UNIVERSIDAD DE LA LAGUNA

DEPARTAMENTO DE ESTADÍSTICA,INVESTIGACIÓN OPERATIVA Y COMPUTACIÓN

ÁRBOLES Y HEURÍSTICAS EN LOCALIZACION. EL MODELO CENTDIAN MÚLTIPLE.

Dionisio Pérez Brito
Memoria para optar al grado de Doctor en Ciencias Matemáticas realizada bajo la dirección del Dr. D. José A. Moreno Pérez

Contenido

PRÓLOGO.

Los problemas de localización han sido objeto de estudio durante siglos, pero no fue hasta la emergencia de la Investigación Operativa cuando estos problemas tomaron mayor interés, siendo actualmente muchas las disciplinas en las que se estudian estos tipos de problemas. Arquitectos, informáticos, economistas, ingenieros, matemáticos y geógrafos han descubierto que tienen un interés común, en lo que concierne a la teoría de la localización.

En un problema de localización, dado un conjunto de puntos de demanda y un conjunto de posibles localizaciones, se desea hallar la ubicación de varios servicios de forma que, al asignar los puntos de demanda a los servicios, se optimice alguna función de distancia, costo o tiempo del trayecto. Esta función suele denominarse función objetivo, y entre las más estudiadas cabe destacar las funciones Centro y Mediana.

El problema de la Mediana consiste en minimizar la distancia media entre el usuario y el servicio a localizar. Este tipo de función objetivo mini-sum es apropiado para localizar servicios que deben satisfacer una demanda continua y estable, como, por ejemplo, escuelas, centros comerciales, terminales de transporte, etc. Por otro lado, el problema del Centro estriba en localizar un punto de tal manera que la distancia del usuario más alejado a él sea mínima. Esta función objetivo mini-max es utilizada con frecuencia en servicios de emergencias tales como estaciones de ambulancias, de bomberos o de policia.

En muchos problemas reales el objetivo es una mezcla de estos dos modelos, muchas veces antagónicos. Por ejemplo, a la hora de ubicar un puesto de bomberos se puede pretender minimizar el tiempo de viaje al posible punto de demanda más alejado, siempre y cuando el puesto esté suficientemente cerca de las áreas más pobladas. El problema consiste en minimizar la función Centro con la restricción de que la función Mediana no supere un determinado nivel. Otro ejemplo es el de minimizar la distancia media que deben recorrer los niños para llegar a la escuela sin que a ninguno le quede demasiado lejos. El objetivo aquí es minimizar la función Mediana imponiendo una cota superior a la función Centro.

Normalmente, cada usuario lleva asignado un valor positivo, llamado peso o ponderación, que a menudo representa el número de veces que dicho usuario visita el servicio.

El modelo que sugirió Halpern (1976) consiste en minimizar una combinación

convexa de la distancia media ponderada y la distancia máxima no ponderada; la solución a este problema recibe el nombre de Centdian. Con ello se consigue un equilibrio entre la distancia media y la distancia mayor.

El presente trabajo se ha realizado en el marco de la localización en grafos, haciendo hincapié en una clase especial de grafo llamada árbol, y en el modelo Centdian. Este grafo debe su nombre a la posibilidad de poder ser dibujado imitando a un árbol genealógico, donde los vértices representan a los miembros de la familia y las aristas la relación entre padres e hijos. Los árboles genealógicos al igual que los grafos árboles no tienen ciclos, pues un ciclo significaría ser padre e hijo de sí mismo. Como ejemplos de árboles tenemos, entre otros, algunas carreteras y autopistas, la mayoría de las vías ferroviarias y casi todos los ríos.

Con el objetivo de reflejar el estado actual de los problemas de localización en árboles, el capítulo 1 se dedica al repaso de los modelos clásicos tanto en el árbol como en su representación más simple, la recta real, o lo que es lo mismo, un árbol en el que cada padre sólo puede tener un hijo. Además, se recuerdan otros problemas que también tienen gran relevancia en el ámbito de la localización en árboles y a los que se hace referencia a lo largo del presente trabajo.

Muchos problemas de localización, como los problemas mencionados en su versión múltiple del Centro y la Mediana, son NP-*Hard*, y no se espera que sean resueltos de forma exácta por ningún algoritmo que tenga complejidad polinomial. Estos problemas pueden ser analizados desde el punto de vista de la optimización combinatoria llegando a resolverse con técnicas de enumeración, sin embargo, estas técnicas no son prácticas en problemas de dimensiones reales, pues no generan una solución óptima en un tiempo razonable. Es en estos casos en los que deben utilizarse los métodos heurísticos, con el fin de encontrar soluciones satisfactorias que puedan estar en el camino de la solución óptima del problema en cuestión.

Cuando el grafo no es árbol, se puede considerar aproximaciones a éste. Una de tales aproximaciones es el llamado árbol generador o de expansión, el cual es un subgrafo que contiene todos los vértices del grafo original, pero con sólo algunas de sus aristas. Estas aproximaciones nos permiten utilizar algoritmos para resolver problemas de localización en árboles, cuyas soluciones lo son aproximadamente del mismo problema en el grafo original.

En esta línea, se proponen en el capítulo 2 varias estrategias heurísticas para resolver problemas de localización en grafos, haciendo uso de los algoritmos construidos para árboles, obteniéndose en muy poco tiempo soluciones próximas a la óptima. El código de las mismas se halla en el apéndice A. También hay que resaltar en este capítulo, los resultados de la heurística VNDS, que ha sido diseñada para resolver problemas de optimización combinatoria en grafos de dimensiones considerables. Esta ha sido probada con grafos del orden de 6000 vértices, mejo-

rando apreciablemente los resultados obtenidos con otras heurísticas. El código de VNDS se muestra en el apéndice B.

En el capítulo 3 se estudia la función Centdian en un grafo considerando la función Centro ponderada, generalizando así el modelo de Halpern. Se entiende que las ponderaciones de la función Centro no tienen que ser iguales a las ponderaciones de la función Mediana. Por simplicidad este nuevo problema se sigue denotando Centdian. Este capítulo comienza analizando el caso de localizar un sólo servicio, extendiendo todas las propiedades del modelo propuesto por Halpern, para el nuevo modelo que aquí se presenta. A continuación, se estudia el problema del 2-λ-Centdian, mediante un análisis de la función objetivo, y se propone un algoritmo de complejidad $O(m^2n^4)$, donde m y n son respectivamente el número de aristas y vértices del grafo considerado.

Exceptuando un apartado del artículo de Hooker y otros (1991), en el cual se propone un conjunto finito dominante para el p-λ-Centdian, no se conoce literatura que haga mención al λ-Centdian múltiple. Este conjunto finito dominante propuesto por Hooker, es erróneo, como se muestra en el contraejemplo que aparece en este capítulo 3. También, se presenta un nuevo conjunto finito dominante para el problema p-λ-Centdian en un grafo con una demostración detallada del mismo. Finalmente, como consecuencia de éste, se propone un algoritmo exacto.

En el cuarto y último capítulo, se estudia el problema p-λ-Centdian en un árbol, y se propone el primer algoritmo polinomial para el problema p-λ-Centdian (generalizado o no) en árboles. En el caso generalizado, es decir, cuando la función Centro se considera con pesos, la complejidad de este algoritmo es de $O(pn^6)$, para $p \geq 6$, y $O(n^p)$, si $p < 6$. En el caso no generalizado, es decir, cuando la función Centro es sin pesos, la complejidad disminuye a $O(pn^5)$, para $p \geq 5$, y $O(n^p)$, si $p < 5$.

También se considera la versión discreta del modelo, donde los p puntos deben seleccionarse entre los vértices. En este caso, la complejidad se reduce a $O(pn^4)$, para $p \geq 4$, y $O(n^p)$, si $p < 4$. El capítulo termina con el estudio de la función Centdian en la recta real.

Con el objetivo de que el lector pueda seguir cualquier capítulo sin necesidad de leer los previos, se han redactado de forma independiente conservando a su vez la coherencia del trabajo en su conjunto. Esto implica que en algunos casos se repitan conceptos ya expuestos anteriormente.

Capítulo 1

LOCALIZACIÓN EN ÁRBOLES.

1.1 Introducción.

Si en la literatura de investigación operativa se realizara una búsqueda con los localizadores **árboles** y **localización**, obtendríamos una visión de la relevancia que han tenido y tienen los problemas de localización en el ámbito de los árboles. Entre los modelos de localización más tratados aparecen:

- Los clásicos problemas del p-Centro y de la p-Mediana, así como los relativamente recientes problemas del p-Anticentro y de la p-Antimediana, tratados tanto en el árbol como en su representación más sencilla, la recta real.
- Los modelos multiobjetivos, apropiados para optimizar simultáneamente más de una función objetivo.
- Los modelos de Programación Matemática, donde formulaciones alternativas permiten estudiar los típicos problemas de localización como problemas de Programación Matemática.
- Los modelos con distancias restringidas, aplicables a problemas en los que existe algún tipo de acotamiento entre la distancia desde el usuario al servicio.

Con el objetivo de reflejar el estado actual de la localizacion en árboles, en este capítulo se analizan cada uno de estos modelos, resaltando las aportaciones más significativas de cada uno de ellos acaecidas en la literatura.

1.1.1 Los árboles en localización.

Un caso especial, e interesante, de localización en grafos tiene lugar cuando la estructura del grafo no contiene ciclos, esto es, cuando el grafo es un árbol. Las propiedades y resultados de los árboles, además de ser elegantes y útiles, de proporcionar reglas y dar soporte a la teoría, constituyen el primer paso natural para el estudio de estos problemas en grafos generales, por ello árboles o grafos acíclicos son estructuras familiares en análisis de grafos.

Los árboles generadores de un grafo corresponden a soluciones básicas de importantes problemas de programación matemática en grafos; los árboles generadores mínimos y los árboles de caminos mínimos son clásicas construcciones en Teoría de Grafos. En la teoría de la localización en grafos, los árboles son metodológicamente importantes debido a la propiedad de convexidad que ellos poseen lo cual a menudo permite procedimientos muy eficientes para localizar servicios en ellos. Un ejemplo de ello son los algoritmos de Goldman [58] y Handler [69] para la Mediana, y el Centro respectivamente.

Por otra parte, un caso particular de problemas de localización son los problemas con distancias restringidas, en los que se desea localizar nuevos servicios en un grafo dentro de una distancia especificada tanto en los servicios existentes como entre ellos. Goldman and Dearing [59] mostraron los motivos y la necesidad que inducen a usar tales problemas. La distancia restringida en un grafo fue formalmente definida por Dearing, Francis, y Lowe [33]. En dicho trabajo establecieron que la distancia restringida define siempre un conjunto convexo si, y sólo si, el grafo es un árbol. Hay que tener en cuenta, no obstante, que no todos los resultados de convexidad asociados a espacios Euclídeos, ni siquiera los familiares resultados de la programación no lineal, se pueden aplicar en el contexto de la localización en grafos. Por ejemplo, la multiplicación de un punto en un árbol por un número real no tiene sentido, como tampoco lo tiene la noción de hiperplano.

1.1.2 Conceptos básicos.

Recordemos ahora algunos conceptos básicos de teoría de grafos, que se requerirán a lo largo de este trabajo.

Definición 1.1.1 *Un* **grafo** $G = (V, E)$ *es una colección de puntos o vértices* $V = \{v_1, .., v_n\}$, *y una colección de líneas o aristas* $E = \{e_1, .., e_m\}$, *que unen todos o algunos de estos puntos. Cada arista tiene asignada una longitud positiva.*

Definición 1.1.2 *Un* **punto interior***a una arista se referencia por su distancia a los extremos de dicha arista. $P(G)$ denota el conjunto continuo de puntos en las aristas de G.*

Definición 1.1.3 *Un grafo G es* **conexo** *si el conjunto de vértices V no se puede separar en dos subconjuntos no vacíos V_1 y V_2 de forma que cualquier arista $e \in E$ una dos vértices de V_1 o dos vértices de V_2.*

Definición 1.1.4 *Un* **subgrafo** *de G es un grafo $G' = (V', E')$ tal que $V' \subseteq V$ y $E' \subseteq E$.*

Definición 1.1.5 *Un* **camino** *C entre dos vértices i y j de un grafo G es un subgrafo minimal conexo conteniendo a los vértices i y j. (Es decir, que cualquier grafo parcial de C distinto de C conteniendo a i y a j es no conexo).*

Definición 1.1.6 *Una* **subarista** *es el segmento de arista desde un punto interior de la misma a uno de sus vértices.*

Definición 1.1.7 *La* **longitud de un camino entre dos puntos**, *es la suma de las longitudes de las aristas y subaristas que lo componen.*

Definición 1.1.8 *La* **longitud del camino más corto** *entre cualesquiera $x, y \in P(G)$ se denota por $d(x, y)$.*

Definición 1.1.9 *Un* **ciclo** *es un camino de longitud no nula entre un vértice i y si mismo.*

Definición 1.1.10 *Un* **grafo parcial** *de G es un subgrafo con su mismo conjunto de vértices; $V' = V$.*

Definición 1.1.11 *Un* **árbol** *T es un grafo minimal conexo, es decir, tal que T es conexo, pero cualquier grafo parcial de T, distinto del propio T, no es conexo. Un árbol se puede definir también como un grafo conexo sin ciclos no vacíos.*

Definición 1.1.12 *Un* **árbol generador** *T de G es un grafo parcial de G que es árbol.*

Definición 1.1.13 *La* **distancia entre dos árboles** $T_1 = (V_G, E_1)$ *y* $T_2 = (V_G, E_2)$ *del grafo* G *se denota por* $d(T_1, T_2)$ *y es igual al número de aristas de* T_1 *que no están en* T_2*, que coincide con el número de aristas que están en* T_2*, pero no en* T_1*.*

Si la distancia $d(T_1, T_2) = 1$ entonces T_1 podría obtenerse de T_2 reemplazando una arista por otra, ya que existen $e_1 \in E_1$ y $e_2 \in E_2$ tales que:

$$(E_1 \cup E_2) \backslash (E_1 \cap E_2) = \{e_1, e_2\}.$$

Definición 1.1.14 *Cuando la distancia entre dos árboles generadores es uno, se dice que se ha realizado una* **transformación elemental** *entre árboles generadores.*

Por lo tanto $d(T_1, T_2)$ es el mínimo número de transformaciones elementales para pasar del árbol T_1 al árbol T_2.

Definición 1.1.15 *El* **grado** *de un vértice de un grafo* G*, es el número de aristas que tiene asociado dicho vértice.*

1.1.3 Convexidad en árboles.

En este apartado se considera la noción de combinación convexa en grafos. Sean cualesquiera dos vectores, digamos V y W, en el espacio Euclídeo E^r. Para $0 \leq \lambda \leq 1$, la ecuación $U = \lambda V + (1 \Leftrightarrow \lambda) W$ es equivalente a $|v_j \Leftrightarrow u_j| + |u_j \Leftrightarrow w_j| = |v_j \Leftrightarrow w_j|$ y $|u_j \Leftrightarrow w_j| = \lambda |v_j \Leftrightarrow w_j|$, con $j = 1, .., r$. Para obtener la noción de combinación convexa en un grafo G, consideramos cualquier par de vectores de r puntos Y, $Z \in G^r$, y cualquier λ, $0 \leq \lambda \leq 1$. Denotamos por $L_\lambda(Y, Z)$ a:

$$L_\lambda(Y, Z) = \{X \in G^r : d(y_j, x_j) + d(x_j, z_j) = d(y_j, z_j), \\ d(x_j, z_j) = \lambda d(y_j, z_j), j = 1, .., r\}.$$

Se puede interpretar que la combinación convexa de Y y Z son los puntos de $L_\lambda(Y, Z)$ para algún $\lambda, 0 \leq \lambda \leq 1$. Es más:

$$L(Y, Z) = \bigcup_{0 \leq \lambda \leq 1} L_\lambda(Y, Z)$$

es el segmento de línea que une Y con Z. En particular, para $r = 1$, si G es un árbol T, entonces $L(y, z)$ es el camino en T que une y con z. Además $\forall \lambda \in [0, 1]$, $L_\lambda(y, z)$ es único y se puede denotar x como $x = \lambda y + (1 \Leftrightarrow \lambda) z$.

Definición 1.1.16 *El conjunto $S \in G^m$ es un* **conjunto convexo** *si $L(Y,Z) \subset S$ para cualesquiera $Y, Z \in S$.*

Definición 1.1.17 *Una* **función** *f definida en el conjunto S es* **convexa** *en G^m, si para $Y, Z \in S$, $f(X) \leq \lambda f(Y) + (1 \Leftrightarrow \lambda) f(Z)$ con $X \in L_\lambda(Y,Z)$ y $0 \leq \lambda \leq 1$.*

Para problemas de localización en grafos serán de interés las funciones convexas en árboles, a la vista de dos resultados básicos que se exponen a continuación.

1) la función $f_1(x) = d(x,a)$ es convexa en G, $\forall a$.

2) la función $f_2(x_1,x_2) = d(x_1,x_2)$ es convexa en $G^2 \Longleftrightarrow G$ es un árbol T.

Estos resultados dan pie a las siguientes conclusiones:

Sea $S \subset T^r$ un conjunto convexo. Si f es cualquier función convexa de S, entonces cualquier mínino local de f en S es un mínimo global en S. Dadas las funciones convexas $f_1, ..., f_p$ de S, la suma de las funciones, y el máximo de las mismas, es convexa en S. También si h es una función convexa en S y g es una función convexa no decreciente, entonces la función $f(X) = g(h(X))$ es convexa en S. Por lo tanto, para cualquier par de números w, w', la función $w\, d(x,a)$ es convexa en T, y la función $w' d(x_1,x_2)$ es convexa en T^2.

Estos resultados nos permiten asegurar que los problemas de la *Mediana* y *Centro* en árboles son de minimización convexa.

Hay que hacer hincapié en que los problemas convexos de localización conocidos son tratables en grafos, puesto que existen algoritmos de orden bajo para resolverlos, así como teorías elegantes. En cualquier caso, encontramos problemas de localización que no son convexos, como el problema del p-Centro ($p \geq 2$), que pueden ser tratados en grafos.

1.2 Los problemas clásicos en Localización.

Como se ha comentado anteriormente, para cualesquiera $x, y \in P(G)$, $d(x,y)$ denota la longitud del camino más corto entre ambos. También para cualquier subconjunto $Y \subseteq P(G)$, $d(x,Y) = \min\{d(x,y) : y \in Y\}$.

Sea $U \subseteq V$ el conjunto de vértices clientes, cuya cardinalidad es $O(n)$, dado que $|V| = n$.

Definición 1.2.1 *Una función g definida en R^k es* **monótona** *si para cualquier w, $z \in R^k$, $w \leq z \Rightarrow g(w) \leq g(z)$.*

Dado el conjunto finito de puntos $X = \{x_1, .., x_p\}$ de $P(G)$, se definen las siguientes matrices:

$$d(X) = \{d(x_i, x_j) : x_i, x_j \in X\}$$

$$d^1(X) = \{d(v_i, x_j) : v_i \in U, x_j \in X\}.$$

Sea $f(U, X) = f(d^1(X), d(X))$ una función real monótona en las componentes de ambas matrices. Una amplia variedad de modelos de localización en la literatura son definidos optimizando varias formas de f sobre clases de subconjuntos $X \subseteq P(G)$, $|X| = p$.

El conjunto X hace el papel de los nuevos servicios, es decir, servidores o fuentes, mientras que el conjunto U es identificado como el conjunto de servicios existentes o clientes.

Convencionalmente los modelos de localización ordinarios suelen ser definidos minimizando o maximizando alguna función de las distancias entre servicios y servicios-clientes. Los elementos de U pueden tener o no asignado un parámetro llamado peso, el cual puede tener varios significados, densidad poblacional, número de veces que se visita el servicio, tiempo de trayecto, etc.

Los problemas más populares entre los investigadores interesados en localización son: el p-Centro, el p-Anticentro, la p-Mediana y la p-Antimediana.

Por caso discreto se entiende el caso en que los nuevos puntos deben ser seleccionados entre los elementos del conjunto U.

Dependiendo si los servicios son distinguibles o no, se estudian dos casos: el homogéneo y el no homogéneo.

En el caso homogéneo, los servicios, es decir, los elementos de X, son indistinguibles desde el punto de vista del servicio que prestan. En este caso, cada cliente, elegirá para ser atendido el servicio más cercano. Ejemplos en los que el modelo homogéneo se refleja en la realidad son la ubicación de colegios, buzones, hospitales, basureros, etc.

En la versión no homogénea de estos modelos, los nuevos centros de servicios, es decir, los elementos de X, son distinguibles en función del servicio que prestan. Por tanto, cada cliente es atendido por todos los servicios. Un ejemplo práctico del modelo lo encontramos, al resolver el problema de localizar dos estaciones de bomberos especializadas. La estación química debe estar en las zonas más densamente pobladas, mientras que la estación forestal debe estar próxima a los

bosques, y además no deben estar muy separadas. Sin embargo en cualquier punto puede haber ambos tipos de incendio.

A continuación, se presentan los modelos anteriormente mencionados tanto en la recta real como en un árbol, clasificados según su homogenidad y su peso. Al mismo tiempo de cada uno de ellos se comentan brevemente las últimas aportaciones que se han presentado en la literatura.

1.2.1 Localización en la Recta Real.

Siguiendo con la notación anterior, se denota por $G = (V, E)$, un grafo no dirigido donde el conjunto V de vértices son n puntos de la recta real, y donde E es el conjunto de $n \Leftrightarrow 1$ aristas que une cada par de puntos consecutivos. La pecularidad de este grafo reside en que todos los vértices menos dos tienen grado dos, y estos dos son precisamente los vértices extremos los cuales tienen grado uno. En definitiva G se puede representar gráficamente como un segmento de la recta real.

El objetivo de un problema de localización en G es seleccionar un conjunto $X \in P(G)^p$, $|X| \leq p$ de puntos de la recta real de manera que una determinada función objetivo sea optimizada. Se asume que los vértices están ordenados de menor a mayor, dado que estos representan números reales, es decir, $v_i \leq v_{i+1}$, $\forall i$.

Hay una pequeña diferencia entre resolver un problema en la recta real con los vértices ordenados y resolver el mismo problema en un camino de un grafo. Asumiendo que este camino viene dado como una lista de $(n \Leftrightarrow 1)$ aristas, ésta requerirá un tiempo $O(n)$ para ordenar los n vértices. En muchos algoritmos añadir este factor $O(n)$ no afecta a la complejidad total. Pero éste no es el caso, por ejemplo, del problema del p-Centro no pesado homogéneo.

Exponemos a continuación los resultados más relevantes para los problemas clásicos de localización en este caso especial de árbol, considerando primero el caso homogéneo y a continuación el no homogéneo.

1.2.2 CASO HOMOGÉNEO.

En el caso homogéneo, los servicios, es decir, los elementos de X, son indistinguibles desde el punto de vista del servicio que prestan. En este caso, cada cliente, elegirá para ser atendido el servicio más cercano.

El problema del p-Centro pesado.

En el problema del p-Centro pesado, también conocido como problema minimax, la función objetivo trata de evaluar la situación del punto de demanda más perjudicado, y es aplicable a la localización de servicios de emergencias. A las correspondientes soluciones óptimas se les denominan Centros. La función objetivo a minimizar sería:

$$f_c(U,X) = \max_{v_i \in U} \{w_i\, d(v_i, X)\}.$$

donde w_i, es el peso no negativo asociado a $v_i \in U$.

El tiempo que se requiere para resolver el problema cuando $p = 1$ es $O(n)$ incluso cuando los puntos de U no están ordenados, Megiddo (1983) [118]. Para $p = 2$, la complejidad dada por Ko y Ching (1992) [96] es $O(n)$. Sharir y Welzl (1996) [156] probaron que para $p = 3$ también el tiempo requerido es $O(n)$.

La generalización de este problema para un p fijo arbitrario, se encuentra en el desarrollo de Sharir y Welzl (1996) [156].

En el caso de que p fuese variable, la complejidad que tendríamos para la versión discreta sería $O(n \log n)$, Megiddo, Tamir, Zemel y Chandransekaran (1981) [115]. Algoritmos alternativos han sido dados por Megiddo y Tamir (1983) [117], y Cole (1987) [17] con complejidades $O(n \log^2 n)$ y $O(n \log n)$ respectivamente.

El problema del p-Centro no pesado.

El problema del p-Centro no pesado consiste en minimizar la siguiente función objetivo:

$$f_c(U,X) = \max_{v_i \in U} \{\, d(v_i, X)\}.$$

Megiddo y Tamir (1983) [118] propusieron un algoritmo con una complejidad del orden $O(p^2 log^2 n)$. Otra alternativa la dio Frederickson en 1990 [51] con un algoritmo de complejidad $O(n)$. Si $p = 1, 2$, la complejidad alcanzada sería $O(\log n)$ Frederickson (1990) [51]. La solución del problema del 2-Centro se obtiene por la descomposición de éste en dos problemas de 1-Centro, Handler (1978) [70]. Esta descomposición no es válida para el caso discreto. Los resultados de Handler no son extensibles a valores mayores de p. Existe un algoritmo de complejidad $O(p\, n)$ basado en los métodos de búsqueda en vectores monótonos, Olstad y Manne (1993) [133].

El problema del p-Anticentro.

Este problema es también conocido como el problema p-Dispersión o p-Maximín, y tiene como objetivo maximizar la siguiente función objetivo:

$$f_{ac}(U,X) = \min_{v_i \in U} \{w_i\, d(v_i, X)\},$$

donde w_i, es el peso no negativo asociado a $v_i \in U$.

Es decir, se trata de hallar p servicios lo más lejos posible de los puntos de demanda $v_i \in U$.

El problema del 1-Anticentro no pesado puede ser resuelto en tiempo lineal, buscando los dos puntos consecutivos más alejados de U, y hallando el punto medio del segmento que los une. Aplicando un procedimiento divide y vencerás estándar, se obtiene una cota del tiempo de ejecución de $O(n \log n)$.

El problema de dispersión, cuando queremos seleccionar p puntos de $P(G)$, maximizando la distancia entre los pares de puntos más cercanos, es el dual del problema del ($p \Leftrightarrow 1$)-Centro, Chandrasekaran y Tamir (1980,1982) [25],[26], Kolen y Tamir (1990) [98], y por lo tanto puede ser resuelto usando los algoritmos del p-Centro con un esfuerzo total de $O(p\,n + p^2 \log p)$.

El problema de la p-Mediana pesada.

En el problema de la p-Mediana pesada, también conocido como problema minisum o problema de Weber, la función objetivo trata de evaluar la situación media de los puntos de demanda. A las correspondientes soluciones óptimas se le denominan Medianas. El criterio minisum refleja la política de optimizar los gastos totales y es aplicable a localizaciones del sector privado. Luego la función objetivo a minimizar sería:

$$f_m(U,X) = \sum_{v_i \in U} w_i\, d(v_i, X),$$

donde w_i, es el peso no negativo asociado a $v_i \in U$.

Para $p = 1$, un algoritmo de tiempo $O(n)$, incluso cuando los puntos de U no están ordenados, se consigue aplicando el algoritmo de la Mediana de Blum y otros (1972) [9]. Para $p > 1$, un algoritmo de complejidad $O(p\,n)$ es dado por Hassin y Tamir (1991) [80].

El problema de la p-Mediana no pesada.

Ahora la función objetivo a minimizar es:

$$f_m(U,X) = \sum_{v_i \in U} d(v_i, X),$$

pues los pesos $w_i = 1$, para todo $v_i \in U$.

Contrariamente a lo que ocurre en los problemas de tipo minimizar f_c, donde el modelo sin pesos tiene menor complejidad que el mismo modelo con pesos, en los problemas minimizar f_m el hecho de tener o no peso asignado a cada vértice, no hace variar la complejidad de sus respectivos algoritmos. Por ello, los algoritmos empleados son los que se citaron para el problema de la p-Mediana pesada. El de Blum y otros (1972) [9] para $p = 1$, y el de Hassin y Tamir (1991) [80] para un p cualquiera.

El problema de la p-Antimediana.

En este problema también conocido como problema p-Maxisum, se pretende maximizar la siguiente función objetivo:

$$f_{am}(U,X) = \sum_{v_i \in U} w_i\, d(v_i, X).$$

donde w_i, es el peso positivo asociados a $v_i \in U$. Dado que la función objetivo es convexa [159] la 1-Antimediana es v_1 ó v_n.

Si los elementos de X pueden ser iguales, la solución de la p-Antimediana es la misma que la del problema de la 1-Antimediana. Si los elementos de X son distintos, la solución óptima no tiene por qué estar necesariamente ubicada en los vértices terminales. Por ejemplo, si $n = 4$, $v_i = i \Leftrightarrow 1$, $i = 1,2,3,4$, $w_1 = w_3 = 1$, $w_2 = 5$, $w_4 = 3$, y $p = 2$, el conjunto óptimo es $S = \{v_1, v_3\}$.

El mejor algoritmo que se conoce es el dado por Tamir (1991) [159], que por medio de un desarrollo en programación dinámica alcanza una complejidad $O(n\,p)$.

1.2.3 CASO NO HOMOGÉNEO.

En la versión no homogénea de estos modelos, los nuevos centros de servicio, es decir, los elementos de X, son distinguibles en función del servicio que prestan. Por tanto, cada cliente es atendido por todos los servicios. Esto hace que los

modelos no homogéneos tengan asociado dos pesos, uno entre el cliente v_i y el servicio x_j y al que denotaremos w_{ij}, y otro entre los servicio x_i y x_j al que denotaremos b_{ij}. Siguiendo con el ejemplo de los bomberos supongamos que la estación química es la x_1 y la forestal la x_2 de esta manera el cliente v_i es atendido por los dos servicios, cuando es atendido por x_1, el peso asociado es w_{i1}, y cuando es atendido por x_2 el peso sería w_{i2}. La distancia entre x_1 y x_2 va ponderada por b_{12} y la distacia entre x_2 y x_1 va ponderada por b_{21}.

El problema del p-Centro pesado

.

El problema es minimizar la función objetivo:

$$f_c(U,X) = \max \begin{Bmatrix} \max\{w_{ij}\, d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \max\{b_{ij}\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{Bmatrix}.$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente. Erkut, Francis y Tamir (1992) [44] suministraron un algoritmo de búsqueda binaria de complejidad $O(p\,n\,(p+n))$. Francis, Lowe y Ratliff (1978) [49] habían obtenido ya el mismo resultado por medio de la implementación del teorema de dualidad, con el inconveniente de que no es extensible al caso discreto. Sin embargo, el desarrollo paramétrico que realiza Megiddo (1979) [114], de orden $O(p\,(n+p))$ [49], puede ser utilizado en el desarrollo de Francis, Lowe y Ratliff (1978) [49] y es aplicable tanto al caso discreto como al caso continuo. La complejidad total de este desarrollo paramétrico es $O(p^2(n+p)^2)$.

El problema del p-Centro no pesado.

No se han encontrado referencias bibliográficas donde este modelo esté especificado.

$$f_c(U,X) = \max \begin{Bmatrix} \max\{\, d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \max\{\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{Bmatrix}.$$

Dado que el problema de minimizar la función objetivo $f_c(U,X)$ en la recta real es equivalente al problema del 1-Centro homogéneo, éste se puede resolver en tiempo $O(n)$, Tamir, (1997) [164].

El problema del p-Anticentro.

El problema del p-Anticentro consiste en maximizar la siguiente función objetivo:

$$f_{ac}(U,X) = \min \left\{ \begin{array}{l} \min\{w_{ij}\, d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \min\{b_{ij}\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{array} \right\}.$$

Este problema es NP-*Hard*, incluso cuando el segmento de recta real, se reduce a una sola arista, Tamir (1991) [159].

El problema de la p-Mediana pesada.

Gusfield y Tardos (1990) [60], Gusfield y Martel (1992) [61] suministraron un algoritmo de complejidad $O(p^3 + p^2 \log m + p\, n)$ para el problema de minimizar la siguiente función objetivo:

$$f_m(U,X) = \sum_{v_i \in U} \sum_{x_j \in X} w_{ij}\, d(v_i,x_j) + \sum_{x_i \in X} \sum_{x_j \in X} b_{ij}\, d(x_i,x_j).$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j.

Tamir (1993) [161], propone un algoritmo alternativo con la misma complejidad que el algoritmo de Gusfield y Martel (1992).

El problema de la p-Mediana no pesada.

En el problema de minimizar la función objetivo:

$$f_m(U,X) = \sum_{v_i \in U} \sum_{x_j \in X} d(v_i,x_j) + \sum_{x_i \in X} \sum_{x_j \in X} d(x_i,x_j),$$

el hecho de tener o no peso asignado a cada vértice, no influye en la complejidad de sus respectivos algoritmos. Así pues, el modelo sin pesos tiene la misma complejidad que el mismo modelo con pesos. Ello nos lleva a citar los mismos algoritmos que en el problema de la p-Mediana pesada. Es decir el de Gusfield y Tardos (1990) [60], el de Gusfield y Martel (1992) [61] y el de Tamir (1993) [161].

El problema de la p-Antimediana.

En el problema de la p-Antimediana el objetivo es maximizar:

$$f_{am}(U,X) = \sum_{v_i \in U} \sum_{x_j \in X} w_{ij}\, d(v_i,x_j) + \sum_{x_i \in X} \sum_{x_j \in X} b_{ij}\, d(x_i,x_j),$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j. O lo que es lo mismo, maximizar la suma de las distancia pesada entre todos lo pares de puntos. El problema de la p-Antimediana es NP-*Hard*, incluso, cuando $n = 2$, Tamir (1991) [159].

1.2.4 Localización en un Árbol.

Exponemos en esta subsección los resultados más relevantes para los problemas clásicos en árboles generales, considerando primero el caso homogéneo y a continuación el no homogéneo análogamente a lo expuesto para la recta real.

1.2.5 CASO HOMOGÉNEO.

En el caso homogéneo, los servicios, es decir, los elementos de X, son indistinguibles desde el punto de vista del servicio que prestan. En este caso, cada cliente, elegirá para ser atendido el servicio más cercano.

El problema del p-Centro pesado.

El problema del p-Centro consiste en encontrar p puntos $X = \{x_1, ..., x_p\}$ del árbol que minimicen la máxima distancia pesada. Si w_i es el peso positivo asignado al vértice $v_i \in U$, el modelo considerado trata de minimizar la siguiente función objetivo:

$$f_c(U, X) = \max_{v_i \in U} \{w_i \, d(v_i, X)\}.$$

El mejor algoritmo para el problema del p-Centro pesado lo ofrecen Megiddo y Tamir [118]. Éste es de complejidad $O(n \log^3 n)$, pero si se le aplica el procedimiento de Cole (1987) [17], la complejidad conseguida para este problema es $O(n \log^2 n)$.

Para el caso discreto, no se ha podido mejorar la complejidad del algoritmo de Megiddo y Tamir (1983) [118] la cual es $O(n \log^2 n)$.

El problema del p-Centro no pesado.

El problema del p-Centro no pesado consiste en minimizar la función objetivo:

$$f_c(U, X) = \max_{v_i \in U} \{ d(v_i, X)\}.$$

El algoritmo de Frederickson (1990) [51] para el p-Centro no pesado en la recta real es extrapolable a árboles, por lo tanto, su complejidad es de $O(n)$. También los resultado de Handler (1978) [70] son extrapolables a árboles para el problema del 1-Centro y del 2-Centro.

El problema del p-Anticentro.

Tamir (1991) [159], reformuló el problema de maximizar la función objetivo:

$$f_{ac}(U, X) = \min_{v_i \in U} \{w_i \, d(v_i, X)\},$$

donde w_i es el peso positivo asignado al vértice $v_i \in U$, como un problema de programación matemática para poder hacer uso de la concavidad de la función objetivo. Como resultado de su desarrollo propuso un algoritmo recursivo de orden $O(n\ p)$. Esta complejidad se puede mejorar en casos especiales como los árboles estrellas, árboles cuyo único vértice no hoja es la raíz. Además, este problema puede reducirse al problema de asignación discreto estándar con un objetivo cuadrático cóncavo separable, éste último resuelto en tiempo $O(n)$ Ibaraki (1988) [86].

El problema de la p-Mediana pesada.

Kariv y Hakimi (1979) [90] probaron que el problema de minimizar la función objetivo:

$$f_m(U, X) = \sum_{v_i \in U} w_i \, d(v_i, X),$$

donde w_i es el peso positivo asignado al vértice $v_i \in U$, es NP-*Hard* en un grafo general, y Lin (1980) [104], presenta algunos esquemas de aproximación.

Para un árbol, Kariv y Hakimi (1979) [90] describieron un algoritmo $O(p^2 n^2)$. Un algoritmo diferente de orden $O(p\, n^3)$ puede encontrarse en Hsu (1982) [84]. Tamir (1996) [163], mostró cómo por medio de un algoritmo de programación dinámica, el tiempo necesario para resolver el problema de la p-Mediana en un árbol es sólo $O(p\, n^2)$, este algoritmo de Tamir es el de mejor complejidad conocida. Si se compara, la complejidad dada por Hassin y Tamir (1991) [80], para el problema de la p-Mediana pesada en un camino, se encuentra que el tiempo requerido es de $O(p\, n)$.

El problema de la p-Mediana no pesada.

En el problema de minimizar la función objetivo:

$$f_m(U,X) = \sum_{v_i \in U} d(v_i, X),$$

el hecho de tener o no peso asignado a cada vértice, no repercute en la complejidad de su algoritmo, de manera que, el modelo sin pesos tiene la misma complejidad que el mismo modelo con pesos. Ello nos lleva a citar los mismos algoritmos que en el problema de la p-Mediana pesada.

El problema de la p-Antimediana.

Este problema también es conocido en la literatura como el problema p-Maxisum, y consiste en maximizar la función objetivo:

$$f_{am}(U,X) = \sum_{v_i \in U} w_i \, d(v_i, X).$$

donde w_i es el peso positivo asignado al vértice $v_i \in U$. En un grafo general es NP-*Hard*, pero en un árbol Tamir [159], solucionó el problema en tiempo $O(n\,p)$ valiéndose de la programación dinámica. Su procedimiento trata de identificar las útiles propiedades de concavidad, y reformular el modelo como un problema de flujo, cuadrático separable y de máxima concavidad.

1.2.6 CASO NO HOMOGÉNEO.

En la versión no homogénea de estos modelos, los nuevos centros de servicios, es decir, los elementos de X, son distinguibles en función del servicio que prestan. Por tanto, cada cliente es atendido por todos los servicios. Esto hace que los modelos no homogéneos tengan asociado dos pesos, uno entre el cliente v_i y el servicio x_j y al que denotaremos w_{ij}, y otro entre los servicio x_i y x_j al que denotaremos b_{ij}.

El problema del p-Centro pesado.

El desarrollo hecho en la recta real puede aplicarse para la resolución de problemas continuos en árboles. Haciendo uso de la descomposición del centroide[1]

[1] En la literatura se denomina centroide, al vértice que al separarlo del árbol, no deja ninguna componente conexa con más de $n/2$ vértices. Coincide con el vértice mediana.

[161] se obtiene, para el problema de minimizar la función objetivo:

$$f_c(U,X) = \max \left\{ \begin{array}{c} \max\{w_{ij}\, d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \max\{b_{ij}\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{array} \right\},$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente, un algoritmo de complejidad $O(np^2 + pn\log^2 n)$, Tamir (1993) [161].

El problema del p-Centro no pesado.

No existen referencias donde este modelo esté especificado. El objetivo del modelo es, minimizar la función objetivo:

$$f_c(U,X) = \max \left\{ \begin{array}{c} \max\{d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \max\{\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{array} \right\}.$$

El problema puede resolverse en tiempo $O(n)$, pues éste sería equivalente al problema del 1-Centro homogéneo, Tamir (1997) [164].

El problema del p-Anticentro.

El problema del p-Anticentro trata de maximizar la función objetivo:

$$f_{ac}(U,X) = \min \left\{ \begin{array}{c} \min\{w_{ij}\, d(v_i,x_j) : v_i \in U, x_j \in X\}, \\ \min\{b_{ij}\, d(x_i,x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{array} \right\}.$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente. Este poblema es NP-*Hard*, incluso cuando el árbol considerado consiste en una sóla arista, Tamir (1991) [159].

El problema de la p-Mediana pesada.

El algoritmo de Tamir (1993) [161], que resuelve el problema de minimizar la función objetivo:

$$f_m(U,X) = \sum_{v_i \in U} \sum_{x_j \in X} w_{ij}\, d(v_i,x_j) + \sum_{x_i \in X} \sum_{x_j \in X} b_{ij}\, d(x_i,x_j),$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente, mejora la complejidad $O(n\, p^3)$ de Kolen (1982) [97] y de Picar y otros (1978) [135]. Para ello, considera una arista $[v_i, v_j]$

y la elimina de E, particionando el conjunto de vértices en dos subconjuntos, $V(i,j)$ y $V(j,i)$, donde $v_i \in V(i,j)$ y $v_j \in V(j,i)$. Siguiendo a Picard (1978) [135] define el problema local-$[i,j]$ como:

$$\min_x \{ \sum_{\substack{v_t \in V(i,j) \\ m=1,\ldots,p}} w_{tm}\, d(v_i, x_m) + \sum_{\substack{v_t \in V(j,i) \\ m=1,\ldots,p}} w_{tm}\, d(v_j, x_m) + \sum_{k,m=1,\ldots,p} b_{km}\, d(x_k, x_m) \},$$

donde $x_1, ..., x_p$ están en $\{v_i, v_j\}$. Este problema es la reducción del problema original obtenido identificando todos los vértices de $V(i,j)$ con el vértice v_i, y todos los vértices de $V(j,i)$ con el vértice v_j.

Dado que la solución del problema local-$[i,j]$ es independiente de la longitud de la arista $[v_i, v_j]$ se concluye, como se muestra en [135], que el problema de la p-Mediana pesada no homogénea en un árbol es independiente de la longitud de las aristas, y que sólo depende de la topología del árbol. Este resultado justifica el que Tamir (1993) [161], haya utilizado el método de descomposición del centroide para resolver el mencionado problema en tiempo $O((p^3 + n)\log n + np)$.

El problema de la p-Mediana no pesada.

El problema de la p-Mediana no pesada, consiste en minimizar la función objetivo:

$$f_m(U,X) = \sum_{v_i \in U} \sum_{x_j \in X} d(v_i, x_j) + \sum_{x_i \in X} \sum_{x_j \in X} d(x_i, x_j).$$

El hecho de tener peso o no, no afecta a la complejidad de los algoritmos que resuelven el problema de la p-Mediana. Ello nos lleva a citar los mismos algoritmos que en el problema de la p-Mediana pesada.

El problema de la p-Antimediana.

El problema de la p-Antimediana consiste en maximizar la función objetivo:

$$f_{am}(V,X) = \sum_{v_i \in U} \sum_{x_j \in X} w_{ij}\, d(v_i, x_j) + \sum_{x_i \in X} \sum_{x_j \in X} b_{ij}\, d(x_i, x_j),$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente, es NP-*Hard*, incluso cuando el grafo considerado consiste en una sola arista, Tamir (1991), [159].

1.3 Modelos Multiobjetivo (Centdian).

En muchos problemas de localización, especialmente en el sector público, las decisiones se toman atendiendo a un número considerable de objetivos. Un ejemplo cotidiano lo tenemos cuando un grupo de pequeñas comunidades quieren construir un servicio común a todas ellas. Como es lógico, cada comunidad tiene su propio objetivo con respecto a la localización del servicio, pero diferentes comunidades conllevan diferentes objetivos. En un supuesto como éste, el analista tiene que encontrar la localización del servicio de tal forma que lo que beneficie a una comunidad no perjudique a ninguna otra.

En términos generales un problema multiobjetivo es aquel cuyo objetivo es optimizar un número k de funciones $f_1, f_2, ..., f_k$ sobre un conjunto no vacío X. Generalmente no existe ninguna solución de X que simultáneamente optimice todas las funciones. En estos casos, el primer paso habrá de ser encontrar el conjunto de soluciones eficientes de la colección de funciones.

Sea $f(x) = (\ f_1(x), ..., f_k(x))$ para $x \in X$. Una solución $x^* \in X$, es eficiente si $x \in X$ y $f(x) \leq f(x^*)$ implica $f(x) = f(x^*)$. Una solución x es dominada si para algún $x' \in X$, $f(x') \leq f(x)$ y $f(x') \neq f(x)$.

1.3.1 El Centdian.

El *Centdian*, es el nombre que Halpern [65] utiliza para el problema de localizar un servicio bajo dos objetivos: el minimax $f_c(U, X) = \max\limits_{v_i \in U} \{\, d(v_i, X)\}$ y el minisum $f_m(U, X) = \sum\limits_{v_i \in U} w_i\, d(v_i, X)$. Los primeros en considerar este problema en un árbol fueron Halpern [65] y Handler[69].

Halpern desarrolló el problema minimizando:

$$f_\lambda(U, x) = \lambda f_c(U, x) + (1 \Leftrightarrow \lambda) f_m(U, x)$$

sobre x para un λ dado, y parametrizando para $0 \leq \lambda \leq 1$. Probó que para cualquier λ dado, si x_c es el centro de un árbol T y x_m es un vértice mediana de T, entonces un mínimo de $f_\lambda(.)$ se alcanza en un punto del camino que conecta x_c con x_m. Además demostró que un mínimo de $f_\lambda(.)$ puede ser hallado resolviendo el problema de la mediana en un árbol T' que contenga un vértice adicional, el vértice $v_{n+1} = x_c$, con peso asociado $w_{n+1} = \lambda^{-1} \Leftrightarrow 1$.

