55-61.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11), 1097-1100.

Revelle, C. S., & Laporte, G. (1996). The plant location problem: New models and research prospects. *Operations Research*, 44(6), 864-874.

Sierra García, R. (2018). Problemas de Localización de Servicios Públicos y Privados. Métodos Heurísticos Basados en Relajación Lagrangiana. *UVadoc*.

Soriano, M. A. (2007). Algoritmos Voraces. Facultat d'Informática, UPC.

Sun, M. (2006). Solving the uncapacitated facility location problem using tabu search. *Computers & Operations Research*, 33(9), 2563-2589

Tapias-Isaza, C. J., Galeano-Ossa, A. A., & Hincapié-Isaza, R. A. (2011). Planeación de sistemas secundarios de distribución usando el agoritmo Branch and Bound. *Ingeniería y Ciencia*, 7(13).

Tuzun, D., & Burke, L. I. (1999). A two-phase tabu search approach to the location routing problem. *European journal of operational research*, 116(1), 87-99.

http://people.brunel.ac.uk/~mastjjb/jeb/info.html

https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/principal.html

https://www.fico.com/es/products/fico-xpress-optimization

https://statgraphics.net