Handler [69] consideró el problema sin pesos $w_i = 1$ y usó la función mediana como una restricción, es decir minimizó $f_c(U, x)$ sujeto a que: $f_m(U, x) \leq \alpha, x \in$

T, para cada valor de α, y parametrizó sobre α.

Ambos autores explotaron intrínsicamente la propiedad de convexidad de los árboles en la obtención de sus resultados.

Halpern [66] también estudió el problema en una red general, considerando que $f_\lambda(U,x)$ es una función continua y lineal a trozos en cada arista que alcanza su mínimo en uno de sus puntos de rupturas o en uno de sus vértices. Esto le llevó a un algoritmo que computa todas los puntos de rupturas del problema.

Empleando un desarrollo similar al de Handler, Halpern [67] obtuvo la relación de dualidad para los problemas de la Mediana y el Centro restringidos. Para λ, μ reales se define:

$$\begin{aligned} f_m^*(\lambda) = \quad & \min_{x \in P(T)} \quad f_m(U,x) \\ & s.a: \quad f_c(U,x) \leq \lambda \end{aligned}$$

$$\begin{aligned} f_c^*(\mu) = \quad & \min_{x \in P(T)} \quad f_c(U,x) \\ & s.a: \quad f_m(U,x) \leq \mu \end{aligned}$$

Sea m^+ el mínimo valor de $f_m(.)$ sobre cualquier $x \in P(T)$, y m^{++} el mínimo valor de $f_m(.)$ sobre cualquier mínimo de $f_c(.)$. Definimos c^+ y c^{++} de forma similar.

Teorema 1.3.1 Teorema de dualidad

a) Dado $\mu \in [m^+, m^{++}]$, con $\lambda = f_c^*(\mu)$, tenemos que $f_c^*(f_m^*(\lambda)) = \lambda$.

b) Dado $\lambda \in [c^+, c^{++}]$, con $\mu = f_m^*(\lambda)$, tenemos que $f_m^*(f_c^*(\mu)) = \mu$.

En un árbol, f_m^* y f_c^* son funciones biyectivas (uno a uno) en los intervalos $[m^+, m^{++}], [c^+, c^{++}]$ respectivamente, y son inversas una de la otra. En un grafo general, la propiedad de la inversa se verifica sólo para algunos miembros de $[m^+, m^{++}], [c^+, c^{++}]$ pues en este caso las funciones no son necesariamente biyectivas.

1.3.2 Modelos Multiobjetivo en árboles.

Lowe [108] consideró un problema multiobjetivo muy general en un árbol T dando un método para construir el conjunto de todos los puntos eficientes. Dadas $f_1, ...f_k$, k funciones continuas y acotadas del tipo de la Mediana o del tipo del Centro, todas ellas definidas en T, y dado Q_i un subconjunto factible no vacío

de T (se asume que Q_i es convexo y compacto), entonces se define:

$$Q = \cap\{Q_i : 1 \leq i \leq k\} \neq \emptyset.$$

El problema consiste en minimizar:

$$f(x) : x \in Q \subset T, \text{ donde } f(x) = (f_1(x), ..., f_k(x)).$$

Bajo la suposición de convexidad, Lowe demostró que un subconjunto compacto convexo T^* de T puede ser identificado como el conjunto de todos los puntos eficientes. Para obtener T^*, se define R_i^* como el conjunto de todos los mínimos del problema $\min\{f_i(x) : x \in T\}$. Si R_i^* intersecta el conjunto factible Q, S_i^* representa dicha intersección; en otro caso, S_i^* es la solución más cercana en Q a R_i^*, y además es única. Una vez definido cada S_i^*, $1 \leq i \leq k$, si sus intersecciones son no vacías, entonces el conjunto de todos los puntos eficientes viene dado por $T^* = \cap\{S_i^* : 1 \leq i \leq k\}$. Si, por el contrario, esta intersección es vacía, entonces T^* es el subárbol convexo y compacto más pequeño de T que intersecta cada S_i^*. Téngase en cuenta que las funciones $f_c(U, x)$ y $f_m(U, x)$ y la del Centdian $f_\lambda(U, x)$ son convexos en T. El teorema de Lowe es otro método para obtener los mismos resultados de Halpern y Handler.

1.4 Modelos de Programación Matemática.

En los problemas continuos se permite que los servicios puedan ser establecidos en cualquier punto de la red, deseándose encontrar p localizaciones óptimas, que minimicen la función objetivo y que al mismo tiempo sastifagan ciertas restricciones. Estas restricciones son normalmente cotas superiores de las distancias entre servicios.

En el caso de que la red sea un árbol, el propósito de este apartado es formular los típicos problemas de localización como problemas de programación matemática, los cuales, en algunos casos, son lineales y conllevan un número polinomial de variables y restricciones.

Sea T un árbol con n vértices $v_1, ..., v_n$, y sea $X = (x_1, ..., x_p)$ el vector de los servicios a localizar en dicho árbol T, $D(X) = d(v_i, X)$ representa el vector de todas las distancias de interés.

Denotaremos por PM (problema monotónico) el problema siguiente:

$$\begin{array}{lll} \min & f(D(X)) & \\ s.a: & f_i(D(X)) & \leq b_i,\ i = 1, ..., k. \end{array}$$

Se asume que cada función real f_i es monótona en cada componente de $D(X)$. Erkut y otros (1989) [43] prueban que cuando en las funciones monótonas f_i interviene las operaciones sumar y/o maximizar, PM es equivalente a un problema de programación lineal, que puede resolverse mediante el algoritmo del simplex.

Si PM se plantea sobre una red no árbol, problemas como el p-Centro y la p-Mediana son NP-*Hard*, Hsu (1979), Kariv (1979a), Kariv (1979b) ([83][89][90]). También determinar si hay o no soluciones factibles para estos problemas es NP-*Hard*, Kole (1982) [97].

La literatura en teoría de localización discreta refleja un vasto número de trabajos que usan algún tipo de relajación, Francis (1989) [50]. Pero el objetivo de este apartado no es estudiar tales problemas sino aquéllos que son esencialmente continuos por naturaleza, y expresarlos como modelos de programación matemática, sin relajaciones.

Lema 1.4.1 Lema de monoticidad.- Sea f una función monótona definida en un espacio euclideo $E^r \to E^s$. Para cualquier $b \in E^s$ y $D \in E^r$, los siguientes resultados son equivalentes.

(a) $f(D) \leq b$.

(b) Existe $Z > 0$, $Z \in E^r$ tal que $D \leq Z$, $f(Z) \leq b$.

Con este resultado y asumiendo que cada función f_i es monótona, $i = 0, 1, ..., k$, el problema PM se puede expresar como sigue:

$$\begin{array}{ll} \min & b_0 \\ s.a: & f_i(D(X)) \quad \leq b_i,\ i = 0, ..., k. \end{array}$$

El lema de monotocidad nos permite expresar la restricción como sigue:

$$\begin{array}{ll} f_i(Z) & \leq b_i,\ i = 0, ..., k. \\ D(X) & \leq Z. \end{array}$$

La ventaja de estos modelos de programación matemática reside en que para ellos se dispone de una herramienta teórica que facilita su resolución, ejemplo de ello lo tenemos en la teoría de dualidad, y en el método de simplex revisado. Otra ventaja es el análisis de sensibilidad y el estudio paramétrico de los mismos.

1.5 Modelos con distancias restringidas.

Los problemas de localización multiservicios con distancias restringidas consisten en localizar p nuevos servicios, satisfaciendo que la distancias entre los clientes y servicios, así como las distancias entre servicios no superen una cota dada. Específicamente, dado el grafo no dirigido $G = (V, E)$ con conjunto de vértices $V = \{v_1, ..., v_n\}$ y conjunto de aristas E, y dado $\{r_{ij}\}$ y $\{r'_{ik}\}$ números no negativos, el problema radica en localizar p servicios $x_1, ..., x_p \in P(G)$ tales que:

$$\begin{array}{ll} d(v_i, x_j) & \leq r_{ij},\ v_i \in U, x_j \in X, \\ d(x_i, x_k) & \leq r'_{ik},\ x_i, x_k \in X,\ i \lneq k. \end{array}$$

La primera de las cotas viene motivada por servicios que tienen un tiempo máximo de tolerabilidad. Por ejemplo, cuando se quiere establecer una estación de bomberos, es deseable que la respuesta de los bomberos se produzca antes de que el fuego esté fuera de control. La segunda de ellas, resulta apropiada cuando hay interacciones entre los servicios, como en el caso de establecer servicios con fines militares. Kolen (1982) [97] demostró que el problema es NP-*Hard*, pues lo redujo al problema del CLIQUE, es decir, al problema del subgrafo completo maximal (Garey y Johnson, 1979 [54]).

Francis, Lowe y Ratliff (1978) [49] consideraron el problema en el que G es un árbol. El algoritmo que propusieron para el caso continuo es de complejidad $O(np + p^2)$. Este algoritmo es fácilmente adaptable al caso discreto.

Si el objetivo es minimizar:

$$f_c(U, X) = \max \left\{ \begin{array}{c} \max\{w_{ij}\, d(v_i, x_j) : v_i \in U, x_j \in X\}, \\ \max\{b_{ij}\, d(x_i, x_j) : x_i, x_j \in X, \forall\, i \neq j\} \end{array} \right\},$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente, el problema se conoce como el problema del p-Centro restringido. Erkut y otros (1992) [44], presentaron dos algoritmos polinomiales para resolverlo en un árbol. El primer algoritmo está basado en una búsqueda binaria, con un esfuerzo de $O(n(m+n)[m+n\log M])$ (M como en [44]), cuya complejidad es polinomial, pero dependiente de los datos. El segundo algoritmo está basado en desarrollos paramétricos, y su complejidad es de $O(mn(m + n\log m) + n^3 \log n)$.

Si el objetivo es minimizar:

$$f_m(U, X) = \sum_{v_i \in U} \sum_{x_j \in X} w_{ij}\, d(v_i, x_j) + \sum_{x_i \in X} \sum_{x_j \in X} b_{ij}\, d(x_i, x_j).$$

donde w_{ij} es el peso asociado a v_i con respecto el servicio x_j y b_{ij} es el peso entre los servicios x_i y x_j respectivamente, el problema es comúnmente llamado el problema de la p-Mediana restringida. Por las propiedades de la función objetivo de la p-Mediana, una p-Mediana óptima se encuentra en algún conjunto de:

$$Y = \bigcup_{\substack{v_i \in U \\ j=1..p}} Y_i(r_{ij}).$$

donde:

$$Y_i(r_{ij}) = V \cup \{y : y \in P(G), w_i d(v_i, y) = r_{ij}, v_i \in U, j = 1, .., p\},$$

donde w_i es el peso positivo asociados a $v_i \in U$.

Este conjunto es de cardinalidad $O(n^2)$. En Kim y otros (1996), [94], se muestra como computar y añadir el conjunto $Y_i(r_{ij})$, al conjunto de vértices del árbol T en tiempo $O(n^2)$. Por lo tanto si se aplica el reciente algoritmo de Tamir (1996) [163], el problema se resuelve en tiempo $O(pn^4)$. Este procedimiento y una mejora del mismo están detallados en el capítulo 3 de esta memoria.

1.6 Conclusiones.

En este capítulo se han considerados los problemas clásicos de localización en sus diferentes versiones. Las siguientes tablas recogen las referencias y complejidades de cada uno de ellos, clasificados por su homogeneidad, peso y repulsividad. El modelo repulsivo es equivalente al modelo Anticentro o Antimediana.

El problema del p-Centro en la Recta Real				
	Homogeneo		No-Homogeneo	
Pesado	$O(nlogn)$	Megiddo (1991) [115]	$O(pn(p+n))$	Francis (1978) [49]
No Pesado	$O(n)$	Frederickson (1990)[51]	$O(n)$	Tamir (1997) [164]
Repulsivo	NP-*Hard*	Tamir (1991) [159]	NP-*Hard*	Tamir (1991) [159]

El problema de la p-Mediana en la Recta Real				
	Homogeneo		No-Homogeneo	
Pesado	$O(pn)$	Hassin (1991) [80]	$O(p^3 + p^2 logm + pn)$	Gusfield (1992) [61]
No Pesado	$O(pn)$	Hassin (1991) [80]	$O(p^3 + p^2 logm + pn)$	Gusfield (1992) [61]
Repulsivo	$O(n^3)$	Tamir (1997) [164]	$O(pn)$	Hulme (1981) [85]

El problema del p-Centro en el arbol				
	Homogeneo		No-Homogeneo	
Pesado	$O(nlog^2n)$	(1983)[113]+ [17]	$O(pn(p+\log^2 n)$	Tamir (1993) [161]
No Pesado	$O(n)$	Frede. (1990) [51]	$O(n)$	Tamir (1997) [164]
Repulsivo	$O(np)$	Tamir (1991) [159]	NP-*Hard*	Tamir (1991) [159]

El problema de la p-Mediana en el árbol				
	Homogeneo		No-Homogeneo	
Pesado	$O(pn^2)$	Tamir (1996) [163]	$O((p^3+n)logn+np)$	Tamir (1993) [161]
No Pesado	$O(pn^2)$	Tamir (1996) [163]	$O(np)$	Tamir (1991) [159]
Repulsivo	$O(np)$	Tamir (1991) [159]	NP-*Hard*	Tamir (1991) [159]

Basándonos en los resultados de Lowe [108] se han construido el conjunto de los puntos eficientes de un problema multiobjetivo en un árbol. Se han reformulado los modelos clásicos de localización en árboles como problemas de Programación Matemática. Y finalmente se han tratados los algoritmos más eficientes tanto para el problemas del p-Centro restringido como para el problema de la p-Mediana restrigida.

Capítulo 2

HEURÍSTICAS Y ÁRBOLES.

2.1 Introducción.

Muchos problemas de localización, como es el caso de los clásicos problemas del p-Centro y de la p-Mediana son NP-*Hard* y no se espera que sean resueltos de forma exacta por ningún algoritmo que tenga complejidad polinomial. Estos problemas pueden ser analizados desde el punto de vista de la optimización combinatoria llegando a resolverse con técnicas de enumeración, sin embargo estas técnicas no son prácticas en problemas de dimensiones reales, pues no generan una solución óptima en un tiempo razonable.

Es en estos casos en los que deben utilizarse los métodos heurísticos, con el fin de encontrar soluciones satisfactorias que puedan estar en el camino de la solución óptima del problema en cuestión. Un algoritmo heurístico puede definirse como sigue:

> *Una heurística es una técnica para buscar buenas soluciones (es decir, próximas a la solución óptima) en un tiempo computacional razonable renunciando a la garantía de su factibilidad y/o de su optimalidad e incluso, en algunos casos, de lo cerca que están de la solución óptima.*

Un campo donde se han aplicado con éxito los métodos heurísticos son los problemas en redes. Se consideran redes de interés las de carreteras, la de transporte aéreo, las de ríos o las de ordenadores. En un problema de localización en red, la red está representada por un grafo no dirigido, y tanto los nuevos servicios como los existentes están usualmente idealizados como puntos. Así mismo, los

vértices del grafo son generalmente considerados potenciales puntos de demanda (clientes o usuarios).

Un caso especial tiene lugar cuando el grafo en el que se trabaja tiene estructura de árbol (un grafo conexo sin ciclos). Este caso tiene interés, no sólo por su sencillez, sino también por su creciente importancia en el mundo científico, hecho que queda corroborado por la extensa bibliografía existente y la de reciente aparición. La gran eficiencia de los algoritmos para árboles reside primordialmente en la propiedad de convexidad que éstos poseen. Por otra parte, los algoritmos para árboles pueden aplicarse en grafos generales, cuando el problema puede descomponerse en subproblemas sobre árboles, o si el grafo tiene pocos ciclos y el algoritmo puede adaptarse a él. Muchas redes rurales y de regiones periféricas son grafos casi acíclicos.

En este capítulo se realiza un repaso de la historia de los métodos de solución, tanto del problema del p-Centro como de la p-Mediana, obteniendo así, una visión general de los mismos. Además se aborda una nueva vía a la hora de diseñar heurísticas para resolver los problemas de localización en grafos, haciendo uso de los algoritmos construidos para árboles. En esta línea se abarcan las diferentes estrategias heurísticas propuestas, aportando los correspondientes resultados computacionales.

Entre las heurísticas que no utilizan esta alternativa de aprovechar los algoritmos eficientes para árboles hay que resaltar los resultados de la heurística VNDS mostrados en este capítulo. La heurística VNDS ha sido diseñada para resolver problemas de optimización combinatoria en grafos de dimensiones considerable. Esta ha sido probada para el problema de la p-Mediana con grafos del orden de 6000 vértices, mejorando considerablemente los resultados obtenidos con otras heurísticas.

2.1.1 Formulación y Notación.

En este apartado se da un repaso a la notación utilizada en este capítulo ya introducida anteriormente. Por $G(V, E)$ se denota un grafo no dirigido con un conjunto de vértices $V = \{ v_1, ..., v_n\}$ y conjunto de aristas $E = \{e_1, ..., e_m\}$. El conjunto $X = \{x_1, ..., x_p\} \subseteq P(G)$ representa las p nuevas localizaciones en el grafo G.

El problema de localización en consideración, que podemos denominar **problema del p-servicio**, consiste en determinar la mejor ubicación X para p servicios que atenderán a un conjunto de usuarios U. Formalmente se trata de encontrar el conjunto X de p puntos del grafo G que más se aproxima a un con-

junto finito de puntos U dado; generalmente vértices. La aproximación será mejor cuanto menor sean las distancias entre el conjunto X y cada uno de los puntos $u \in U$, determinada por $d(X,u) = \min_{x \in X} d(x,u)$. Por tanto se trata de minimizar una cierta función $f(X)$ obtenida a través de una función globalizadora g de las distancias $d(X,u), u \in U$. El problema se formula de la forma siguiente:

$$\min\{f(X) = g(d(X)) : X \subseteq P(G), |X| = p\} \tag{2.1}$$

donde $d(X) = (d(X,u) : u \in U)$ es el conjunto de distancias de interés.

En estos problemas de localización donde la función f de las ubicaciones de los servicios X a minimizar representa un costo que aumenta con las distancias a los puntos de demanda U, la función globalizadora g es no decreciente en las distancias, es decir, $\forall X_1, X_2 \subseteq G$:

$$d(X_1,u) \leq d(X_2,u), \forall u \in U, \Rightarrow g(d(X_1)) \leq g(d(X_2)).$$

Estos problemas vienen caracterizados por el hecho de que, al ser la función objetivo monótona no decrecientes en la distancia, cada vértice usuario $u \in U$ es asignado al punto de servicio $x \in X$ más cercano.

Casos especiales de este problema son, entre otros, los problemas de la p-Mediana, del p-Centro y del p-Centdian dados, respectivamente por la funciones a minimizar:

- $f_m(X) = \sum_{u \in U} w_u \, d(X,u)$
- $f_c(X) = \max_{u \in U} w'_u \, d(X,u)$
- $f_\lambda(X) = \sum_{u \in U} w_u \, d(X,u) + \max_{u \in U} w'_u \, d(X,u)$

donde los pesos w_u y w'_u vienen dados y van asociados a cada vértice usuario $u \in U$. Estos pesos pueden representar la cantidad de demanda que se produce en cada vértice.

2.1.2 Métodos de solución.

En este apartado se describen los aspectos más importantes de los métodos de solución para los problemas aquí considerados atendiendo a su vertiente histórica y metodológica.

El problema de la p-Mediana.

Los orígenes de este problema se remontan a principios del siglo XVII cuando Fermat lo propuso de la siguiente forma: *dados tres puntos en el plano, encontrar un cuarto punto tal que minimice la suma de las distancias a los tres puntos dados.* Antes de 1610, Torricelli propuso una solución geométrica. En el año 1750 aparece el problema de la p-Mediana con pesos. Weber utilizó estos modelos en 1909 para determinar la localización óptima de una fábrica que atendía a un sólo servicio y tenía dos fuentes de suministro. El problema de la 1-Mediana en el plano es el denominado el problema de Weber.

Hakimi en 1964 [62] probó que las soluciones al problema de la p-Mediana para un conjunto de vértices usuario cualquiera puede encontrarse entre los vértices del grafo. Este resultado permite transformar el problema continuo de la p-Mediana en un problema discreto donde sólo se contempla la localización en los vértices. Por tanto, los principales métodos para resolver el problema de la p-Mediana operan con la matriz de distancia entre vértices y son los mismos para los problemas discretos formulados en grafos que en cualquier otro espacio, como es el caso del problema de la p-Mediana en el plano. El problema de la p-Mediana es NP-*Hard*, Kariv y Hakimi (1979) [90]. Los métodos exactos para resolver el problema de la p-Mediana utilizan principalmente tres tipos de herramientas: árboles de búsqueda, técnicas de programación lineal y dualidad.

Algoritmos exactos que usan **árboles de búsqueda** pueden encontrarse en los trabajos de Jarvinen, Rajala y Sinervo (1972) [87], El-Shaieb (1973) [40] y Khumawala (1972) [92]. Las herramientas fundamentales utilizadas fueron: penalizaciones, acotamiento, generación de soluciones factibles bajo ciertas condiciones, y reglas de ramificación en una enumeración implícita. Otras aportaciones fueron debidas a Christofides y Beasley (1982) [30], y Hanjoul y Peeters (1985) [73]. Los ejemplos que muestran estos trabajos alcanzan grafos de hasta 400 vértices, excepto Beasley (1985) [7] que utilizando un ordenador vectorial, pudo trabajar con grafos de hasta 900 vértices. Referencias de trabajos como éstos y de áreas relacionadas están contenidos en Brandeau y Chiu (1989) [12], Mirchandani y Francis (1990) [124] y Cornuejols, Fisher y Nemhauser (1977) [19].

Entre las primeras contribuciones sobre el uso de técnicas de **programación lineal** están las de ReVelle y Swain (1970) [151]. La más conocida es la relajación lineal sobre las variables enteras, con la que generalmente se obtienen soluciones óptimas enteras, o una cota ajustada de las mismas. Debido al gran número de variables y restricciones de los problemas reales, el uso del Simplex estándar es prohibitivo. Para paliar este problema, ReVelle y otros (1976) [152] reducen el número de filas y columnas del problema original usando un método de descomposición que aprovecha la peculiar estructura del problema (Garfinkel, Neebe y

Rao (1974) [53], Swain (1974) [157], Magnanti y Wong (1981) [109], etc.).

En el uso de la **dualidad** destaca el trabajo de Galvao (1980) [52] quien aporta una heurística para el dual del correspondiente problema de Programación Lineal; su método está relacionado con el hallazgo de Erlenkotter (1978) [45] para un problema muy cercano al de la p-Mediana, el problema de localización simple de plantas o problema de Weber. Cornuejols y otros (1977) [19] y Narula, Ogbu y Samuelson (1977) [132] sugirieron el desarrollo de la dualidad lagrangiana obtenida al aplicar multiplicadores de Lagrange al conjunto de restricciones. Otra alternativa de relajar las restricciones se puede ver en Krarup y Pruzan (1983) [99].

El problema del p-Centro.

El origen de este problema se puede adjudicar a la publicación del artículo de Silvester (1857) [158] en la Quartely Journal of Pure and Applied Mathematics, en el que planteaba un procedimiento para encontrar el menor círculo que contuviera un conjunto de puntos del plano. En 1869 Jordan [88] presenta el problema del vértice Centro. En este problema, el mayor esfuerzo computacional viene por el cálculo de la matriz de distancias (Christofides (1975) [29], Minieka (1977) [121], Handler y Mirchandani (1979) [71], entre otros). El problema del centro absoluto (cuando se pueden seleccionar puntos interiores a las aristas) fue planteado y resuelto por Hakimi en 1964 [62]. El concepto de *centro local* es fundamental para la solución eficiente del problema del centro absoluto y del p-Centro absoluto. Minieka (1970) [120] introdujo el concepto de centro local en el interior de una arista como un punto equidistante de dos puntos de demanda y tal que, en ninguna de las dos direcciones en que puede ser desplazado dicho punto, decrezcan a la vez las distancias a dichos puntos. Con esta noción de centro local se demuestra que el centro absoluto de un grafo se encuentra en un vértice o en un centro local con respecto a dos vértices usuario. Para Hakimi (1965) [63], y Kariv y otros (1979) [89], centro local en una arista es la mejor localización posible dentro de esa arista. En 1980 Minieka [122] presenta un algoritmo de centro absoluto, usando su concepto de centro local, que se puede implementar con la misma complejidad que el de Kariv y Hakimi (1979) [89].

Para el problema del p-Centro absoluto también se puede encontrar una solución óptima en el conjunto de vértices y centros locales. Sin embargo, este conjunto tiene un tamaño mucho mayor que el de los vértices por lo que se ha desarrollado un tipo de procedimiento distinto a los de la p-Mediana. Kariv y Hakimi (1979) [89] han mostrado, el problema del p-Centro también es NP-*Hard*.

Una característica común a los procedimientos eficientes para el problema

del p-Centro es que resuelven una secuencia de problemas de cubrimiento (Teitz y Bartz (1968) [168], Drezner (1984) [35], y Eiselt y Charlesworth (1986) [38]). Christofides y Viola (1971) [28], proponen un procedimiento por medio del cual se resuelve una secuencia de problemas de r-Cubrimiento con incrementos sucesivos del valor de r. La solución de cada problema de r-Cubrimiento tiene dos fases. En la primera, se obtienen todas las soluciones factibles, encontrando todas las regiones que pueden ser alcanzadas por un vértice con un radio r. En la segunda, se calculan todas las soluciones factibles con mínima cardinalidad, resolviendo una serie de problemas de cubrimiento. Handler (1979) [71] trata de aprovechar el hecho de que, para valores moderados de p, estos problemas de cubrimiento sólo afectan a vértices de demanda muy alejados entre sí. Para ello extiende el concepto de centro local a los vértices para sólo considerar como localizaciones posibles aquellos puntos (vértices o puntos interiores) que estén en equilibrio con respecto a los puntos de demanda más alejados. Sin embargo la definición proporcionada en dicho trabajo y que se continua proponiendo (Handler (1990) [124] tiene que ser corregida según se muestra en Moreno (1985) [128] para que el algoritmo de relajación propuesto, y que es el más eficiente de los aparecidos en la literatura, sea correcto. Sin embargo la eficiencia del algoritmo no se ve afectada por esta corrección.

Una muestra del desarrollo existente para ambos problemas, puede verse en Francis, McGinnis y White (1974) [48], en Handler y Mirchandani (1979) [71], en Tansel, Francis y Lowe (1983) [166], en el texto *Facilities Location, Models and Methods* de Love, Morris y Wesolowsky (1988) [105]. En 1990 se edita una colección de trabajos con título: *Discrete Location Theory* de Mirchandani y Francis [124] que realiza un estado del arte de la teoría de localización. Pero el más reciente (1995) es el libro de Drezner [37] titulado *Facility Location* el cual aparte de dar un amplio repaso a la teoría de localización, aporta una vasta bibliografía con más de 1200 referencias.

Métodos Heurísticos.

Dado que ambos problemas son NP-*Hard* las heurísticas son la mejor alternativa a problemas de dimensiones considerables. Muchas de las heurísticas diseñadas para cualquiera de estos problemas tienen características fundamentales que los hacen esencialmente aplicables a cualquier otro problema de localización de p servicios con función objetivo monótona en las distancias. Por ejemplo, aquellas que parten de una partición inicial del conjunto de usuarios que se va iterativamente transformando. Las heurísticas se van a diferenciar en la forma de manipular la partición para conseguir una secuencia de particiones, de tal forma que el valor de la función objetivo se vaya minimizando.

El **método de Adición** o *heurística de Babel* (Kuehn y Hamburger (1963) [100]), empieza localizando un servicio de modo que minimice la función objetivo. A continuación se van añadiendo, uno a uno, los servicios hasta alcanzar los p deseados. El nuevo servicio se elige de forma que la función objetivo en cada paso disminuya lo más posible.

La **heurística alternante** que combina iterativamente fases de localización y asignación fue inicialmente propuesta por Maranzana (1964) [110]. En la primera iteración, se eligen p puntos de servicio, se le asignan los usuarios más cercanos a ellos, y se resuelve el problema de localización simple en cada uno de los p conjuntos que se forman. Entonces el procedimiento es iterado con las nuevas localizaciones de los servicios, hasta que no ocurran más cambios en las asignaciones. En la versión de arranque múltiple de esta heurística, el procedimiento se repite un número determinado de veces partiendo de diferentes conjuntos de p servicios, manteniendo la que mejor solución produzca.

La **heurística de eliminación o sustracción**, también denominada *método Attila o Stingy*, propuesto inicialmente por Feldman, Lehrer y Ray (1966) [46], supone que inicialmente en todos los puntos de demanda se establece un servicio. En cada iteración el servicio que produzca el menor incremento en la función objetivo es eliminado. El proceso se repite hasta que el número de servicios se reduce a p.

Teist y Bart (1968) [168], propusieron la llamada **heurística de intercambio**, en la que inicialmente se parte de p localizaciones posibles para los servicios. A continuación se mueven iterativamente los servicios, uno por uno, reduciendo lo más posible el valor de la función objetivo. Esta operación se conoce como el 1-intercambio, ya que se intercambia un elemento que está en la solución por otro que no lo está. La **heurística de Montecarlo** consiste en generar en cada iteración un 1-intercambio y aceptarlo si se produce una mejora. Estas dos heurísticas son comúnmente utilizadas como un estándar de referencia en la comparación de heurísticas y son las utilizadas en la experiencia computacional descrita al final de este capítulo.

Este tipo de movimiento estándar se ha utilizado en otras heurísticas que se enmarcan dentro de las distintas estrategias metaheurísticas, como las búsquedas Tabú, recocido simulado, algoritmos genéticos, métodos de arranque múltiple y la reciente heurística de entorno variable.

Los métodos **heurísticos de búsqueda Tabú** han sido recientemente considerados por Mladenovic, Moreno y Moreno-Vega (1996) [126] para el problema de la p-Mediana, en los que el movimiento de intercambio ha sido extendido al llamado movimiento de sustitución en cadena. Esto permite escapar con más facilidad de los mínimos locales. Otro procedimiento Tabú para la p-Mediana es

sugerido por VoB (1996) [172], en el cual se discuten también algunas variantes del método de eliminación inversa.

La búsqueda **heurística de arranque múltiple** ha sido analizada en M. Moreno (1996) [127]. Estos métodos constan generalmente de dos fases. En la fase global del procedimiento, se genera una solución de la región factible. En la fase local, se aplica una búsqueda local desde la solución encontrada en la fase global, que acabará en un mínimo local con respecto a la estructura de entorno considerada. Estos dos pasos se reiteran hasta que se satisfaga algún criterio de parada. El mínimo local obtenido con menor valor de la función objetivo es la solución propuesta por el algoritmo. La búsqueda con arranque múltiple combina apropiadamente la exploración del espacio de soluciones con la explotación en el entorno de la solución actual; ahí radica en parte su éxito. Una de las principales cuestiones en este tipo de búsqueda es cuando parar. Siguiendo los trabajos de Boender y otros (1987) [10] y Bruno y otros (1987) [13] pueden obtenerse diversas reglas de parada para la búsqueda de arranque múltiple.

Las estrategias heurísticas del recocido simulado o *simulated annealing* han sido aplicadas con éxito relativo por Aarts y Laarhoven [3] y Chams, Herts y Werra [16], y los **algoritmos genéticos** se han mostrado efectivos al combinarlos con técnicas de paralelización ver Moreno, Roda y Moreno [131]

La utilización de algunas de estas heurísticas en problemas de localización puede encontrarse en Pérez-Brito (1998) [6]

El procedimiento heurístico más efectivo ha sido el de la **búsqueda de entorno variable** denominado *VNS* de Mladenovic (1995) [125][77]. Esta heurística consiste brevemente en la idea siguiente. Dada la solución actual, se generan aleatoriamente un número especificado de soluciones en el primer entorno definido. Si se encuentra una solución mejor en este entorno, se toma ésta como la solución actual. En otro caso se cambia de entorno aumentando el radio del mismo. El algoritmo termina si no se ha encontrado ninguna mejora al llegar al último entorno definido. En la siguiente sección se ofrecen más detalles de la misma incorporando una técnica de descomposición que la hace significativamente más eficiente para obtener la heurística denominada **VNDS** que es la que mejores resultados computacionales ofrece.

2.2 Heurísticas basadas en árboles

Cualquier problema de optimización se puede entender como la minimización de una función objetivo sobre un espacio de búsqueda dotado de cierta estructura de entornos ligada a un conjunto de movimientos posibles. La mayoría de los pro-

cedimientos heurísticos se pueden entender como la aplicación de algún tipo de movimiento o transformación sobre el correspondiente espacio de búsqueda. Los espacios de búsqueda más utilizados para los problemas de localización han sido el espacio de las selecciones de puntos de servicio y las particiones. Los métodos de búsqueda generales, **metaestrategias** o estrategias **meta-heurísticas** consisten en reglas para partir de una solución inicial y realizar una sucesión de movimientos y pueden formularse para distintos conjuntos de movimientos o estructuras de entornos con diversas variantes, y pueden combinarse o integrarse en un sistema experto muy genérico. En esta sección se analiza la consideración del espacio de búsqueda constituido por los subárboles para la optimización de problemas de localización en grafos. Esto permite aprovechar los algoritmos extremadamente eficientes para los problemas sobre árboles y aplicar heurísticas de búsqueda mediante el movimiento basado en intercambio de aristas. Ver Pérez-Brito y otros (1994) [138], Pérez-Brito y otros (1994) [139] y Pérez-Brito (1995) [141]

En los grafos planares sobre los que se establecen los modelos de localización, el número de aristas es del mismo orden que el número de vértices. Todo grafo planar con n vértices y m aristas verifica $m \leq 3n \Leftrightarrow 6$ (Biggs et al. (1976) [8], Tucker (1980) [169]). Especialmente frecuentes son los grafos que son casi árboles porque siendo conexos tienen muy pocas aristas más que un árbol.

Los problema de localización en un grafo en los que se trata de determinar las localizaciones X para p puntos de servicio que minimicen una función $f(X)$ obtenida a través de una función globalizadora g no decreciente en las distancias a los puntos de demanda se pueden formalizar de la forma siguiente. Se denota por $P(G)^p$ a todos los conjuntos de p puntos del grafo G; $P(G)^p = \{X \subseteq P(G) : |X| = p\}$. Sea, para cualquier $X \in P(G)^p$, $f_G(X)$ el valor de la función globalizadora g cuando las distancias están calculadas como la longitud del camino del grafo G más corto; $f_G(X) = g(d_G(X,u) : u \in U)$. El problema de localización de p servicios en G consiste en obtener el conjunto $X^* \in P(G)^p$ tal que:

$$f_G(X^*) = \min_{X \in P(G)^p} f(X).$$

Para estos problemas se verifica que, para cualquier grafo existe un subárbol, cuya solución es también solución del problema en el propio grafo. Ver Pérez-Brito y otros (1995) [141] y Pérez-Brito y otros (1997) [145].

Teorema 2.2.1 *Sea el problema de localización en el grafo G dado por:*

$$\min_{X \in P(G)^p} g(d_G(X,u) : u \in U)$$

donde la función globalizadora g es no decreciente en las distancias. Entonces existe un subárbol T^* del grafo G, tal que la solución óptima de este problema

en T^* es solución óptima del mismo problema en G; es decir,

$$\min_{X \in P(T^*)^p} g(d_{T^*}(X,u) : u \in U) = \min_{X \in P(G)^p} g(d_G(X,u) : u \in U).$$

Si X^* es una solución óptima del problema en G y este árbol T^* es el que contiene a todos los caminos mínimos desde X^* a todos los puntos de U.

Demostración:

Sea $T(G)$ el conjunto de todos los subárboles de G. Al considerar el problema en cada subárbol $T \in T(G)$ entonces $X \in P(T)^p$ y las distancias están calculadas por las longitudes de los caminos mínimos en T denotadas por $d_T(X,u)$. Se tiene que, para cada $T \in T(G)$ y para cualquier conjunto de localizaciones $X \in P(T)^p$ es

$$d_G(X,u) \leq d_T(X,u), \forall u \in U.$$

Si por $f_T(.)$ se denota la función objetivo del problema en T; es decir:
$f_T(X) = g(d_T(X,u) : u \in U)$, se sigue que:

$$f_G(X) \leq f_T(X).$$

En particular, para la solución óptima X^* del problema en G se verifica:
$f_G(X^*) \leq f_T(X^*)$, si $X^* \in P(T)$.

Para cualquier $X \in P(G)^p$, sea $T(X)$ el conjunto de todos los subárboles T que contienen caminos mínimos desde X a todos los puntos de demanda; es decir, tales que: $d_G(X,u) = d_T(X,u)$, $\forall u \in U$. Si se producen empates en las distancias el conjunto de todos los caminos mínimos desde X no es un árbol, sin embargo se obtiene un árbol que contiene desde X a todos los vértices usuario si los empates se resuelve de forma única aunque arbitraria. Si no hay empates, el único árbol de $T(X)$ es el formado por los caminos mínimos desde X a todos los puntos de U.

Se tiene que, para cualquier $X \in P(G)^p$ y para cualquier $T \in T(X)$ es $f_T(X) = f_G(X)$. En particular, para la solución óptima X^*, si $T \in T(X^*)$, se verifica:

$$f_G(X^*) = f_T(X^*).$$

Luego $\forall T \in T(G)$ y $\forall X \in P(T)^p$ se tiene que $\forall T^* \in T(X^*)$:

$$f_{T^*}(X^*) \leq f_T(X).$$

Por f_G^* se denota el valor óptimo de la función objetivo del problema en G;

$$f_G^* = \min_{X \in P(G)^p} g(d_G(X,u) : u \in U).$$

Entonces se tiene, $\forall T \in T(G)$:

$$f^*_{T^*} \leq f_{T^*}(X^*) \leq f^*_T$$

de donde

$$f^*_{T^*} = \min_{T \in T(G)} f^*_T = f^*_G.$$

Luego el valor óptimo para el problema en T^* es el mismo que para el problema en G.

Finalmente X^* es también solución óptima del problema en T^*, pues, para cualquier $X \in T^*$ se tiene:

$$f_{T^*}(X^*) \leq f_G(X^*) \leq f_G(X) \leq f_{T^*}(X).$$

Recuérdese que T^* es un árbol con los caminos mínimos desde X^* a los puntos de U pero también es el árbol para el que f^*_T es mínimo;

$$f^*_{T^*} = \min_{T \in T(G)} f^*_T.$$

□

De este resultado se deduce que, en tales problemas de localización sobre un grafo, existe un subárbol en el que la solución óptima del problema coincide con la del problema en el grafo completo. Este árbol es precisamente en el que el problema da el mejor valor de la función objetivo; es decir el árbol óptimo. Se propone como procedimiento para resolver el problema del grafo general encontrar el subárbol óptimo y resolver el problema en él.

Una heurística general para este procedimiento consiste en partir de un árbol inicial, que se va modificando sucesivamente en busca del árbol óptimo. La modificación del árbol T_i para obtener T_{i+1} se puede realizar por cualquier procedimiento que ejecute una transformación elemental.

Este tipo de búsqueda consistente en partir de un árbol generador inicial T_0, y modificarlo mediante una sucesión de transformaciones elementales ha sido propuesta por Mayeda (1972) [111] y Chen (1971) [27]. Este método de búsqueda se adopta en muchas estrategias heurísticas, como las ya mencionadas: búsqueda de Montecarlo, Algoritmos Genéticos, métodos Multiarranque, búsquedas Tabú, Recocido Simulado, y Búsquedas Voraces (o Greedy).

El árbol inicial T_0, puede obtenerse por procedimientos del tipo del algoritmo de Prim (1957) [148] o del algoritmo de Dijkstra (1959) [34]; es decir, mecanismos que parten de un subgrafo sin aristas y se van añadiendo aristas evitando formar ciclos hasta alcanzar todo el conjunto U.

Para explorar árboles del entorno pueden usarse cualquier tipo de transformación que garantice un recorrido eficiente del espacio de árboles, en particular alguna de las dos transformaciones elementales siguientes:

Primera transformación: *Add-Drop*

1. Se añade una arista del grafo G que no esté en el árbol.
2. Se determina el ciclo formado al incorporar esta arista al árbol.
3. Se elimina del árbol una arista de dicho ciclo.

Segunda transformación: *Drop-Add*

1. Se elimina una arista del árbol T_i.
2. Se determinan las dos componentes conexas en que se divide el árbol.
3. Se añade una arista que conecte dichas componentes.

El procedimiento heurístico se describe especificando la estrategia, es decir, la forma en que se implementan los cuatro elementos del proceso de búsqueda general: 1) la forma de obtener el árbol inicial, 2) el conjunto de transformaciones posibles, 3) el método de elección de la transformación, y 4) el criterio de parada. Este esquema tiene un gran abanico de posibilidades que permiten incorporar conocimientos heurísticos, puesto que nos podemos plantear distintos diseños al decidir sobre:

- Cómo generar el árbol inicial.
- Cómo ejecutar la transformación elemental.
- Cómo seleccionar la transformación.
- Cómo diseñar el criterio de parada.

De los resultados anteriores se derivan dos condiciones que debe cumplir el árbol óptimo. Por un lado, que el árbol con los caminos mínimos en el grafo desde la solución óptima del problema a todos los usuarios vuelva a ser el mismo árbol y, por otro lado, que el valor de la función objetivo del problema, tanto en el grafo, como en el propio árbol, sea mínimo entre todos los posibles árboles.

Una primera aproximación a las heurísticas denominadas **heurísticas árbol** se encuentra en la Heurística AG (ver figura 1. Heurística AG).

Heurística AG

Paso 0. (Inicialización):

Sea T_0 un árbol inicial de G. Tomar $i \leftarrow 0$.

Paso 1. (Localización):

Sea $X_i = \{x_i^1, ..., x_i^p\}$ la localización óptima de los p servicios en T_i.

Paso 2. (Asignación):

Sea T_{i+1} un árbol con los caminos mínimos desde X_i a los puntos de U.

Paso 3. (Terminación):

Si $T_{i+1} = T_i$ parar. En otro caso, hacer $i \leftarrow i + 1$ y volver al paso 1.

Figura 1. **Heurística AG**

En el Paso 0, el árbol inicial se puede obtener al aplicar el algoritmo de Prim, el de Kruskal, el de Dijkstra u otro cualquiera que de como resultado un árbol. Una selección inteligente de este primer árbol puede contribuir a mejorar la eficiencia del procedimiento; por ejemplo, utilizando un árbol de caminos mínimos de cualquier solución dada por una heurística muy rápida.

En el Paso 1 se aplica el algoritmo eficiente de la p-Mediana [163], el p-Centro [118] o el p-Centdian en árboles [165] si se quiere resolver el problema de la p-Mediana, el p-Centro o el p-Centdian en el grafo, respectivamente. Además, en el paso 2, el algoritmo de Dijkstra proporciona el árbol de caminos mínimos desde la solución dada por uno de los anteriores algoritmos.

2.3 Estrategias de Búsqueda.

A continuación se describen las estrategias de búsqueda más importantes para determinar el árbol óptimo en el que poder determinar la solución óptima del problema de localización en el grafo. Ver Pérez-Brito y otros (1992) [137] y Pérez-Brito (1996) [140]

2.3.1 Búsquedas al azar.

Existe un grupo importante de estrategias que tienen fundamentalmente una componente aleatoria y que se engloban dentro de las denominadas búsquedas al azar.

a) Búsqueda por muestreo al azar.

Una estrategia de búsqueda primitiva, denominada de prueba y error, de Montecarlo o simplemente **búsqueda aleatoria** es la consistente en generar árboles al azar y seleccionar el mejor (Andersen (1972) [4]. La búsqueda es **aleatoria pura** o uniforme si la distribución de probabilidad en el espacio de los árboles es la uniforme o equiprobable. En caso de que la distribución de probabilidad sea cualquier otra, es una búsqueda por muestreo aleatorio.

b) Búsqueda por recorrido al azar.

Frecuentemente el término *búsqueda* se aplica exclusivamente a los procedimientos que recorren el espacio de búsqueda usando movimientos o transformaciones para pasar de un árbol al siguiente. La **búsqueda (por recorrido) al azar** consiste en, partiendo de un árbol inicial, seleccionar iterativamente al azar una transformación elemental para transformar el árbol actual e ir guardando la mejor solución encontrada.

2.3.2 Búsquedas Monótonas.

Las estrategias de búsquedas al azar no utilizan la información proporcionada por los árboles encontrados. Esta información se puede incorporar al método de búsqueda para guiar las transformaciones elementales aplicadas. Las estrategias pueden ser principalmente de dos tipos: monótonas o no monótonas. Una estrategia monótona, también llamada escaladora de colinas (*hill-climbing*), miope o **greedy**, es la que no admite transformaciones que den lugar a una solución peor. Con búsquedas monótonas sólo realizarían transformaciones elementales del árbol actual si con ello se mejora la solución. Una estrategia no monótona admite, aunque con restricciones o en determinadas circunstancias, transformaciones desfavorables.

a) Búsqueda aleatoria monótona.

Una búsqueda aleatoria monótona se obtiene a partir de un recorrido aleatorio introduciendo una condición para que sólo acepte mejoras. En la **Heurística AG-Azar** (ver Figura 2.) se utiliza una estrategia aleatoria monótona. La transformación aplicada consiste en introducir aleatoriamente una arista entre las

posibles candidatas y del ciclo que se forma, se elimina otra, también seleccionada aleatoriamente. Si la modificación no mejora el árbol actual, se rechaza. Entre las posibles reglas de parada para realizar los estudios computacionales para realizar comparaciones se ha optado por fijar el número máximo de iteraciones.

Heurística AG-Azar.

Paso 0. (Inicialización):

Sea T_0 un árbol inicial de G. Sea k el máximo número de iteraciones. Tomar $i \leftarrow 0$.

Paso 1. (Localización):

Sea X_i la localización óptima de los p servicios en el árbol T_i.

Paso 2. (Transformación):

Determinar al azar una transformación elemental T_{i+1} del árbol T_i.

Paso 3. (Terminación):

Si $i \geq k$, parar. En otro caso, hacer $i \leftarrow i + 1$ y volver al paso 1.

Figura 2. **Heurística AG-Azar**

b) Búsqueda Greedy parcial.

La búsqueda se puede intensificar en torno a cada árbol actual seleccionando la mejor entre una serie de transformaciones elementales. (Charon and Hudry (1993) [24]). Una búsqueda de tipo *Greedy* es la que realiza siempre la mejor transformación de entre todas las posibles, realizando una selección exhaustiva en el entorno del árbol actual. Si la búsqueda en el entorno del árbol actual se hace de forma que las transformaciones elementales entre las que se selecciona la mejor sean distintas se tiene una búsqueda monótona localmente intensa o greedy parcial. Esta búsqueda se puede implementar de cualquier forma que garantice que las transformaciones sean distintas. En la heurística AG-GP (ver figura 3. Heurística AG-GP) se introduce aleatoriamente la arista que entra y se elige la que se elimina por un procedimiento de tipo *Greedy*. Al igual que la heurística AG-Azar, la regla de parada aplicada viene dada por el número máximo de iteraciones.

Heurística AG-GP.

Paso 0. (Inicialización):

Sea T_0 un árbol inicial de G, y k el máximo número de iteraciones. Tomar $i \leftarrow 0$.

Paso 1. (Localización):

Sea X_i la localización óptima de los p servicio en el árbol T_i.

Paso 2. (Transformación):

Determinar al azar una arista e_i que no esté en T_i.

Hallar la transformación óptima en T_i, para obtener el árbol T_{i+1} que contenga a e_i.

Paso 3. (Terminación):

Si $i \geq k$, parar. En otro caso, $i \leftarrow i + 1$ y volver al paso 1.

Figura 3. **Heurística AG-GP**

c) Búsqueda Greedy.

La clásica búsqueda localmente exhaustiva o heurística greedy pura se obtiene seleccionando de manera exhaustiva la mejor transformación del árbol actual. Para ello en el paso 2 de la heurística AG-G (ver figura 4. Heurística AG-G) se analizan todas las posibilidades de cambiar una arista del árbol por otra que no lo sea. La heurística termina cuando se ha alcanzado el número máximo de iteraciones.

Heurística AG-G.

Paso 0. (Inicialización):

Sea T_0 un árbol inicial de G, y k el máximo número de iteraciones. Tomar $j \leftarrow 0$.

Paso 1. (Localización en un árbol):

Sea X_j la localización óptima de los p servicio en el árbol T_j.

Paso 2. (Movimiento en el entorno de soluciones):

Determinar la mejor modificación T_{j+1} del árbol T_j.

Paso 3. (Terminación):

Si $j \geq k$ parar. En en otro caso, $j \leftarrow j + 1$ y volver al paso 1.

Figura 4. **Heurística AG-G**

2.3.3 Búsquedas No Monótonas.

El inconveniente de las búsquedas monótonas es que si se aproximan a un árbol localmente óptimo (es decir, un árbol para el que ninguna transformación de las contempladas produce una mejora) el procedimiento queda atrapado en él. Las tres formas principales de escapar de esta situación consisten en volver a comenzar la búsqueda desde otro árbol inicial, permitir de forma controlada movimientos que den lugar al empeoramiento del árbol actual, o modificar el conjunto de transformaciones contempladas.

a) Búsquedas de Arranque Múltiple.

Los procedimientos de arranque múltiple (MultiStart) realizan varias búsquedas monótonas partiendo de diferentes árboles iniciales. Una forma simple de realizar esto consiste en generar una muestra de árboles de arranque (Rinnooy Kan and Timmer (1987) [153]), o equivalentemente generar al azar un nuevo árbol de partida cada vez que la búsqueda quede estancada. El criterio de parada se convierte entonces en un criterio para reiniciar la búsqueda por lo que hay que incorporar un nuevo criterio de finalización de todo el proceso de búsqueda.

Las herramientas para mejorar la generación totalmente al azar de los árboles iniciales persiguen que los recorridos de las sucesivas búsquedas se distribuyan lo más posible en todo el espacio de posibles árboles. Una posibilidad consiste en seleccionar una muestra representativa de un conjunto más grande de árboles iniciales procurando que éstos sean lo más diferentes posible. También se puede tener en cuenta por qué parte del espacio de búsqueda han transcurrido los recorridos ya realizados para no insistir en zonas exploradas.

b) Búsquedas por Recorrido no monótonos.

Otra de las técnicas para evitar quedarse atrapados en un árbol óptimo local, es la de admitir la posibilidad de dar pasos hacia atrás. Sin embargo, hay que controlar de alguna forma estas transformaciones para que, al menos a la larga, se vayan mejorando los árboles. Esto se realiza principalmente por medio de dos herramientas: la primera, asignando distintas probabilidades a las transformaciones elementales en función de la mejora que produzcan, y la segunda, utilizando la memoria histórica del recorrido para evitar la reincidencia en los mismos árboles.

La **Búsqueda Probabilística** se obtiene utilizando una probabilidad no nula de aceptar el nuevo árbol aún cuando éste no mejore al actual. Esta probabilidad suele determinarse teniendo en cuenta el valor de la función objetivo en ambos árboles. Los parámetros de esta función de aceptación se pueden establecer estáticamente o modificarse dinámicamente para regular la diversificación e

intensificación en la búsqueda.

La heurística AG-GP-P que aplica esta estrategia se diferencia sólo en el paso 2 de la heurística AG-GP, descrita anteriormente. En este caso, la arista se introduce cuando mejora la función objetivo, y si esto no ocurre cuando supera una determinada probabilidad fijada de antemano.

Una de las funciones de aceptación más famosa es la emanada del Recocido Simulado o **Simulated Annealing** (Laarhoven and Aarts (1987) [102], Vidal (1993) [170]). Básicamente, se trata de usar una probabilidad de aceptación de empeoramientos que es función exponencial de la modificación de la función objetivo. Se utiliza un parámetro de control c o temperatura para modular la intensidad de la búsqueda que se va modificando de acuerdo con cierto plan de enfriamiento que trata de garantizar con probabilidad 1 la convergencia hacia el árbol óptimo.

Una estrategia de **búsqueda con memoria**, como la búsqueda Tabú, trata de utilizar la memoria del proceso de búsqueda para evitar la reiteración en una misma zona de búsqueda (Glover (1989) [56], Glover et al. (1993) [57]). La forma más elemental de introducir la memoria en el procedimiento de búsqueda no monótona es considerar una función de aceptación binaria (tomando sólo los valores 0 y 1) que tenga en cuenta la historia de la búsqueda. La heurística AG-GP-T que emplea esta estrategia se deriva de la heurística AG-GP al añadir en el paso 2 una lista tabú para la arista que es candidata a entrar.

c) Búsquedas de Entorno Variable.

La técnica heurística de propósito general que se ha llegado a aplicar con éxito más reciente es la denominada *VNS*, (*Variable Neighbourhood Search*) Mladenovic (1995) [125]. La idea principal consiste en modificar la estructura de entorno al atascarse la búsqueda en el entorno de un óptimo local. La forma más elemental de aplicar esta estrategia se obtiene al considerar una serie de entornos definidos de forma anidada, fundamentalmente usando explícitamente o implícitamente un concepto de distancia que asocie un radio diferente a las diferentes estructuras de entorno. Este tipo de estrategia se ha utilizado con éxito en combinación con un proceso de descomposición para el problema de la p-Mediana sobre el espacio de soluciones explícitas.

Para aplicar esta estrategia a las búsquedas basadas en árboles se consideran las transformaciones del árbol consistentes en reemplazar varias aristas a la vez. Si se denota por k el número de aristas a sustituir la transformación se denomina k-intercambio. La estrategia consiste en incrementar el valor de k cuando las transformaciones del k-intercambio no están suficientemente lejanas para escapar del árbol óptimo local (fase de diversificación), mientras que cuando se consiga

eludir esta situación se vuelva a tomar $k = 1$ (fase de intensificación).

2.3.4 El árbol inicial.

En el paso 0 de inicialización de los anteriores procedimientos es importante aplicar algoritmos apropiados para proporcionar un árbol inicial. Las características importantes de estos algoritmos son su eficiencia, el grado de aleatoriedad y la calidad de la solución asociada al árbol. Se pueden contemplar diversas alternativas entre las que se han seleccionado las siguientes: a) el árbol de longitud mínima, obtenido por el clásico algoritmo de Prim, b) un árbol obtenido al azar por una adaptación del algoritmo de Prim, y c) un árbol paramétrico que constituye una combinación regulable de los anteriores. Se describen con más detalle estos algoritmos.

a) **Árbol mínimo.**

El árbol de longitud total mínima se obtiene aplicando al algoritmo de Prim. Partiendo de un vértice usuario cualquiera como subgrafo árbol de longitud nula, en cada iteración del algoritmo se conecta al árbol un nuevo vértice usuario por medio del camino de menor longitud; si todos los vértices son usuarios, se trata de elegir la arista más corta incidente en un sólo vértice del árbol. Se van incorporando los vértices usuario en función de unos niveles que se actualizan en cada iteración hasta que estén conectados todos los vértices usuario. El nivel de cada vértice usuario representa la longitud mínima del camino que lo enlaza al árbol y el enlace en el árbol es el vértice al que lo une dicho camino (ver figura 5. Algoritmo de Prim).

Algoritmo de Prim

Paso 1. Elegir $v_1 \in V$ cualquiera.
Poner $Nivel[v] := \infty$, $\forall v \in V$.
$V_T = \{v_1\}$, $E_T := \emptyset$.
$Nivel[v_1] := 0$, $i := 1$.
Paso 2. Para cada $v \notin V_T$:
Si $[v_i, v] \in E$ entonces:
Si $Long[v_i, v] < Nivel[v]$ entonces:
$Enlace[v] := v_i$, $Nivel[v] := Long[v_i, v]$.
Paso 3. Hacer $i := i + 1$.
Sea $NivelMin := \infty$.
Para cada $v \notin V_T$.
Si $Nivel[v] < NivelMin$ entonces:
$v_i := v$.
$NivelMin := Nivel[v]$.
Paso 4. Hacer $V_T := V_T \cup \{v_i\}$ y $E_T = E_T \cup \{[Enlace[v_i], v_i]\}$.
Si $V_T \neq V$ entonces volver al paso 2.

Figura 5. **Algoritmo de Prim**

b) **árbol aleatorio.**

Una generación al azar del árbol de partida se realiza con una adaptación del algoritmo de Prim. La modificación consiste en que el vértice usuario que se conecta al árbol en cada iteración se selecciona al azar (ver figura 6. Algoritmo de Prim-Rand). Sin embargo, se sigue utilizando la forma más corta de conectar cada vértice incorporado.

Algoritmo de Prim-Rand

Paso 1. Elegir $v_1 \in V$ cualquiera.
Poner $Nivel[v] := \infty$, $\forall v \in V$.
$V_T = \{v_1\}$, $E_T := \emptyset$.
$Nivel[v_1] := 0$, $i := 1$.
Paso 2. Para cada $v \notin V_T$:
Si $[v_i, v] \in E$ entonces:
Si $Long[v_i, v] < Nivel[v]$ entonces:
$Enlace[v] := v_i$, $Nivel[v] := Long[v_i, v]$
Paso 3. Hacer $i := i + 1$.
Sea $v \notin V_T$ con $Nivel[v] < \infty$ elegido al azar
$v_i := v$
Paso 4. Hacer $V_T := V_T \cup \{v_i\}$ y $E_T = E_T \cup \{[Enlace[v_i], v_i]\}$.
Si $V_T \neq V$ volver al paso 2.

Figura 6. **Algoritmo de Prim-Rand**

c) **árbol paramétrico.**

Con objeto de conseguir un árbol generador al azar que no sea precisamente el de longitud mínima, pero cuya diferencia de éste pueda ser controlada, se introducen dos parámetros α y β en el algoritmo de Prim. Los parámetros α y β nos permiten controlar respectivamente, las probabilidades de que se incorpore al árbol el vértice usuario de menor nivel y de que se enlace al árbol con el camino de menor longitud (ver figura 7. Algoritmo de Prim$_{\alpha,\beta}$). El parámetro α fija la probabilidad con la que se modifica el enlace de un vértice si el enlace al vértice que se conecta mejora el existente. El parámetro β establece la probabilidad de que se seleccione el vértice con mejor enlace. Ver Pérez-Brito y otros (1994) [140]

Algoritmo de Prim$_{\alpha,\beta}$

Paso 1. Elegir $v_1 \in V$ cualquiera.
Poner $Nivel[v] := \infty$, $\forall v \in V$.
$V_T = \{v_1\}$, $E_T := \emptyset$.
$Nivel[v_1] := 0$, $i := 1$.
Paso 2. Para cada $v \notin V_T$:
Si $[v_i, v] \in E$ entonces:
Si $Long[v_i, v] < Nivel[v]$ entonces:
Si $(Random < \alpha)$ o $(Nivel[v] = \infty)$ entonces:
$Enlace[v] := v_i$, $Nivel[v] := Long[v_i, v]$
Paso 3. Hacer $i := i + 1$.
Sea $NivelMin := \infty$
Para cada $v \notin V_T$
Si $Nivel[v] < NivelMin$ entonces:
Si $(Random < \beta)$ o $(NivelMin = \infty)$ entonces:
$v_i := v$, $NivelMin := Nivel[v]$
Paso 4. Hacer $V_T := V_T \cup \{v_i\}$ y $E_T = E_T \cup \{[Enlace[v_i], v_i]\}$.
Si $V_T \neq V$ volver al paso 2.

Figura 7. **Algoritmo de Prim$_{\alpha,\beta}$**

2.3.5 Heurísticas del p-bosque.

Un procedimiento heurístico diferente propuesto consiste en reemplazar el subárbol T_i por p subárboles T_i^j, $j = 1..p$ para plantear el problema simple en cada uno de ellos. A este conjunto de p subárboles se le denomina p-**bosque** y se puede construir eliminando $p \Leftrightarrow 1$ aristas del árbol T_i consiguiendo así el bosque F_i, ver Pérez-Brito y otros (1996) [142]. Se asigna cada usuario al servicio más cercano con respecto a G, obteniéndose el bosque F_{i+1}. Si a F_{i+1} se le añaden $p \Leftrightarrow 1$ aristas sin formar ciclo, se obtiene el correspondiente árbol T_{i+1}.

Las dos operaciones elementales para obtener el bosque F_{i+1} a partir del bosque F_i son las siguientes:

Primera transformación: *Add-Drop*

1. Se añade una arista que no esté en el bosque F_i.
2. Pueden ocurrir dos cosas; que los extremos de la arista estén en un mismo árbol T_i^j o que conecte dos subárboles T_i^j y T_i^k.
   (a) Si la arista forma un ciclo en el subárbol T_i^j entonces se elimina una arista de dicho ciclo.
   (b) Si la arista conecta dos subárboles T_i^j y T_i^k entonces se elimina una arista cualquiera.

Segunda transformación: *Drop-Add*

1. Se elimina una arista del bosque F_i.
2. Se determinan las $p+1$ componentes conexas que se forman.
3. Se añade una arista que conecte dos de dichas componentes.

Heurística AG-bosque

Paso 0. (Inicialización):

Sea F_0 un bosque inicial de G. Tomar $i \leftarrow 0$.

Paso 1. (Localización):

Sea $X_i = \{x_i^1, ..., x_i^p\}$ la localización óptima de los p servicios en F_i, obtenida de resolver el problema de un servicio en cada árbol T_i^j, $j = 1, ..., p$.

Paso 2. (Asignación):

Sea, para cada $j = 1, ..., p$, el conjunto U_i^j con los puntos de demanda asignados a la localización $x_i^j \in X_i$. Sea $T_{i+1}^j \in T(\{x_i^j\}, U_i^j)$ y

$$F_{i+1} = T_{i+1}^1 \cup \cdots \cup T_{i+1}^p.$$

Paso 3. (Terminación):

Si $F_{i+1} = F_i$ parar. En otro caso, hacer $i \leftarrow i+1$ y volver al paso 1.

Figura 8. **Heurística AG-bosque**

2.3.6 Experiencia computacional.

Con la experiencia computacional se pretende conseguir dos objetivos, el primero es comprobar la eficiencia de las heurísticas de búsqueda, y el segundo es corroborar la utilidad del árbol y del bosque como herramientas algoritmicas para la resolución de problemas de localización en grafos.

Para la experimentación se consideran modelos estándares sobre grafos planares conexos, generados aleatoriamente. Denotamos por $G(n,m)$ a los grafo de n vértices y m aristas. Cada grafo de este modelo se obtiene seleccionando al azar n puntos de un cuadrado unidad, usando el generador del Turbo Pascal. Los segmentos rectilíneos que unen los n puntos son las aristas con su longitud real. Se obtiene un árbol generador mínimo y del resto se van tomando las más cortas sin cruzar las anteriores hasta complementar el número m de aristas. Los pesos de los vértices se generaron también independientemente con distribución uniforme en el intervalo unidad $[0,1]$.

Las heurísticas AG, AG-G y AG-GP presentan el grave inconveniente de que, al no incorporar ninguna componente aleatoria, pueden quedarse rápidamente estancadas en un mínimo local. En la tabla que se muestra a continuación se reflejan los resultados experimentales obtenidos con las heurísticas AG-Azar, AG-GP, AG-GP-T y AG-GP-P partiendo de los tres árboles iniciales anteriormente descritos para el problema del 1-λ-Centdian. En dicha tabla se exponen los porcentajes de aciertos y los tiempos medios en segundos, obtenidos al aplicar estas heurísticas a 100 grafos por modelo. Las heurísticas han sido implementadas en Turbo Pascal 7.0.

	AG-Azar		AG-GP		AG-GP-T		AG-GP-P	
Prim	%	CPU	%	CPU	%	CPU	%	CPU
$G(25,30)$	87%	2.06	87%	2.84	87%	2.36	88%	1.54
$G(25,35)$	72%	1.66	72%	2.74	72%	2.30	74%	1.50
$G(25,40)$	55%	1.77	53%	2.61	55%	2.77	59%	1.82
$G(25,45)$	28%	1.84	28%	2.78	28%	3.28	35%	2.22
$G(25,50)$	34%	1.76	30%	2.74	33%	2.41	35%	1.34

Resultados de las heurísticas.

	AG-Azar		AG-GP		AG-GP-T		AG-GP-P	
Prim-Rand	%	CPU	%	CPU	%	CPU	%	CPU
$G(25,30)$	85%	1.36	84%	2.89	85%	3.05	89%	1.84
$G(25,35)$	68%	2.24	69%	2.72	71%	3.04	75%	2.37
$G(25,40)$	39%	2.05	44%	2.97	49%	3.60	41%	1.61
$G(25,45)$	51%	1.81	46%	3.37	42%	3.26	46%	1.14
$G(25,50)$	25%	1.83	30%	2.69	24%	2.58	33%	1.49

Resultados de las heurísticas.

	AG-Azar		AG-GP		AG-GP-T		AG-GP-P	
Prim$_{\alpha,\beta}$	%	CPU	%	CPU	%	CPU	%	CPU
$G(25,30)$	90%	1.53	86%	2.70	86%	2.85	88%	1.70
$G(25,35)$	67%	1.71	70%	2.65	65%	2.86	70%	1.61
$G(25,40)$	62%	2.06	55%	2.93	52%	2.50	51%	1.86
$G(25,45)$	42%	1.38	40%	2.71	44%	2.87	44%	1.44
$G(25,50)$	26%	1.73	20%	3.27	23%	3.05	31%	1.84

Resultados de las heurísticas.

De la tabla se deduce que no hay diferencias significativas ni entre las diferentes heurísticas, ni entre los diferentes árboles iniciales. Los resultados no son del todos satisfactorios debido a que es un modelo sencillo, en el que la utilidad de la heurística no está muy justificada y presenta dificultades estructurales.

La siguiente experiencia computacional se ha realizado aplicando la heurística H2CDHG (ver apéndice A) a el problema del 2-λ-Centdian en un grafo. La heurística parte de un árbol inicial obtenido por el algoritmo de Prim-p, que consiste en crear un lista vacía y en cada iteración se van introduciendo las aristas candidatas a formar parte del árbol generador, de forma que elegir una de ellas, sea siempre equiprobable. Para explorar los árboles vecinos se utiliza la transformación Add-Drop en el que hay que seleccionar la arista que entra y la que la sale del árbol. Para seleccionar la que entra se emplea un procedimiento ansioso, es decir, se introduce la primera arista que produzca una mejora en la función objetivo. Y para elegir la arista que sale se utiliza un procedimiento Greedy, es decir, se saca la arista del ciclo que se produce, que mayor valor ocaciona a la función objetivo. La peculiaridad del procedimiento consiste en no explorar más de una vez el mismo entorno del árbol, es decir, si el procedimiento ansioso a encontrado la primera mejora en la arista que está en la posición quinta, en la siguiente iteración continua la búsqueda en la posición sesta y no en la primera, como suele hacerse normalemente en esto procedimientos. Esto permite un ahorro considerable de tiempo de ejecución. En cada árbol se resuelve el problema del 2-λ-Centdian por medio del algoritmo 2CDT expuesto en la sección 4.8 del capítulo 4. La heurística finaliza cuando ha complementado un ciclo, es

decir, se ha realizado un cambio de árbol sin obtener ninguna mejora o cuando sobrepasa el tiempo máximo permitido.

En las siguientes tablas se refleja los resultados obtenidos al ejecutar la heurística H2CDHG a el problema del 2-λ-Centdian en un grafo. Los grafos son obtenidos de la misma manera que en la experiencia computacional anterior. Cada columna de las tablas muestra el valor de la función objetivo y el tiempo de CPU empleado en un grafo. Las filas representan el número de veces que se ha ejecutado la heurística en dicho grafo.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1221.20	2.75	1560.80	1.76	1354.00	1.43	1664.80	2.64	1418.00	1.48
1221.20	1.59	1560.80	1.98	1354.00	1.65	1664.80	2.58	1418.00	2.25
1221.20	1.98	1560.80	1.42	1354.00	1.59	1664.80	2.30	1418.00	1.65
1221.20	1.42	1560.80	3.41	1354.00	1.15	1664.80	2.09	1418.00	1.37
1221.20	2.75	1560.80	1.48	1354.00	0.66	1664.80	2.69	1418.00	1.32
1221.20	4.06	1560.80	2.04	1354.00	2.20	1664.80	1.59	1418.00	1.27
1221.20	2.42	1560.80	1.48	1354.00	1.37	1664.80	2.03	1418.00	1.48
1221.20	2.42	1560.80	1.87	1354.00	1.43	1664.80	2.25	1418.00	1.59
1221.20	2.19	1560.80	1.15	1354.00	0.93	1664.80	1.27	1418.00	1.70
1221.20	2.53	1560.80	0.93	1354.00	1.48	1664.80	2.58	1418.00	1.93

5 grafos de 25 vértices y 30 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1634.40	2.58	1256.00	1.21	1345.60	1.87	1323.40	1.32	1255.40	1.92
1634.40	2.63	1256.00	1.54	1345.60	1.76	1323.40	1.15	1255.40	1.70
1634.40	2.04	1256.00	1.65	1345.60	4.61	1323.40	0.77	1255.40	1.65
1634.40	1.86	1256.00	1.59	1345.60	2.09	1323.40	1.38	1255.40	1.87
1634.40	2.04	1256.00	1.21	1345.60	1.76	1323.40	1.75	1255.40	1.15
1634.40	1.42	1256.00	1.75	1345.60	1.20	1323.40	1.43	1255.40	2.09
1634.40	1.54	1256.00	1.21	1345.60	3.90	1323.40	1.05	1255.40	1.97
1634.40	1.49	1256.00	1.71	1345.60	1.82	1323.40	1.15	1255.40	2.04
1634.40	1.86	1256.00	0.87	1345.60	1.75	1323.40	1.26	1255.40	1.97
1634.40	0.99	1256.00	1.38	1345.60	1.32	1323.40	1.16	1255.40	1.98

5 grafos de 25 vértices y 30 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1333.00	18.95	1239.60	49.05	1360.80	5.55	1466.80	8.24	1384.00	3.84
1333.00	4.01	1227.20	18.73	1380.80	14.39	1466.80	21.14	1384.00	20.10
1333.00	13.29	1227.20	35.98	1360.80	15.44	1466.80	5.55	1384.00	4.01
1426.40	24.22	1239.60	4.89	1380.80	5.44	1455.20	25.54	1384.00	6.53
1333.00	6.48	1227.20	4.06	1380.80	4.40	1466.80	5.98	1384.00	4.94
1426.40	5.11	1239.60	14.77	1360.80	5.82	1466.80	4.94	1384.00	24.00
1333.00	19.23	1208.80	33.84	1360.80	6.92	1407.20	17.96	1419.20	5.16
1360.80	4.45	1239.60	11.97	1360.80	4.72	1407.20	5.39	1384.00	10.06
1333.00	11.92	1239.60	6.37	1380.80	5.16	1466.80	23.07	1384.00	4.73
1333.00	7.45	1215.60	42.35	1360.80	4.72	1466.80	5.60	1349.60	9.17

5 grafos de 25 vértices y 40 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1042.40	5.11	1042.40	22.58	1014.40	9.45	1590.00	14.17	1060.40	12.85
1042.40	32.68	989.60	13.35	1014.40	7.85	1614.20	5.44	1060.40	14.45
1042.40	16.80	989.60	37.08	1014.40	6.32	1628.00	6.70	1072.40	4.94
989.60	45.47	989.60	6.20	1014.40	7.63	1590.00	5.82	1072.40	33.01
989.60	10.76	989.60	6.32	1014.40	7.63	1614.20	7.25	1072.40	11.98
1042.40	12.41	1042.40	32.24	1014.40	7.86	1628.00	23.13	1072.40	22.19
989.60	32.63	1042.40	3.79	1014.40	7.80	1628.00	5.71	1072.40	6.75
1042.40	5.82	989.60	11.92	1014.40	8.29	1590.00	6.15	1072.40	16.48
989.60	6.97	1042.40	11.48	1014.40	7.80	1628.00	6.26	1072.40	9.94
989.60	8.85	989.60	5.49	1014.40	8.62	1590.00	6.15	1060.40	57.90

5 grafos de 25 vértices y 40 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1513.20	34.99	1141.20	17.30	1235.20	65.97	997.60	10.60	1050.80	22.74
1492.80	38.78	1118.40	26.91	1187.60	22.41	1048.40	33.67	1050.80	36.69
1472.40	52.34	1118.40	13.02	1296.40	24.49	1048.40	19.94	1054.00	74.48
1498.80	13.45	1141.20	15.21	1264.00	23.89	1048.40	22.03	1054.00	52.62
1452.00	34.83	1118.40	26.37	1213.20	10.10	1048.40	12.19	1050.80	9.17
1452.00	15.21	1118.40	15.88	1218.00	76.79	1048.40	23.51	1050.80	36.64
1452.00	70.96	1112.40	9.06	1264.00	31.31	1048.40	68.83	1050.80	15.33
1472.40	65.30	1112.40	53.88	1218.00	35.98	1048.40	9.95	1099.20	23.62
1452.00	81.07	1118.40	74.37	1170.40	55.86	1002.40	9.83	1064.40	63.55
1513.20	13.46	1124.80	44.43	1170.40	89.03	1002.40	8.01	1170.00	29.11

5 grafos de 25 vértices y 50 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
1255.60	30.10	1093.20	71.18	1221.20	29.00	1047.40	17.30	1264.00	81.18
1196.40	14.55	1106.00	15.77	1218.80	87.33	1047.40	7.19	1213.20	32.69
1196.40	97.93	1093.20	35.70	1218.80	87.33	1047.40	8.34	1170.40	7.63
1304.80	21.31	1093.20	25.43	1209.20	99.74	1047.40	11.21	1213.20	20.65
1293.20	51.25	1093.20	26.80	1232.40	14.94	1096.40	18.01	1213.20	35.26
1231.20	36.31	1089.20	98.32	1230.80	56.69	1099.80	7.75	1213.20	12.68
1300.80	33.67	1106.00	15.93	1209.20	43.94	1110.40	16.64	1264.00	22.47
1300.80	80.35	1093.20	63.00	1218.80	25.15	1078.00	13.51	1164.80	91.56
1255.60	12.47	1089.20	91.83	1239.60	54.65	1047.40	5.99	1170.40	70.75
1255.60	38.83	1093.20	62.34	1218.80	83.27	1087.80	7.75	1213.20	82.50

5 grafos de 25 vértices y 50 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
2179.20	18.35	2305.60	36.03	2294.40	27.90	2237.60	14.28	2338.80	14.66
2179.20	14.66	2305.60	34.82	2294.40	31.74	2237.60	11.53	2328.00	42.02
2179.20	15.92	2305.60	29.72	2294.40	38.62	2237.60	14.11	2338.80	37.46
2179.20	17.41	2305.60	30.86	2294.40	29.28	2237.60	13.13	2328.00	21.20
2179.20	17.30	2305.60	29.06	2294.40	98.64	2237.60	16.15	2338.80	19.94
2179.20	17.14	2305.60	32.08	2294.40	27.63	2237.60	15.66	2338.80	17.69
2179.20	18.78	2305.60	30.81	2280.80	102.60	2237.60	15.76	2328.00	37.74
2179.20	16.48	2305.60	31.96	2280.80	100.79	2237.60	12.30	2328.00	58.88
2179.20	18.45	2305.60	31.47	2294.40	43.01	2237.60	16.48	2328.00	21.69
2179.20	19.78	2305.60	21.42	2294.40	26.04	2237.60	18.90	2328.00	42.19

5 grafos de 45 vértices y 65 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
2511.20	24.22	2224.80	18.29	2994.80	28.18	2230.00	84.04	2608.60	43.23
2511.20	17.35	2224.80	101.34	2994.80	24.39	2288.40	38.39	2608.60	54.92
2511.20	11.43	2224.80	82.83	2994.80	35.15	2230.00	28.83	2608.60	51.58
2511.20	7.96	2224.80	25.54	2994.80	97.39	2230.00	39.77	2608.60	50.47
2511.20	22.96	2224.80	14.11	2994.80	117.21	2230.00	52.67	2608.60	45.37
2511.20	14.28	2224.80	23.45	2994.80	45.26	2230.00	54.87	2608.60	48.34
2511.20	20.66	2224.80	14.17	2994.80	61.46	2230.00	92.00	2608.60	48.50
2511.20	20.81	2224.80	20.87	2994.80	37.45	2230.00	85.46	2608.60	62.34
2511.20	15.38	2224.80	18.79	2994.80	78.65	2230.00	52.24	2608.60	48.12
2511.20	17.03	2224.80	14.94	2994.80	71.35	2230.00	72.61	2608.60	64.26

5 grafos de 45 vértices y 65 aristas.

V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje.	CPU	V.Obje	CPU
2396.80	39.33	2195.20	55.64	2413.20	208.44	2159.20	98.70	2099.20	57.51
2396.80	42.41	2195.20	84.53	2462.80	87.39	2436.40	109.08	2208.80	72.34
2219.20	82.56	2195.20	165.38	2304.00	191.75	2180.80	117.81	2099.20	63.11
2216.00	46.96	2195.20	76.78	2375.20	125.83	2269.20	81.95	2099.20	61.13
2396.80	64.16	2195.20	100.24	2413.20	123.04	2180.80	117.48	2208.80	66.02
2216.00	104.53	2206.00	91.06	2465.20	176.86	2368.40	62.23	2099.20	73.60
2216.80	51.13	2195.20	105.90	2465.20	82.82	2135.60	88.05	2099.20	76.62
2216.00	126.66	2195.20	148.08	2413.20	173.02	2180.80	68.71	2099.20	83.11
2470.00	56.51	2195.20	69.10	2413.20	167.30	2175.60	151.70	2099.20	49.71
2216.00	56.19	2195.20	58.99	2304.00	100.30	2175.60	140.83	2208.80	49.05

5 grafos de 50 vértices y 100 aristas.

2.4 La heurística *VNDS*.

En esta sección se realiza una descripción en detalle de la implementación de un procedimiento heurístico derivado de la técnica de búsqueda de entorno variable (*Variable Neighbourhood Search*). Este procedimiento está basado en el clásico movimiento del k-intercambio para las búsquedas en espacios de soluciones constituidos por selecciones de un número fijo de elementos de un universo. A esta técnica general se le incorpora un proceso de descomposición, por lo que se denomina *VNDS* (*Variable Neighbourhood Decomposition Search*), y se aplica al problema de la p-Mediana mostrándose superior a las otras heurísticas propuestas para este problema. Ver Hansen, Mladenovic y Pérez-Brito (1997) [78].

2.4.1 Introducción.

Las funciones objetivo de importantes problemas de optimización, como son los modelos clásicos de localización en un grafo, no son ni cóncavas ni convexas (en el conjunto de sus variables), y en general el número de óptimos locales es elevado. Algunas técnicas de optimización global han jugado un papel importante en la resolución de estos modelos, Sin embargo, en problemas de tamaño moderado, el tiempo computacional se ve incrementado exponencialmente. Algoritmos recientes (ver Chen y otros (1992) [20] y Hansen y Mladenovic (1995) [75]) han obtenidos soluciones exactas para problemas de tamaño mediano. Sin embargo en problemas reales donde el número de posibles clientes pueden ser miles, sigue siendo necesario el uso de procedimentos heurísticos.

Mladenovic (1995) [125] sugirió una nueva heurística para resolver problemas continuos de localización-asignación por medio de estructuras variables de

entornos. La idea, brevemente, es la siguiente: un número especificado de puntos alrededor de la solución actual es generado aleatoriamente obteniendose así el primer entorno. Si es posible un movimiento descendente hacia una mejor solución desde uno de estos puntos, ésta se realiza; en otro caso, se cambia de entorno ampliando el radio del mismo. El algoritmo termina si no se consige una mejora en el último entorno definido. Alternativamente, se permiten movimientos ascendente como en una búsqueda tabú. La motivación de la estructura de entorno variable, es aportar un mecanismo que metodológicamente extienda el espacio de búsqueda de las soluciones, para evitar quedar atrapado en un mínimo local. La heurística es denotada *VNS* (*Variable Neighbourhood Search*), y se describe en las próximas secciones.

En este apartado se muestra que una idea similar a la de *VNS*, puede ser aplicada con éxito en la resolución de problemas de grandes dimensiones, mediante la descomposición de los mismos. Para ello se ha desarrollado una nueva heurística denominada *VNDS (Variable Neighbourhood Decomposition Search)* y se ha implementado para el problema de la p-Mediana (ver apéndice B). El correspondiente algoritmo heurístico *VNDS* ha sido probado en la resolución de problemas de la p-Mediana para grafos de 1400, 3038 y 5934 vértices. Estos grafos han sido elegidos de la librería del problema del viajante de comercio TSPLIB [150].

2.4.2 La operación del Intercambio.

El problema estándar de la p-Mediana en un grafo consiste en, dado el conjunto de vértices usuario, seleccionar p puntos del grafo que hacen mínima la suma de la distancia a los usuarios. Cada solución viene dada por estos p puntos llamados medianas y es conocido que basta considerar los vértices como posibles medianas. A partir del trabajo de Teist y Bart (1968) [168], se han propuesto diversas heurísticas de búsqueda en las que la solución actual se mueve iterativamente cambiando las medianas una a una. En esta sección se discute la implementación eficiente de este movimiento (intercambio de servicios), que se ha convertido en un movimiento básico en muchas heurísticas para el problema de la p-Mediana. En particular, usando sólo este movimiento, reduciendo el valor de la función objetivo desde una solución inicial aleatoria dada, la heurística es bastante buena. A ello se debe el esfuerzo en mejorar la eficiencia de la implementación de este movimiento, denominado 1-Intercambio (ver Moreno, Mladenovic y Moreno (1997) [126]. El movimiento se ha extendido al considerar la sustitución simultánea de un número k fijo de puntos de servicio, denominándose k-intercambio.

En la implementación del 1-intercambio realizada se incorporan tres ingre-

dientes para mejorar la eficiencia en su aplicación:

Evaluación del movimiento: Una vez conocido la mediana que entra a formar parte de la solución, se determina la mediana que debe salir de la misma para que el intercambio sea óptimo.

Actualización: Se actualiza la lista de las medianas más cercanas a cada usuario, así como la lista de las segundas más cercanas.

Mejor mejora: Se examinan todas las mejoras posibles y se selecciona la mejor de ellas.

Los dos primeros ingredientes ya fueron sugeridos por Whitaker (1983) [176], mientras que para la tercera se proponía pasar a la primera solución que produzca mejora.

Cada instancia del problema de la p-Mediana viene especificado por un número n de usuarios, la matriz de distancias entre ellos d y el número p de medianas a localizar. En la descripción de la heurística se usa la siguiente notación paralela a los identificadores de las variables usadas en la implementación que se ofrece en el apéndice B. Una permutación $x = (x_1, x_2, ..., x_n)$ de los números del 1 al n representa a la vez el problema y una solución; sus componentes indican las ubicaciones de los usuarios y las p primeras son las seleccionadas como medianas. Generalmente, el índice $i = 1, ..., n$ recorre los usuarios mientras que $j = 1, ..., p$ recorre las medianas. La distancia entre dos ubicaciones viene dada por la matriz d de forma que $d(x_i, x_j)$ es la distancia entre x_i y x_j.

La distancia de un usuario x_i a la solución viene dada por

$$\min\{d(x_i, x_j) : j = 1, ..., p\}.$$

La mediana donde se alcanza este mínimo es la mediana que corresponde al usuario x_i y se denota por $m(x_i)$; es decir,

$$m(x_i) = \text{argmin}\{d(x_i, x_j) : j = 1, ..., p\}.$$

La segunda mediana más cercana a cada usuario x_i viene dada por el vector $s(.)$ definido por

$$s(x_i) = \text{argmin}\{d(x_i, x_j) : x_j \neq m(x_i), j = 1, ..., p\}.$$

La función objetivo se evalua en tiempo $O(pn)$. Con el mismo esfuerzo se determina la mediana $m(x_i)$ que corresponde a cada punto x_i. La misma complejidad lleva encontrar el vector s que da la segunda mediana más cercana. Una vez conocidas las medianas, la función objetivo se evalua en $O(n)$. Aunque se pueden descomponer en varias partes, la subrutina que realizaría esta tarea sería la descrita en la siguiente figura.

```
Algoritmo Evaluate (f, x, m, s, d, n, p)
    f ← 0
    do i = 1, n
        d_min ← ∞
        do j = 1, p
            if d(x_i, x_j) < d_min then
                d_min ← d(x_i, x_j)
                m(x_i) ← x_j
            endif
        enddo
    enddo
    do i = 1, n
        d_min ← ∞
        do j = 1, p
            if x_j ≠ m(x_i) then
                if d(x_i, x_j) < d_min then
                    d_min ← d(x_i, x_j)
                    s(x_i) ← x_j
                endif
            endif
        enddo
    enddo
    f ← 0
    do i = 1, n
        f ← f + d(x_i, m(x_i))
    enddo
```

Figura 9. Pseudocódigo para la evaluación de una solución.

La selección del Movimiento.

En el procedimiento que ejecuta el movimiento del 1-intercambio (denominado $MedianOut$), se calcula la modificación w que experimenta la función objetivo al incluir un nuevo punto de servicio a la solución actual (x_{in}), mientras se encuentra el mejor servicio candidato a salir (x_{out}).

En este procedimiento se usan las siguientes variables. El vector $v(x_j), j = 1, ..., p$, que da el cambio del valor de la función objetivo si la mediana sustituida es x_j y w que da el valor de la modificación de la función objetivo obtenido por el mejor intercambio. La subrutina correspondiente viene descrita en la siguiente figura.

```
Algoritmo MedianOut (w, x, m, s, d, n, p, x_in, x_out)
    do j = 1, p
        v(x_j) ← 0
    enddo
    do i = 1, n
        if d(x_i, x_in) < d(x_i, m(x_i))
            then w ← w + d(x_i, x_in) − d(x_i, m(x_i))
        else
            if d(x_i, x_in) < d(x_i, s(x_i)) then
                v(m(x_i)) ← v(m(x_i)) + d(x_i, x_in) − d(x_i, m(x_i))
            else
                v(m(x_i)) ← v(m(x_i)) + d(x_i, s(x_i)) − d(x_i, m(x_i))
            endif
        endif
    enddo
    w ← ∞
    do j = 1, p
        if v(x_j) < w then
            w ← v(x_j)
            x_out ← x_j
        endif
    enddo
```

Figura 10. Pseudocódigo para el procedimiento *MedianOut*

Con el algoritmo aplicado en *MedianOut* se examinan pn diferentes soluciones del entorno de la solución actual. La complejidad de este procedimiento es $O(n)$. Nótese que en VoB (1996) [172], se emplea la misma complejidad para determinar w cuando x_{in} y x_{out} son conocidos.

La Actualización

En la subrutina *MedianOut* tanto la primera como la segunda mediana más cercana a cada usuario x_i debe conocerse a priori. La subrutina *Update* realiza la actualización correspondiente al movimiento ejecutado en tiempo $O(pn)$. Esta subrutina viene descrita en la figura siguiente.

```
Algoritmo Update (x, m, s, d, n, p, x_in, x_out)

   do i = 1, n do
      if m(x_i) = x_out then
* ____________________________________usuarios que pierden su mediana
         if d(x_i, x_in) ≤ d(x_i, s(x_i)) then
            m(x_i) ← x_in
         else
            m(x_i) ← s(x_i)
            s(x_i) ← argmin{d(x_i, x_j) : x_j ≠ m(x_i)}
         endif
      else
* ____________________________________usuarios que NO han perdido su mediana
         if d(x_i, x_in) ≤ d(x_i, m(x_i)) then
            s(x_i) ← m(x_i)
            m(x_i) ← x_in
         else
            if d(x_i, x_in) ≤ d(x_i, s(x_i)) then
               s(x_i) ← x_in
            else
               if s(x_i) = x_out then
                  s(x_i) ← argmin{d(x_i, x_j) : x_j ≠ m(x_i)}
               endif
            endif
         endif
      endif
   enddo
```

Figura 11. Pseudocódigo para el procedimiento *Update*

La complejidad del procedimiento *Update* es $O(np)$, ya que para cada usuario, a lo sumo hay que calcular un mínimo entre p distancias. Si se utiliza una estructura de datos ordenada, como un *heap* o un árbol binario ver [1], para la actualización de la segunda mediana más cercano $s(x_i)$ la complejidad del procedimiento *Update* es $O(n \log p)$.

La Heurística del intercambio simple.

El algoritmo heurístico del Intercambio simple que se ha implementado utiliza los procedimientos *MedianOut* y *Update* para ejecutar una búsqueda local descendente de tipo *Greedy*. El procedimiento se ejecuta a partir de una solución de arranque arbitraria y ejecuta el mejor 1-intercambio posible. A continuación se muestra la descripción de esta heurística por su pseudocódigo.

```
Heurística Interchange (f, x, m, s, d, n, p)
   w* ← −1
   do while w* < 0
      w* ← ∞
      do i = p + 1, n
         x_in ← x_i
         MedianOut(w, x, m, s, d, n, p, x_in, x_out)
         if w* ≤ w then
            w* ← w, x*_in ← x_in, x*_out ← x_out
         endif
      enddo
      if w* < 0 then
*  ____________________________________Si no mejora en el entorno, parar
         f ← f + w*
         Update(x, m, s, d, n, p, x*_in, x*_out)
         Intercambiar en x las posiciones de x*_in y x*_out
      endif
   enddo
```

Figura 12. Pseudocódigo para el procedimiento *Interchange*

La complejidad de cada iteración del algoritmo es $O(n^2)$ ya que se llama a la subrutina *MedianOut* en $n \Leftrightarrow p$ ocasiones, cuya complejidad es de $O(n)$.

2.4.3 La Heurística *VNS* para la p-Mediana.

La idea básica de la heurística *VNS* es cambiar sistemáticamente el radio del entorno en un algoritmo de búsqueda local. La exploración de estos entornos puede ser realizada de dos maneras. Una exploración sistemática si se está cerca de la solución actual, o una exploración parcial si se está lejos de la misma. La exploración parcial se realiza generando una solución aleatoria en el entorno y comenzando una búsqueda local a partir de ella. El algoritmo mantiene la misma solución hasta que se alcance otra mejor, cuando esto ocurre ésta se toma como solución actual continuando la búsqueda desde ella. Los entornos son usualmente ordenados de forma que cada vez se exploran soluciones más alejadas de la actual. De este modo la intensificación de la búsqueda alrededor de la solución actual precede a la diversificación. El nivel de intensificación o diversificación puede ser controlado a través de unos parámetros.

Sea el espacio de soluciones del problema de la p-Mediana X formado por los conjuntos de p localizaciones para los servicios. La distancia entre dos soluciones $x, x' \in X$ es igual a k si y sólo si difieren en k localizaciones. Formalmente

$\rho(x,x') = |x \setminus x'| = |x' \setminus x|\,, \forall x, x' \in X$. Las estructuras de entorno que utiliza *VNS* están inducidas por la métrica ρ, es decir, $k \leq p$ localizaciones de servicios de la solución actual son reemplazadas por otros k. Se denota por N_k, $k = 1, ..., k_{\max}$, $(k_{\max} \leq p)$ el conjunto de tales estructuras y con $N_k(x)$ el conjunto de soluciones que forman un entorno N_k de la solución actual x. Formalmente $x^\circ \in N_k(x) \Leftrightarrow \rho(x^\circ, x) = k$. La cardinalidad de $N_k(x)$ es: $|N_k(x)| = \binom{p}{k}\binom{n-p}{k}$.

Obsérvese que estos conjuntos son disjuntos $(N_k(x) \cap N_l(x) = \emptyset, \forall l \neq k)$, y su unión junto con $N_0(x)$ da X, donde $N_0(x) = \{x\}$; es decir:

$$\bigcup_{k=0}^{p} N_k(x) = X, \quad \sum_{k=0}^{p} \binom{p}{k}\binom{n \Leftrightarrow p}{k} = \binom{m}{p} = |X|.$$

El pseudocódigo de *VNS* para la p-Mediana se muestra en la figura siguiente.

Heurística *VNS* para la p-Mediana

(1) Inicialización
(1a) Generar al azar una solución $x = (x_1, ..., x_p, x_{p+1}, ..., x_n)$
(1b) Evaluar $f(x)$ determinando $m(x_i)$ y $s(x_i)$, $i = 1, ..., n$.
(1c) Almacenar la solución inicial x como solución actual.

(2) Paso principal
Tomar $k \leftarrow 1$ y repetir los pasos siguientes:
(2a) **Agitar.**
Generar una solución aleatoria del entorno N_k.
(2b) **Búsqueda local.**
Aplicar la búsqueda greedy del intercambio a esta solución.
(2c) **Moverse o no moverse.**
Si la búsqueda mejora la solución x entonces
tomarla como solución actual y volver a tomar $k \leftarrow 1$,
en otro caso, hacer $k \leftarrow k + 1$.

Figura 13. Heurística *VNS* para la p-Mediana

En el paso (2a) de agitación de la heurística *VNS* se ejecuta con la subrutina *Shake*. La solución x implicada es perturbada de forma que la nueva solución x° verifica $\rho(x^\circ, x) = k$. Entonces x° es utilizada como solución inicial para el procedimiento de 1-intercambio con estrategia *greedy* del paso 2b. Si se obtiene una solución mejor que la actual x nos movemos a ella, y comenzamos otra vez con pequeñas perturbaciones, es decir, $k \leftarrow 1$. En otro caso incrementamos la distancia entre x y la nueva solución generada aleatoriamente, es decir, $k \leftarrow k+1$.

El algoritmo de agitación implementado con la subrutina *Shake* consiste en repetir k veces la operación de elegir cual será la nueva mediana al azar y seleccionar la mediana a sustituir que de lugar a la mejor solución posible. Para ello se ejecutan sucesivamente los procedimientos $MedianOut$ y $Update$ junto con las reasignaciones correspondientes. La subrutina correspondiente viene descrita en la siguiente figura donde ξ representa un número al azar en [0,1] elegido con distribución uniforme.

Algoritmo de agitación *Shake*

Subrutina $Shake(f, x, m, s, d, n, p, k)$
  do $r = 1, k$
    $i \leftarrow p + 1 + (n - p)\xi$,
    $x_{in} \leftarrow x_i$
    $MedianOut(w, x, m, s, d, n, p, x_{in}, x_{out})$
    $Update(w, x, m, s, d, n, p, x_{in}, x_{out})$
      Intercambiar en x las posiciones de x_{in} y x_{out}
  enddo

Figura 14. Pseudocódigo de *Shake*

Para detener la búsqueda en el VNS se puede limitar el valor que alcance k. También existe la posibilidad de aplicar otras reglas de parada como el limitar el número de iteraciones, limitar el tiempo máximo de CPU, o limitar el máximo número de iteraciones entre dos mejoras. Adviértase que en el paso (2a) se genera la solución aleatoriamente con el objetivo de evitar ciclos, los cuales suelen ocurrir con cualquier regla determinística.

2.4.4 El Algoritmo VNDS

Para problemas de tamaño pequeño y mediano, las búsquedas descendentes empleadas en VNS son muy rápidas, pero no suele ocurrir lo mismo en problemas de tamaño considerable. En estos casos, es cuando se recomienda utilizar métodos de descomposición.

La heurística *VNDS* (*Variable Neighbourhood Decomposition Search*) tiene una estructura muy similar al *VNS*. La diferencia entre *VNS* y *VNDS* estriba en el paso (2b). Si en *VNS* se aplica una búsqueda local en el espacio de soluciones, comenzando con $x^\circ \in N_k(x)$; en *VNDS* se resuelve el problema descompuesto en el subespacio de $N_k(x)$ determinado por x°. El problema descompuesto consiste en determinar la mejor solución donde las $p \Leftrightarrow k$ componentes de $x \cap x^\circ$ están fijas y sólo pueden variar las k que se han intercambiado para obtener x°. Para

construir este subproblema se considera el conjunto de usuarios que tienen la mediana en $x \cap x^\circ$ siendo ésta la solución de arranque.

Si en el paso 2b de *VNDS* se utiliza una heurística de búsqueda local, como la del 1-intercambio, para el problema descompuesto entonces sólo hay que incorporar al método el criterio de parada global, como el de usar el parámetro $k_{\max}$. Sin embargo si se utiliza una heurística de búsqueda como *VNS*, hay que considerar un criterio o parámetro específico que mantenga un equilibrio entre el número de descomposiciones y la calidad de la solución en cada subproblema. Esta limitación fijaría el tiempo máximo de CPU permitido en resolver cada subproblema, o el tamaño máximo en que puede descomponerse el problema. En este último caso, si el tamaño del subproblema excede su máximo, simplemente se comienza de nuevo la descomposición con $k = 1$, es decir, se comienza de nuevo con subproblemas pequeños.

El algoritmo *VNDS* completo se describe en la siguiente figura.

Heurística *VNDS* para la p-Mediana

(1) Inicialización
(1a) Generar al azar una solución $x = (x_1, ..., x_p, x_{p+1}, ..., x_n)$
(1b) Evaluar $f(x)$ determinando $m(x_i)$ y $s(x_i)$, $i = 1, ..., n$.
(1c) Almacenar la solución inicial x como solución actual.

(2) Paso principal
Tomar el primer entorno ($k \leftarrow 1$) y repetir los pasos:
(2a) **Descomponer.**
Generar un subproblema descompuesto de dimensión k
(2b) **Búsqueda heurística.**
Aplicar la heurística al subproblema.
(2c) **Moverse o no moverse.**
Si la búsqueda mejora la solución entonces
tomarla como solución actual y volver a tomar $k \leftarrow 1$,
en otro caso, hacer $k \leftarrow k + 1$.

Figura 15. Algoritmo *VNDS* para la p-Mediana

El procedimiento heurístico comienza con una solución aleatoria $x^\circ \in N_1(x)$, pero en lugar de realizar una búsqueda en todo el espacio de soluciones, resuelve el problema 1-dimensional, en el subespacio de x°. La nueva solución obtenida es evaluada en la función objetivo. El resto de los pasos son los mismos que *VNS*: si la nueva solución no es mejor que la solución actual, se hace $k = k + 1$, es decir, se busca una mejora en un subespacio donde todas, menos dos componentes de

la solución son fijadas, y así sucesivamente. En otro caso, se cambia de entorno y $k = 1$ una vez más.

Como se mencionaba previamente *VNDS* resuelve el problema descomponiendo el espacio de búsqueda en subespacios de $N_k(x)$ donde de cada solución sólo hay que buscar k componentes. El procedimiento *Decompose* es el que obtiene el subproblema descompuesto. Dado el número k, se seleccionan al azar las k medianas que van a variar en el subproblema y se unen los k conjuntos de usuarios asociados a estas medianas. Las k medianas seleccionadas constituyen la solución de arranque para el subproblema. El vector x se reordena de forma que las k primeras componentes sean estas medianas y a continuación estén los puntos asignados a ellas. El valor de p para el nuevo subproblema es k y el valor de n es el tamaño del conjunto de puntos que se le asignan.

El procedimiento *Decompose*.

```
Decompose(f, x, m, s, d, n, p, f', x', m', s', d', n', p', k)
   Colocar a true k indices del vector Merged(.) entre 1 y p
   p' ← k
   f' ← 0
   j ← 0
   do i = 1, n
      If Merged(m(x_i)) then
         j ← j + 1
         x'_j ← x_i
         m'(x_i) ← m(x_i)
         f' ← f' + d(x_i, m(x_i))
      endif
   enddo
   n' ← j
   FindSecond(x', m', s', d, p', n')
```

Figura 16. Pseudocódigo para el procedimiento *Decompose*

Para realizar la experiencia computacional se probaron diversos valores para los parámetros que determinan los criterios de parada. Las limitaciones en el número de iteraciones, tanto desde la última mejora como en la cantidad total, resultaron menos efectivas que las dependientes del tiempo. Para realizar las comparaciones se concedía a la heurística *VNDS* el mismo tiempo que el empleado por la aplicación una sola vez de la heurística del 1-intercambio desde una solución obtenida al azar. Con este control del tiempo empleado, las otras

limitaciones no tuvieron efecto alguno y los resultados obtenidos, como se muestran en la siguiente sección, indican que este esfuerzo es suficiente para alcanzar, y en algunas ocasiones superar, a la mejor solución conocida en la gran mayoría de los casos.

2.4.5 Experiencia computacional.

Para realizar la experiencia computacional se ha elegido el problema de la p-Mediana en grafos de 1400, 3038 y 5934 vértices. Estos grafos han sido elegidos de la librería del problema del viajante de comercio ORLIB [150]. La heurística *VNDS* ha sido codificada en FORTRAN. En el apéndice B se muestra el código que se ha ejecutado sobre una estación Sun Ultra I System (143-Mhz) del Groupe de Études et de Recherche en Analyse des Décisions (GERAD) en Montreal, Canadá. La compilación de los programas se ha realizado con la opción optimizadora *f77 -cg92 -O4* la cual hace disminuir el tiempo de ejecución.

Las siguientes tablas aportan los resultados obtenidos de ejecutar este programa del *VNDS* en los tres grafos elegidos. En la primera columna de la tabla se da el mejor valor conocido de la función objetivo. Estos valores objetivos para los grafos de 1400 y 3038 fueron obtenidos, cuando $p \leq 500$, del análisis empírico de [77]. Las siguiente tres columnas muestran el tiempo de CPU (en segundos) de las heurísticas, de intercambio (*Int*) (Teitz and Bart [168]), de Montecarlo (*MC*) y la de *VNDS* respectivamente. La heurística del intercambio selecciona, desde una solución elegida al azar, el mejor 1-intercambio entre todos los posibles hasta llegar a un mínimo local. La heurística de Montecarlo toma un 1-intercambio elegido al azar siempre que mejore la solución.

Para obtener los datos de estas tablas la regla de parada es el tiempo de CPU empleado. El límite de tiempo utilizado para detener al *VNDS* ha sido el requerido por la heurística del Intercambio siguiendo el método de Teitz y Bart en realizar una búsqueda local descendente para cada uno de los problemas. La heurística del intercambio se ha detenido cuando ha alcanzado esta misma cantidad de tiempo. Las tres últimas columnas muestran el porcentaje de desviación de cada método respecto al mejor valor de la función objetivo conocido. Por último se lista, el tiempo medio de CPU y la desviación media de los tres métodos. Nótese que el tiempo incluido en esta columna es inferior al de la columna del intercambio dado que, para que la comparación sea válida, se utiliza el tiempo hasta que se realizó la última mejora del objetivo.

	Valor Obj.	Tiempo de CPU			% de error		
p	mejor con.	Int	MC	$VNDS$	Int	MC	$VNDS$
10	101248.13	14.78	6.34	9.25	0.69	0.03	0.00
20	57856.32	25.60	13.55	13.55	1.36	0.00	0.00
30	44086.53	35.61	15.93	18.65	0.69	0.03	0.00
40	35005.82	45.62	17.59	25.73	0.07	0.09	0.02
50	29130.10	62.02	13.64	22.06	-0.16	0.19	-0.04
60	25166.15	90.05	15.90	31.41	0.63	1.29	-0.04
70	22125.53	101.05	23.87	96.96	0.89	0.09	-0.27
80	19877.88	110.07	17.46	50.09	0.58	0.45	-0.11
90	17987.94	137.52	18.63	46.78	0.86	0.40	-0.38
100	16551.20	165.53	20.17	39.41	1.01	0.55	0.21
150	12032.65	215.90	35.04	150.26	0.72	0.71	-0.02
200	9360.01	283.13	42.93	148.83	1.11	0.62	-0.03
250	7742.70	362.21	31.75	122.69	0.62	0.33	-0.06
300	6624.52	389.38	43.47	366.37	0.79	0.70	-0.07
350	5727.02	407.88	38.97	360.74	1.63	1.07	-0.21
400	5020.50	479.77	49.42	136.51	0.77	0.66	-0.50
450	4487.73	467.14	33.58	77.58	0.55	0.70	-0.05
500	4049.03	423.53	27.94	285.66	0.78	0.44	-0.34
Media Total		228.70	27.10	241.47	0.76	0.96	-0.11

Tabla 2.1: **Grafo de 1400 vértices.**

	Valor obj.	Tiempo de CPU			% de error		
p	mejor con.	*Int*	*MC*	*VNDS*	*Int*	*MC*	*VNDS*
50	507655.19	612.87	60.71	311.15	0.50	0.47	-0.03
100	353255.22	1248.77	158.95	1062.94	0.43	0.61	-0.35
150	281772.09	1896.87	167.12	1862.07	0.80	0.39	-0.05
200	238622.98	2526.70	139.85	2335.87	0.65	0.95	-0.19
250	209343.34	3114.28	195.36	2846.61	0.43	0.48	-0.18
300	187807.06	3358.40	169.75	1913.19	0.70	0.30	-0.18
350	171009.30	3645.59	199.01	2951.38	0.55	0.39	-0.42
400	157079.67	5308.28	206.29	4772.22	0.40	0.57	-0.53
450	145448.98	5491.01	233.34	2148.47	0.44	0.67	-0.44
500	135467.97	6044.12	272.65	3379.78	0.44	0.43	-0.45
550	126867.38	6407.13	318.19	2883.01	0.74	0.88	0.00
600	119107.99	4845.88	196.81	3392.02	1.03	1.09	0.00
650	112090.28	5275.87	243.98	5160.90	1.16	0.98	0.00
700	105893.39	5346.42	188.82	3421.69	1.11	1.01	0.00
750	100362.55	5108.20	202.88	5095.70	1.22	1.15	0.00
800	95445.06	5024.91	164.18	4143.99	1.21	1.30	0.00
850	91023.87	5023.56	157.50	4733.62	1.10	1.42	0.00
900	87041.84	5656.95	193.38	5549.94	1.16	1.27	0.00
950	83310.19	5079.25	238.98	4328.75	1.07	1.09	0.00
1000	79900.52	4883.87	205.55	4879.89	0.99	0.95	0.00
Media Total		4584.96	201.27	3624.84	0.86	0.61	-0.12

Tabla 2.2: **Grafo de 3038 vértices.**

	Valor obj.	Tiempo de CPU			% de error		
p	mejor con.	Int	MC	VD	Int	MC	VD
50	4053917.75	4257.26	418.05	550.18	0.72	0.06	0.00
100	2733817.25	6637.48	510.20	6087.75	0.36	0.15	0.00
150	2151018.50	9971.28	373.54	2657.35	0.90	0.86	0.00
200	1809064.38	14966.05	663.69	14948.37	0.79	0.36	0.00
250	1571813.50	17118.27	620.14	11042.62	0.73	0.40	0.00
300	1394715.12	20127.91	541.76	17477.51	0.65	0.51	0.00
350	1257900.00	22003.88	868.85	21769.56	0.82	0.55	0.00
400	1145669.38	23630.95	618.62	22283.04	0.82	0.59	0.00
450	1053450.88	66890.41	1898.83	21683.38	0.90	0.79	0.00
500	974275.31	29441.97	954.10	10979.77	0.98	0.51	0.00
600	848459.38	32957.36	768.95	18996.37	0.78	0.47	0.00
700	752068.38	36159.45	768.84	32249.00	0.64	0.50	0.00
800	676846.12	38887.40	813.38	20371.81	0.61	0.53	0.00
900	613367.44	41607.78	731.71	27060.09	0.55	0.53	0.00
1000	558802.38	44176.27	742.70	26616.96	0.73	0.66	0.00
1100	511813.19	45763.18	740.33	28740.89	0.90	0.63	0.00
1200	470295.38	46387.06	674.87	15886.20	0.99	0.91	0.00
1300	433597.44	46803.63	740.48	26150.85	1.00	0.82	0.00
1400	401853.00	47184.10	708.90	44944.41	0.87	1.18	0.00
1500	374061.41	47835.35	823.95	44727.36	0.98	0.93	0.00
Total average		39940.03	742.69	25650.25	0.85	0.71	0.00

Tabla 2.3: **Grafo de 5934 vértices.**

2.4.6 Conclusiones.

Hay que destacar que la heurística *VNDS* resuelve en tiempos muy razonables problemas de localización en Grafos del orden de 6000 vértices. La calidad de la soluciones dadas por este método de entorno varible es muy buena. El motivo del éxito de las heurística de entorno variable parece estar, entre otras cosas, en la proximidad que existe entre las soluciones buenas de la p-Mediana. Una propiedad similar tienen otros problemas de optimización combinatoria como el problema del viajante de comercio o el problema de la bipartición de un grafo [11]. En otras palabras existen localizaciones cuyas componentes pertenecen en su mayoría a los mínimos locales obtenidos por los procedimiento descendentes. Explotar esta propiedad de proximidad de los mínimos locales produce varias ventajas a la heurística *VNDS*,

- La búsqueda de la nueva solución se realiza en un área atractiva del espacio de búsqueda formada por buenas soluciones.
- Las soluciones desde las que se inician los descensos son buenas soluciones y por tanto son cortos.
- La descomposición se centra específicamente en las componentes que varían entre las buenas soluciones.
- El estudio del entorno se realiza eficientemente por medio de la heurística VNS.

En consecuencia el método visita muchos óptimos locales próximos al óptimo global, en el mismo tiempo de CPU en el cual el procedimiento descendente visita un mínimo local comenzando desde una solución inicial aleatoria. Esta operación es denominada un descenso, por su analogía con la representación gráfica de la función objetivo en estos mínimos locales.

Si realizamos un análisis específico de cada una de las tablas presentadas, se sigue lo siguiente:

tabla 1.

Para valores de p menores de 50, la heurística *VNDS* alcanza, en menos tiempo, los valores obtenidos por Hansen y Mladenovic (1996) [77], pero, para p mayor que 50, *VNDS* mejora el mejor valor de la función objetivo conocido, y por ello para $p \geq 50$ se han actualizado los valores objetivos de [77] con los valores objetivos obtenidos de *VNDS*. Se concluye que la heurística *VNDS* es un 11% mejor que la heurística de Hansen y Mladenovic

(1996) [77], en el tiempo que ésta última gasta en realizar una sóla búsqueda local descendente.

tabla 2.

En la tabla 2, se puede observar que para cada valor de $p \leq 500$, la heurística *VNDS* mejora siempre el mejor valor objetivo conocido citados en Hansen y Mladenovic (1996) [77]. Por ello los mejores valores objetivos conocidos para el problema, son precisamente, los obtenidos por medio de *VNDS*. Cuando $p > 500$, no se conoce referencia alguna, para el problema de la p-Mediana en un grafo de 3038 vértices. Por ello se compara *VNDS* con las heurísticas de intercambio y Montecarlo. Es claro que los resultados que se obtienen de *VNDS* son los mejores. Esta heurística en media para $p \leq 500$ es un 28% mejor que la heurística de Hansen y Mladenovic (1996) [77], en el tiempo que ésta última gasta en realizar una sóla búsqueda local descendente.

tabla 3.

Al igual que en la tabla 2, los mejores valores objetivos conocidos para el problema de la p-Mediana en un grafo de 5934 vértices, son los obtenidos por medio de *VNDS*. Una vez más, la comparación se ha realizado con respecto a las heurísticas de intercambio y Montecarlo, dado que no se conoce referencia alguna.

Capítulo 3

EL PROBLEMA DEL CENTDIAN EN GRAFOS.

3.1 Introducción.

Los problemas de localización del Centro y de la Mediana de un grafo han sido objeto de profundos estudios desde que fueron formulados por Hakimi (1964) [62], lo cual ha dado lugar a diversos algoritmos para su resolución así como a numerosas aplicaciones de estos conceptos. Antes de acometer el estudio del problema del Centdian, conviene recordar brevemente el concepto de cada uno de ellos.

El problema de la Mediana consiste en minimizar la distancia media entre el usuario y el servicio a localizar. Este tipo de función objetivo mini-sum es apropiado para localizar servicios que deben satisfacer una demanda continua y estable, como, por ejemplo, escuelas, centros comerciales, terminales de transporte, etc. En la literatura es comúnmente conocido también como el problema de Fermat-Weber (Wendell and Hurter (1973) [173]). Por otro lado, el problema del Centro estriba en localizar un punto en el grafo de tal manera que la distancia del vértice más alejado a él sea mínima. Esta función objetivo mini-max es frecuentemente usada en servicios de emergencias tales como estaciones de ambulancias, de bomberos o de policia. Este problema también es conocido como el problema de Rawl [15]. En este contexto, el modelo incorpora una medida de la aversión por la desigualdad. Sin embargo, este modelo fracasa si hay variaciones de las distancias dentro de la población de los usuarios.

En muchos problemas reales el objetivo es una mezcla de estos dos modelos, muchas veces antagónicos. Por ejemplo, a la hora de ubicar un puesto de

bomberos se puede pretender minimizar el tiempo de viaje al posible punto de demanda más alejado, siempre y cuando el puesto esté suficientemente cerca de las áreas más pobladas. El problema consiste en minimizar la función Centro con la restricción de que la función Mediana no supere un determinado nivel. Otro ejemplo es minimizar la distancia media que deben recorrer los niños para llegar a la escuela sin que a ninguno le quede demasiado lejos. El objetivo aquí es minimizar la función Mediana imponiendo una cota superior a la función Centro.

El modelo que sugirió Halpern (1976) [65] consiste en minimizar una combinación convexa de la distancia media ponderada y la distancia máxima no ponderada; la solución a este problema recibe el nombre de Centdian. Con ello se consigue un equilibrio entre las distancias media y la distancia más alejada.

En Carrizosa y otros, [15] se encuentra un desarrollo axiomático en el que se justifica el uso del criterio Centdian.

En este capítulo se considera la función Centro ponderada generalizando así el modelo de Halpern. Se entiende que las ponderaciones de la función Centro son distintas que las ponderaciones de la función Mediana. Por simplicidad este nuevo problema se sigue denotando Centdian.

Se comienza analizando el caso de localizar un sólo servicio, extendiendo todas las propiedades del modelo de Halpern, para el nuevo modelo aquí presentado.

A continuación, se estudia el problema del 2-λ-Centdian. Realizando un análisis de la función objetivo, y proponiendo un algoritmo cuya complejidad es de $O(m^2n^4)$.

Exceptuando un apartado del artículo de Hooker y otros (1991) [82], en el cual se propone un conjunto finito dominante para el p-λ-Centdian, no se conoce literatura que haga mención al λ-Centdian múltiple. Este conjunto finito dominante propuesto por Hooker, es erróneo, como se muestra en el contraejemplo presentado en este capítulo.

Finalmente se propone un nuevo conjunto finito dominante para el problema p-λ-Centdian en un grafo. Y como consecuencia del mismo se propone un algoritmo exacto.

3.2 Formulación del problema.

Sea $P(G)$ el conjunto continuo de puntos de las aristas de G. Para cualquier número $p \geq 1$, sea $X = (x_1, x_2, ..., x_p)$ una colección de p puntos de $P(G)$. La distancia de X a un vértice v_i perteneciente al conjunto de vértices clientes $U \subseteq V$

viene dada por:

$$d(X, v_i) = \min_{v_i \in U} \{d(x_j, v_i) : j = 1, 2, ..., p;\}.$$

El problema de la p-Mediana reside en encontrar p puntos servicios X en G que minimicen la función:

$$f_m(U; X) = \sum_{v_i \in U} w_i \, d(X, v_i).$$

El problema del p-Centro consiste en encontrar p puntos servicios X en G que minimicen la función:

$$f_c(U; X) = \max_{v_i \in U} \; w_i' \, d(X, v_i) \, .$$

Esto implica que hay que abrir p servicios y asignar cada usuario a uno de ellos, optimizando las correspondientes funciones objetivos.

Dado un número p de servicios a ser localizados y un valor λ, $0 \leq \lambda \leq 1$, el problema p-λ-Centdian consiste en encontrar p puntos servicios X en G que minimizen la función objetivo:

$$f_\lambda(U; X) = \lambda \; f_c(U; X) + (1 \Leftrightarrow \lambda) \; f_m(U; X).$$

Denotamos con p-λ-CD el conjunto de todos los p-λ-Centdian del grafo para un λ dado.

El valor de λ representa el peso atribuído a la función Centro con respecto al peso de la función Mediana. Generalmente es difícil para el decisor determinar exactamente el peso que debe ser asociado a cada uno de los dos criterios. Por tanto, es importante encontrar una expresión sencilla de la ubicación del λ-Centdian para todos los pesos relativos. En particular, si $\lambda = 0$ los λ-Centdian son las Medianas, y si $\lambda = 1$ el λ-Centdian es un Centro. Cuando el $0 < \lambda < 1$, el λ-Centdian es la solución óptima para un problema de localización en el que los dos criterios de eficiencia y equidad son importantes.

Una solución $X \in P(G)$ es eficiente, respecto al problema bicriterio $\min \{f_c, f_m\}$ si no existe ninguna solución $Y \in P(G)$ tal que $f_c(Y) \leq f_c(X)$ y $f_m(Y) \leq f_m(X)$ siempre que, al menos, una de las dos desigualdades sea estricta. En Pérez-Brito y Moreno Pérez (1997) [146] se encuentra un procedimiento geométrico que caracteriza los puntos eficientes de este problema bicriterio. Sea EF el conjunto de

todos las soluciones eficientes del grafo. El conjunto de los puntos λ-Centdian del grafo para todo $\lambda \in [0,1]$ es un subconjunto del conjunto de puntos eficientes del $\min\{f_c, f_m\}$, es decir,

$$\cup \{p\text{-}\lambda\text{-CD} : 0 \leq \lambda \leq 1\} \subseteq EF.$$

3.2.1 Puntos especiales.

A continuación se definen algunos conjuntos de puntos de $P(G)$ que serán utilizados para caracterizar el conjunto finito dominante del problema del p-λ-Centdian.

- Un punto $x \in P(G)$ es un **centro local** con rango r asociado a los vértices v_k, $v_l \in U$, (el cual es representado por $x \in B_c(r; v_k, v_l)$) si x es un punto interior de una arista $[i,j]$ tal que :

  1. $r = w'_k\, d(x, v_k) = w'_k\,(l(x,i) + d(i,v_k)) < w'_k\,(l(x,j) + d(j,v_k))$ y
  2. $r = w'_l\, d(x, v_l) = w'_l\,(l(x,j) + d(j,v_l)) < w'_l\,(l(x,i) + d(i,v_l))$.

  Para cada valor $r \geq 0$, sea:

  $$LC(r) = \bigcup_{v_k, v_l \in U} B_c(r; v_k, v_l)$$

- Un punto $x \in P(G)$ es un **punto pendiente** con rango r asociado a los vértices v_k, $v_l \in U$, (el cual es representado por $x \in B_p(r; v_k, v_l)$) si x es un punto interior de una arista $[i,j]$ tal que :

  1. $w'_k\,(l(x,i) + d(i,v_k)) = w'_l\,(l(x,i) + d(i,v_l))$ con $w'_k \neq w'_l$ y / ó
  2. $w'_k\,(l(x,j) + d(j,v_k)) = w'_l\,(l(x,j) + d(j,v_l))$ con $w'_k \neq w'_l$.

  Para cada valor $r \geq 0$, sea:

  $$PP(r) = \bigcup_{v_k, v_l \in U} B_p(r; v_k, v_l)$$

- Un punto $x \in P(G)$ es un **cuello de botella** con rango r asociado a un vértice $v_i \in U$, (el cual es representado por $x \in B_a(r; v_i)$) si x es un punto interior de una arista $[i,j]$ tal que :

  1. $r = w'_i\, d(x, v_i) = w'_i\,(l(x,i) + d(i,v_i))$ y

2. $r = w_i' d(x, v_i) = w_i' (l(x, j) + d(j, v_i))$.

- Un punto $x \in P(G)$ es un **punto extremo** con rango r asociado a un vértice $v_i \in U$, (el cual es representado por $x \in E_p(r; v_i)$) si x es un punto interior de una arista $[i, j]$ tal que :

  1. $r = w_i' d(x, v_i) = w_i' (l(x, i) + d(i, v_i))$ ó
  2. $r = w_i' d(x, v_i) = w_i' (l(x, j) + d(j, v_i))$.

Para cada valor $r \geq 0$, sea:

$$EP(r) = \underset{v_i \in U}{\cup} E_p(r; v_i).$$

Existe a lo sumo un centro local y dos puntos pendientes con respecto a cada pareja de vértices en cada arista, por tanto, el número de los diferentes radios de los centros locales y de los puntos pendientes es finito. El conjunto de distancias ponderadas entre vértices y vértices usuarios junto con el de rangos de centro locales y rangos de los puntos pendientes viene determinado por:

$$R = \{r : LC(r) \neq \emptyset\} \cup \{r : PP(r) \neq \emptyset\} \cup \{r : r = w_i' d(v_i, v_j), v_i \in U, v_j \in V\}.$$

Este conjunto será denominado **conjunto canónico de distancias**. Los puntos extremos con rango en este conjunto son los **puntos extremos canónicos**. Entonces, el conjunto de centros locales y el conjunto de puntos extremos canónicos asociados a los vértices usuarios son:

$$LC = \underset{r \in R}{\cup} LC(r)$$

$$PP = \underset{r \in R}{\cup} PP(r)$$

$$EP = \underset{r \in R}{\cup} EP(r).$$

3.3 El 1-λ-Centdian.

El objetivo del 1-λ-Centdian es encontrar un punto $x^* \in P(G)$ tal que:

$$f_\lambda(U;(x^*)) = \min_{x \in P(G)} \{f_\lambda(U;(x))\}.$$

Si bien, sobre el problema 1-λ-Centdian la bibliografía es escasa, en este apartado se considera los resultados más relevantes del modelo de Halpern, adaptándolos para el nuevo modelo con la función Centro pesada.

3.3.1 Propiedades del 1-λ-Centdian.

Sea $e = [v_i, v_j]$ una arista de E, y Q_e el conjunto de puntos de ruptura, o de no linealidad de $f_c(U,x)$ sobre e además de v_i y v_j.

El siguiente resultado se deduce de las propiedades de $f_c(U,x)$ y $f_m(U,x)$.

Propiedad 3.3.1 Dado λ, $0 \leq \lambda \leq 1$, la función:

$f_\lambda(U,x) = \lambda f_c(U,x) + (1 \Leftrightarrow \lambda) f_m(U,x)$, $x \in e = [v_i,\ v_j]$, es una función de x continua y lineal a trozos, con un número finito de puntos de ruptura sobre e, todos alcanzados en puntos de Q_e.

Demostración: Dado que la función $f_c(x)$ es continua y lineal a trozos, y con un número finito de puntos de ruptura en $e \cap (LC \cup PP \cup V)$ y, dado que la función $f_m(x)$ es continua y lineal a trozos, y con un número finito de puntos de ruptura en $e \cap V$, la suma es continua y lineal a trozos, y con un número finito de puntos de ruptura en $Q_e \subset e \cap (LC \cup PP \cup V)$, por lo que se sigue la propiedad 3.3.1. □

De la propiedad anterior se deduce que para un λ dado, $0 \leq \lambda \leq 1$, y para dos puntos consecutivos x e y de Q_e si tenemos que $f_\lambda(U,x) = f_\lambda(U,y)$, entonces la función f_λ es constante en $[x,\ y]$. Esto implica que todos los puntos de $[x,\ y]$ son puntos de ruptura de f_λ (Hansen y otros (1991)[74]).

Propiedad 3.3.2 *El conjunto $LC \cup PP \cup V$ es un conjunto finito dominante para el problema 1-λ-Centdian.*

Demostración: Es consecuencia de la propiedad 3.3.1 □

Sea $f_\lambda = \ min\{f_\lambda(U,x) : x \in G\}$. Para un λ dado, f_λ denota el valor de la función 1-λ-Centdian en el punto λ-Centdian.

Usando la propiedad 3.3.1, podemos escribir f_λ como:

$$f_\lambda = \min\{\lambda\, f_c(q) + (1 \Leftrightarrow \lambda)\, f_m(q) : q \in \bigcup_{e\in E} Q_e\}.$$

Al ser Q_e un conjunto finito, se sigue que:

Propiedad 3.3.3 La función f_λ (como función de λ) es continua, lineal a trozos y cóncava con $0 \leq \lambda \leq 1$.

Demostración: Para cada $q \in \bigcup_{e\in E} Q_e$ la expresión $\lambda\, f_c(q) + (1 \Leftrightarrow \lambda)\, f_m(q)$ representa un segmento de recta, por lo tanto es continua y lineal. El mínimo de todos estos segmentos de recta es continuo, lineal a trozos y concavo. □

Definición 3.3.1 *Una solución x* **domina** *a otra solución y sobre el intervalo $[\lambda', \lambda''] \subseteq [0,1]$, si $f_\lambda(x) \leq f_\lambda(y)$ para todo λ tal que $\lambda' \leq \lambda \leq \lambda''$.*

Definición 3.3.2 *Una solución x es* **dominante** *sobre el intervalo $[\lambda',\ \lambda'']$, si domina a todos las soluciones del grafo sobre ese intervalo.*

Definición 3.3.3 *Una solución y está* **uniformemente dominada** *por un subconjunto de soluciones $J \subset G$, si para todo $\lambda \in [0,\ 1]$ existe un $x \in J$ tal que $f_\lambda(y) \geq f_\lambda(x)$; este caso lo representaremos por $y \dashv J$.*

Propiedad 3.3.4 Existen conjuntos finitos:

a) $\{\lambda_i : i = 0, ..., r\}$ de valores comprendidos en el intervalo $[0,1]$ y

b) $\{x_i : i = 1, ..., r\}$ de puntos de G, que tienen las siguientes propiedades:

(i) $\lambda_i < \lambda_{i+1}$ para todo i; $\lambda_0 = 0$, $\lambda_r = 1$.

(ii) $x_1 = x_m$ (Mediana), $x_r = x_c$ (Centro).

(iii) $f_\lambda = f_\lambda(x_i)$ para todo λ tal que $\lambda_{i-1} \leq \lambda \leq \lambda_i$, $i = 1, ..., r$.

Demostración: Sean $\{\lambda_i : i = 0, ..., r\}$ los puntos de ruptura de f_λ, de la propiedad 3.3.2, se deduce que si λ_{i-1} y λ_i son dos puntos adyacentes de discontinuidad en la derivada de f_λ, entonces existe algún $x_i \in P(G)$ tal que x_i es dominante en $[\lambda_{i-1},\ \lambda_i]$ y $f_\lambda = f_\lambda(x_i)$ para todo $\lambda_{i-1} \leq \lambda \leq \lambda_i$. □

La siguiente propiedad facilita encontrar el 1-λ-Centdian en un grafo.

Propiedad 3.3.5 El 1-λ-Centdian de un grafo, para cualquier $\lambda \in [0, 1]$, está localizado en un camino que une un Centro y una Mediana del grafo.

Demostración: El camino que une el Centro y una Mediana del grafo es el camino de los puntos x_i de la propiedad 3.3.4, donde para cada λ_i los puntos de $[x_i, x_{i+1}]$ son $\lambda_i \Leftrightarrow CD$. □

Propiedad 3.3.6 Si denotamos por $x^*(\lambda)$ el 1-λ-Centdian para un λ dado, entonces $f_c(U, x^*(\lambda))$ es una función no creciente de λ, mientras que $f_m(U, x^*(\lambda))$ es una función no decreciente de λ.

Demostración: Para todo $\lambda \in [0, 1]$ los puntos $\{x_i : i = 1, ..., r\}$ que aparecen en la propiedad 3.3.4 son algunos puntos λ-Centdian y además verifican que $f_c(U, x_i) \geq f_c(U, x_{i+1})$ y $f_m(U, x_i) \leq f_m(U, x_{i+1})$, $i = 1, ..., r \Leftrightarrow 1$. Como $x^*(0) = x_m = x_1$ y $x^*(1) = x_c = x_r$, es fácil comprobar lo que se establece en esta propiedad 3.3.6. □

Para resolver el problema del 1-λ-Centdian en grafos generales se han propuesto dos métodos con idéntica complejidad $O(mn \log mn)$: el de Halpern (1976) [65] y el de Hansen y otros (1991) [74]. En ambos casos los algoritmos propuestos dan como solución un conjunto de puntos λ-Centdian, para todos los $\lambda \in [0, 1]$.

El método de Halpern se basa en la construcción de cotas superiores para el valor mínimo de f_λ, mientras que Hansen y otros, abordan el problema desde el punto de vista geométrico, resultando este último conceptualmente más sencillo.

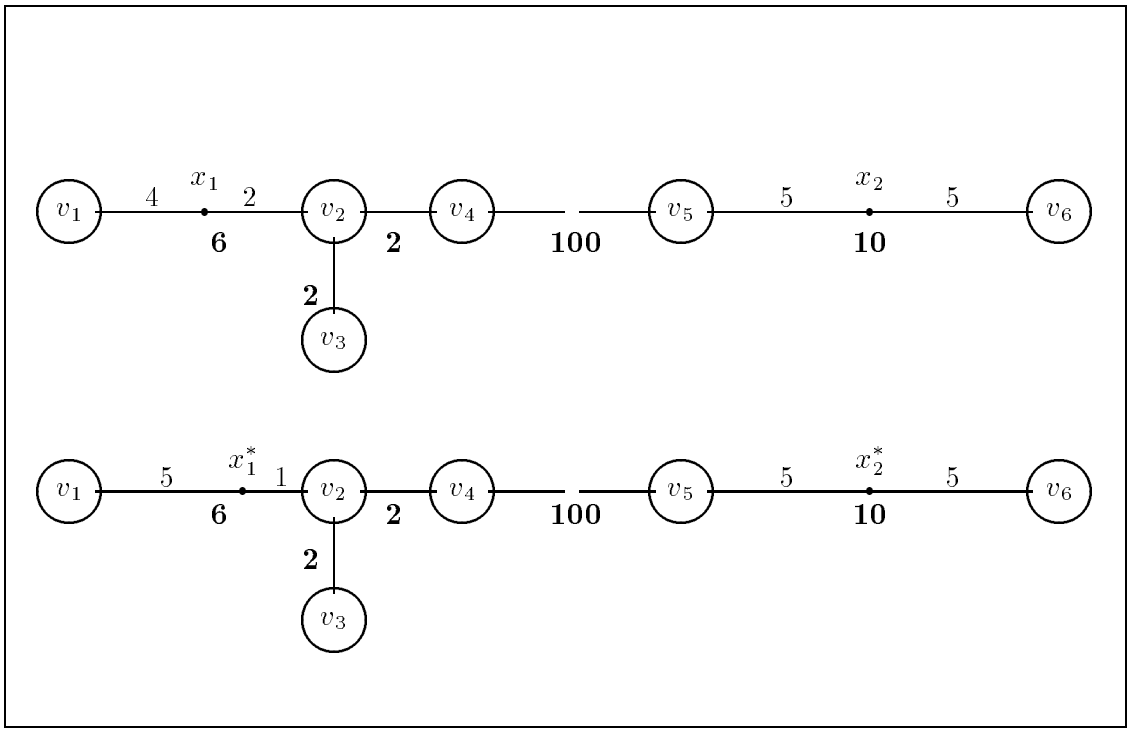
3.4 El 2-λ-Centdian.

El objetivo del 2-λ-Centdian es encontrar dos puntos $x_1^*, x_2^* \in P(G)$ de tal forma que:

$$f_\lambda(U; (x_1^*, x_2^*)) = \min_{(x_1, x_2) \in P(G)} \{f_\lambda(U; (x_1, x_2))\}.$$

Exceptuando el artículo de Hooker, Garfinkel y Chen (1991) [82], no se conoce literatura que haga mención al λ-Centdian múltiple. Hooker y otros, utilizan un resultado teórico (lema 10) que extiende el conjunto finito dominante del problema λ-Centdian al problema del p-λ-Centdian con $p \geq 2$. Como se señala en el apartado anterior, el conjunto de vértices, centros locales y puntos pendientes es un conjunto finito dominante para el λ-Centdian, en el caso del centro no

ponderado el conjunto $PP = \emptyset$ pero $V \cup LC$ no es un conjunto finito dominante para el problema del p-λ-Centdian (con la función centro sin peso), como se afirma en [82]. El siguiente contraejemplo demuestra tal error.



Consideremos el problema del 2-λ-Centdian con $\lambda = 0.8$ en un árbol con seis vértices usuarios (ver figura.- El contraejemplo), donde las longitudes de las aristas se indican debajo de las mismas y todos los pesos son iguales a 1. Sin dificultad encontramos dos grupos de vértices, el primero de ellos lo forman los vértices v_1, v_2, v_3 y v_4, y el segundo lo forman los vértices v_5 y v_6. Para este segundo grupo, la mejor localización se halla en el punto $x_2 = p([v_5, v_6], 5)$, es decir, en el punto que dista 5 del vértice v_5 en la arista $[v_5, v_6]$. Este es el único centro local asociado a v_5 y v_6; $\{x_2\} = B_c(5; v_5, v_6)$. Los centros locales asociados al primer grupo de vértices son:

$$
\begin{aligned}
&p([v_1, v_2], 3) \in B_c(3; v_1, v_2),\\
&p([v_1, v_2], 4) \in B_c(4; v_1, v_3) = B_c(4; v_1, v_4),\\
&p([v_2, v_3], 1) \in B_c(1; v_2, v_3)\\
&p([v_2, v_4], 1) \in B_c(1; v_2, v_4).
\end{aligned}
$$

El mejor de estos candidatos es $x_1 = p([v_1, v_2], 4)$. Existen otros centros locales sobre la arista de mayor longitud, pero resulta evidente que éstos no

pueden estar en la solución óptima. El valor de la función objetivo para $X = \{x_1, x_2\}$ es $f_\lambda(U, X) = 8.8$

$$f_\lambda(U,X) = 0.8 \cdot d(x_2, v_6) + 0.2 \cdot \left[\sum_{i=1}^{4} d(x_1, v_i) + \sum_{i=5}^{6} d(x_2, v_i)\right] =$$

$$= 0.8 \cdot 5 + 0.2 \cdot (4 + 2 + 4 + 4 + 5 + 5) = 8.8$$

Sin embargo, la solución óptima es $X^* = \{x_1^*, x_2^*\}$, donde $x_1^* = p([v_1, v_2], 5)$ y $x_2^* = x_2$, con un valor de la función objetivo para X^* de $f_\lambda(U, X^*) = 8.4$

$$f_\lambda(U,X^*) = 0.8 \cdot d(x_1^*, v_1) + 0.2 \cdot \left[\sum_{i=1}^{4} d(x_1^*, v_i) + \sum_{i=5}^{6} d(x_2^*, v_i)\right] =$$

$$= 0.8 \cdot 5 + 0.2 \cdot (5 + 1 + 3 + 3 + 5 + 5) = 8.4$$

Nótese que x_1^* no es un centro local.

3.4.1 Análisis de la función objetivo.

Adelantamos en este apartado que el conjunto finito dominante D que se propone para el 2-λ-Centdian es $D = V \cup LC \cup PP \cup EP$. En este apartado lo que se pretende es mostrar mediante ejemplos el comportamiento de la función objetivo del 2-λ-Centdian con objeto de justificar éste conjunto finito dominante. La demostración formal del mismo aparece en la subsección 3.5.1.

Sea $X = \{x_1, x_2\} \subset P(G)$ un conjunto formado por dos puntos del grafo. Para analizar la forma en que varía la función f_λ al desplazar uno o ambos de los elementos del conjunto X, es conveniente tener en cuenta qué vértices se pueden considerar asignados a cada uno de los elementos de este conjunto.

Sean:

$$U_1 = \{v \in U : d(x_1, v) < d(x_2, v)\}$$

$$U_2 = \{v \in U : d(x_1, v) > d(x_2, v)\}$$

$$U_0 = \{v \in U : d(x_1, v) = d(x_2, v)\}$$

Los vértices de U_0 pueden ser asignados tanto a x_1 como a x_2. Si suponemos que son asignados a x_2, las funciones objetivos del 2-Centro y la 2-Mediana quedarían como sigue:

$$f_c(U;(x_1,x_2)) = \max\{f_c(U_1;x_1), f_c(U_2\cup U_0;x_2)\} =$$
$$= \max\{\underbrace{\max_{v_i\in U_1}\{w_i' d(x_1,v_i)\}}_{r_1}, \underbrace{\max_{v_i\in U_2\cup U_0}\{w_i' d(x_2,v_i)\}}_{r_2}\} = \max\{r_1,r_2\} = r$$

y

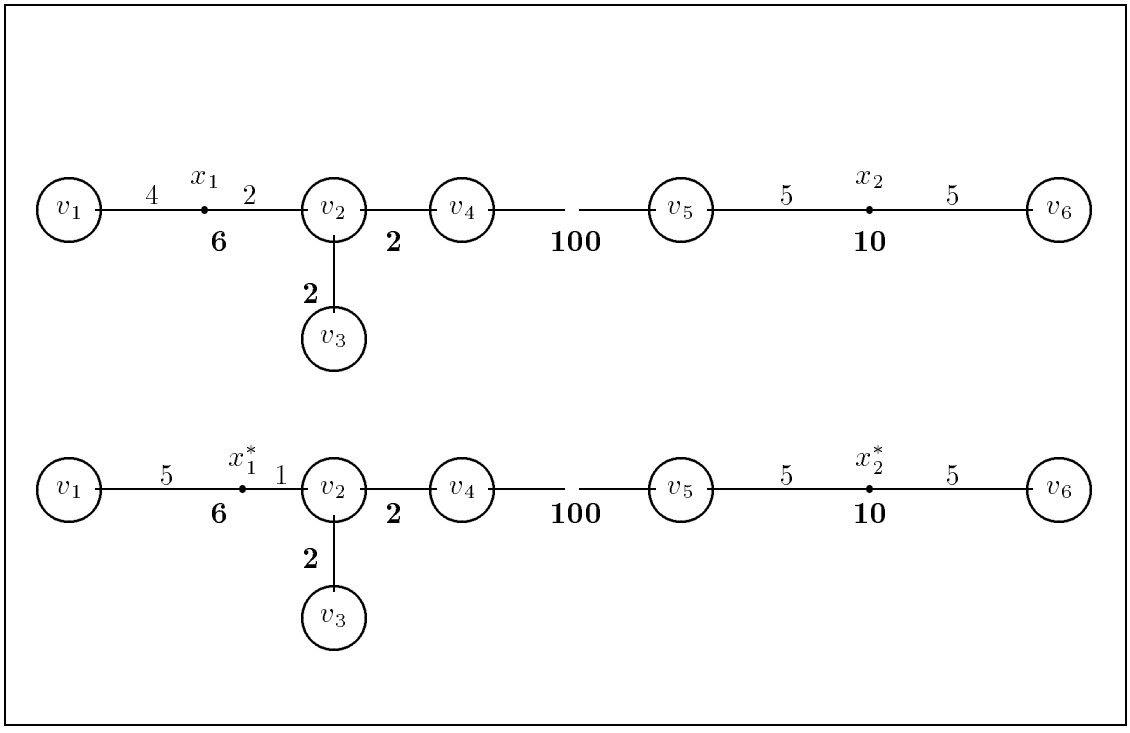
$$f_m(U;(x_1,x_2)) = f_m(U_1;x_1)+f_m(U_2\cup U_0,x_2) = \sum_{v_i\in U_1} w_i\, d(x_1,v_i)+\sum_{v_i\in U_2\cup U_0} w_i\, d(x_2,v_i)$$

Sea $X=\{x_1,x_2\}$ una solución de f_λ. Si X no pertenece al conjunto D, se desplaza X de forma que la nueva solución $X^*\subset D$ verifique:

$$f_\lambda(U;X^*)\le f_\lambda(U;X).$$

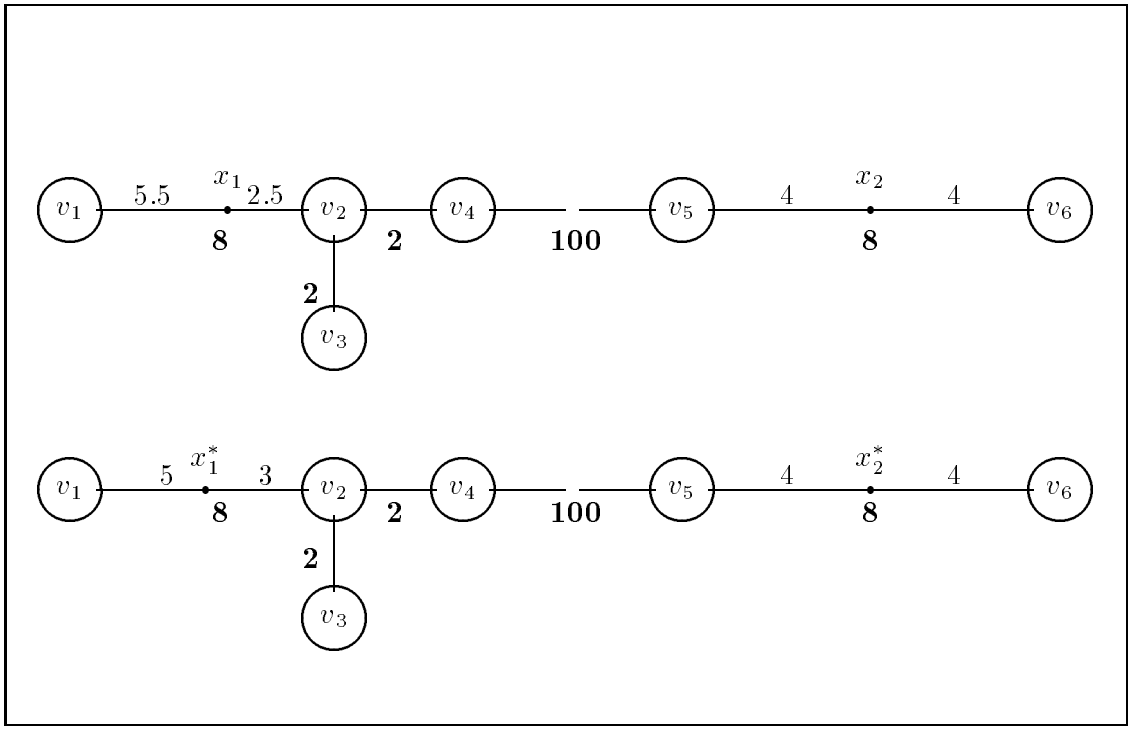
Supongamos que $x_1\notin D$ y consideremos los siguientes casos:. $r_1<r_2$, $r_1>r_2$ y $r_1=r_2$.

Caso 1. Sea $r_1<r_2=r$.



Vamos a mostrar con un ejemplo (ver figura.-Caso1) como desplazando el punto x_1 sobre la arista podemos conseguir que la función $f_\lambda(U;X)$ disminuye hasta que x_1 alcance un vértice o un r_2-extremo. (ver figura.- Caso 1) Como se observa $\{x_1\} \in B_c(4;v_1,v_3)$ y $\{x_2\} \in B_c(5;v_5,v_6)$. Puesto que los radios $r_1 = 4 < r_2 = 5$, x_1 se puede mover hacia v_2 hasta que $r_1 = r_2 = 5$, lo cual hace que la función Centro no se vea afectada, y siga valiendo $max\{r_1,r_2\} = 5$, pero, al mismo tiempo, este movimiento favorece a la función Mediana en dos unidades pues incrementa una unidad respecto a v_1 y disminuye una unidad respecto a cada uno de los vértices v_2, v_3 y v_4. El resultado global de este movimiento hace disminuir la función objetivo en 0.4 unidades, pasando de 8.8 a 8.4. La solución óptima para $\lambda = 0.8$ consiste en $\{x_1^*, x_2^*\}$. El punto x_2^* es un centro local con rango 5 y el punto x_1^* es un r_2^*-extremo.

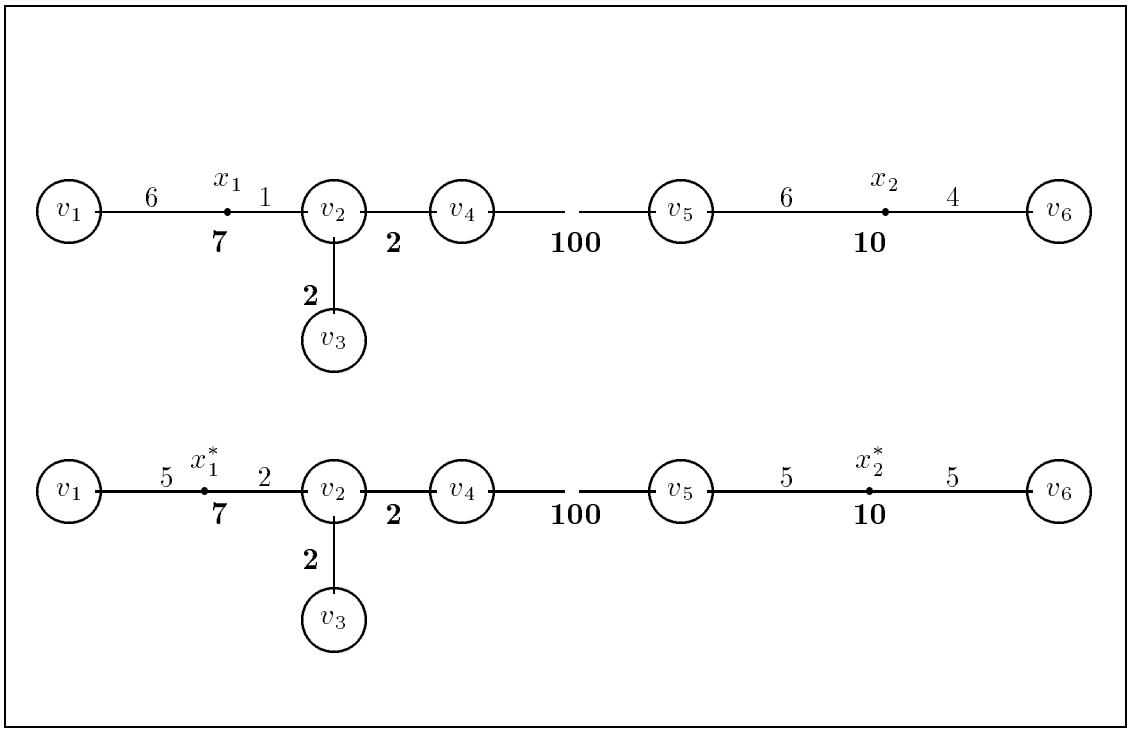
Caso 2. Sea $r_2 < r_1 = r$.



Vamos a mostrar con otro ejemplo (ver figura.-Caso 2) cómo desplazando el punto x_1 sobre la arista podemos conseguir que la función $f_\lambda(U,X)$ no aumente hasta que x_1 alcance un vértice o un centro local. (ver figura-Caso 2-) Se tiene que $\{x_1\}$ no es un centro local respecto a ningún par de vértices, y $\{x_2\} \in B_c(4;v_5,v_6)$. Dado que los radios $r_1 = 5.5 > r_2 = 4$, si se mueve x_1 hacia v_1 hasta que $x_1 \in B_c(5;v_1,v_3)$, es decir, hasta que $r_1 = 5$, se consigue que la función Centdian pase de valer 9.4 a valer 9.2. La solución óptima para $\lambda = 0.8$ consiste en $\{x_1^*, x_2^*\}$.

El punto x_2^* es un centro local con rango 4 y el punto x_1^* es un centro local con rango 5.

Caso 3. Sea $r_1 = r_2 = r$.



Vamos a mostrar con un tercer ejemplo (ver figura.- Caso3), cómo partiendo de dos puntos $\{x_1, x_2\}$ donde ningúno es vértice, o centro local, al desplazarlo simultáneamente sobre sus respectivas aristas podemos conseguir que la función $f_\lambda(U, X)$ no aumente hasta que uno de los dos puntos, $\{x_1, x_2\}$ sea un centro local o un vértice, en este ejemplo, es el punto x_2 el que alcanza un centro local respecto a v_5 y v_6. Como se observa, los respectivos radios pasan de 6 a valer $r_1 = r_2 = 5$, y por tanto la función objetivo del 0.8-Centdian pasa de valer 9.4, a valer 9.0. La única solución óptima para $\lambda = 0.8$ consiste en $\{x_1^*, x_2^*\}$. El punto x_2^* es un centro local con rango 5 y el punto x_1^* es un punto r_2^*-extremo.

3.4.2 Un algoritmo para el 2-λ-Centdian.

La primera consecuencia directa de obtener el conjunto finito dominante para un problema de optimización es que éste se puede resolver de forma exacta con una simple búsqueda exahustiva en dicho conjunto. Y, como es lógico, la complejidad

de tal búsqueda dependerá del tamaño del conjunto finito dominante. Sea $|V| = O(n)$ y $|E| = m$.

Proposición 3.4.1 *El tamaño del conjunto finito dominante para el problema del 2-λ-Centdian en un grafo es:* $|D| = O(m^2n^3)$.

Demostración: Los valores de R son rangos de centros locales, rangos de puntos pendientes y distancias entre vértices. Para cada arista, hay a lo sumo un centro local con respecto a cada pareja de vértices, entonces $|LC| = O(mn^2)$. Para cada arista, hay a lo sumo dos puntos pendientes con respecto a cada pareja de vértices, entonces $|PP| = O(mn^2)$. Por tanto el tamaño de R es $O(mn^2)$. Para cada $r \in R$, hay a lo sumo dos puntos extremos con rango r en cada arista con respecto a cada vértice, esto es $|EP(r)| = O(mn)$. Luego de la unión de $O(mn^2)$ conjuntos de tamaño $O(mn)$ se obtiene que $|D| = O(m^2n^3)$. □

La búsqueda exhaustiva de pares de punto de D para encontrar la solución óptima del 2-λ-Centdian requiere un tiempo $O(m^4n^7)$, porque se necesita un tiempo de $O(n)$ para evaluar la función objetivo en cada par. Sin embargo la complejidad de la búsqueda puede ser reducida, porque no todos los pares de punto de D son candidatos para el del análisis de la función objetivo, se infiere el siguiente resultado.

Proposición 3.4.2 *Sea (x_1^*, x_2^*) una solución óptima del 2-λ-Centdian en G y $r^* = f_c(U; X^*)$. Entonces:*

1. $x_1^* \in LC(r^*) \cup PP(r^*) \cup (V \cap EP(r^*))$ *y* $x_2^* \in V \cup EP(r^*)$, *o*
2. $x_2^* \in LC(r^*) \cup PP(r^*) \cup (V \cap EP(r^*))$ *y* $x_1^* \in V \cup EP(r^*)$.

Demostración: Se sigue del análisis de la función objetivo del 2-λ-Centdian. □

Como lectura de este resultado se tiene que uno de los puntos de servicios de la solución óptima (el que tiene mayor radio), tiene que ser un vértice, un centro local, o un punto pendiente. Este punto fija la distancia máxima a un vértice usuario, es decir, fija el r^*. El otro punto servicio es un vértice o un punto extremo con rango r^*.

En este resultado se sustenta el algoritmo 2-CD-G que resuelve el problema del 2-λ-Centdian en un grafo. (ver figura.- Algoritmo 2-CD-G)

Proposición 3.4.3 *El algoritmo 2-CD-G tiene complejidad $O(m^2n^4)$.*

Algoritmo.- 2-CD-G.

Paso 0. Sea $f^* = \infty$.

Paso 1. Calcular el conjunto P_1 de pares (x, r), donde $x \in LC(r) \cup PP(r)$. Calcular el conjunto P_2 de pares (v_i, r), donde $v_i \in V$ y $r = w'_j d(v_i, v_j)$ para algún $v_j \in U$. Sea L_1 la lista de los pares de $P_1 \cup P_2$.

Paso 2. Tomar un par (x_1, r_1) de la lista L_1, y calcular el conjunto $EP(r_1)$. Sea L_2 la lista de los puntos de $EP(r_1) \cup V$.

Paso 3. Tomar un punto x_2 de la lista L_2, y calcular $f = f_\lambda(U; x_1, x_2)$. Si $f < f^*$ hacer $f^* = f$, $x_1^* = x_1$ y $x_2^* = x_2$, y eliminar el punto x_2 de la lista L_2.

Paso 4. Si L_2 no está vacío, ir al paso 3.

Paso 5. Eliminar (x_1, r_1) de la lista L_1. Si la lista L_1 no está vacía, ir al paso 2. En otro caso, parar.

Figura 3.1: Algoritmo 2-CD-G

Demostración: En primer lugar se necesitan las distancias entre vértices. Calcular esto requiere un tiempo del orden $O(mn \log n)$, usando, por ejemplo el algoritmo de Dijkstra. Los conjuntos P_1 y P_2 son de tamaño $O(mn^2)$ y $O(n^2)$ respectivamente y pueden ser calculados con la misma complejidad. Para cada r, el conjunto $EP(r)$ es calculado en un tiempo de orden $O(mn)$ comprobando cada vértice usuario y cada arista. Por lo tanto, el número de pares de puntos testeados en el paso 3 es del orden $O(m^2n^3)$. Como la función objetivo es evaluada en un tiempo $O(n)$, la complejidad total es del orden $O(m^2n^4)$. □

3.5 El p-λ-Centdian.

El objetivo del p-λ-Centdian es encontrar p puntos $X = (x_1^*, x_2^*, ..., x_p^*) \in P(G)^p$ tal que:

$$f_\lambda(U; (x_1^*, x_2^*, ..., x_p^*)) = \min_{X \in P(G)^p} \{f_\lambda(U; X)\}.$$

De forma natural en este apartado se establecen los resultados apuntados para el 2-λ-Centdian.

3.5.1 El conjunto finito dominante.

Utilizando las definiciones de los conjuntos LC, PP y EP dados en el apartado puntos especiales, y teniendo en cuenta los resultados del apartado el 2-λ-Centdian, se enuncia el siguiente teorema.

Teorema 3.5.1 Teorema p-CD-G

El conjunto $D = V \cup LC \cup PP \cup EP$ consistente en los vértices, los centros locales, los puntos pendientes y los puntos extremos canónicos asociados a los vértices usuarios es un conjunto finito dominante para el problema del p-λ-Centdian en un grafo.

Demostración: Sea $X \in P(G)^p$, una solución candidata al problema del p-λ-Centdian, es decir, una selección de p puntos servicios $X = (x_1, x_2, ..., x_p)$ con $x_k \in P(G)$, $k = 1, ..., p$. Cualquier vértice usuario $v_i \in U$ es asignado a la componente más cercana de $X \in P(G)^p$. El resultado de esta asignación es una serie de conjuntos $U(X) = (U_1(X), U_2(X), ..., U_p(X))$ definidos como sigue:

$$U_k(X) = \{v_i \in U : w_i' \, d(x_k, v_i) = \min_{x \in X} w_i' \, d(x, v_i)\}.$$

Estos conjuntos constituyen una partición del conjunto de vértices usuarios U si y sólo si no existen empates; en cualquier otro caso, una partición óptima se establecerá deshaciendo arbitrariamente dichos empates, obviamente la partición no será única. La asignación dada por: $U^*(X) = (U_1^*(X), U_2^*(X), ..., U_p^*(X))$ es una partición óptima para $X \in P(G)^p$ si ésta es una partición del conjunto de vértices usuarios que verifica:

$$v_i \in U_k^*(X) \Rightarrow w_i' \, d(x_k, v_i) = \min_{x \in X} w_i' \, d(x, v_i).$$

Dada una partición óptima del conjunto de vértices usuarios U, las funciones objetivos del p-Centro y de la p-Mediana pueden ser calculadas como se muestra a continuación:

$$f_c(U; X) = \max_{k=1,...,p} f_c(U_k^*(X); x_k).$$

$$f_m(U; X) = \sum_{k=1}^{p} f_m(U_k^*(X); x_k).$$

Dada una solución candidata X y su correspondiente partición óptima $U^*(X)$, se tiene que asociado con cada x_k existe un radio $r_k(X)$ que representa la mayor distancia en la que se puede encontrar un vértice de $U_k^*(X)$ al punto x_k.

$$r_k(X) = \max_{v_i \in U_k^*(X)} w_i' \, d(x_k, v_i).$$

Entonces:

$$f_c(U;X) = \max_{k=1,...,p} r_k(X) = r^*(X),$$

y

$$f_\lambda(U;X) = \lambda \cdot \max_{k=1,...,p} r_k(X) + (1 \Leftrightarrow \lambda) \cdot f_m(U;X) = \lambda \cdot r^*(X) + (1 \Leftrightarrow \lambda) \cdot f_m(U;X).$$

Supongamos que $x_k \notin D$ para algún $k \in \{1, ..., p\}$. Se probará que el conjunto X puede ser modificado sin incrementar la función $f_\lambda(U;X)$ hasta que $x_k \in D$ para todo k. En la demostración se considerarán dos casos: caso 1.) $r_k(X) < r^*(X)$ y caso 2.) $r_k(X) = r^*(X)$.

Pasamos a ver detalladamente cada uno de estos casos.

Caso 1.) Sea $r_k(X) < r^*(X)$.

Como $x_k \notin D$, éste debe ser un punto interior en la arista $[i,j]$. Se probará cómo se puede mover este punto x_k en su arista sin incrementar la función $f_\lambda(U;X)$ hasta que un vértice sea alcanzado o $r_k(X)$ sea igual a $r^*(X)$. Para ello se necesita analizar la pendiente de $f_\lambda(U;X)$ en términos de la distancia desde x_k al extremo i.

Para cualquier punto $x \in [i,j]$, el conjunto de vértices usuarios que son óptimamente cubiertos por x a través de i se representa por $U^i(x)$; es decir,

$$U^i(x) = \{v_i \in U : w_i' d(x,v_i) = w_i' (l(x,i) + d(i,v_i)) \leq w_i' (l(x,j) + d(j,v_i))\}.$$

Análogamente, a través de j,

$$U^j(x) = \{v_i \in U : w_i' d(x,v_i) = w_i' (l(x,j) + d(j,v_i)) \leq w_i' (l(x,i) + d(i,v_i))\}.$$

Sea $U_k^=(X)$ el conjunto de vértices usuarios que pueden ser asignados a x_k y también a otro x_m, para algún $m \neq k$; es decir,

$$U_k^=(X) = \{v_i \in U : w_i' d(x_k,v_i) = \min_{x \in X} w_i' d(x,v_i) =$$

$$= w_i' d(x_m,v_i), \text{ para algún } m \neq k\}.$$

El siguiente conjunto está formado por los vértices asignados a x_k y que no son asignables a otro servicio:

$$U_k^<(X) = U_k(X) \Leftrightarrow U_k^=(X) =$$

$$= \{v_i \in U : w_i' d(x_k,v_i) = \min_{x \in X} w_i' d(x,v_i) < w_i' d(x_m,v_i), \forall m \neq k\}.$$

Los vértices de $U_k^{=}(X)$ son asignables a x_k y a otro servicio de X, pero si x_k se mueve una pequeña cantidad ξ hacia j o hacia i, encontramos que algunos de estos vértices pueden permanecer asignados a x_k y mientras que otros pueden cambiar de asignación, e incluso vértices que estaban en equilibrio con respecto a más de un servicio pueden dejar de estarlo. El nuevo punto $x_k(\xi)$ denota a x_k cuando éste se ha movido una pequeña cantidad ξ hacia j; es decir, si $x_k = p([i,j],t)$ entonces $x_k(\xi) = p([i,j],t+\xi)$. Luego, aquellos vértices usuarios de

$$U_k^{=}(X) \cap U^j(x_k)$$

son asignados sólo a $x_k(\xi)$ y aquellos de

$$U_k^{=}(X) \Leftrightarrow U^j(x_k)$$

no son asignados a $x_k(\xi)$.

Si $X(\xi)$ denota la nueva solución $(x_1, x_2, ..., x_k(\xi), ..., x_p)$, entonces una nueva partición óptima para $X(\xi)$ es dada por:

$$U_m^*(X(\xi)) = U_m^*(X) \Leftrightarrow [U_k^{=}(X) \cap U^j(x_k)]; \forall m \neq k,$$

y

$$U_k^*(X(\xi)) = U_k^*(X) \Leftrightarrow [U_k^{=}(X) \Leftrightarrow U^j(x_k)].$$

La pendiente de $f_m(U; X(\xi))$, como una función de ξ, depende de los vértices usuarios asignados a x_k (aquellos de $U_k(X(\xi)) = U_k^{<}(X) \cup U_k^{=}(X)$) óptimamente cubiertos a través de j. El valor de esta pendiente es:

$$D_m^j(\xi) = w(U_k^{<}(X) \Leftrightarrow U^j(x_k)) \Leftrightarrow w([U_k^{<}(X) \cup U_k^{=}(X)] \cap U^j(x_k)).$$

Donde $w(W) = \sum_{v_i \in W} w_i, \forall W \subset U$.

Si el movimiento es hacia el vértice i, el nuevo punto de servicio es $x_k(\Leftrightarrow\xi) = p(e, t \Leftrightarrow \xi)$ y, análogamente, $X(\Leftrightarrow\xi)$ indicará la solución $(x_1, x_2, ..., x_k(\Leftrightarrow\xi), ..., x_p)$. La pendiente de $f_m(U; X(\Leftrightarrow\xi))$, se puede expresar como una función de ξ, como se muestra a continuación:

$$D_m^i(\xi) = w(U_k^{<}(X) \Leftrightarrow U^i(x_k)) \Leftrightarrow w([U_k^{<}(X) \cup U_k^{=}(X)] \cap U^i(x_k)).$$

Dado que la suma de las pendientes es:

$$D_m^i(\xi) + D_m^j(\xi) = \Leftrightarrow 2w([U_k^{<}(X) \cup U_k^{=}(X)] \cap U^i(x_k) \cap U^j(x_k)) \leq 0.$$

se deduce que una de los dos pendientes es no positiva. Se asume (el otro caso es similar) que $D_m^j(\xi) \leq 0$.

La pendiente $D_m^j(\xi)$ podría cambiar de no positiva a positiva si y sólo si ocurre uno de los siguientes casos:

a) el conjunto $U_k^<(X) \Leftrightarrow U^j(x_k)$ gana un vértice, o

b) el conjunto $[U_k^<(X) \cup U_k^=(X)] \cap U^j(x_k)$ pierde un vértice.

A continuación se analiza cada uno de ellos.

a) $U_k^<(X) \Leftrightarrow U^j(x_k)$ gana un vértice. Las únicas posibilidades para que esto ocurra son:

   a1) Un vértice deja $U^j(x_k(\xi))$, pero esto no es posible porque el movimiento de x_k se está produciendo hacia j.

   a2) Un vértice que no pertenece a $U^j(x_k(\xi))$ se le asigna a $U_k^<(X)$, pero esto tampoco es posible, pues para este vértice, la distancia a $x_k(\xi)$ aumenta mientras que la distancia a x_m, para $m \neq k$, permanece inalterable.

b) $[U_k^<(X) \cup U_k^=(X)] \cap U^j(x_k)$ pierde un vértice. Esto es imposible porque la distancia desde $x_k(\xi)$ a los vértices de $U^j(x_k)$ decrece, dado que $x_k(\xi)$ se está acercando a j, por lo tanto, ningún vértice de $U^j(x_k)$ puede salir de $U_k^<(X) \cup U_k^=(X)$.

Mientras $r_k(X(\xi)) \leq r^*(X(\xi)) = r^*(X)$ la pendiente de f_c es cero. Como consecuencia de ello, la pendiente de f_λ es:

$D_\lambda^j(\xi) = (1 \Leftrightarrow \lambda) \cdot D_m^j(\xi)$, si el movimiento es hacia j, y

$D_\lambda^i(\xi) = (1 \Leftrightarrow \lambda) \cdot D_m^i(\xi)$, si el movimiento es hacia i.

La suma de estas dos pendientes es:

$$D_\lambda^j(\xi) + D_\lambda^i(\xi) = \underbrace{(1 \Leftrightarrow \lambda)}_{\geq 0} \cdot \underbrace{(D_m^i(\xi) + D_m^j(\xi))}_{\leq 0} \leq 0.$$

Por ello, en cualquier caso, una de las pendientes es no positiva. Esto significa que siempre se puede mover x_k en la dirección en la que la pendiente es no positiva y, por consiguiente, en la dirección donde f_λ no aumenta. Este movimiento se puede realizar hasta que x_k alcance un vértice o hasta que su correspondiente radio r_k sea igual a r^*. Si esto último ocurre, estaríamos en el caso 2.

Caso 2. Sea $r_k(X) = r^*(X)$.

Se distinguen dos subcasos:

2a) $r_m(X) < r_k(X)$, para todo $m \neq k$; es decir, $r^*(X) = \max_{i\in\{1,..,p\}} r_i(X)$ es igual sólo a $r_k(X)$, y

2b) $r_m(X) = r_k(X)$, para algunos $m \neq k$; es decir, $r^*(X) = \max_{i\in\{1,..,p\}} r_i(X)$ es igual a varios $r_i(X)$, $i \in \{1,..,p\}$.

Caso 2a) $r_m(X) < r_k(X) = r^*(X)$, para todo $m \neq k$.

Como en el caso 1, se analiza la pendiente de la función f_λ cuando se mueve $x_k \in [i,j]$ una cantidad ξ hacia j, o hacia i. Sea $x_k(\xi)$ que denota x_k cuando éste ha sido movido una cantidad ξ hacia j, y $x_k(\Leftrightarrow\xi)$ que representa a x_k cuando éste ha sido movido una cantidad ξ hacia i.

Sea $U^* = \{v_i \in U_k^*(X) : w_i' \, d(x_k, v_i) = r^*(X)\}$. Como $x_k \notin D$, no es un centro local, entonces sólo los dos casos siguientes son posibles:

a) $U^* \subset U^j(x_k)$ y $U^* \cap U^i(x_k) = \emptyset$, entonces:

$$r^*(X(\xi)) = r_k(X(\xi)) = r_k(X) \Leftrightarrow w_i' \, \xi.$$

b) $U^* \subset U^i(x_k)$ y $U^* \cap U^j(x_k) = \emptyset$, entonces:

$$r^*(X(\xi)) = r_k(X(\xi)) = r_k(X) + w_i' \, \xi.$$

Por lo tanto la pendiente de $f_c(U, X(\xi))$ puede ser w_i' ó $\Leftrightarrow w_i'$, (y la de $f_c(U, X(\Leftrightarrow\xi))$ es $\Leftrightarrow w_i'$ ó w_i' respectivamente) y la pendiente de f_λ se puede expresar de una de las formas siguientes:

a) $D_\lambda(\xi) = D_\lambda^j(\xi) = \Leftrightarrow\lambda \, w_i' + (1 \Leftrightarrow \lambda) \cdot D_m^j(\xi)$ y

$D_\lambda(\Leftrightarrow\xi) = D_\lambda^i(\xi) = +\lambda \, w_i' + (1 \Leftrightarrow \lambda) \cdot D_m^i(\xi)$.

b) $D_\lambda(\xi) = D_\lambda^j(\xi) = +\lambda \, w_i' + (1 \Leftrightarrow \lambda) \cdot D_m^j(\xi)$ y

$D_\lambda(\Leftrightarrow\xi) = D_\lambda^i(\xi) = \Leftrightarrow\lambda \, w_i' + (1 \Leftrightarrow \lambda) \cdot D_m^i(\xi)$.

En ambos casos, como se puede deducir del análisis del caso 1, la suma de las pendientes tanto cuando x_k se mueve hacia j, como cuando se mueve hacia i es:

$$D_\lambda^i(\xi) + D_\lambda^j(\xi) = \underbrace{(1 \Leftrightarrow \lambda)}_{\geq 0} \underbrace{(D_m^i(\xi) + D_m^j(\xi))}_{\leq 0} \leq 0.$$

Moviendo x_k en una de estas direcciones (hacia i o hacia j) la función f_λ no aumenta hasta que se presenta una de las dos situaciones siguientes:

i) que $x_k(\xi)$ alcance un extremo de la arista (i o j), un centro local, o un punto pendiente con respecto a dos vértices usuarios de U^*, donde la pendiente de $f_c(U_k(X(\xi)); x_k(\xi))$ pueda ser w_i' en ambas direcciones, o

ii) que $r_k(X(\xi))$ tome el mismo valor que algún r_m, con $m \neq k$. Si esto último ocurre, estaríamos en el caso 2b).

Caso 2b) $r_m(X) = r_k(X) = r^*(X)$, para algunos $m \neq k$.

Sea K el conjunto de índices donde $r^*(X)$ es alcanzado, es decir, $k \in K$ si y sólo si, $r_k(X) = r^*(X)$. Si para algún $k^* \in K$, $x_{k^*} \in LC(r^*) \cup PP(r^*) \cup V$. Entonces $r^* \in R$ y $x_k \in EP(r^*)$ $\forall k \in K$, es decir, todos son extremos canónicos. Por tanto $x_k \notin LC(r^*) \cup PP(r^*) \cup V, \quad \forall k \in K$.

Para cada $k \in K$, x_k es un punto interior de una arista, y $x_k \notin LC(r^*) \cup PP(r^*)$. Debido al análisis del caso 2a), si se denota la arista que contiene a x_k por $[i_k, j_k]$, y por w_k' el peso del vértice de $U_k(x)$ más alejado de x_k, y en el caso de que éste no sea único, se elige el de mayor peso, se tiene que la pendiente de $f_c(U_k(X); x_k)$ es w_k' cuando el movimiento de x_k es hacia j_k, y es $\Leftrightarrow w_k'$, cuando el movimiento de x_k es hacia i_k (puede ser necesario el intercambio de los nombres i_k y j_k para algún $v_i \in U$). En este último caso w_k' es el peso del vértice de $U_k(x)$ más alejado de x_k, y en el caso de que éste no sea único, se elige el de menor peso. Se moverán al mismo tiempo todos los puntos x_k, $k \in K$, una cantidad ξ / w_k' en su arista en la dirección en la que simultáneamente cada r_k aumente ó disminuya. Adviertase que el peso w_m' de x_m con $m \neq k$ ha de coincidir con el de w_k, pues en caso contrario x_m sería un punto pendiente.

La pendiente de la función objetivo f_λ puede ser tal y como se muestra en los dos casos siguientes:

a) Si todos los x_k, para $k \in K$, son movidos una cantidad ξ / w_k hacia j_k:

$$D_\lambda(\xi) = +\lambda \; + \; (1 \Leftrightarrow \lambda) \sum_{k \in K} 1/w_k' \; D_k^{j_k}(\xi)$$

b) Si todos los x_k, para $k \in K$, son movidos una cantidad ξ / w_k' hacia i_k:

$$D_\lambda(\Leftrightarrow \xi) = \Leftrightarrow \lambda \; + \; (1 \Leftrightarrow \lambda) \sum_{k \in K} 1/w_k' \; D_k^{i_k}(\xi).$$

Se concluye que una de estas dos pendiente debe ser no positiva, porque la suma de ellas es:

$$(1 \Leftrightarrow \lambda) \sum_{k \in K} 1/w'_k \underbrace{[D_k^{i_k}(\xi) + D_k^{j_k}(\xi)]}_{\leq 0} \leq 0.$$

Por lo tanto, moviendo al mismo tiempo todos los x_k, $k \in K$, en sus correspondientes direcciones (hacia i_k o hacia j_k) la función f_λ no aumenta hasta:

i) que algún x_k alcance un extremo de su arista (i_k o j_k) o

ii) que algún x_k alcance un punto pendiente o un centro local con respecto a dos vértices de U^* con rango $r_k(X(\xi)) = r^*(X(\xi))$ donde la pendiente de $f_c(U_k(X(\xi)); x_k(\xi))$ es 1 en ambas direcciones, o

iii) que $r_k(X(\xi))$ disminuya hasta alcanzar el mismo valor que algún nuevo r_m, con $m \notin K$.

En este último caso un nuevo índice se añade a K y el procedimiento es iterado. Esto completa la demostración. □

3.5.2 Un Algoritmo para el p-λ-Centdian.

El conjunto finito dominante D proporciona un procedimiento rudimentario para resolver el problema: una búsqueda exhaustiva en el conjunto de todas las combinaciones de p puntos de D. La complejidad de este algoritmo depende del tamaño de D. Tal como se demostró anteriormente el conjunto finito dominante D para el problema del p-λ-Centdian en un grafo G tiene tamaño $O(n^3m^2)$, donde n es el cardinal de V y m el número de aristas.

Un algoritmo exacto para resolver el problema del p-λ-Centdian, basado en la búsqueda exhaustiva en el conjunto D^p, tiene complejidad $O(n^{3p+1}m^{2p})$ porque $|D^p| = O(n^{3p}m^{2p})$ y la función objetivo es evaluada en tiempo $O(n)$.

El siguiente resultado permite reducir la complejidad de la búsqueda en el conjunto de candidatos.

Proposición 3.5.1 *Existe una solución óptima X^* para el problema del p-λ-Centdian tal que, si $r^* = f_c(U; X^*)$ es el máximo radio de la solución, entonces cada $x^* \in X^*$ es un centro local, un punto pendiente, un vértice, o un punto extremo con rango r^*. Esto es, $X^* \subset V \cup LC \cup PP \cup EP(r^*)$.*

Demostración: Se sigue de la demostración del teorema p-CD-G. □

Para cada $r \in R$, $|D(r)| = |V \cup LC \cup PP \cup EP(r)| = O(n^2m)$, por la proposición 3.3.1, se puede reducir la búsqueda de la solución óptima al conjunto de combinaciones de p puntos de $D(r)$, para cada $r \in R$. Estas $|R|$ búsquedas de combinaciones de p puntos implica comprobar $O(n^{2p+2}m^{p+1})$ soluciones candidatas. Además la función objetivo es evaluada en un tiempo $O(n)$, esto conlleva que la complejidad total del procedimiento sea $O(n^{2p+3}m^{p+1})$.

La complejidad de la búsqueda puede reducirse aún más, si se observa que uno de los puntos de X^* tiene que ser un centro local, un punto pendiente o un vértice que determine el valor de r^*, y los otros $p \Leftrightarrow 1$ puntos de X^* tienen que ser vértices o puntos extremos con rango r^*.

Proposición 3.5.2 *Sea X^* una solución óptima para el problema del p-λ-Centdian y $r^* = f_c(U; X^*)$ el máximo radio de la solución. Entonces, existe un punto $x^* \in X^*$ tal que $x^* \in LC(r^*)$, $x^* \in PP(r^*)$ o $x^* \in V$ con $w_i' d(x^*, v_i) = r^*$ para algún vértice $v_i \in U$; además los otros $p \Leftrightarrow 1$ puntos de servicios de X^* son vértices o puntos extremos con rango r^*.*

Demostración: Se sigue de la demostración del teorema p-CD-G. □

Proposición 3.5.3 *La complejidad del algoritmo p-CD-G es de $O(n^{p+2}m^p)$*

Demostración:

Considérese DR como el conjunto de pares (*punto*, *rango*) dado por:

$$DR = \{(x,r) : x \in LC(r) \cup PP(r)\} \cup \{(v_i, r) : v_i \in V, r = w_j' d(v_i, v_j), \text{ para } v_j \in U\}.$$

Para cada $(x,r) \in DR$ sólo se necesita buscar aquellas soluciones que contengan a x y $p \Leftrightarrow 1$ puntos de $V \cup EP(r) \cup PP(r)$. Luego $|DR| = O(n^2m)$ pues $|LC| = O(n^2m)$, $|PP| = O(n^2m)$ y $|V| = O(n)$. Además $|EP(r)| = O(nm)$ para cada $r \geq 0$. Por lo tanto, hay que realizar $O(n^2m)$ búsquedas en $(V \cup EP(r))^{p-1}$. Además evaluar cada candidato requiere un tiempo $O(n)$. Por lo tanto, la complejidad del algoritmo exacto es: $O(n^2m \cdot (nm)^{p-1} \cdot n) = O(n^{p+2}m^p)$. □

El siquiente resultado nos permite mejorar un algoritmo para el problema del p-λ-Centdian

Proposición 3.5.4 *Sea $r^* = r^*(U; X^*)$, $r_c^* = r^*(X_c)$ y $r_m^* = r^*(X_m)$ los radios respectivos de una solución óptima para los problemas p-λ-Centdian, del p-Centro, y de la p-Mediana respectivamente. Entonces $r_c^* \leq r^* \leq r_m^*$.*

Demostración: Se sigue de la demostración del teorema p-CD-G □

(ver figura.- Algoritmo p-CD-G).

Algoritmo.- p-CD-G

Paso 1. Obtener la lista L de pares (x, r), consistente en centros locales y puntos pendientes x y sus respectivos radios r. Añadir a L todos los pares (v_i, r), donde v_i es un vértice y $r = w'_j\, d(v_i, v_j)$ para algún $v_j \in U$.

Paso 2. Calcular el p-Centro X_c y la p-Mediana X_m. Sea $r_c = r^*(X_c)$ y $r_m = r^*(X_m)$ sus correspondientes radios máximos.

Paso 3. Para cada $(x, r) \in L$, con $r_c \leq r \leq r_m$ hacer lo siquiente:

i) obtener el conjunto $EP(r)$ de puntos extremos con rango r, y

ii) para cada selección Y de $p-1$ puntos de $U \cap EP(r)$,calcular $f_\lambda(U; \{x\} \cap Y)$.

Paso 4. Mantener el mejor conjunto de p puntos de los evaluados en el Paso 3.

Figura 3.2: Algoritmo p-CD-G

3.6 Conclusiones.

El hecho de introducir pesos a la función Centro en el modelo propuesto por Halpern, pesos que no necesariamente tienen que ser iguales a los existentes para la función Mediana, hace que se pierda simplicidad y se gane generalidad.

Además de generalizar el modelo, se ha estudiado por primera vez el caso múltiple, en el que el objetivo es encontrar p nuevos servicios que minimicen la función λ-Centdian.

Un contraejemplo sobre el conjunto finito dominante existente en la literatura, demuestra por un lado que el estudio del modelo múltiple no es fácil, y por el otro, el interés que este modelo suscita.

El estudio de la función objetivo Centdian múltiple, nos lleva a presentar, **un conjunto finito verdaderamente dominante** para el p-λ-Centdian en un grafo, y como consecuencia del mismo un algoritmo exacto.

Capítulo 4

EL PROBLEMA DEL CENTDIAN EN EL ÁRBOL.

4.1 Introducción.

En este capítulo se resuelve el problema Centdian en un grafo, pero a diferencia del capítulo 3, en éste se estudia el problema en un grafo especial conocido en la literatura como árbol, y del cual ya se ha hablado anteriormente. Dada la importancia que tienen estos tipos de grafos especiales, se ha dedicado un cápitulo al estudio del problema Centdian en el mismo. Proponiendo el primer algoritmo polinomial para el problema p-Centdian (generalizado o no) en árboles. Este es fruto del trabajo en colaboración con A.Tamir. Para el caso generalizado, es decir, cuando la función Centro se considera con pesos, la complejidad de este algoritmo es de $O(pn^6)$, para $p \geq 6$, y $O(n^p)$, para $p < 6$. Para el caso no generalizado, es decir, para el modelo con la función Centro sin pesos, la complejidad disminuye a $O(pn^5)$, para $p \geq 5$, y $O(n^p)$, para $p < 5$.

También se considera la versión discreta del modelo, donde los p puntos deben seleccionarse entre los vértices, en este caso la complejidad se reduce a $O(pn^4)$, para $p \geq 4$, y $O(n^p)$, para $p < 4$.

4.2 Formulación del problema.

Sea $T = (V, E)$ un árbol, donde $U \subseteq V = \{v_1, ..., v_n\}$ representa el conjunto de vértices clientes y E el conjunto de aristas. Cada arista tiene asignada una longitud positiva. Un punto interior de una arista se representa por su distancia

a los extremos de dicha arista. Por $P(T)$ se denota el conjunto continuo de puntos de T. Las longitudes de las aristas inducen una función de distancia en $P(T)$: para cada par de puntos x, y en $P(T)$, tenemos que $d(x, y)$ designa la longitud de $P(x, y)$, el único camino que conecta x con y. También, para cualquier subconjunto $Y \subseteq P(T)$, y $x \in P(T)$ se define:

$$d(x, Y) = d(Y, x) = \min\{d(x, y) | y \in Y\}.$$

Entonces $P(T)$ es un espacio métrico con respecto a la función de distancia mencionada.

Una vez más se van a considerar los principales problemas estudiados en teoría de localización, el p-Centro y la p-Mediana, restringiéndolos a un árbol y usando la notación anterior.

El problema del p-Centro *(w'-pesado)* consiste en seleccionar un subconjunto de p puntos $X \subseteq P(T)$, que minimicen la función objetivo $f_c(U, X)$, donde:

$$f_c(U, X) = \max_{v_i \in U} \{ w_i' d(X, v_i)\}.$$

Recordemos que dado $e = [v_i, v_j] \in E$, Q_e representa el conjunto de puntos mínimos locales de $f_c(U, x)$ sobre e además de v_i y v_j.

Recuérdese también los siguientes conjuntos de puntos especiales:

i) El conjunto de todos los centros locales de rango $r \geq 0$ viene determinado por:

$$LC(r) = \bigcup_{v_k, v_l \in U} B_c(r; v_k, v_l)$$

donde un punto $x \in B_c(r; v_k, v_l)$, si x es un punto interior de un arista $[i, j]$ tal que :

1. $r = w_k' \, d(x, v_k) = w_k' \, (l(x, i) + d(i, v_k)) < w_k' \, (l(x, j) + d(j, v_k))$ y
2. $r = w_l' \, d(x, v_l) = w_l' \, (l(x, j) + d(j, v_l)) < w_l' \, (l(x, i) + d(i, v_l))$.

ii) El conjunto de todos los puntos pendiente con rango $r \geq 0$ viene determinado por:

$$PP(r) = \bigcup_{v_k, v_l \in U} Bp(r; v_k, v_l)$$

donde un punto $x \in Bp(r; v_k, v_l)$, si x es un punto interior de una arista $[i, j]$ tal que :

1. $w'_k\,(l(x,i) + d(i,v_k)) = w'_l\,(l(x,i) + d(i,v_l))$ con $w'_k \neq w'_l$ y / ó
2. $w'_k\,(l(x,j) + d(j,v_k)) = w'_l\,(l(x,j) + d(j,v_l))$ con $w'_k \neq w'_l$.

Además,

$$LC = \underset{r \in R}{\cup} LC(r)$$
$$PP = \underset{r \in R}{\cup} PP(r)$$

siendo R como sigue:

$$R = \{r : LC(r) \neq \emptyset\} \cup \{r : PP(r) \neq \emptyset\} \cup \{r : r = w'_i d(v_i, v_j), v_i \in U, v_j \in V\}.$$

El problema de la p-Mediana (w-pesada) consiste en seleccionar un subconjunto de p puntos $X \subseteq P(T)$, que minimicen la función objetivo $f_m(U, X)$, donde:

$$f_m(U, X) = \sum_{v_i \in U} w_i\, d(X, v_i).$$

Teniendo en cuenta las ventajas y los inconvenientes de cada uno de estos modelos, Halpern [65, 66, 67] sugirió considerar una combinación convexa de la función Centro y la función Mediana, a la que denominó función Centdian. Esta función usada por Halpern tiene una peculiaridad, y es que la función Centro se considera sin pesos ($w'_i = 1, i = 1, ..., n$) y la función Mediana se considera con pesos.

En este capítulo se considera la función Centdian como una combinación convexa de las funciones Centro y Medianas ambas ponderadas generalizando así el modelo de Halpern. Por simplicidad este nuevo problema se seguirá llamando Centdian.

Dado un número p de servicios a ser localizados y un valor λ, $0 \leq \lambda \leq 1$, el problema p-λ-Centdian consiste en encontrar p puntos servicios X en G que minimize la función objetivo:

$$f_\lambda(U; X) = \lambda\, f_c(U, X) + (1 \Leftrightarrow \lambda)\, f_m(U, X).$$

Siguiendo la línea del capítulo anterior, a lo largo de este capítulo a no ser que se especifique lo contrario se tratará siempre la función Centro con pesos

(generalizado), estos pesos de la función Centro no tiene porqué coincidir con los de la función Mediana.

Existen algoritmos eficientes para resolver los problemas del p-Centro y de la p-Mediana en árboles. El algoritmo más eficiente conocido para el problema del p-Centro pesado es de complejidad $O(n \log^2 n \log\log n)$ y es debido a Megiddo y Tamir [118]. Este algoritmo puede ser mejorado a $O(n\ log^2 n)$ si se aplica la modificación de Cole [17]. Para el caso no pesado, es decir, cuando $w_i' = 1,\ i = 1, ..., n$, Frederickson [47] desarrolló un algoritmo óptimo de complejidad $O(n)$.

El algoritmo más eficiente conocido para la p-Mediana pesada es el algoritmo de Tamir [163] el cual tiene complejidad $O(pn^2)$. No existe en la literatura un algoritmo de tiempo polinomial para el problema del p-Centdian (generalizado o no) en un árbol, para un p general. Algoritmos eficientes para el caso $p = 1$ (no generalizado) se encuentran en Halpern [65] y Handler [72].

4.3 El 1-λ-Centdian.

Es bien conocido que la 1-Mediana de un grafo tiene la propiedad de ser localizada en al menos un vértice del grafo. El 1-Centro sin embargo, tiene que estar localizado en un vértice o en un centro local, y por tanto puede estar localizado a lo largo de la arista de un grafo. El Centdian posee una propiedad similar a la del Centro, ésta queda recogida en el teorema 4.3.2.

Recordemos que el objetivo del 1-λ-Centdian reside en encontrar un punto $x^* \in P(T)$ tal que:

$$f_\lambda(x^*) = \min_{x \in P(T)} \{\lambda\, f_c(U, x) + (1 \Leftrightarrow \lambda)\, f_m(U, x)\}.$$

4.3.1 Propiedades del 1-λ-Centdian.

Tomando como base los resultados de mayor relevancia para el problema del 1-λ-Centdian (con $w_i' = 1, \forall v_1 \in U$), a continuación se presentan su reformulación para la extensión aquí propuesta.

Proposición 4.3.1 *Para cualesquiera dos puntos $x, y \in T$, y para $\lambda \in [0, 1]$ dado, la función f_λ es, a lo largo del camino $P(x, y)$, continua, convexa y lineal a trozos, cuyos puntos de ruptura pertenecen a $V \cup LC \cup PP$.*

Demostración: La función Centro es continua, convexa y lineal a trozos, y con puntos de ruptura en $V \cup LC \cup PP$ y la función Mediana es continua, convexa y lineal a trozos, y con puntos de ruptura en V, esto implica que la suma de ambas funciones verifique el enunciado de la proposición 4.3.1. □

De la proposición anterior se tiene que, un mínimo local es un mínimo global. Halpern [65] afinó aún más este resultado, y lo aplicó de forma concreta al camino que une el centro absoluto del árbol, x_c, y la mediana más cercana a él, x_m.

Proposición 4.3.2 *Dado un* $\lambda \in [0,1]$ *la función* $f_\lambda(U,x) = \lambda f_c(U,x) + (1 \Leftrightarrow \lambda) f_m(U,x)$, *con* $x \in P(x_m, x_c)$, *es continua, convexa y lineal a trozos, con puntos de ruptura en los vértices, centros locales y puntos pendientes de* $P(x_m, x_c)$.

Demostración: Es consecuencia de la proposición 4.3.1. □

Sea EF_T el conjunto de todos los puntos eficientes del problema de minimizar $\{f_c, f_m\}$ en el árbol T.

Teorema 4.3.1 *Sea* x_c *el centro del árbol* T, x_m *la mediana más cercana a* x_c, *y* λ-CD *el conjunto de todos los* λ-*Centdian para un* $\lambda \in [0,1]$. *Entonces:*

$$EF_T = P(x_m, x_c) = \cup\{\lambda \Leftrightarrow CD : \lambda \in [0,1]$$

Demostración: Dado un $\lambda \in [0,1]$ la función $f_\lambda(U,x) = \lambda f_c(U,x) + (1 \Leftrightarrow \lambda) f_m(U,x)$, con $x \in P(x_m, x_c)$, es continua, convexa y lineal a trozos. Además, por la proposición 4.3.2, se sabe que los puntos de ruptura son los vértices, los centros locales y los puntos pendientes de $P(x_m, x_c)$. □

Teorema 4.3.2 *Dado* $\lambda \in [0,1]$ *un* 1-λ-*Centdian está localizado en uno de los vértices, en un* LC *o en un* PP *del camino* $P(x_m, x_c)$.

Demostración: Es consecuencia de las proposiciones 4.3.1 y 4.3.2. □

Vamos a ver a continuación una proposición que generaliza el resultado de Halpern (1976) [65], proporcionándonos una regla sencilla para localizar el 1-λ-Centdian, pero antes necesitamos alguna notación:

Sea $\{v_0, v_1, ..., v_k\}$ el conjunto de vértices del camino $P(x_m, x_c)$, con $x_m = v_0$ y $x_c \in [v_k, v_{k+1}]$ y quizás $x_c = v_{k+1}$. Con U_j nos referimos al conjunto de vértices

usuarios que permanecen conectados a v_j si eliminamos del árbol las aristas $[v_{j-1}, v_j]$ y $[v_j, v_{j+1}]$. Así mismo, U_0 (U_{k+1}) representa el subconjunto de vértices que permanecen conectados a v_0 (v_{k+1}) si eliminamos la arista $[v_0, v_1]$ ($[v_k, v_{k+1}]$). Definimos: $w(U_j) = \sum_{v_j \in U_j} w_j$.

Proposición 4.3.3 *La pendiente de $f_m(U, x)$ en $[v_t, v_{t+1}]$ para cada $t = 0, ..., k$ es $D_m(t)$ donde:*

$$D_m(t) = \sum_{j=0}^{t} w(U_j) \Leftrightarrow \sum_{j=t+1}^{k+1} w(U_j) = w(T_t) \Leftrightarrow w(T'_{t+1}), t = 0, ..., k,$$

siendo T_t el subárbol generado por $\bigcup_{j=0}^{t} U_j$ y T'_{t+1} el generado por $\bigcup_{j=t+1}^{k+1} U_j$.

Demostración: Se tiene que la función Mediana:

$$f_m(U, x([v_t, v_{t+1}], s)) = \sum_{v_i \in T_t} w_i[d(v_i, v_t) + s] \Leftrightarrow \sum_{v_i \in T'_{t+1}} w_i[d(v_i, v_{t+1}) + l(v_t, v_{t+1}) \Leftrightarrow s].$$

□

Por otra parte, existen $q_i \in Q_e \cap P(x_m, x_c), \quad i = 1, ..., s$ puntos de ruptura de la función f_c en el camino $P(x_m, x_c)$.

Proposición 4.3.4 *La pendiente $D_c(q_i)$ de $f_c(U, X)$ en $[q_i, q_{i+1}]$ es el mayor de los pesos de los vértice $v_j \in U$ donde se alcanza el máximo:*

$$\max\{w'_j d(q_{i+1}, v_j) : v_j \in U\}.$$

Demostración: Se tiene que la función Centro:

$$f_c(U, x([q_i, q_{i+1}], s)) = w_{j^*}[d(q_{i+1}, v_j) + l(q_i, q_{i+1}) \Leftrightarrow s]$$

donde

$$j^* = argmax\{w'_j : w'_j d(q_{i+1}, v_j) = f_c(q_{i+1})\}.$$

□

Proposición 4.3.5 *La pendiente de la función f_λ en cada $[q_i, q_{i+1}]$ es:*

$$D_\lambda(q_i) = \lambda D_c(q_i) + (1 \Leftrightarrow \lambda) D_m(q_i)), \; i = 1, ..., r$$

Demostración: Se tiene de las dos proposiciones anteriores. □

La siguiente proposición se basa en el hecho de que, al ser f_λ una función convexa, un mínimo local es también un mínimo global. Por lo tanto, podemos hallar una solución óptima buscando una dirección en la que f_λ sea decreciente. Antes de pasar al siguiente resultado, se requiere aclarar la notación.

Si algún q_i, con $i = 1, ..., s$, no es un vértice, entonces $q_i \in [v_t, v_{t+1}]$, para algún $t = 0, ..., k$, y $D_m(q_i) = D_m(t)$.

Proposición 4.3.6 *Sea* $\lambda \in [0, 1]$.

(i) *Si* $D_\lambda(v_t) \leq 0 < D_\lambda(v_{t+1})$. *Entonces los λ-Centdianes están en la arista* $[v_t, v_{t+1}]$.

(ii) *Si* $D_\lambda(q_i) < 0 < D_\lambda(q_{i+1})$. *Entonces el único λ-Centdian es el punto* q_i.

(iii) *Si* $D_\lambda(q_i) = 0$. *Entonces todos los puntos de la subarista* $[q_i, q_{i+1}]$ *son λ-Centdianes.*

Demostración: La proposición se sigue del desarrollo de la función $f_\lambda(x)$ con $x \in P(x_m,\ x_c)$, como se afirma en la proposición 4.3.2, y del hecho de que la pendiente de la función $f_\lambda(x)$ a lo largo del camino $P(x_m,\ x_c)$ sea:

$$D_\lambda(q_i) = \lambda D_c(q_i) + (1 \Leftrightarrow \lambda) D_m(q_i), \forall x \in [q_i, q_{i+1}] \subset [v_t, v_{t+1}].$$

□

Con este resultado es sencillo determinar la localización de los 1-λ-Centdian como una función de λ y de w_j^*.

Esta proposición nos permite dar un algoritmo para localizar los 1-λ-Centdian de un árbol para un λ dado que denominamos Algoritmo de Halpern generalizado, (ver figura.-Algoritmo de Halpern generalizado) y está basado en el algoritmo de Halpern (1976) [65].

Algoritmo de Halpern generalizado.

Paso 1. Usar el algoritmo el algoritmo de Megiddo [112], para hallar el Centro pesado x_c de T.

Paso 2. Usar el algoritmo de Goldman [58] para hallar las Medianas de T. Sea x_m la mediana más cercana a x_c.

Paso 3. Calcular los puntos de ruptura $q_i, i = 1..r$ de la función f_c en el camino $P(x_m, x_c)$.

Paso 4. Realizar una búsqueda dicotómica en dichos puntos de ruptura a lo largo del camino $P(x_m, x_c)$, hasta encontrar el punto donde la pendiente de la función f_λ cambie de signo. Dicho punto representa el 1-λ-Centdian del árbol T.

Figura 17. **Algoritmo de Halpern generalizado**

El Algoritmo de Halpern consistía en hallar el Centro no pesado x_c de T, usando el algoritmo de Handler [68]. Si x_c es un vértice hacer $w(x_c) = w(x_c) + \lambda/(1 \Leftrightarrow \lambda)$. En otro caso, añadir x_c al conjunto de vértices de T (reemplazando la arista $[v_k, v_{k+1}]$ que contiene a x_c por dos nuevas aristas $[v_k, x_c]$ y $[x_c, v_{k+1}]$) y asignarle el peso $w(x_c) = \lambda/(1 \Leftrightarrow \lambda)$; obtenemos así un nuevo árbol T'). Finalmente, aplicar el algoritmo de Goldman [58] para hallar las Medianas de T'. El conjunto de puntos λ-Centdian de T coincide con el conjunto de Medianas de T'.

Proposición 4.3.7 *La complejidad del algoritmo de Halpern generalizado es de* $O(n \log n)$.

Demostración: Por ser los puntos de ruptura de la función centro, centros locales y puntos pendientes se tiene que $r = O(n^2)$. Por tanto la búsqueda dicotómica realiza $O(\log n)$ cálculos de las pendientes D_m y D_c. Dado que éstas se calculan en tiempo $O(n)$, la complejidad del algoritmo de Halpern generalizado es $O(n \log n)$. □

Recuérdese que el algoritmo de Halpern es de complejidad $O(n)$. En la siguiente sección, se va a extender un algoritmo $O(n)$ para el λ-CD en un árbol T.

4.3.2 Un algoritmo $O(n)$ para el 1-λ-Centdian en un árbol.

Denotemos por x^*, y^* y z^* al Centro, a la Mediana y al λ-Centdian del árbol T. Como ya se ha expuesto, el Centdian z^* siempre está en el camino

del Centro a la Mediana. Sea C^* dicho camino. Para cada valor r del intervalo $[f_c(T,x^*), f_c(T,y^*)]$ existe un único punto, que denotaremos $z(r)$ del camino C^* tal que $f_c(T,z(r)) = r$.

El Centro x^* de T se determina por el algoritmo de Megiddo [112] y la Mediana y^* de T por el algoritmo de Goldman [58]. Para hallar el 1-λ-Centdian, se precisa de la pendiente de la función objetivo de la Mediana en las aristas del camino $C(x^*,y^*)$. Para ello el algoritmo de Goldman no se detiene al encontrar la(s) mediana(s) sino que continua hasta que se hayan eliminado todas las aristas del árbol.

Recordemos que cuando en el algoritmo de Goldman se elimina la arista $[v,u]$ incidente en la hoja v, el valor del peso $w(v)$ acumulado en v es la suma de los pesos de todos los vértices que se han contraido en v. Entonces, para los puntos interiores a la arista $[v,u]$, el valor $w(v)$ representa la suma de las pendientes de las distancias ponderadas a todos los vértices usuarios a los que se accede a través de v. Al resto de los vértices usuarios se accede a través de u y la suma de las pendientes correspondientes es la diferencia entre el peso total de todos los usuarios $w(U)$ y el valor actual de $w(v)$. Por tanto, la pendiente de $f_m(x)$ en dicha arista recorrida de v a u es el valor de $2w(v) \Leftrightarrow w(U)$.

La pendiente de la función Centdian es

$$D_\lambda(x) = (1 \Leftrightarrow \lambda) D_m(x) \Leftrightarrow \lambda D_c(x)$$

siendo D_m la pendiente de la función Mediana y D_c la pendiente de la función Centro. Sean $q_1, q_2, ..., q_r$ los puntos de no derivabilidad de la función Centro f_c del camino C de x^* a y^* (es decir, tal que $x^* = q_1$ y $y^* = q_r$) proporcionados por el algoritmo de Megiddo. En cada arista $[v_i, v_{i+1}]$ del camino de C, la pendiente de la función Mediana $f_m(x)$ viene dada por el algoritmo de Goldman. Si el algoritmo de Goldman se implementa de forma que el vértice en el que se contrae el árbol sea la Mediana entonces la Mediana queda como raíz del árbol y la pendiente en $[v_i, v_{i+1}]$ es $D_i = 2w(v_i) \Leftrightarrow w(U)$.

El λ-Centdian se determina calculando el punto del camino C^* del Centro x^* a la Mediana y^* en el que la pendiente de $f_\lambda(x)$ cambia de signo. Por tanto basta recorrer los puntos q_i del camino C desde x^* hacia y^* hasta que $D_\lambda = (1 \Leftrightarrow \lambda) D_m(x) \Leftrightarrow \lambda D_c(x) \geq 0$.

El algoritmo de Megiddo y el de Goldman extendido hasta comprimir todo el árbol en la Mediana son de complejidad $O(n)$. El camino de x^* a y^* se obtiene en tiempo $O(n)$. El número de puntos q_i de ruptura de la función Centro f_c en dicho camino es $O(n)$ ya que se trata del máximo de n funciones lineales. Por tanto el esfuerzo para determinar el λ-centdian es $O(n)$.

4.4 El p-λ-Centdian.

Cada vértice $v_i \in U$ tiene asociado dos pesos no negativos, (w_i', w_i). El primero respecto la función Centro y el segundo respecto a la función Mediana. Con el objetivo de simplificar el desarrollo se considera que:

$$\begin{aligned} w_i' &= \lambda w_i' \\ w_i &= (1 \Leftrightarrow \lambda) w_i. \end{aligned}$$

Nuestra consideración específica del problema p-Centdian en un árbol T estriba en encontrar un conjunto de p puntos $X \subseteq P(T)$, que minimicen la función objetivo:

$$f_\lambda(U, X) = f_c(U, X) + f_m(U, X).$$

Existe una estrecha relación entre el problema del p-Centdian y el problema de la p-Mediana restringida, hecho que permite encontrar algoritmos polinomiales para el problema tratado. Para cada real r no negativo, se define el problema de la p-Mediana r-restringida como el problema de minimizar la suma de las distancias de los vértices w-pesados al conjunto de las p medianas, dado que las distancias w_i'-pesadas de cada vértice al conjunto de las p Medianas es a lo sumo r. El conjunto de las p Medianas restringidas no es necesariamente un subconjunto de vértices, como ocurría en las p Medianas no restringidas. Por $m(r)$ se denota el valor óptimo del problema de la p-Mediana r-restringida. Entonces,

$$m(r) = \min_{\{X : X \subseteq P(T), |X|=p\}} \{f_m(U, X) : w_i' d(X, v_i) \leq r, \quad \forall v_i \in U\}$$

Para cada r no negativo sea $g(r) = r + m(r)$. Lo cual nos permite reformular el problema del p-Centdian como:

$$f_\lambda^* = \min_{|X|=p} f_\lambda(U, X) = \min_r \{g(r)\}.$$

El valor en el que se alcanza el mínimo de la función $g(r)$, se representará por r^*.

4.4.1 Un Conjunto Finito Dominante

En esta sección identificamos un conjunto R de cardinalidad $O(n^3)$ que contiene a r^*, como la solución óptima del problema p-Centdian.

Teorema.-(p-CD-árbol) *Sea r^* el valor que minimiza la función $g(r) = r + m(r)$. Entonces, r^* es un elemento del conjunto*

$$R = R_1 \cup R_2 \cup R_3,$$

donde

$$R_1 = \{w_i' \, d(v_i, v_j) : v_i \in U, v_j \in V\}$$

$$R_2 = \{d(v_i, v_j)/(1/w_i' + 1/w_j') : v_i, v_j \in U\}$$

$$R_3 = \{(d(v_j, v_k) \Leftrightarrow d(v_i, v_k))/(1/w_j' \Leftrightarrow 1/w_i') : \ v_i, v_j \in U, v_k \in V, v_k \in P(v_i, v_j)\}$$

Demostración: Sea X' una solución óptima del problema del p-Centdian. Entonces, existe una partición de G en p subárboles, $T_1, ..., T_p$, donde todos los vértices usuarios de cada subárbol T_j son servidos por el mismo punto, digamos x_j', de X', es decir, $d(v_i, X') = d(v_i, x_j')$, para cada vértice v_i en $U_j = U \cap T_j$. En particular, el subproblema correspondiente a $U_j, j = 1, ..., p$, es un problema de la 1-Mediana r^*-restringida. Para un real no negativo r, consideremos $m_j(r)$ como el valor objetivo del subproblema de la 1-Mediana r-restringida, definida en U_j.

Veamos que $m_j(r)$ es una función convexa, decreciente, y lineal a trozos de r, con puntos de rupturas en el conjunto R definido en el enunciado del teorema. Sea x_j^* la única solución del problema 1-Centro w'-pesado en U_j, y sea y_j^*, el vértice de U_j solución del problema de la 1-Mediana w-pesadas (no restringido) de T_j, más cercana a x_j^*. Considérese el camino $P(x_j^*, y_j^*)$, que conecta x_j^* e y_j^* desde y_j^* hacia x_j^* como un segmento de longitud $d(x_j^*, y_j^*)$, y analicemos la función 1-Centro w'-pesado, $f_c^j(x_j)$, y la función 1-Mediana w-pesada (no restringida) $f_m^j(x_j)$, definida en este camino (segmento). La función Mediana, $f_m^j(x_j)$, es convexa, decreciente desde y_j^* hacia x_j^*, y lineal a trozos en x_j. Sus puntos de rupturas se corresponden a los vértices de T_j en el camino $P(x_j^*, y_j^*)$. Similarmente, la función Centro, $f_c^j(x_j)$, es convexa, creciente desde y_j^* hacia x_j^*, y lineal a trozos en x_j. Cada punto de ruptura de esta función es un punto x en $P(x_j^*, y_j^*)$ cuyas distancias w'-pesadas desde dos vértices de U_j, digamos v_i y v_t son iguales, es decir, $w_i' d(v_i, x) = w_t' d(v_t, x)$.

Sean r_j' y r_j'' que representan, respectivamente, los valores de $f_c^j(x_j)$ en x_j^* e y_j^*. Para cada valor de r, en el intervalo $[r_j', r_j'']$, $x_j(r)$ denota la única solución de la ecuación $f_c^j(x_j) = r$ en $P(x_j^*, y_j^*)$. En particular, $x_j(r)$ es una función de r concava, monótona creciente, y lineal a trozos. La concavidad se deriva de la convexidad de $f_c^j(x_j)$. Luego, si r es un punto de ruptura de $x_j(r)$, entonces: $w_t' d(v_t, x_j(r)) = r$.

Para cada valor de r, en $[r_j', r_j'']$, la solución del problema de la 1-Mediana r-restringida se alcanza en el punto más cercano a y_j^* en el conjunto $\{z : z \in P(T_j) : w_i' d(v_i, z) \leq r, \forall v_i \in U_j\}$. Como este conjunto contiene a x_j^*, la solución

óptima del problema de la 1-Mediana r-restringida en U_j se alcanza en un punto x_j de $P(x_j^*, y_j^*)$ para el cual $f_c^j(x_j) = r$, es decir, en $x_j(r)$. En definitiva,

$$m_j(r) = \begin{cases} \infty & r < r'_j \\ f_m^j(x_j(r)) & r'_j \leq r < r''_j \\ m_j(r''_j) & r \geq r''_j \end{cases}$$

De lo expuesto anteriormente se sigue que $m_j(r)$ es una función convexa, decreciente y lineal a trozos. Además, si r es un punto de ruptura de $m_j(r)$, entonces pueden ocurrir dos casos:

a) $x_j(r)$ es un vértice de T_j en $P(x_j^*, y_j^*)$, o

b) $x_j(r)$ es un punto de ruptura de la función Centro $f_c^j(x_j)$.

Veamos más detalladamente cada uno de estos dos casos:

a) Si $x_j(r)$ es un vértice de T_j, digamos v_k, entonces de la definición de $f_c^j(x_j)$, existen algunos vértices, digamos v_i de U_j, tales que $r = w'_i d(v_i, v_k)$. Así que, en este caso $r \in R_1$.

b) Si $x_j(r)$ es un punto de ruptura de $f_c^j(x_j)$, entonces existen un par de vértices en U_j, digamos v_i y v_t, tales que $w'_i d(v_i, x_j(r)) = w'_t d(v_t, x_j(r)) = r$.

b1) Si $x_j(r)$ está en $P(v_i, v_t)$, el camino que une v_i con v_t, entonces: $d(v_i, v_t) = d(v_i, x_j(r)) + d(x_j(r), v_t) \Rightarrow r = d(v_i, v_t)/(1/w'_i + 1/w'_t)$ y, por lo tanto, $r \in R_2$.

b2) Si $x_j(r)$ no está en $P(v_i, v_t)$, cosideremos v_k el vértice más cercano a $x_j(r)$ en $P(v_i, v_t)$. Por lo tanto, tenemos $r = (d(v_t, v_k) \Leftrightarrow d(v_i, v_k))/(1/w'_t \Leftrightarrow 1/w'_i)$, y $r \in R_3$.

Se ha mostrado que para cada $j = 1, ..., p$, los puntos de ruptura de la función $m_j(r)$ pertenecen al conjunto R definido en el enunciado del teorema. Para concluir la demostración, obsérvese que r^*, la solución óptima del problema p-Centdian, es el valor que minimiza la función:

$$g'(r) = r + \sum_{j=1}^{p} m_j(r).$$

$g'(r)$ es una función lineal, y sus puntos de ruptura coinciden con los puntos de ruptura de las funciones $m_j(r)$, $j = 1, ..., p$. Por lo tanto, el punto que minimiza a $g'(r)$ es uno de estos puntos de ruptura. Esto concluye la demostración del teorema. □

Nótese que la casuística de este teorema para el caso no pesado, es decir,

cuando $w_i' = w_i = 1$, para todo $i = 1, ..., n$, puede ser obtenida de los resultados de Pérez-Brito y otros. [143, 144]. Adviértase que en el modelo no generalizado, es decir, cuando $w_i' = 1$, para todo $i = 1, ..., n$, el valor óptimo r^* se encuentra en el conjunto $R = R_1 \cup R_2$.

4.5 La p-Mediana r-restringida.

Para resolver el problema del p-Centdian en un árbol T, será suficiente computar la solución de la p-Mediana r-restringida para todos los valores de $r \in R$, especificados en el Teorema p-CD-árbol. En este apartado se muestra cómo resolver el problema r-restringido en tiempo polinomial, adaptando y modificando el algoritmo de la p-Mediana de Tamir [163]. El algoritmo de Tamir [163] resuelve el problema de la p-Mediana en árboles en un tiempo $O(pn^2)$.

Recuérdese que el conjunto de todos lo puntos extremos de rango r viene determinado para cada valor de $r \geq 0$ por:

$$EP(r) = \bigcup_{v_i \in U} E_p(r; v_i).$$

donde $x \in E_p(r; v_i)$, si x es un punto interior de un arista $[i, j]$ tal que :

1. $r = w_i' d(x, v_i) = w_i'(l(x, i) + d(i, v_i))$ o
2. $r = w_i' d(x, v_i) = w_i'(l(x, j) + d(j, v))$.

Proposición 4.5.1 *El conjunto: $Y(r) = V \cup EP(r)$, es finito dominante para el problema de la p-Mediana r-restringida.*

Demostración: Dado que $EP(r)$ contiene todos los puntos extremos a distancia w_i'-pesado de r de un vértice de U, y dado que una solución se encuentra en el conjunto de vértices usuarios, Hakimi (1964) [62] se sigue que $Y(r)$ es un conjunto finito dominante para el problema de la p-Mediana r-restringida. □

Proposición 4.5.2 *El conjunto $Y(r)$ es de cardinalidad $O(n^2)$.*

Demostración: Por cada arista, hay a lo sumo un punto extremo de radio r con respecto a cada vértices, por lo tanto la cardinalidad de $Y(r)$ es de $O(n^2)$. □

En Kim y otros, [94], se muestra cómo calcular el conjunto $Y(r)$, y aumentar con sus puntos el conjunto de vértices de T, todo ello en un tiempo $O(n^2)$.

Representamos por $T(r)$ el árbol al que se le ha insertado como nuevos vértices el conjunto de puntos $EP(r)$.

Como $T(r)$ tiene $O(n^2)$ vértices, el algoritmo de Tamir (1996) [163], resuelve el problema de la p-Mediana sin restringir, en un tiempo $O(pn^4)$.

Sabiendo que para resolver el problema del p-Centdian en T se requiere la solución de la p-Mediana r-restringida, para los $O(n^3)$ valores de r, la complejidad total es $O(pn^7)$. En la siguiente sección se muestra de que manera se puede reducir esta complejidad a $O(pn^6)$, mejorando la complejidad de la p-Mediana r-restringida a $O(pn^3)$.

Para el modelo no generalizado la complejidad sería de $O(pn^5)$. Ya que los posibles valores de r^* ($r^* \in R_1 \cup R_2$) pasan a ser del orden $O(n^2)$.

4.5.1 Un algoritmo de complejidad $O(pn^3)$ para el problema de la p-Mediana r-Restringida.

En este apartado se propone un algoritmo dinámico, para el problema de la p-Mediana r-restringida.

El algoritmo consta de tres fases previas: en la primera se transforma el árbol inicial en un árbol binario, en la segunda se prepara el árbol binario para resolver el problema de la p-Mediana r-restringida, y en la tercera se calculan y ordenan las distancias entre los vértices del árbol. Sea v_1 la raíz del árbol.

Definición 4.5.1 *Para cada par de vértices v_i, v_j, el vértice v_i es* **descendiente** *de v_j, si v_j está en el único camino que conecta v_i con la raíz v_1.*

Definición 4.5.2 *El vértice v_i es un* **hijo** *de v_j, si v_i es descendiente de v_j y v_i está conectado a v_j con una arista.*

Definición 4.5.3 *El vértice v_j es* **padre** *de v_i, si v_i es descendiente de v_j y v_i está conectado a v_j con una arista.*

Definición 4.5.4 *Una* **hoja** *del árbol, es un vértice que no tiene hijos.*

Fase 1. Transformación en un árbol binario

Como muestra Tamir [163], es conveniente transformar el árbol dirigido T en un árbol binario equivalente T_{bin}, en el que cada vértice no hoja v_j tenga exáctamente dos hijos, $v_{j(1)}$, $v_{j(2)}$. Al primero se le denominará **hijo izquierdo**, y al segundo **hijo derecho**.

Definición 4.5.5 *El conjunto V_j es el que contiene todos los vértices descendientes del vértice v_j.*

Definición 4.5.6 *El subárbol T_j es el inducido por el conjunto V_j. El vértice v_j será considerado la raíz de T_j.*

La transformación se realiza como sigue:

1. Se considera cada vértice no hoja v_j del árbol original con un sólo hijo, digamos $v_{j(1)}$, se introduce un nuevo vértice $u_{j(1)}$ con peso cero, y se conecta a v_j con una nueva arista de longitud cero. Por lo que, $u_{j(1)}$ además de ser el segundo hijo de v_j, será una hoja del nuevo árbol.

2. Se considera cada vértice v_j del árbol original con al menos tres hijos, digamos $v_{j(1)}, ..., v_{j(t)}$, $t \geq 3$, se añaden t-2 nuevos vértices, $u_{j(2)}, ..., u_{j(t-1)}$ todos ellos con peso cero, para cada $z = 2, ..., t$-1, se reemplaza la arista $[v_j, v_{j(z)}]$ por la arista $[u_{j(z)}, v_{j(z)}]$, también, se reemplaza la arista $[v_j, v_{j(t)}]$ por esta otra arista $[u_{j(t-1)}, v_{j(t)}]$. La longitud de cada una de estas nuevas aristas es la longitud de la arista que sustituye.

3. Finalmente, se añaden las siguientes aristas de longitud cero:

$$[v_j, u_{j(2)}], [u_{j(2)}, u_{j(3)}], ..., [u_{j(t-2)}, u_{j(t-1)}].$$

La figura 24 (Árbol binario) muestra un ejemplo donde un vértice con cinco hijos $t = 5$ que al añadirle $t \Leftrightarrow 2 = 3$ nuevos vértices con peso cero, $u_{j(2)}, u_{j(3)}, u_{j(4)}$, del modo descrito anteriormente, se transforma en un árbol binario.

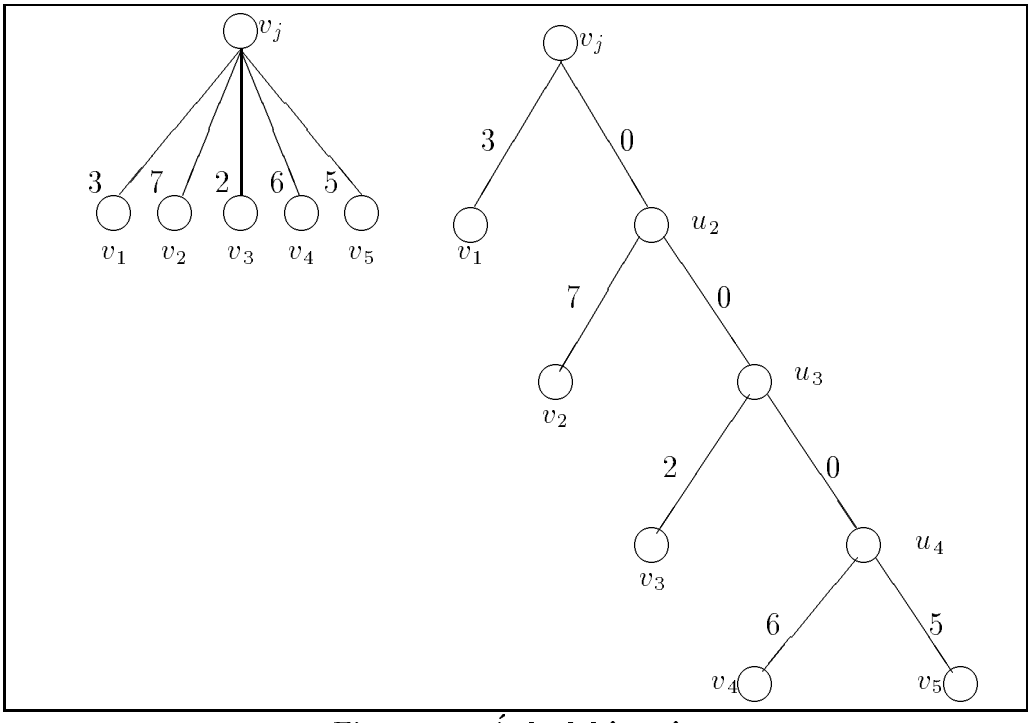


Figura 18. **Árbol binario**

Despúes de realizar este proceso para cada vértice v_j del árbol original T, el nuevo árbol tendrá a lo más $2n \Leftrightarrow 3$ vértices.

Proposición 4.5.3 *Resolver el problema de la p-Mediana (restringida o no) en el árbol original equivale a resolverlo en el nuevo árbol binario* T_{bin}.

Demostración: Dado que se verifica

$$d_T(v_i, v_j) = d_{T_{bin}}(v_i, v_j), \forall v_i, v_j \in T$$

la función objetivo en ambos árboles es la misma. □

Fase 2. Inserción del conjunto dominante

En la segunda fase aumentamos el conjunto de vértices de T_{bin}, con el conjunto de puntos $Y = Y(r)$, definidos en la sección previa. (No debe olvidarse que hay una solución óptima al problema r-restringido, donde los p puntos seleccionados están en Y).

Definición 4.5.7 *Un* **semi-vértice** *es un punto del conjunto* $Y = V + EP(r)$.

Definición 4.5.8 *Para cada vértice v_j, Y_j representará el subconjunto de semi-vértices y tales que v_j está en el único camino de y a la raíz v_1.*

Como se ha mencionado anteriormente la cardinalidad de Y es $m = O(n^2)$.

Incrementar los vértices de T_{bin} con el conjunto $EP(r)$ puede realizarse en tiempo $O(n^2)$, como se describe en Kim y otros [94].

Fase 3. Lista ordenada de distancias

En la tercera fase, para cada vértice v_j se calculan y ordenan las distancias desde v_j a todos los semi-vértices de Y. Esta secuencia es representada por $L_j = \{r_j^1, ..., r_j^m\}$, donde $r_j^i \leq r_j^{i+1}$, $i = 1, ..., m \Leftrightarrow 1$, y $r_j^1 = 0$. Por conveniencia, y para poder controlar el caso de empates entre los elementos de L_j, se establece una correspondencia uno a uno (biyectiva) entre los elementos de L_j y los semi-vértices de Y, de tal forma que:

1. Si $y_k \in Y$ corresponde a r_j^i, entonces $r_j^i = d(v_j, y_k)$.

2. Si y_k e y_t son dos semi-vértices distintos de Y_j, e y_k está en el único camino que conecta y_t con v_j, entonces el elemento de L_j que representa y_k precederá al que representa a y_t. En particular, a v_j le corresponde el primer elemento de la lista, es decir, a r_j^1.

3. Si v_j no es una hoja, y además tanto y_k como y_t están en $V_{j(1)}$ $(V_{j(2)})$, donde el elemento que representa a y_k en $L_{j(1)}$ $(L_{j(2)})$ precede al que representa a y_t, entonces el elemento de L_j que representa a y_k precederá al que representa a y_t.

4. Si y_k está en Y_j, e y_t está en $Y \Leftrightarrow Y_j$, y $d(v_j, y_k) = d(v_j, y_t)$, entonces el elemento de L_j que representa a y_k precederá al que representa a y_t.

Para $i = 1, ..., m$, los correspondientes semi-vértices de r_j^i se denotan por y_j^i.

Proposición 4.5.4 *Las listas ordenadas L_j de los vértices v_j a los semivértices se obtiene en tiempo $O(n^3)$.*

Demostración: Para cada vértice v_j,

L_j^+ representa la secuencia de distancias ordenadas desde v_j a todos los semivértices de Y_j, y

L_j^- representa la secuencia de distancias ordenadas desde v_j a todos los semivértices de $Y \Leftrightarrow Y_j$.

Se comienza con las hojas de T, y se procede recursivamente hacia la raíz, calculando primero la lista L_j^+, como se muestra a continuación:

Cálculo de la lista L_j^+.

Si v_j es una hoja, $L_j^+ = [0]$. En otro caso, hay que considerar las listas $L_{j(1)}^+$ y $L_{j(2)}^+$, asociadas respectivamente a los dos hijos de v_j, $v_{j(1)}$ y $v_{j(2)}$.

1. Sumar la distancia $d(v_{j(1)}, v_j)$ a cada elemento de $L_{j(1)}^+$. Con ello se obtiene la lista $L_{j(1)}^{++}$. Y sumar la distancia $d(v_{j(2)}, v_j)$ a cada elemento de $L_{j(2)}^+$ para obtener la lista $L_{j(2)}^{++}$.

2. Construir las listas ordenadas $L_{j(1)}^{+1} = \{d(y_i, v_j) : y_i \in E_p(r) \cap \left[v_{j(1)}, v_j\right]\}$, y $L_{j(2)}^{+1} = \{d(y_i, v_j) : y_i \in E_p(r) \cap \left[v_{j(2)}, v_j\right]\}$.

3. Concatenar las listas $L_{j(1)}^{++}$ con $L_{j(1)}^{+1}$ llamando a la nueva lista $L_{j(1)}^{+2}$, y concatenar $L_{j(2)}^{++}$ con $L_{j(2)}^{+1}$ para obtener $L_{j(2)}^{+2}$.

4. Usando el algoritmo de ordenación por mezcla construir la lista ordenada $L_j^+ = L_{j(1)}^{+2} \cup L_{j(2)}^{+2}$.

La complejidad del procedimiento depende del punto 4. Este último punto es del orden de la suma del tamaño de las respectivas listas $L_{j(1)}^{+2}$ y $L_{j(2)}^{+2}$. Al ser las listas de tamaño $|Y| = O(n^2)$, se tiene: $\left|L_{j(1)}^{+2}\right| + \left|L_{j(2)}^{+2}\right| = O(n^2)$.

Cálculo de la lista L_j^-.

Para generar las listas L_j^-, comenzamos por la raíz v_1, y de forma recursiva se procede hacia las hojas.

Si $v_j = v_1$, L_j^- es vacío. En otro caso v_j es un hijo de v_k, supóngase que es el hijo izquierdo, $v_j = v_{k(1)}$. Se considerarán las listas L_k^- y $L_{k(2)}^+$.

1. Sumar las distancias $d(v_{k(1)}, v_k)$ a cada elemento de L_k^-. Con ello se obtiene la lista L_k^{-*}. Y sumar las distancias $d(v_{k(1)}, v_{k(2)})$ a cada elemento de $L_{k(2)}^+$, para obtener la lista $L_{k(2)}^{+*}$.

2. Construir las listas ordenadas $L_k^{-1} = \{d(y_i, v_j) : y_i \in E_p(r) \cap \left[v_{k(1)}, v_k\right]\}$, y $L_{k(2)}^{-1} = \{d(y_i, v_j) : y_i \in E_p(r) \cap \left[v_{k(2)}, v_k\right]\}$.

3. Concatenar las listas L_k^{-*}con L_k^{-1} llamando a la nueva lista L_k^{-2}, y concatenar $L_{k(2)}^{+*}$ con $L_{k(2)}^{-1}$ para obtener $L_{k(2)}^{-2}$.

4. Usando el algoritmo ordenación por mezcla construir $L_j^- = L_k^{-2} \cup L_{k(2)}^{-2}$.

Esta unión tiene por complejidad el tamaño de las respectivas listas L_k^{-2} y $L_{k(2)}^{-2}$. Al ser las listas de tamaño $|Y| = O(n^2)$, se tiene $\left|L_k^{-2}\right| = \left|L_{k(2)}^{-2}\right| = O(n^2)$.

Luego en cada vértice $v_j \in V$, el esfuerzo total, mientras se generan las listas L_j^+ y L_j^-, es de $O(n^2)$. En definitiva la complejidad total es de $O(n^3)$. □

Ejemplo

En el siguiente ejemplo (ver figura.- Ejemplo para ilustrar la generación de listas), donde $Y = \{v_1, ..., v_7\}$ se muestra cómo generar dichas listas.

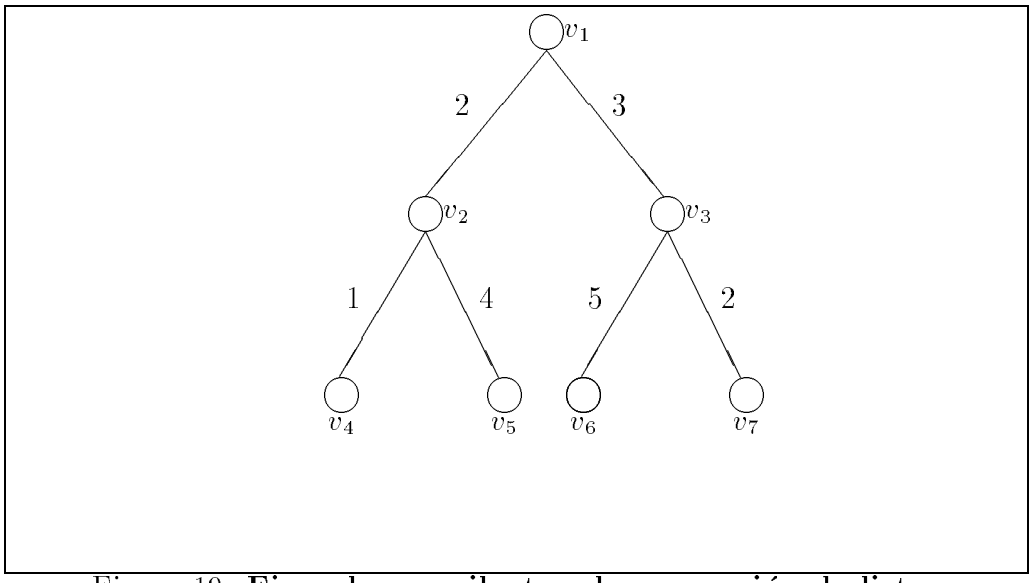


Figura 19. **Ejemplo para ilustrar la generación de listas**

Comenzamos generando L_j^+,

$L_4^+ = L_5^+ = L_6^+ = L_7^+ = \{0\}$,

$L_4^{++} = \{1\}$, $L_5^{++} = \{4\}$,

$L_6^{++} = \{5\}$, $L_7^{++} = \{2\}$,

$L_2^+ = \{0,1,4\}$, $L_3^+ = \{0,2,5\}$,

$L_2^{++} = \{2,3,6\}$, $L_3^{++} = \{3,5,8\}$,

$L_1^+ = \{0,2,3,3,5,6,8\}$.

Pasamos a generar L_j^-,

$L_1^- = \emptyset$,

$L_2^- = \{L_1^- + d(v_2,v_1)\} \cup \{L_3^+ + d(v_2,v_3)\} \cup d(v_2,v_1) = \{2,5,7,10\}$,

$L_3^- = \{L_1^- + d(v_3,v_1)\} \cup \{L_2^+ + d(v_3,v_2)\} \cup d(v_3,v_1) = \{3,5,6,9\}$,

$L_4^- = \{L_2^- + d(v_4,v_2)\} \cup \{L_5^+ + d(v_4,v_5)\} \cup d(v_4,v_2) = \{1,3,5,6,8,11\}$,

$L_5^- = \{L_2^- + d(v_5,v_2)\} \cup \{L_4^+ + d(v_5,v_4)\} \cup d(v_5,v_2) = \{4,5,6,9,11,14\}$,

$L_6^- = \{L_3^- + d(v_6,v_3)\} \cup \{L_7^+ + d(v_6,v_7)\} \cup d(v_6,v_3) = \{5,7,8,10,11,14\}$,

$L_7^- = \{L_3^- + d(v_7,v_3)\} \cup \{L_6^+ + d(v_7,v_6)\} \cup d(v_7,v_3) = \{2,5,7,7,8,11\}$.

Con lo anterior se está en disposición de describir el algoritmo de programación dinámica denominado **De las hojas a la raíz** que resuelve el problema de la p-Mediana r-restringida. Este algoritmo está inspirado en el algoritmo de Tamir [163], para el problema de la p-Mediana. Para el desarrollo del mismo se requieren unas definiciones previas.

Definición 4.5.9 *Para cada vértice v_j, un entero $q = 1,...,p$, y un radio $r_j^i \in L_j$.*

$G(v_j,q,r_j^i)$ es el valor óptimo del subproblema definido en el subárbol T_j, dado que el total de semi-vértices que pueden ser seleccionados en T_j, es al menos 1 y a lo más q (centros de servicios), y que al menos uno de ellos tiene que estar en $\{y_j^1, y_j^2, ..., y_j^i\} \cap Y_j$.

Es decir:

$$G(v_j,q,r_j^i) = \min_{\substack{1 \le |T_j| \le q \\ |X \cap \{y_j^1,...,y_j^i\} \cap Y_j| \ge 1}} \{ \sum_{v_i \in U_i} w_i' \, d(X,v_i) \}.$$

En el subproblema mencionado implícitamente se asume que no hay interrelaciones entre los semi-vértices de T_j, y el resto de los semi-vértices de T. La función $G(v_j,q,r)$ es calculada sólo para $q \le |V_j|$. También, para cada vértice v_j definimos:

$$G(v_j,0,r) = \infty.$$

Definición 4.5.10 *$F(v_j, q, r)$ es el valor óptimo del subproblema definido en el subárbol T_j, bajo las condiciones siguientes:*

1. *Un total de a lo más q semi-vértices pueden ser seleccionados en Y_j.*

2. *Existen ya algunos semi-vértices (centros de servicios) seleccionados en $Y \Leftrightarrow Y_j$, y el más cercano de ellos a v_j está a una distancia de exáctamente r desde v_j.*

Es decir:

$$F(v_j, q, r) = \min_{\substack{|T_j| \leq q \\ \min_{X \cap \{Y - Y_j\}} \{w'_i d(X, v_i)\} = r}} \{ \sum_{v_i \in U_i} w'_i d(X, v_i) \}.$$

La función recursiva $F(v_j, q, r)$ será calculada sólo para $q \leq |V_j|$, y $r = r_j^i$, donde r_j^i se corresponde con un semi-vértice $y_j^i \in Y \Leftrightarrow Y_j$.

El motivo de esta última definición, viene dado porque si todos los elementos de L_j son distintos, entonces $G(v_j, q, r_j^i)$ es el valor óptimo del subproblema definido en T_j, dado que al menos 1 y a los más q semi-vértices son seleccionados en T_j, y el más cercano de ellos a v_j está a una distancia de al menos r_j^i desde v_j. La condición de L_j, requirida anteriormente, asegura que la misma interpretación de $G(v_j, q, r_j^i)$ puede ser hecha para *distintos* elementos de L_j, es decir, elementos r_j^i, satisfaciendo $r_j^i < r_j^{i+1}$.

El algoritmo define las funciones G y F en todas las hojas de T, y recursivamente procede desde las hojas a la raíz calculando los valores relevantes de estas dos funciones recursivas en todos los vértices de T. El valor óptimo del problema vendrá dado por $G(v_1, p, r_1^m)$, donde v_1 es la raíz del árbol. (Ver figura.-Algoritmo de las hojas a la raíz)

Algoritmo De las hojas a la raíz.

Paso 0. Inicialización Si v_j es una hoja de T_{bin}. Entonces: $\forall i = 1, ..., m$ tal que $y_j^i \in Y - Y_j$

$$G(v_j, 1, r_j^i) = 0,\ i = 1, ..., m\ .$$

$$F(v_j, 0, r_j^i) = w_j'\, d(v_j, y_j^i) = w_j'\, r_j^i,\ y$$

$$F(v_j, 1, r_j^i) = \min\{F(v_j, 0, r_j^i), G(v_j, 1, r_j^i)\}.$$

Paso 1. Paso arriba Si v_j es un vértice no hoja de T_{bin}
(1.a) calcular el valor de G en v_j para cada r_j^i de interés correspondiente a Y_j^i.
Caso 1.- Si $y_j^i = v_j$, entonces calcular:

$$G(v_j, q, r_j^1) = \min_{\substack{q_1+q_2=q-1 \\ q_1 \le |V_{j(1)}| \\ q_2 \le |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\ .$$

Caso 2.- Si $y_j^i \in Y - Y_j$, entonces calcular:

$$G(v_j, q, r_j^i) = G(v_j, q, r_j^{i-1}).$$

Caso 3.- Si $y_j^i \in Y_j(1)$, entonces calcular: $G(v_j, q, r_j^i) =$

$$= \min\{G(v_j, q, r_j^{i-1}),\quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q \\ 1 \le q_1 \le |V_{j(1)}| \\ q_2 \le |V_{j(2)}|}} \{G(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

Caso 4.- Si $y_j^i \in [v_j, v_j(1)] - v$, entonces calcular: $G(v_j, q, r_j^i) =$

$$= \min\{G(v_j, q, r_j^{i-1}),\quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q-1 \\ q_1 \le |V_{j(1)}| \\ q_2 \le |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

Caso 5.- Si $y_j^i \in Y_j(2)$, entonces calcular: $G(v_j, q, r_j^i) =$

$$= \min\{G(v_j, q, r_j^{i-1}),\quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q \\ q_1 \le |V_{j(1)}| \\ 1 \le q_2 \le |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + G(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

Caso 6.- Si $y_j^i \in [v_j, v_j(2)] - v$, entonce calcular: $G(v_j, q, r_j^i) =$

$$= \min\{G(v_j, q, r_j^{i-1}),\quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q-1 \\ q_1 \le |V_{j(1)}| \\ q_2 \le |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

(1.b) calcular el valor de F en v_j para cada r_j^i de interés correspondiente a y_j^i.

$$F(v_j, q, r_j^i) = \min\{\quad G(v_j, q, r_j^i),\quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q \\ q_1 \le |V_{j(1)}| \\ q_2 \le |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

A continuación se detalla cada uno de los casos del paso 1 del algoritmo de las hojas a la raíz.

En los seis casos que se muestran a continuación asociado a cada semi-vértice y_j^i hay dos radios, a los que denotamos $r_{j(1)}^k$, $r_{j(2)}^t$ perteneciente a las listas $L_{j(1)}$ y $L_{j(2)}$ respectivamente. El radio $r_{j(1)}^k$ se obtiene de la ecuación:

$$w_j' \, r_j^i = w_{j(1)}' \, (r_{j(1)}^k + d(v_{j(1)}, v_j)$$

y el radio $r_{j(2)}^t$ viene dado por la ecuación:

$$w_j' \, r_j^i = w_{j(2)}' \, (r_{j(2)}^k + d(v_{j(2)}, v_j)$$

Caso 1.- Si r_j^1 es el radio correspondiente al semi-vértice $y_j^1 = v_j$, entonces, este radio también le corresponde a un par de elementos, digamos $r_{j(1)}^k$, $r_{j(2)}^t$ de $L_{j(1)}$ y $L_{j(2)}$ respectivamente. De esta manera,

$$G(v_j, q, r_j^1) = \min_{\substack{q_1+q_2=q-1 \\ q_1 \leq |V_{j(1)}| \\ q_2 \leq |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\} \ .$$

Esto significa que un servicio puede establecerse en v_j (la raíz del subárbol T_j) y para evaluar la función objetivo en él, hay que evaluar todas las posibilidades en sus respectivos hijos. Con tal fin se utilizan las funciones $F(v_{j(1)}, q_1, r_{j(1)}^k)$ y $F(v_{j(2)}, q_2, r_{j(2)}^t)$, ya que v_j está fuera de los subárboles $T_{j(1)}$ y $T_{j(2)}$.

En los casos venideros, consideramos r_j^i, para $i = 2, ..., m$.

Caso 2.- Si r_j^i es el radio correspondiente al semi-vértice $y_j^i \in Y \Leftrightarrow Y_j$, entonces:

$$G(v_j, q, r_j^i) = G(v_j, q, r_j^{i-1}) \ .$$

En este caso no es necesario realizar nuevos cálculos, pues la función vale lo mismo que cuando ésta fue evaluada con el radio que le precede.

Caso 3.- Si r_j^i es el radio correspondiente al semi-vértice $y_j^i \in Y_{j(1)}$, entonces, y_j^i también le corresponde a un elemento, digamos $r_{j(1)}^k$ de $L_{j(1)}$, y a un elemento, digamos $r_{j(2)}^t$ de $L_{j(2)}$, entonces:

$$G(v_j, q, r_j^i) = \min \quad \{G(v_j, q, r_j^{i-1}), \quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q \\ 1 \leq q_1 \leq |V_{j(1)}| \\ q_2 \leq |V_{j(2)}|}} \{G(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

Es decir, hay dos posibilidades, la primera de ellas, es no establecer un servicio en y_j^i, esta posibilidad es evaluada con la función $G(v_j, q, r_j^{i-1})$. La segunda de las posibilidades, es establecer un servicio en y_j^i, con un coste de $w_j \cdot r_j^i$, y establecer los q-1 restantes entre las dos ramas de v_j. La rama correspondiente a $v_{j(1)}$ se evalúa con $G(v_{j(1)}, q_1, r_{j(1)}^k)$, $1 \leq q_1 \leq |V_{j(1)}|$, pues ya se ha establecido un servicio en el subárbol $T_{j(1)}$. La rama correspondiente a $v_{j(2)}$ se evalúa con $F(v_{j(2)}, q_2, r_{j(2)}^t)$, $q_2 \leq |V_{j(2)}|$ ya que no se ha establecido un servicio en el subárbol $T_{j(2)}$.

Caso 4.- Si r_j^i es el radio correspondiente al semi-vértice $y_j^i \in (v_j, v_{j(1)})$, e $y_j^i \neq v_{j(1)}$, entonces:

$$G(v_j, q, r_j^i) = \min\{ \quad G(v_j, q, r_j^{i-1}), \quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q-1 \\ q_1 \leq |V_{j(1)}| \\ q_2 \leq |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

De forma similar al caso 3, se tienen dos posibilidades, pero en este caso, el servicio se establece fuera de $T_{j(1)} \cup T_{j(2)}$, por ello ambos subárboles se evalúan con la función F.

Caso 5.- Este caso es simétrico al caso 3, es decir, lo que acontecía para el subárbol $T_{j(1)}$ ahora acontece para el subárbol $T_{j(2)}$. Es decir:

Si r_j^i es el radio correspondiente a un semi-vértice $y_j^i \in Y_{j(2)}$, entonces:

$$G(v_j, q, r_j^i) = \min\{ \quad G(v_j, q, r_j^{i-1}), \quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q \\ q_1 \leq |V_{j(1)}| \\ 1 \leq q_2 \leq |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + G(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

Caso 6.- Este caso es simétrico a lo ocurrido en el caso 4.

Si r_j^i es el radio correspondiente a un semi-vértice $y_j^i \in (v_j, v_{j(2)})$, e $y_j^i \neq v_{j(2)}$, entonces:

$$G(v_j, q, r_j^i) = \min\{ \quad G(v_j, q, r_j^{i-1}), \quad w_j \cdot r_j^i + \min_{\substack{q_1+q_2=q-1 \\ q_1 \leq |V_{j(1)}| \\ q_2 \leq |V_{j(2)}|}} \{F(v_{j(1)}, q_1, r_{j(1)}^k) + F(v_{j(2)}, q_2, r_{j(2)}^t)\}\}.$$

(1.b) Calcular el valor de la función F en v_j. Teniendo definida la función G en v_j, se puede calcular la función F en v_j para todos los argumentos relevantes. Sea y_j^i un semi-vértice de $Y \Leftrightarrow Y_j$. Entonces y_j^i le corresponde a un par de elementos, digamos $r_{j(1)}^k$, $r_{j(2)}^t$ de $L_{j(1)}$ y $L_{j(2)}$, respectivamente. Luego,

$$F(v_j,q,r_j^i)=\min\{\quad G(v_j,q,r_j^i),\\ w_j\cdot r_j^i+\min_{\substack{q_1+q_2=q\\ q_1\le|V_{j(1)}|\\ q_2\le|V_{j(2)}|}}\{F(v_{j(1)},q_1,r_{j(1)}^k)+F(v_{j(2)},q_2,r_{j(2)}^t)\}\}.$$

Para la función F no es necesario hacer el desgloce anterior, pues el único caso relevante es cuando $y_j^i \in Y \Leftrightarrow Y_j$. Y como en el caso 2, no es necesario realizar nuevos cálculos, pues el valor de la función objetivo coincide con el valor de la función evaluada en el radio que le precede.

Proposición 4.5.5 *La complejidad del Algoritmo de las hojas a la raíz es* $O(p^2n^3)$.

Demostración: Se sigue directamente de la ecuación recursiva, que el esfuerzo total de computar las funciones G y F en un vértice dado v_j, para todo valor relevante de q y r es $O(n^2 \min\{(|V_{j(1)}|,p)\} \min\{(|V_{j(2)}|,p)\})$. De esta manera, el esfuerzo total del algoritmo es $O(p^2n^3)$. □

Si al algoritmo de las hojas a la raíz se le aplica el siguiente análisis de Tamir (1996) [163], la complejidad de éste mejora a $O(pn^3)$.

El análisis está basado en la partición del conjunto de vértices en dos subconjuntos, y fundamentado en la siguiente definición.

Definición 4.5.11 *Un vértice v_j se le denomina* **rico**, *si no es una hoja, y sus dos hijos $v_{j(1)}$ y $v_{j(2)}$ satisfacen las desigualdades siguientes: $\left|V_{j(1)}\right| \ge p/2$, y $\left|V_{j(2)}\right| \ge p/2$.*

Si un vértice no es rico se le denomina pobre.

Proposición 4.5.6 El número de vértices ricos es del orden $O(\frac{n^2}{p})$.

Demostración: Ver Tamir (1996). [163] □

Por esta proposición el esfuerzo total necesario para calcular G y F en todos los vértices ricos es de $O(pn^3)$. Veámos ahora que la complejidad a la hora de calcular estas funciones en todos los vértices pobres es también $O(pn^3)$.

Para cada vértice $v_j \in Y$, H_j denotará el esfuerzo total resultante al calcular las funciones G y F en todos los vértices pobres de Y. Entonces, si v_j es rico,

$$H_j \leq H_{j(1)} + H_{j(2)},$$

y si v_j es pobre,

$$H_j \leq H_{j(1)} + H_{j(2)} + c \cdot n^2 \cdot \min\{\left|V_{j(1)}\right|, p/2\} \cdot \min\{\left|V_{j(2)}\right|, p/2\},$$

para alguna constante c.

Proposición 4.5.7 Para cada vértice $v_j \in Y$, $H_j \leq cpn^2 \left|V_j\right|$.

Demostración: Ver Tamir (1996). [163] □

De estas dos proposiciones se concluye que la complejidad del algoritmo de programación dinámica es $O(pn^3)$.

4.6 Un algoritmo alternativo para el problema del p-λ-Centdian.

Para un p general se ha desarrollado un algoritmo de $O(pn^6)$ para el p-Centdian en un árbol. Pero cuando p es pequeño hay un desarrollo alternativo, basado en la demostración del Teorema (p-CD-árbol) cuya complejidad es $O(n^p)$ para cada valor fijo de p.

Consideremos todas las $O(n^{p-1})$ particiones del árbol T en p subárboles, los cuales son obtenidos borrando un subconjunto de $p \Leftrightarrow 1$ aristas de T. Para alcanzar la complejidad $O(n^p)$, se presenta una algoritmo lineal para el problema del p-Centdian en una partición de T en p subárboles, $T_1, ..., T_p$, esta partición es conocida como bosque.

Para cada subárbol T_j, sea x_j^* la solución (única) del problema 1-Centro w'-pesado en T_j, y sea y_j^*, un vértice de T_j, la solución de la 1-Mediana w-pesada (no restringida), como ésta no tiene por qué ser única, asumimos que es la más cercana a x_j^*. Se considera el camino $P(x_j^*, y_j^*)$, que conecta x_j^* con y_j^* como un segmento lineal de longitud $d(x_j^*, y_j^*)$, luego $f_c^j(x_j)$ denota la función 1-Centro w'-pesado, y $f_m^j(x_j)$ denota la función 1-Mediana w-pesada (no restringida), definidas en este camino (segmento).

La función $f_m^j(x_j)$, es convexa, decreciente y lineal a trozos. Sus puntos de rupturas se corresponden con los vértices de T_j en el camino $P(x_j^*, y_j^*)$.

Similarmente, la función Centro, $f_c^j(x_j)$, es convexa, monótona creciente y lineal a trozos. Cada punto de ruptura de esta función es un punto x de $P(x_j^*, y_j^*)$ cuyas distancias w'-pesada desde dos vértices de T_j, digamos v_i y v_t son iguales, es decir, $w_i' d(v_i, x) = w_t' d(v_t, x)$ con $w_i' \neq w_t'$.

Proposición 4.6.1 *El problema del p-Centdian en un bosque de p subárboles $T_1, ..., T_p$ se puede formular como un programa lineal expresado en función de las variables $x_1, ..., x_p$, y algunas variables auxiliares, de la siguiente manera:*

$$\min\{r + \sum_{i=1}^{n} f_i\},$$

sujeto a las restricciones:

$$\begin{aligned} r &\geq w_k'((x_j \Leftrightarrow a_{i(k)}) + b_k), \\ r &\geq w_k'(\Leftrightarrow(x_j \Leftrightarrow a_{i(k)}) + b_k), \\ f_k &\geq w_k((x_j \Leftrightarrow a_{i(k)}) + b_k), \\ f_k &\geq w_k(\Leftrightarrow(x_j \Leftrightarrow a_{i(k)}) + b_k), \end{aligned}$$

para todo $v_k \in U^j$, $j = 1, ..., p$.

Demostración: Por conveniencia, y sin perder generalidad, asumimos que cada variable real x_j, $j = 1, ..., p$, está restringida al intervalo (segmento) $[0, d(x_j^*, y_j^*)]$, en el que $x_j = 0$ se corresponde al 1-Centro de T_j, x_j^*, y $x_j = d(x_j^*, y_j^*)$ se corresponde a la 1-Mediana de T_j, y_j^*. Si el 1-Centro, x_j^*, no es un vértice de T_j, aumentamos con este punto el conjunto de vértices de T_j, con pesos w' y w iguales a 0. V^j denotará el conjunto de vértices de T_j. Nótese que el 1-Centro w'-pesado, x_j^* puede ser encontrado en tiempo $O(|U^j|)$ por el algoritmo de Megiddo [112]. Similarmente, la 1-Mediana w-pesada, y_j^*, puede ser encontrada en tiempo $O(|U^j|)$ por el algoritmo de Goldman [58]. En definitiva, el tiempo total para encontrar $\{x_1^*, ..., x_p^*\}$, e $\{y_1^*, ..., y_p^*\}$ es $O(n)$.

Para cada vértice $v_i \in U^j$, en el camino $P(x_j^*, y_j^*)$, sea a_i, el valor correspondiente a la variable real x_j, es decir, $a_i = d(v_i, x_j)$. Para cada vértice $v_k \in U^j$, sea $v_{i(k)}$ el vértice más cercano al vértice v_k en $P(x_j^*, y_j^*)$, y sea $b_k = d(v_k, v_{i(k)})$.

Para cada subárbol T_j, $j = 1, ..., p$, la función 1-Centro en el camino $P(x_j^*, y_j^*)$ está definida por

$$f_c^j(x_j) = \max_{\{v_k \in U^j\}} \{w_k'(|x_j \Leftrightarrow a_{i(k)}| + b_k)\}.$$

Similarmente, la función 1-Mediana está definida por

$$f_m^j(x_j) = \sum_{\{v_k \in U^j\}} \{w_k(|x_j \Leftrightarrow a_{i(k)}| + b_k)\}.$$

El problema del p-Centdian en un bosque reside en encontrar p reales, $x_1, ..., x_p$, que minimicen la función:

$$\max_{\{j=1,...,p\}} f_c^j(x_j) + \sum_{j=1}^{n} f_m^j(x_j). \tag{4.1}$$

Una reformulación del problema puede ser obtenida asociando una variable real f_k a cada vértice v_k de T. El problema estriba en encontrar las variables reales, $x_1, ..., x_p$, $f_1, ..., f_n$ y r, minimizando $\{r + \sum_{i=1}^{n} f_i\}$, cuando éste esté sujeto a las restricciones,

$$r \geq w'_k(|x_j \Leftrightarrow a_{i(k)}| + b_k),$$
$$f_k \geq w_k(|x_j \Leftrightarrow a_{i(k)}| + b_k),$$

para todo $v_k \in U^j$, $j = 1, ..., p$,

El problema expresado como un programa lineal será como sigue:

$$\min\{r + \sum_{i=1}^{n} f_i\},$$

sujeto a las restricciones:

$$r \geq w'_k((x_j \Leftrightarrow a_{i(k)}) + b_k),$$
$$r \geq w'_k(\Leftrightarrow(x_j \Leftrightarrow a_{i(k)}) + b_k),$$
$$f_k \geq w_k((x_j \Leftrightarrow a_{i(k)}) + b_k),$$
$$f_k \geq w_k(\Leftrightarrow(x_j \Leftrightarrow a_{i(k)}) + b_k),$$

para todo $v_k \in U^j$, $j = 1, ..., p$. □

Este programa lineal es un ejemplo del modelo presentado en Zemel (1984) [177]. Específicamente es un caso especial del dual del problema de Programación Lineal de Elección Múltiple p-dimensional, discutido en la sección 3 del trabajo de Zemel (1984) [177]. Luego, para cada valor fijo de p, el anterior problema lineal puede ser resuelto en tiempo $O(n)$. Se concluye que para cada valor fijo de p, el problema del p-Centdian en el árbol T puede resolverse en un tiempo $O(n^p)$. Esta última complejidad domina a la obtenida anteriormente de $O(pn^6)$ cuando $p < 6$.

4.7 Un algoritmo eficiente para el 2-λ-Centdian.

El algoritmo alternativo para el problema del p-Centdian presentado en la sección anterior puede mejorarse en su eficiencia práctica con algunas sencillas reglas heurísticas. En esta sección se describe una versión para el caso $p = 2$ en el que además, algunas de las operaciones implicadas se pueden ejecutar de manera alternativa, sin que por ello se altere la complejidad.

Para el caso $p = 2$ este método alternativo consiste en resolver problemas del 2-Centdian en bosques obtenidos al eliminar una arista del árbol T y eligiendo el que de lugar a una mejor solución. Si el orden en que se seleccionan las aristas a ser eliminadas se elige de forma inteligente empezando por las aristas más prometedoras, es de esperar que la solución óptima se encuentre pronto y además es posible establecer una cota, que determina cuando no se puede mejorar la solución al eliminar otras aristas.

4.7.1 El algoritmo para el 2-λ-Centdian en un 2-bosque (T_1, T_2).

En este apartado se generaliza el algoritmo $O(n)$ de la sección 4.3.2 para el 1-λ-Centdian en un árbol T al caso del 2-λ-Centdian.

Concretamos una vez más el problema al que nos referimos. El problema del 2-λ-Centdian en un 2-bosque $B = (T_1, T_2)$ consiste en hallar $z_1 \in T_1$, $z_2 \in T_2$ que minimizen

$$f(z_1, z_2) = \lambda f_c(z_1, z_2) + (1 \Leftrightarrow \lambda) f_m(z_1, z_2)$$

donde $f_c(z_1, z_2) = \max\{f_c(T_1, z_1), f_c(T_2, z_2)\}$ y $f_m(z_1, z_2) = f_m(T_1, z_1) + f_m(T_2, z_2)$, siendo

$$f_c(T_i, z_i) = \max_{v \in T_i} d(z_i, v) \text{ y } f_m(T_i, z_i) = \sum_{v \in T_i} d(z_i, v).$$

Sea (z_1, z_2) el 2-λ-Centdian del 2-bosque $B = (T_1, T_2)$.

Proposición 4.7.1 *Para $i = 1, 2$, el punto z_i está en el camino del centro x_i^* a la mediana y_i^* del árbol T_i.*

Demostración: Si el punto z_1 no está en el camino C_1 de x_1^* a y_1^*, sea z_1' el punto más cercano a z_1 en el camino C_1. La solución (z_1', z_2) verifica: $f_c(T_1, z_1') < f_c(T_1, z_1)$ y $f_m(T_1, z_1') < f_m(T_1, z_1)$. Por tanto es mejor que (z_1, z_2).

Analogamente se prueba que z_2 tiene que estar en el camino C_2 de x_2^* a y_2^*. □

Para $i = 1, 2$, sea $r_i = f_c(T_i, z_i)$ donde (z_1, z_2) es el 2-λ-Centdian del 2-bosque $B = (T_1, T_2)$. Denotemos por x_i^*, y_i^* y z_i^* al centro, a la mediana y al λ-centdian del árbol T_i.

Proposición 4.7.2 *Si $r_1 > r_2$ entonces $z_1 = z_1^*$ y $z_2 = y_2^*$; es decir, z_1 es el 1-λ-Centdian de T_1 y z_2 es la mediana de T_2.*

Demostración: El punto z_1 está en el camino C_1 de x_1^* a y_1^*. Si z_1 no es el 1-λ-Centdian z_1^* de T_1 sea $z_1' = z_1(r_2)$ el punto de C_1 con $f_c(T_1, z_1) = r_2$. Entonces, tanto para el par (z_1, z_2) como para (z_1', z_2) se puede expresar:

$$f_\lambda(z_1, z_2) = \lambda f_c(z_1, z_2) + (1 \Leftrightarrow \lambda) f_m(z_1, z_2) =$$

$$= \lambda f_c(T_1, z_1) + (1 \Leftrightarrow \lambda)[f_m(T_1, z_1) + f_m(T_2, z_2)] = f_\lambda(T_1, z_1) + (1 \Leftrightarrow \lambda) f_m(T_2, z_2).$$

Dado que la función f_λ es convexa, se tiene que: $f_\lambda(T_1, z_1\prime) < f_\lambda(T_1, z_1^*)$ por lo que el par (z_1, z_2) sería peor solución que (z_1', z_2).

Supongamos que $r_1 > r_2$ y $z_1 = z_1^*$ es el 1-λ-Centdian de T_1 pero z_2 no es la Mediana y_2^* de T_2. Sea z_2' el punto del camino C_2 tal que $f_c(T_2, z_2') = r_1$. Entonces, tanto para el par (z_1, z_2) como para (z_1, z_2') se puede expresar:

$$f_\lambda(z_1, z_2) = \lambda f_c(z_1, z_2) + (1 \Leftrightarrow \lambda) f_m(z_1, z_2) =$$

$$= \lambda f_c(T_1, z_1) + (1 \Leftrightarrow \lambda)(f_m(T_1, z_1) + f_m(T_2, z_2)) = f_\lambda(T_1, z_1) + (1 \Leftrightarrow \lambda) f_m(T_2, z_2).$$

Pero z_2' es la solución óptima del problema de la Mediana r_1-restringida en T_2, que es única; es decir no hay otro punto $z_2 \in T_2$ con $f_c(T_2, z_2) \leq r_1$ tal que $f_m(T_2, z_2) \leq f_m(T_2, z_2')$, por tanto $f_m(T_2, z_2') < f_m(T_2, z_2)$, por lo que el par (z_1, z_2) sería peor solución que (z_1, z_2'). □

Proposición 4.7.3 *Si $f_c(T_1, x_1^*) \geq f_c(T_2, y_2^*)$ entonces (z_1^*, y_2^*) es el 2-λ-Centdian de $B = (T_1, T_2)$.*

Demostración: Sea (z_1, z_2) el 2-λ-Centdian del 2-bosque $B = (T_1, T_2)$. Entonces

$$f_\lambda(B, (z_1, z_2)) = \lambda f_c(B, (z_1, z_2)) + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda \max\{f_c(T_1, z_1), f_c(T_2, z_2)\} + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda f_c(T_1, z_1) + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda f_c(T_1, z_1) + (1 \Leftrightarrow \lambda)[f_m(T_1, z_1) + f_m(T_2, z_2))] =$$

$$= f_\lambda(T_1, z_1) + (1 \Leftrightarrow \lambda) f_m(T_2, z_2)$$

$$\geq f_\lambda(T_1, z_1^*) + (1 \Leftrightarrow \lambda) f_m(T_2, y_2^*).$$

Por tanto, (z_1^*, y_2^*) es el 2-λ-Centdian del 2-bosque $B = (T_1, T_2)$. □

Proposición 4.7.4 *Si $f_c(T_1, z_1^*) \geq f_c(T_2, y_2^*)$ entonces (z_1^*, y_2^*) es el 2-λ-Centdian de $B = (T_1, T_2)$.*

Demostración: Sea (z_1, z_2) el 2-λ-Centdian del 2-bosque $B = (T_1, T_2)$ y sean $r^* = f_c(B, (z_1, z_2))$ $r_1 = f_c(T_1, z_1)$ y $r_2 = f_c(T_2, z_2)$.

Si $r_1 < r_2$ tomando $z_1' \in C_1$ tal que $f_c(T_1, z_1') = r_2$, se tiene:

$$f_\lambda(B, (z_1, z_2)) = \lambda f_c(B, (z_1, z_2)) + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda \max\{f_c(T_1, z_1), f_c(T_2, z_2)\} + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda f_c(T_2, z_2) + (1 \Leftrightarrow \lambda) f_m(B, (z_1, z_2)) =$$

$$= \lambda f_c(T_2, z_2) + (1 \Leftrightarrow \lambda)[f_m(T_1, z_1) + f_m(T_2, z_2))] >$$

$$> \lambda f_c(T_2, z_2) + (1 \Leftrightarrow \lambda)[f_m(T_1, z_1') + f_m(T_2, z_2))] = f_\lambda(B, (z_1', z_2))$$

pues $f_m(T_1, z_1') < f_m(T_1, z_1)$. Además, dado que $f_c(T_1, z_1') = f_c(T_2, z_2) = r_2$, se tiene:

$$f_\lambda(B, (z_1', z_2)) = f_\lambda(T_1, z_1') + (1 \Leftrightarrow \lambda) f_m(T_2, z_2) \geq$$

$$\geq f_\lambda(T_1, z_1^*) + (1 \Leftrightarrow \lambda) f_m(T_2, z_2) = f_\lambda(B, (z_1^*, z_2))$$

pues $f_\lambda(T_1, z_1^*) \leq f_\lambda(T_1, z_1')$. Finalmente, dado que $f_c(T_1, z_1^*) \geq f_c(T_2, y_2^*)$, se tiene:

$$f_\lambda(B, (z_1^*, z_2)) = f_\lambda(T_1, z_1^*) + (1 \Leftrightarrow \lambda) f_m(T_2, z_2) \geq$$

$$\geq f_\lambda(T_1, z_1^*) + (1 \Leftrightarrow \lambda) f_m(T_2, y_2^*) = f_\lambda(B, (z_1^*, y_2^*))$$

pues $f_m(T_2, y_2^*) \leq f_m(T_2, z_2)$.

En otro caso, si $r_1 \geq r_2$ se tiene análogamente $f_\lambda(B, (z_1^*, z_2)) \leq f_\lambda(B, (z_1, z_2))$ pues $f_\lambda(T_1, z_1^*) \leq f_\lambda(T_1, z_1)$. Finalmente también $f_\lambda(B, (z_1^*, y_2^*)) \leq f_\lambda(B, (z_1^*, z_2))$ pues $f_m(T_2, y_2^*) \leq f_m(T_2, z_2)$.

Por tanto en ambos casos $f_\lambda(B, (z_1^*, y_2^*)) \leq f_\lambda(B, (z_1, z_2))$. □

Sea $r_z = \max\{f_c(T_1, z_1^*), f_c(T_2, z_2^*)\}$ y $r_m = \min\{f_c(T_1, y_1^*), f_c(T_2, y_2^*)\}$.

Proposición 4.7.5 *Sea (z_1, z_2) el 2-λ-Centdian de B y $r^* = f_c(B, (z_1, z_2))$. Si $r_z \leq r_m$ entonces $r_z \leq r^* \leq r_m$. Si $r_1 = f_c(T_1, z_1)$ y $r_2 = f_c(T_2, z_2)$ entonces $r_1 = r_2 = r^*$.*

Demostración: Por el mismo tipo de razonamiento que en las proposiciones anteriores, analizando la situación relativa entre r_1 y r_2 con respecto al intervalo $[r_z, r_m]$, se tiene:

1. Si $f_c(B,(z_1,z_2)) > r_m$ entonces tomando $z_1' = z_1(r_m)$ y $z_2' = z_2(r_m)$ se tiene
$$f_\lambda(B,(z_1',z_2')) < f_\lambda(B,(z_1,z_2)).$$

2. Si $f_c(B,(z_1,z_2)) < r_z$ entonces tomando $z_1' = z_1(r_z)$ y $z_2' = z_2(r_z)$ se tiene
$$f_\lambda(B,(z_1',z_2')) < f_\lambda(B,(z_1,z_2)).$$

3. Si $r_z \leq r_1 < r_2 = r^* \leq r_m$ entonces tomando $z_1' = z_1(r_2)$ se tiene
$$f_\lambda(B,(z_1',z_2)) < f_\lambda(B,(z_1,z_2)).$$

4. Si $r_z \leq r_2 < r_1 = r^* \leq r_m$ entonces tomando $z_2' = z_2(r_1)$ se tiene
$$f_\lambda(B,(z_1,z_2')) < f_\lambda(B,(z_1,z_2)).$$

Por tanto, $r_z \leq r_1 = r^* = r_2 \leq r_m$. □

Además, siguiendo los resultados obtenidos al determinar el conjunto finito dominante para el p-Centdian, z_1 o z_2 es un punto de ruptura de la función f_c en su árbol respectivo. Si Q_i denota al conjunto de puntos de ruptura de la función f_c en el camino C_i entonces, los pares de puntos candidatos (z_1,z_2) están formados por un elemento de $z_1 \in Q_1$ y $z_2 = z_2(r_1(z_1))$ o por $z_2 \in Q_2$ y $z_1 = z_1(r_2(z_2))$ donde $r_i(z_i) = f_c(T_i,z_i)$ y $z_i(r)$ es el único punto z_i del camino C_i tal que $f_c(T_i,z_i) = r$.

Luego, el procedimiento para resolver el 2-Centdian en el bosque $B = (T_1,T_2)$ propuesto consiste en determinar las Medianas y Centdianes de ambos árboles y calcular r_z y r_m. Si $r_m < r_z$ la solución es la formada por el Centdian de un árbol y la Mediana del otro. En otro caso basta recorrer los pares de puntos $(z_1, z_2(r_1(z_1)))$, para $z_1 \in Q_1$ y $(z_1(r_2(z_2)), z_2)$, para $z_2 \in Q_2$ desde $z_1 = z_1^*$ o $z_2 = z_2^*$ hasta que la pendiente de $f_\lambda(z_1(r), z_2(r))$ cambie de signo.

Dado que las Medianas y Centdianes se obtienen en tiempo $O(n)$ y los tamaños de los conjuntos Q_1 y Q_2 son $O(n)$, la complejidad de todo el procedimiento es $O(n)$.

4.7.2 Algoritmo 2CDB para el caso no ponderado.

Para implementar el algoritmo 2CDB se ha elegido la versión para el problema del 2-λ-Centdian sin pesos que tiene una complejidad equivalente y la implementación resulta más clara.

El algoritmo para determinar el 2-λ-Centdian de un 2-bosque $B = (T_1, T_2)$ empieza por obtener los Centros, Medianas y λ-Centdianes de T_1 y T_2. Para $i = 1, 2$, sean x_i^*, y_i^* y z_i^* el Centro, Mediana y λ-Centdian de T_i. Sea $r_i^- = f_c(T_i, z_i^*)$ y $r_i^+ = f_c(T_i, y_i^*)$ Sea $r^- = \max\{r_1^-, r_2^-\}$ y $r^+ = \min\{r_1^+, r_2^+\}$. Si $r^- > r^+$ entonces el 2-λ-Centdian esta formado por el λ-Centdian de un árbol y la Mediana del otro. Concretamente si $r^- = r_1^-$ es $z_1 = z_1^*$ y $z_2 = y_2^*$, pero si $r^- = r_2^-$ es $z_1 = y_1^*$ y $z_2 = z_2^*$. En otro caso, se tiene $r^- \leq r^* \leq r^+$.

Sea u_i el vértice de T_i más alejado de la Mediana y_i^*. Se recorren a la vez los caminos C_1^* y C_2^* desde los Centdianes z_1^* y z_2^* hacia las Medianas y_1^* y y_2^* por medio de dos puntos z_1 y z_2 de forma que siempre $t = d(z_1, u_1) = d(z_2, u_2)$. Sean $z_1(t)$ y $z_2(t)$ dichos puntos. La pendiente de $f_\lambda(z_1(t), z_2(t))$ es
$\lambda \Leftrightarrow (1 \Leftrightarrow \lambda)(s^1(z_1(t)) + s^2(z_2(t))$, siendo $D_i(z_i(t))$ la pendiente de $f_m(T_i, z_i(t))$. El recorrido se detiene cuando cambie de signo; es decir cuando $D_1 + D_2 \geq \lambda/(1 \Leftrightarrow \lambda)$, o cuando alguno de estos dos puntos llegue a la mediana respectiva.

Dado que en el caso sin pesos, la función f_c es lineal en el camino entre el Centro y la Mediana, esta pendiente cambia sólo cuando $z_1(t)$ o $z_2(t)$ llegue a un vértice del camino correspondiente.

Algoritmo CDT para el caso no ponderado.

En el caso no ponderado ($w'_u = w_u = 1, \forall u \in V$). El centro x^* de T se determina por el siguiente algoritmo de Handler [68].

Algoritmo de Handler

Paso 1. Sea v un vértice cualquiera.

Paso 2. Sea u el vértice más alejado a v.

Paso 3. Sea w el vértice más alejado a u.

Paso 4. El centro x^* de T es el punto medio del camino de u a w.

Figura 20. **Algoritmo de Handler**

El algoritmo de Goldman [58] extendido hasta comprimir todo el árbol para obtener la pendiente de la función mediana en todas sus aristas es el siguiente:

Algoritmo de Goldman

Paso 1. Sea $W = \sum_{u \in U} w_u = |V| = n$.

Paso 2. Para cada vértice v de T se inicializa $w(v) \leftarrow w_v = 1$.

Paso 3. Mientras queden aristas por contraer en el árbol T repetir la operación siguiente:

Tomar una hoja cualquiera v de T.
Si $w(v) \geq W/2$ entonces v es la mediana de T; $y^* = v$.
En otro caso, sea la arista $[v, u]$ de T, hacer:
$w(u) \leftarrow w(u) + w(v)$
y eliminar $[v, u]$ de T.

Figura 21. **Algoritmo de Goldman**

El λ-Centdian de T se determina calculando el punto del camino C de x^* a y^* en el que la pendiente de $f_\lambda(x)$ se anula. Esta pendiente cuando se recorre desde x^* hacia y^* es:

$$D_\lambda(x) = \lambda \Leftrightarrow (1 \Leftrightarrow \lambda) D_m(x)$$

siendo D_m la pendiente de la función Mediana. En cada arista $[v_j, v_{j+1}]$ del camino de C, la pendiente de la función Mediana $f_m(x)$ es $D_j = 2w(v_i) \Leftrightarrow n$ donde $w(.)$ es la función peso resultante del algoritmo de Goldman.

Se recorren los vértices de C hasta que $D_\lambda = \lambda \Leftrightarrow (1 \Leftrightarrow \lambda) D_m(x) \geq 0$, es decir hasta el primer vértice v_i de C tal que $\lambda \Leftrightarrow (1 \Leftrightarrow \lambda) D_i \geq 0 \Leftrightarrow D_i \leq \lambda/(1 \Leftrightarrow \lambda)$

Algoritmo CDT

Paso 1. Calcular el Centro x^* del árbol T por medio del algoritmo de Handler.

Paso 2. Calcular la Mediana y^* del árbol T por medio del algoritmo de Goldman.

Paso 3. Calcular la pendiente de la función Mediana D_m.

Paso 4. Recorrer los vértices del camino de x^* a y^* hasta que:

$$D_\lambda = \lambda - (1 - \lambda) D_m(x) \geq 0,$$

Figura 22. **Algoritmo CDT**

Algoritmo 2CDB (caso no ponderado).

Paso 1. (i) Aplicar el algoritmo CDT al árbol T_1, para obtener el Centro x_1^*, la Mediana y_1^* y el λ-Centdian z_1^* de dicho árbol.

(ii) Aplicar el algoritmo CDT al árbol T_2, para obtener el Centro x_2^*, la Mediana y_2^* y el λ-Centdian z_2^* de dicho árbol.

Paso 2. (i) Si $f_c(U_1, z_1^*) \geq f_c(U_2, y_2^*)$ entonces $z_1 = z_1^*$ y $z_2 = y_2^*$.

(ii) Si $f_c(U_2, z_2^*) \geq f_c(U_1, y_1^*)$ entonces $z_2 = z_2^*$ y $z_1 = y_1^*$.

En otro caso ir al paso 3.

Paso 3. Para cada arbol T_i sea C_i^* el camino desde z_i^* a y_i^* dado por $C_i = [v_i^0, v_i^1, v_i^2, ..., v_i^{n_i}]$ donde $z_i^* \in (v_i^0, v_i^1]$ y $v_i^{n_i} = y_i^*$. El algoritmo CDT proporciona, junto con este camino, las pendientes D_i^j de $f_m(T_i, z_i(t))$ en la arista $[v_i^j, v_i^{j+1}]$, las distancias $d(v_i^j, u_i)$ donde u_i es el vértice de T_i más lejano a y_i^*; $j = 1, 2, ..., n_i$ y

(i) Inicialización.
Sea $j_1 \leftarrow 0$ y $j_2 \leftarrow 0$. Tomar $s \leftarrow s_1^{j_1} + s_2^{j_2}$

(ii) Iteraciones.
Mientras $s \leq \lambda/(1-\lambda)$ ejecutar los pasos:
(a) Si $d(v_1^{j_1}, u_1) \geq d(v_2^{j_2}, u_2)$ entonces: $j_1 \leftarrow j_1 + 1$, en otro caso $j_2 \leftarrow j_2 + 1$.
(b) Hacer $s \leftarrow s_1^{j_1} + s_2^{j_2}$

(iii) Finalización.
Si $d(v_1^{j_1}, u_1) \leq d(v_2^{j_2}, u_2)$ entonces:
(a) Si $j_1 = 0$ entonces $z_1 = z_1^*$, en otro caso $(j_1 > 0)$ $z_1 \leftarrow v_1^{j_1}$.
(b) Sea $z_2 \in [v_2^{j_2}, v_2^{j_2+1}]$ tal que $d(z_1, u_1) = d(z_2, u_2)$.
En otro caso, $(d(v_1^{j_1}, u_1) > d(v_2^{j_2}, u_2))$,
(a) Si $j_2 = 0$ entonces $z_2 = z_2^*$; en otro caso $(j_2 > 0)$ $z_2^* \leftarrow v_2^{j_2}$.
(b) Sea $z_1 \in [v_1^{j_1}, v_1^{j_1+1}]$ tal que $d(z_1, u_1) = d(z_2, u_2)$.

Figura 23. **Algortimo 2CDB (caso no ponderado)**

4.7.3 Algoritmo 2CDT.

El algoritmo 2CDT que resuelve el problema en el árbol consiste en recorrer el conjunto de aristas del árbol para resolver el problema en el bosque que resulta de eliminar del árbol dicha arista aplicando el algoritmo 2CDB.

Algoritmo 2CDT

Paso 1. Inicialización.

Sea $f^* = \infty$.

Paso 2. Iteraciones.

Para cada arista $[i,j] \in T$, hacer lo siguiente:

Obtener el 2-bosque $B = T - \{[i,j]\}$.

Aplicar el algoritmo 2CDB para obtener el 2-CD (z_1, z_2) de B.

Si $f_\lambda(B,(z_1,z_2)) < f^*$ tomar $z^* = (z_1, z_2)$ y hacer $f^* = f_\lambda(B,(z_1,z_2))$.

Paso 3. Finalización.

El par z^* es el 2-Centdian de T.

Figura 24. **Algoritmo 2CDT**

Proposición 4.7.6 *El algoritmo 2CDT tiene complejidad* $O(n^2)$.

Demostración: Dado que el árbol T tiene $n \Leftrightarrow 1$ aristas, el algoritmo 2CDB se ejecuta $O(n)$ veces. Puesto que la complejidad de 2CDB es $O(n)$ se tiene la complejidad anunciada para 2CDT. □

Una mejora heurística al algoritmo 2CDT se consigue seleccionando las aristas a eliminar de una forma más inteligente. Es intuitivamente claro que los dos árboles que conforman el bosque que da lugar a la solución óptima deben estar equilibrados. Si una arista une dos árboles muy dispares en tamaño y peso seguramente no producirán buenos candidatos. Por tanto, sería conveniente empezar por las aristas que estén por la *mitad* del árbol.

Para ello se seleccionan las aristas $[i,j]$ a eliminar en un orden BFS a partir de una raíz del árbol. Como raíz del árbol se puede usar el Centro, la Mediana o el propio Centdian del árbol. En las experiencias realizadas se han obtenido los mejores resultados utilizando la Mediana del árbol como raíz.

Además si el recorrido BFS del árbol esta suficientemente alejado de la Mediana no se podrá mejorar al candidato a Centdian. En particular, si para una arista $[i,j]$, el Centdian del árbol (T_i o T_j) que contiene a la mediana da lugar a un valor de la función objetivo peor que el mejor candidato actual, el candidato obtenido del bosque correspondiente no mejorará al candidato. Además, tampoco lo mejorará ninguno de los candidatos surgidos al eliminar las aristas de las ramas que emanan de dicha arista en el recorrido BFS por lo que dicho recorrido puede ser truncado en tales aristas. El siguiente algoritmo $2CDTH$ aplica estas mejoras heurísticas.

Algoritmo 2CDTH

Paso 1. Inicialización.

Sea $f^* = \infty$. Obtener la Mediana y^* de T.

Paso 2. Iteraciones.

Desde $i = m$ como raíz del árbol T aplicar el siguiente procedimiento recursivo.

Para cada hijo j de i. Sea T_i el árbol que contiene a i al eliminar la arista $[i,j]$ del árbol T. Sea z_i^* el λ-Centdian de T_i. Si $f_\lambda(T_i, z_i^*) > f^*$ realizar la operación siguiente:

Obtener el 2-bosque $B = T - \{[i,j]\}$.

Aplicar el algoritmo 2CDB para obtener el 2-Centdian (z_1, z_2) de B.

Si $f_\lambda(B,(z_1,z_2)) < f^*$ tomar $z^* = (z_1, z_2)$ y hacer $f^* = f_\lambda(B,(z_1,z_2))$.

Tomar $i \leftarrow j$

Paso 3. Finalización.

El par z^* es el 2-Centdian de T.

Figura 25. **Algoritmo 2CDTH**

4.8 El p-λ-Centdian en la recta real.

En el caso de que cada vértice del árbol tenga a lo sumo un hijo, es decir, que el árbol sea un camino o un trozo de la recta real, la complejidad del problema p-Centdian (continuo y discreto) se reduciría a $O(pn^3)$. Para obtener esta complejidad, hay que tener en cuenta que el vértice v_k, de la definición del conjunto R_3 del teorema p-CD-árbol, debe coincidir con v_i o v_j. Por lo tanto, en este caso $|R_3| = O(n^2)$.

Proposición 4.8.1 *El problema de la p-Mediana r-restringido puede solucionarse en un tiempo $O(pn)$.*

Demostración: Cada vértice contribuye con a lo sumo 2 puntos en el camino cuya distancia desde el vértice es igual a r. Por lo cual, es suficiente considerar un problema discreto con al menos $3n$ puntos. Este problema discreto se soluciona en un tiempo $O(pn)$, si se utiliza el siguiente procedimiento de Hassin y Tamir [80]. Este procedimiento es similar al que se ha mostrado para el problema del Centdian en el árbol.

Sea

$$f_m(U,X) = \sum_{v_i \in U} w_i\, d(X, v_i).$$

con $w_i \geq 0$. Entonces para $v_j, v_k \in U$, $j < k$ se tiene que el coste de atender los puntos de j a k desde v_j o desde v_k es:

$$w^+(j,k) = \sum_{i=j}^{k} w_i(v_i \Leftrightarrow v_j) = \sum_{i=1}^{k} w_i v_i \Leftrightarrow \sum_{i=1}^{j-1} w_i v_i \Leftrightarrow (\sum_{i=j}^{k} w_i) v_j,$$

$$w^-(j,k) = \sum_{i=j}^{k} w_i(v_k \Leftrightarrow v_i) = \Leftrightarrow \sum_{i=1}^{k} w_i v_i + \sum_{i=1}^{j-1} w_i v_i + (\sum_{i=j}^{k} w_i) v_k,$$

Para simplificar la notación se define para $j = 1, ..., n$, lo siguiente:

$$A(j) = \sum_{i=1}^{j} w_i, \qquad AV(j) = \sum_{i=1}^{j} w_i v_i.$$

Por lo tanto, se puede reformular $w(j,k)$ y $w'(j,k)$ como se muestra a continuación:

$$w^+(j,k) = AV(k) \Leftrightarrow AV(j \Leftrightarrow 1) \Leftrightarrow (A(k) \Leftrightarrow A(j \Leftrightarrow 1)) v_j, \qquad (1.1a)$$

$$w^-(j,k) = \Leftrightarrow AV(k) + AV(j \Leftrightarrow 1) + (A(k) \Leftrightarrow A(j \Leftrightarrow 1)) v_k. \qquad (1.1b)$$

Se sigue de (1.1) que después de unos pasos previos para calcular A y AV que consumen un tiempo $O(n)$, tanto $w^+(j,k)$ como $w^-(j,k)$ pueden obtenerse en tiempo constante para un par fijado $j \leq k$. Para resolver el problema del p-Centdian se utilizan dos funciones recursivas denominadas $F(j)$ y $G(j)$ con $j = 1, .., n$. Donde $F(j)$ es el valor objetivo óptimo del subproblema reducido al conjunto de vértices $N^j = \{v_j, ..., v_n\}$ y $G(j)$ es el valor respectivo del mismo subproblema, siempre que un servicio esté en v_j.

$$G(j) = \min_{j < k \leq n+1} \{\Leftrightarrow AV(j \Leftrightarrow 1) + A(j \Leftrightarrow 1) v_j \Leftrightarrow A(k \Leftrightarrow 1) v_j + AV(k \Leftrightarrow 1) + F(k)\}.$$

$$F(j) = \min_{j < k \leq n} \{AV(j \Leftrightarrow 1) \Leftrightarrow A(j \Leftrightarrow 1) v_k + A(k) v_k \Leftrightarrow AV(k) + G(k)\}.$$

Se define para $j = 1, ..., n$,

$$F'(j) = F(j) + AV(j \Leftrightarrow 1),$$

$$G'(j) = G(j) \Leftrightarrow AV(j) + A(j) v_j.$$

Entonces las funciones se pueden reescribir como se muestra a continuación:

$$G(j) = \min_{j < k \leq n+1} \{\Leftrightarrow A(k \Leftrightarrow 1) v_j + F'(k)\}. \quad (2.2a)$$

$$F(j) = \min_{j<k\leq n} \{\Leftrightarrow A(j \Leftrightarrow 1)v_k + G'(k)\}. \quad (2.2b)$$

Para resolver este sistema de ecuaciones recursivas, considérese la ecuación $G(j)$, donde cada $k > j$ representa un punto en el plano con coordenadas $(A(k \Leftrightarrow 1), F'(k))$. Un punto $k = k(j)$ que minimiza la ecuación $G(j)$ se corresponde con un punto extremo de la envoltura convexa del conjunto planar de puntos $(A(k \Leftrightarrow 1), F'(k))$, $j < k \leq n+1$. Por esta razón sólo es necesario mantener estos puntos extremos. Este mínimo de todos ellos es determinado por el coeficiente $\Leftrightarrow v_j$. Este coeficiente es monótono en j y, por lo tanto, se asume que la secuencia de puntos mínimos es monótona. Es decir, $k(j) \leq k(j+1)$. El razonamiento es similar para la ecuación $F(j)$, pero con los puntos del plano, $(v_k, G'(k))$, $j \leq k \leq n$. La monotonía de estos puntos mínimos permite usar procedimientos estándares de geometría computacional, para construir y mantener eficientemente la envoltura convexa, cuando j varía de n a 1.

Todos los valores de $G(j)$ y $F(j)$, para $j = 1, ..., n$, son generados en tiempo lineal. La unión entre las dos envolturas convexas es gobernada por las ecuaciones (2.2a) y (2.2b). Específicamente se comienza con $G(n) = F(n)$, y recursivamente para $j = n \Leftrightarrow 1, ..., 1$ se calcula primero $G(j)$ y después $F(j)$. De esta manera se concluye que el tiempo total empleado es de $O(pn)$. □

4.9 Conclusiones.

En este capítulo se introduce el problema del Centdian generalizado, adaptándo las propiedades del 1-λ-Centdian al problema del 1-λ-Centdian generalizado, así como el algoritmo de Halpern.

Respecto al p-Centdian, primero hay que hacer notar que el modelo discreto en el que el conjunto p-Centdian está restringido al conjunto de vértices U, puede solucionarse en un tiempo $O(pn^4)$. Esto se deduce directamente del hecho de que en este caso, el conjunto R_1, del teorema p-CD-árbol debe incluir el valor óptimo r^*, y el problema de la p-Mediana r-restringido discreto se puede solucionar en un tiempo $O(pn^2)$, si se utiliza el algoritmo de Tamir (1996) [163]. También se ha añadido un algoritmo alternativo que resuelve el modelo discreto en tiempo $O(n^p)$ para $p < 4$.

El modelo (continuo) con $w'_i = 1,\ i = 1, ..., n$, es solucionable en un tiempo $O(pn^5)$. Lo cual se infiere del hecho de que el conjunto $R_1 \cup R_2$ del teorema p-CD-árbol, incluye el valor óptimo r^*, para este caso $R = R_1 \cup R_2$.

En el caso especial de la recta real, la complejidad del problema p-Centdian (continuo y discreto) se reduciría a $O(pn^3)$. Para obtener esta complejidad nótese

que el vértice v_k, de la definición del conjunto R_3 del teorema p-CD-árbol, debe coincidir con v_i o v_j. Por lo tanto, en este caso $|R_3| = O(n^2)$. Además, el problema de la p-Mediana r-restringido puede solucionarse en un tiempo $O(pn)$.

En vista de la relativa alta complejidad $O(pn^6)$, se plantea la relevante cuestión de si podría evitarse el explícito cálculo de la función $g(r)$, para todos lo valores de $r \in R$, definida en el teorema p-CD-árbol. Esto podría ser posible si la función $g(r)$ fuera convexa. Desafortunadamente, no es el caso, como se ilustra con el siguiente ejemplo.

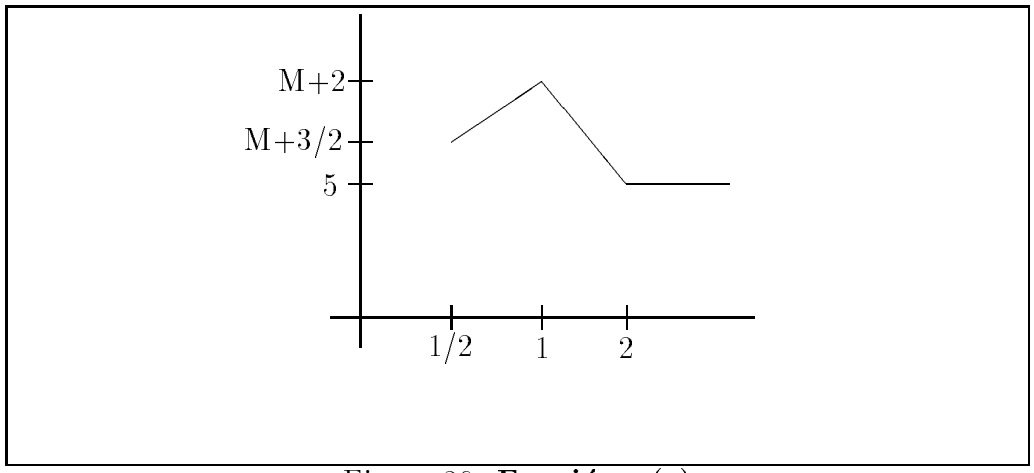


Figura 26. **Función g(r)**

(Ver la gráfica.-Función g(r))Como se puede observar $g(r)$ es lineal (creciente) para $1/2 \leq r \leq 1$, $g(r)$ es lineal (decreciente) para $1 \leq r \leq 2$, y $g(r)$ es constante (=5) para $r \geq 2$.

Apéndice A

CÓDIGO DEL 2-CENTDIAN EN UN GRAFO.

```
PROGRAM Dos_Cent_Dian_Grafo (input, output, Datos);
  {
  { HEURISTICA BASADA EN ARBOLES PARA EL 2CD EN UN GRAFO
  {
  { Dado un grafo con los pesos de los vertices y el valor de lambda
  { Se genera un arbol generador de partida aleatorio usando PRIM-P.
  { Se utilizan los movimientos ADD-DROP, determinando la arista que
  { entra con criterio ansioso (el primero mejor) y la que sale con
  { sale con criterio greedy (el mejor primero). Para cada arbol se
  { resuelve el 2-CentDian correspondiente por el algoritmo 2CDT.
  {
  { El algoritmo 2CDT rompe el arbol en dos formando un 2-bosque
  { al quitar una arista. La arista eliminada se selecciona en
  { orden BFS (Bread First Search) partiendo de la mediana como raiz.
  { Para cada 2-bosque se halla el 2CD por el algoritmo 2CDB.
  {
  { El algoritmo 2CDB calcula, para cada uno de los dos arboles,
  { el centro, mediana y cent-dian (c1,m1,z1) y (c2,m2,z2)
  { y sus radios (rc1,rm1,rz1) y (rc2,rm2,rz2).
  { Si rm1 < rz2 el 2cd es (m1,z2).
  { Si rm2 < rz1 el 2cd es (z1,m2).
  { En otro caso, se recorren a la vez los dos caminos que van
  { desde los centdianes z1 y z2 hacia las medianas m1 y m2
  { hasta que cambia de signo la pendiente de la funcion objetivo.
  }
```

```
USES Dos ;

CONST    Infin = 1000000;
         nMax  =  101;

TYPE     Vertices = 1..nMax ;
         IVector = array[Vertices] of Integer;
         RVector = array[Vertices] of Real;
         BVector = array[Vertices] of Boolean;

         VertPter = ^Vert;
         Vert = Record
                  Vertice : Vertices;
                  Next :    VertPter;
                End;

         NodoPter = ^Nodo;
         Nodo = Record
                  Vertice : Vertices;
                  Long : Real;
                  Next : NodoPter;
                End;
         ListaAdy = array[Vertices] of NodoPter;

         Punto = Record
                   vi, vj : Integer;
                   ti, tj : real;
                 End;

         Solucion = Record
                       punto1,punto2 :  punto;
                       Radio, suma : real;
                       Funcion : real
                     End;

         Arista = Record
                    e1,e2: Vertices ;
                    Long12 : Real ;
                  End ;
         AristVector = array[Vertices] of Arista ;

VAR    NomFich : String[8] ;
```

```
        Datos,Salida : Text ;

        nNodos, nAristas : Integer;
        Grafo,Arbol,Bosque : ListaAdy;
        Pesos : IVector ;
        Lambda : Real ;

        Sol,MejorSol : Solucion ;
        MejorObj : Real ;

        Tiempo,MejorTiempo : Real;

        Fuera : AristVector ;

        k,Entra : Integer ;

        Mejora : Boolean ;

FUNCTION CPUTime : Real ;

VAR h,m,s,c: Word ;

BEGIN
   GetTime(h,m,s,c);
   cputime := h*3600.0 + m*60. + s + 0.01*c ;
END ;

PROCEDURE EscribeSolucion( File : Text, Sol : Solucion ) ;

BEGIN
   With Sol
   do Begin
      writeln (' SOLUCION :');
      writeln (' Radio: ', Radio:6:2);
      writeln (' Suma : ', Suma:6:2);
      writeln (' Funcion lambda-cent-dian: ',funcion:6:2);
      With Punto1
      do writeln (' Punto 1: p([',vi,',',vj,'],',ti:4:2,').');
      With Punto2
      do writeln (' Punto 2: p([',vi,',',vj,'],',ti:4:2,').');
      end;
   Readln;
END ;
```

```
PROCEDURE LeerDatos(Var Adyac : ListaAdy ;
                    Var Pesos : IVector );

                   {  Este procedimiento lee el fichero de datos }
                   {  donde este el grafo y lo almacena en forma }
                   {  de lista de adyacendia. Las longitudes de  }
                   {  las aristas las guarda en un campo de los  }
                   {  registros que forman las listas. Los pesos }
                   {  de los nodos los guarda en un array aparte }

VAR Long : Integer ;
    i, j, k : Vertices ;
    nuevo, ultimo : NodoPter;

BEGIN
  Readln (Datos, nNodos, naristas);

  For i := 1 to nNodos
  do Adyac[i] := nil;               { Inicializa las listas }

  For k := 1 to naristas
  do Begin
     Readln(Datos, i, j, long);
     New(Nuevo);                       {Listas de adyacencia}
     Nuevo^.Vertice := j;
     Nuevo^.Long := long;
     Nuevo^.Next := Adyac[i];
     Adyac[i] := Nuevo ;
     New(Nuevo);                       { Listas de adyacencia }
     Nuevo^.Vertice := i;
     Nuevo^.Long := long;
     Nuevo^.Next := Adyac[j];
     Adyac[j] := Nuevo ;
     End;
END;

PROCEDURE PuntoVertice( Var p : Punto; v : Vertices) ;

BEGIN                        { Crea un punto p como el vertice v }
   With p
   do Begin
      vi := v ; vj := v ;
```

```
      ti := 0 ; tj := 0 ;
      End ;
END ;

PROCEDURE DistArbol( s : Vertices;
                Arbol : ListaAdy ;
          var    Dist : RVector;
          var   Padre : IVector;
          var Visitado : BVector ;
          var   Grado : IVector);

                        { Recorrido en profundidad desde s para }
                        { hallar en el arbol lo siguiente:      }
                        { - las distancias desde s a cada nodo, }
                        { - el vector Padre del camino minimo   }
                        { - los nodos Visitados del arbol y     }
                        { - el Grado de cada nodo.              }

   PROCEDURE DFS( s : Vertices;
              Arbol : ListaAdy ;
       var     Dist : RVector;
       var    Padre : IVector;
       var Visitado : BVector ;
      var    Grado : IVector);
                              { Procedimiento recursivo para       }
                              { el recorrido en profundidad DFS    }
                              { Depth First Search                 }
   VAR v : NodoPter;

   BEGIN { DFS }
      Visitado[s] := True;
      v := Arbol[s];
      While v <> nil
      do Begin
         Grado[s] := Grado[s] + 1;
         If Visitado[v^.Vertice] = False Then
            Begin
            Padre[v^.Vertice] := s;
            Dist[v^.Vertice] := Dist[s] + v^.Long;
            DFS(v^.Vertice, Arbol, Dist, Padre, Visitado, Grado);
            End;
         v := v^.Next;
         End;
```

```
   END; { DFS }

VAR   i : Vertices;

BEGIN { DistArbol }
   For i := 1 to nNodos
   Do Begin                    { Inicializacion de visitado y grado }
      Visitado[i] := False;
      Grado[i] := 0 ;
      End ;
   Padre[s] := s;
   Dist[s] := 0 ;                          { DFS desde el nodo raiz }
   DFS(s,Arbol,Dist,Padre,Visitado,Grado) ;
END ; { DistArbol }


PROCEDURE MasLejos( s : Vertices ;
                    Arbol  : ListaAdy ;
          var Vertmax : Vertices;
          var Dist     : RVector;
          var Padre    : IVector ;
          var Visitado : BVector ;
          var Grado    : IVector ;
          var Distmax : real);

                         { este procedure halla, dado un nodo s, }
                         { el mas alejado a s, las distancias y  }
                         { la distancia maxima.                  }
Var i : Vertices;

BEGIN
   DistArbol(s, Arbol, Dist, Padre, Visitado, Grado);
   Distmax := 0;
   Vertmax := s;
   For i := 1 to nNodos
   do If visitado[i] then
         If (dist[i] > dist[vertmax]) then
            Vertmax := i;
   Distmax := dist[vertmax];
END;

PROCEDURE CentroArbol( s : Vertices;
                   Arbol : ListaAdy;
```

```
          var     Centro : Punto ;
          var      Radio : Real);
                         { Halla el centro del arbol de raiz s }
                         { por el metodo de Handler, 1973.     }
VAR i, t : Vertices;
    dist : RVector ;
    distmax : real;
    Grado,Padre : IVector ;
    Visitado : BVector ;

BEGIN
   MasLejos(s, Arbol, t, Dist, Padre, Visitado, Grado, Distmax);
                               { Halla el vertice t mas alejado de s.}
   MasLejos(t, Arbol, s, Dist, Padre, Visitado, Grado, Distmax);
                               { Halla el vertice s mas alejado de t }
                               { y la distancia ente s y t.          }
   Radio := Distmax / 2;
   i := s;          { Busca el punto medio del camino que une s y t.}
   While Dist[Padre[i]] > Radio
   do i := Padre[i] ;
   With Centro
   do Begin
      ti := Dist[i] - Radio;
      vi := i;
      vj := Padre[i];
      tj := Radio - Dist[Padre[i]] ;
      End ;
END;

PROCEDURE MedianaArbol( Raiz : Vertices;
                        Arbol : ListaAdy;
              var  Pendiente : IVector ;
              var  Padre     : IVector ;
              var  Mediana   : Vertices);
                               { Halla las medianas de un arbol por}
                               { el algoritmo de Goldman. Pero no  }
                               { se detiene al encontrar la mediana}
                               { obteniendo asi las pendientes de  }
                               { la funcion objetivo en la arista  }
                               { que une cada vertice con su padre }
VAR Hojas,HojaPter : VertPter ;
    i, hoja : Vertices ;
    j1,j2,j3 : Vertices ;
```

```
    Encontrado : boolean;
    Activo : BVector;
    Dist : RVector ;
    nNodosArbol : Integer ;
    Pesos,Grado : IVector ;

BEGIN   { MedianaArbol }
   DistArbol(Raiz,Arbol,Dist,Padre,Activo,grado) ;
                              { Calcula los grados y activos }
   nNodosArbol := 0 ;
               { nNodosArbol es el numero de nodos del rbol }
   For i := 1 to nNodos
   do If Activo[i] Then
         nNodosArbol:= nNodosArbol + 1 ;
   For i := 1 to nNodos
   do If Activo[i] Then  { Para la version con pesos no se inicializa }
         Pesos[i] := 1 ;
   If nNodosArbol = 1    { Hay que tener en cuenta aparte este caso }
   Then Begin
        Mediana := Raiz ;
        Padre[Mediana] := Mediana ;
        Pendiente[Mediana] := 0 ;
        Exit ;
        End ;

   Hojas := Nil ;                          { Crea la lista de hojas }
   For i := 1 to nNodos
   do If (Grado[i] = 1) and Activo[i] Then
         Begin
         New(HojaPter) ;
         HojaPter^.Vertice := i ;
         HojaPter^.Next := Hojas ;
         Hojas := HojaPter ;
         End ;

   Encontrado := False ;
   Repeat                                 { Toma una hoja del arbol }
     Hoja := Hojas^.Vertice ;
     If (Encontrado = False) Then
        If ((Pesos[hoja] + Pesos[hoja]) >= nNodosArbol) Then
           Begin                                { La hoja es mediana }
           Encontrado := True ;
           Mediana := Hoja ;
```

```
        End ;
   If Hoja = Raiz Then
      Begin                 { en este caso solo elimina la hoja }
      HojaPter := Hojas ;
      Hojas := Hojas^.Next ;
      Dispose(HojaPter) ;
      End
   Else
      Begin
      Pesos[Padre[hoja]] := pesos[padre[hoja]] + pesos[hoja];
      Grado[Padre[hoja]] := grado[padre[hoja]] - 1;
      If Grado[Padre[hoja]] = 1 Then
         Hojas^.vertice := Padre[hoja]{ el padre de la hoja pasa }
      Else                            { a ser una nueva hoja     }
         Begin
         HojaPter := Hojas ;          { desaparece la hoja y no }
         Hojas := Hojas^.Next ;       { aparece ninguna nueva   }
         Dispose(HojaPter) ;
         End ;
      End ;
Until Hojas = Nil ;
            { Se le da la vuelta al camino de la mediana a la raiz }
            { determinando el nuevo vector Padre[.] para que quede }
            { la mediana como raiz del arbol                       }

j1 := Mediana ;
j2 := Padre[j1] ;
j3 := Padre[j2] ;
While j1 <> Raiz                  {    Reasignacion de enlaces }
do Begin                          {    <--- j3 <--- j2 <--- j1 }
   Padre[j2] := j1 ;              {    <--- j3      j2 ---> j1 }
   j1 := j2 ;
   j2 := j3 ;                     {         j3      j2      j1 }
   j3 := Padre[j2] ;              { j3      j2      j1         }
   End ;
Padre[Mediana] := Mediana ;

j1 := Raiz ;            { Se invierten los pesos de este camino }
While j1 <> Mediana
do Begin
   Pesos[j1] := nNodosArbol-Pesos[Padre[j1]] ;
   j1 := Padre[j1] ;
   End ;
```

```
   For i := 1 to nNodos { Calculo de la pendiente de la funcion mediana }
   do If Activo[i] Then
         Pendiente[i] :=  nNodosArbol- (Pesos[i] + Pesos[i]) ;
   Pendiente[Mediana] := 0 ;
END; { Mediana Arbol }

PROCEDURE CentDianArbol( s : Vertices ;
                     Arbol : ListaAdy ;
          var Centdian     : punto;
          var RadioCD      : real ;
          var Centro       : punto ;
          var RadioCentro  : Real ;
          var Mediana      : Vertices ;
          var RadioMediana : Real ;
          var Pendiente    : IVector;
          var DistMediana  : RVector ;
          var PadreMed     : IVector ;
          var LejosMed     : Vertices );
                                 {  Halla el Cent-dian de un arbol }
                                 {  por el algoritmo CDT, metodo de}
                                 {  de Perez, Moreno & Rguez (1998)}
                                 {  Aportando ademas el centro,    }
                                 {  la mediana (con sus radios) y  }
                                 {  las pendientes en el camino    }
                                 {  hacia la mediana.              }
VAR x : real;
    v : Vertices;
    Visitado : BVector ;
    Padre,Grado : IVector ;

BEGIN { CentDianArbol }

{ 1. Calcula el centro del arbol }

   CentroArbol (s, Arbol, Centro, RadioCentro);

{ 2. Calcula la mediana del arbol }

   MedianaArbol(s, Arbol, Pendiente, PadreMed, Mediana) ;

{ 3. Calcula el vertice mas alejado a la mediana }
```

```
   MasLejos   (Mediana, Arbol, LejosMed, DistMediana,
                        Padre, Visitado, Grado, RadioMediana);

{ 4. Mueve desde el centro hacia la mediana hasta que
     cambia de signo la pendiente de f_lambda;
     cuando la de la f_mediana llega a lambda/(1-lambda) }

   x := lambda/(1-lambda) ;
   With Centro
   do If DistMediana[vi] < DistMediana[vj]
                                { v es un extremo de la arista }
         Then v := vj           { que contiene al centro, el   }
         Else v := vi ;         { mas alejado a la mediana     }

   If Pendiente[v] <= x Then    { En este caso, no se mueve;  }
      Begin                     { el centro es el centdian    }
      CentDian := Centro ;
      RadioCD := RadioCentro ;
      End
   Else
      Begin                     { Recorrido del camino desde }
      While pendiente[v] > x    { el centro hacia la mediana }
      do v := Padre[v] ;        { hasta que la pendiente sea }
      PuntoVertice(CentDian,v) ;  { mayor o igual a "x" }
      RadioCD := RadioMediana - DistMediana[v] ;
      End ;

END; { CentDianArbol }

FUNCTION SumaMediana(p : Punto ;
               Arbol  : ListaAdy ) : Real ;

Var    Suma : Real ;
       i : Vertices ;
       Dist1,Dist2 : RVector ;
       Visitado : BVector ;
       Grado,Padre : IVector ;

BEGIN
   Suma := 0.0 ;                { Calcula las distancias desde p }
   With p                       { a los vertices del arbol de p  }
   do Begin
      DistArbol(vi,Arbol,Dist1,Padre,Visitado,Grado) ;
```

```
      DistArbol(vj,Arbol,Dist2,Padre,Visitado,Grado) ;
      For i := 1 to nNodos
      do If visitado[i] Then
         If ti+ Dist1[i] < tj + Dist2[i] Then
            Suma := Suma + ti + Dist1[i]
         Else
            Suma := Suma + tj + Dist2[i] ;
      End ;
   SumaMediana := Suma ;

END ;

PROCEDURE DisCDBosque ( s1,s2 : Vertices ;
                        Bosque : ListaAdy;
                        Lambda : Real;
                    Var    Sol : Solucion );

{Dado un 2-bosque halla los cent-dianes y medianas de ambos arboles.  }
{Si RadioMediana1 < RadioCentdian2 la solucion es (Mediana1,CentDian2)}
{Si RadioMediana2 < RadioCentdian1 la solucion es (Mediana1,CentDian2)}
{En otro caso se mueven a la vez los centdianes hacia las medianas.   }

VAR
    Lejos1,Lejos2 : Vertices ;
    CentDian1,CentDian2,
    Centro1,Centro2,p : punto ;
    Mediana1,Mediana2 : Vertices ;
    Pendiente1,Pendiente2 : IVector ;
    RadioCentro1,RadioCentro2 : real ;
    RadioMediana1,RadioMediana2 : real ;
    RadioCentDian1 : Real ;
    RadioCentDian2 : real ;
    v10,v20,v1,v2,i : Vertices;
    Dist1,Dist2 : RVector ;
    Visitado1,Visitado2 : BVector ;
    Padre1,Padre2 : IVector ;
    x : Real ;

BEGIN     { DosCDBosque }

   CentDianArbol(s1, Bosque,
                     CentDian1, RadioCentDian1,
                     Centro1, RadioCentro1,
```

```
                  Mediana1, RadioMediana1,
                  Pendiente1, Dist1, Padre1, Lejos1) ;

CentDianArbol(s2, Bosque,
                  CentDian2, RadioCentDian2,
                  Centro2, RadioCentro2,
                  Mediana2, RadioMediana2,
                  Pendiente2, Dist2, Padre2, Lejos2) ;

If RadioMediana2 <= RadioCentDian1 Then
   Begin                  { La solucion es la formada por }
   With Sol               { - El centdian del arbol 1 }
   Do Begin               { - La mediana del arbol 2  }
      Punto1 := Centdian1 ;
      PuntoVertice(p,Mediana2) ;
      Punto2 := p ;
      Radio := RadioCentDian1 ;
      Suma := SumaMediana(Punto1,Arbol) + SumaMediana(Punto2,Arbol) ;
      Funcion := lambda * Radio + (1 - lambda) * Suma ;
      End ;
   Exit ;
   End ;

If RadioMediana1 <= RadioCentDian2 Then
   Begin                  { La solucion es la formada por }
   With Sol               { - El centdian del arbol 2 }
   Do Begin               { - La mediana del arbol 1  }
      PuntoVertice(p,Mediana1) ;
      Punto1 := p ;
      Punto2 := Centdian2 ;
      Radio := RadioCentDian2 ;
      Suma := SumaMediana(Punto1,Arbol) + SumaMediana(Punto2,Arbol) ;
      Funcion := lambda * Radio + (1 - lambda) * Suma ;
      End ;
   Exit ;
   End ;
                     { v1 es el extremo mas alejado a Mediana1 }
                     { de la arista que contiene al CentDian1. }
With CentDian1
do If Dist1[vi] > Dist1[vj]
      Then v10 := vi
      Else v10 := vj ;
                     { v2 es el extremo mas alejado a Mediana2 }
```

```
                    { de la arista que contiene al CentDian2. }
With CentDian2
do If Dist2[vi] > Dist2[vj]
      Then v20 := vi
      Else v20 := vj ;

    { v1 y v2 recorren los vertices de los caminos respectivos }

v1 := v10 ;
v2 := v20 ;
x := lambda/(1-lambda) ;
                { Se van moviendo los dos vertices v1 y v2 }
                { hacia sus medianas m1 y m2 hasta que     }
                { la pendiente cambia de signo             }
                { En cada caso se mueve el de menor radio  }
While Pendiente1[v1] + Pendiente2[v2] > x
do If RadioMediana1-Dist1[Padre1[v1]] <
                           RadioMediana2-Dist2[Padre2[v2]]
      Then v1 := Padre1[v1]
      Else v2 := Padre2[v2] ;

If (v1 = v10) and (RadioCentdian1 > RadioCentdian2)
          { No se ha movido v1. Entonces   }
Then Begin{ La solucion es la formada por  }
          { CentDian1 y z2(RadioCentDian1) }
     While RadioMediana2 - Dist2[Padre2[v20]] < RadioCentDian1
     do v20 := Padre2[v20] ;
     With Sol
     do Begin
        Punto1 := Centdian1 ;
        With Punto2
        do Begin
           vi := v20 ;
           vj := Padre2[v20] ;
           ti := RadioCentDian1 - (RadioMediana2 - Dist2[v20]) ;
           tj := (RadioMediana2- Dist2[Padre2[v20]])
                                    - RadioCentDian1 ;
           End ;
        Radio := RadioCentDian1 ;
        Suma := SumaMediana(Punto1,Arbol)
                                + SumaMediana(Punto2,Arbol) ;
        Funcion := lambda * Radio + (1 - lambda) * Suma ;
        End ;
```

```
     Exit
     End
Else
If (v2 = v20) and (RadioCentdian2 > RadioCentdian1)
          { No se ha movido z2 }
Then Begin{ La solucion es la formada por CD2 }
          { y el punto z1(RadioCentDian2)     }
     While RadioMediana1 - Dist1[Padre1[v10]] < RadioCentDian2
     do v10 := Padre1[v10] ;
     With Sol
     do Begin
        Punto2 := Centdian2 ;
        With Punto1
        do Begin
           vi := v10 ;
           vj := Padre1[v10] ;
           ti := RadioCentDian2 - (RadioMediana1 - Dist1[v10]) ;
           tj := (RadioMediana1 - Dist1[Padre1[v10]])
                                          - RadioCentDian2 ;
           End ;
        Radio := RadioCentDian2 ;
        Suma := SumaMediana(Punto1,Arbol)
                               + SumaMediana(Punto2,Arbol) ;
        Funcion := lambda * Radio + (1 - lambda) * Suma ;
        End ;
     Exit ;
     End
Else
             { La solucion esta formada por uno de los vertices }
             { (el que tenga mayor radio) !! Por eso el "If"    }
             { Y el punto del otro camino que iguala su radio   }

      If RadioMediana1-Dist1[v1] > RadioMediana2-Dist2[v2] Then
         With Sol
         do Begin
            PuntoVertice(p,v1) ;
            Punto1 := p ;
            Punto2.vi := v2 ;
            Punto2.vj := Padre2[v2] ;

            Punto2.ti := (RadioMediana1-Dist1[v1])
                               - (RadioMediana2-Dist2[v2])  ;
```

```
               Punto2.tj := (RadioMediana2-Dist2[Padre2[v2]])
                                  - (RadioMediana1-Dist1[v1]);

               Radio := RadioMediana1-Dist1[v1] ;

               Suma := SumaMediana(Punto1,Arbol)
                                  + SumaMediana(Punto2,Arbol) ;

               Funcion := lambda * Radio + (1 - lambda) * Suma ;
               End
         Else
            With Sol
            do Begin
               PuntoVertice(p,v2) ;
               Punto2 := p ;
               Punto1.vi := v1 ;
               Punto1.vj := Padre1[v1] ;

               Punto1.ti := (RadioMediana2-Dist2[v2])
                                  - (RadioMediana1-Dist1[v1]) ;

               Punto1.tj := (RadioMediana1-Dist1[Padre1[v1]])
                                  - (RadioMediana2-Dist2[v2]) ;

               Radio := RadioMediana2-Dist2[v2] ;

               Suma := SumaMediana(Punto1,Arbol)
                                  + SumaMediana(Punto2,Arbol) ;
               Funcion := lambda * Radio + (1 - lambda) * Suma ;
               End ;
END;  { DosCDBosque }

PROCEDURE QuitaArista(e1,e2 : Vertices ;
          var Arbol : ListaAdy ;
          var long : Real );                  { Variable de salida }

VAR p,q : NodoPter ;

BEGIN
  p := Arbol[e1];
  If p^.vertice = e2 Then{ es el primero en la lista de adyacencia }
     Begin
     Arbol[e1]:= p^.next ;
```

```
     Dispose(p);
     End
  Else Begin          { NO es el primero en la lista de adyacencia }
       while p^.vertice <> e2
       do Begin
          q := p ;
          p := p^.next ;
          End ;
       long := p^.long ;
       q^.next := p^.next;
       dispose(p);
       End ;

  p := Arbol[e2];
  If p^.vertice = e1 Then{ es el primero en la lista de adyacencia }
     Begin
     Arbol[e2]:= p^.next ;
     Dispose(p);
     End
  Else Begin          { NO es el primero en la lista de adyacencia }
       while p^.vertice <> e1
       do Begin
          q := p ;
          p := p^.next ;
          End ;
       long := p^.long ;
       q^.next := p^.next;
       dispose(p);
       End ;
END ; { QuitaArista }

PROCEDURE PoneArista(  e1,e2 : Vertices;
                   Var Arbol : ListaAdy ;
                       long : Real ) ;

VAR Nuevo : NodoPter ;

BEGIN
   new(nuevo);
   nuevo^.vertice := e2;
   nuevo^.long := long;
   nuevo^.next := Arbol[e1] ;
   Arbol[e1] := Nuevo ;
```

```
   new(nuevo);
   nuevo^.vertice := e1;
   nuevo^.long := long;
   nuevo^.next := Arbol[e2] ;
   Arbol[e2] := Nuevo ;

END ;

PROCEDURE DosCDArbol (
          Var           Arbol : ListaAdy ;
                        Pesos : IVector;
          Var           MejorSol : Solucion );

                        { Calcula el 2-CD en el arbol eliminando   }
                        { una arista que recorre el grafo en orden }
                        { BFS partiendo de la mediana como raiz y  }
                        { calculando el 2-CD en cada bosque        }

VAR     v,Mediana : Vertices ;
        Sol : Solucion;
        BFSVisitado : BVector ;
        Radio1,Suma1,Funcion1 : Real ;
        PadreMed,Padre,Grado : IVector ;
        DistMediana : RVector ;

        v1,v2 : Vertices ;
        vList,v2Item,v2Lista,
        item,item1 : VertPter ;
        e : NodoPter;
        Long : Real ;


        Centro1,CentDian1 : Punto ;
        RadioCentDian1, RadioCentro1, RadioMediana1 : Real ;
        Pendiente1 : IVector ;
        Padre1 : IVector ;
        Mediana1,Lejos1: Vertices ;
        Dist1 : RVector ;


BEGIN    { DosCDArbol }
   MejorSol.funcion := Infin ;
```

```
MedianaArbol(1, Arbol, pesos, PadreMed, Mediana) ;

New (vList) ;          { Lista de vertices a visitar por el BFS }
With vList^
do Begin
   Next := Nil ;
   Vertice := Mediana ;
   end ;
For v := 1 to nNodos
do BFSvisitado[v] := False ;

While vList <> Nil
do Begin
   v1 := vList^.vertice ;       { recorriendo por el vertice v1 }
   e := Arbol[v1] ;
   BFSvisitado[v1] := True ;
   v2Lista := Nil ;   { Lista a anadir a los vertices a visitar }
   Repeat             { formada por: los adyacentes a v1        }
      New(v2Item);
      With v2Item^
      do Begin
         vertice := e^.vertice ;
         next := v2lista ;
         end ;
      v2lista := v2Item ;
      e := e^.next ;
   Until e = Nil ;

   v2Item := v2Lista ;
   Repeat
      v2 := v2Item^.vertice ;

      QuitaArista(v1,v2,Arbol,long) ;

{ Se resuelve el Centdian en el arbol que contiene a la mediana }

      CentDianArbol( v1, Arbol,CentDian1, RadioCentDian1,
                     Centro1, RadioCentro1,
                     Mediana1, RadioMediana1,
                     Pendiente1, Dist1, Padre1, Lejos1) ;

      Radio1 := RadioCentDian1 ;
      suma1 := SumaMediana(CentDian1 , Arbol) ;
```

```
      Funcion1 := lambda * radio1 + (1 - lambda) * suma1;

              { Solo si el CentDian en este arbol no es peor }
              { que la mejor solucion encontrada hasta ahora }
              { se continua por la rama BFS del arbol.       }

      If Funcion1 < MejorSol.funcion Then
         Begin
         If Not BFSvisitado[v2] Then
                               { Si v2 aun no ha sido visitado    }
            Begin              { se anade v2 al final de la lista }
            New(item1) ;
            item := vList ;
            While item^.Next <> Nil
            do item := item^.next ;
            item^.next := item1 ;
            with item1^
            do Begin
               vertice := v2 ;
               next := Nil ;
               End ;
            End ;
                                { Se resuelve el DOSCDBOSQUE }
         DosCDBosque (v1,v2,Arbol, lambda, Sol);

         If (Sol.funcion < MejorSol.funcion) Then
            MejorSol := Sol ;
         End ;

      PoneArista(v1,v2, Arbol,long) ;

      v2Item := v2Item^.next ; {Se toma otra arista [v1,.] a quitar}
   Until v2Item = Nil ;

   v2Item := v2lista ;            { Vaciado de la lista v2Lista }
   While v2Item <> nil
   do Begin
      v2lista := v2lista^.next ;
      Dispose (v2Item) ;
      v2Item := v2Lista ;
      End ;

   item := vList ;
```

```
      vList := vList^.next ;
      Dispose(item) ;      { Se elimina elemento de la lista vList }

      End;    {Fin del WHILE que recorre los vertices en BFS }

END ; { DosCDArbol }

PROCEDURE Mueve(
          Var    Arbol : ListaAdy;
          Var    Fuera : AristVector ;
          Var    MejorSol : Solucion );
                              { HEURISTICA ANSIOSA-GREEDY     }
                              { Usa el Movimiento ADD-DROP    }
                              { Selecciona la arista entrante }
                              { con la regla el primero mejor }
                              { y despues la arista saliente  }
                              { con la regla el mejor primero }

Var v1,v2,w1,w2,w1opt,w2opt : Vertices ;
    Dist1 : RVector ;
    Visitado : BVector ;
    Padre, Grado : IVector ;
    InicObj,MejorObj, Lv12,Lw12,Lw12opt : Real ;
    InicEntra,BestEntra : Integer ;
    Sol,InicSol :  Solucion ;

BEGIN
   MejorObj := MejorSol.Funcion ;

   DosCDArbol( Arbol, Pesos , InicSol ) ;

   InicObj := InicSol.Funcion ;
   If (Entra < 1) or (Entra > nAristas-nNodos+1) Then
      Entra := 1 ;
   InicEntra := Entra ;
   Repeat
      v1 := Fuera[Entra].e1 ;
      v2 := Fuera[Entra].e2 ;
      Lv12 := Fuera[Entra].Long12 ;

                                  { Se determina el vector Padre }
      DistArbol(v1,Arbol,Dist1,Padre,Visitado,Grado) ;
```

```
                    { Se aade la arista [v1,v2] de longitud Lv12 }
     PoneArista(v1,v2, Arbol,Lv12 ) ;
     w1 := v2 ;
     Repeat
        w2 := Padre[w1] ;
                    { Se quita la arista [w1,w2] de longitud Lw12 }
        QuitaArista(w1,w2, Arbol,Lw12 ) ;

        DosCDArbol( Arbol, Pesos , Sol ) ;
        If Sol.Funcion < MejorObj
        Then Begin
             MejorObj := Sol.Funcion ;
             MejorSol := Sol ;
             w1opt := w1 ;
             w2opt := w2 ;
             Lw12opt := Lw12 ;
             BestEntra := Entra ;
             Fuera[BestEntra].e1 := w1opt;
             Fuera[BestEntra].e2 := w2opt ;
             Fuera[BestEntra].Long12 := Lw12opt ;
             Exit ;
             End ;
                  { Se re-pone la arista [w1,w2] de longitud Lw12 }
        PoneArista(w1,w2, Arbol,Lw12) ;
        w1 := w2 ;
    Until w1 = v1 ;

    QuitaArista(v1,v2, Arbol,Lv12 ) ;
    Entra := Entra + 1 ;
    If (entra > nAristas-nNodos+1) Then
       entra := 1 ;
  Until Entra = InicEntra ;

  If MejorObj < InicObj Then
     Begin
     v1 := Fuera[BestEntra].e1 ;
     v2 := Fuera[BestEntra].e2 ;
     Lv12 := Fuera[BestEntra].Long12 ;
     PoneArista(v1,v2, Arbol,Lv12 ) ;

               { Se quita la mejor arista [w1,w2] de longitud Lw12 }
     QuitaArista(w1opt,w2opt, Arbol,Lw12opt ) ;
                { La arista quitada ocupa el lugar de la que entra }
```

```
     Fuera[BestEntra].e1 := w1opt;
     Fuera[BestEntra].e2 := w2opt ;
     Fuera[BestEntra].Long12 := Lw12opt ;
     End
  Else MejorSol := InicSol ;

END ;

PROCEDURE SemiPrim ( Grafo : ListaAdy ;
          Var    Arbol : ListaAdy ;
          Var    Fuera : AristVector ) ;
                          { Algoritmo de Prim modificado con }
                          { componente aleatoria             }
  VAR i,j,k,s  : Vertices ;
      Edge,Nuevo : NodoPter ;
      Vert : VertPter ;
      Level : RVector ;
      Dentro : BVector ;
      Padre : IVector ;
      NumEnlaces : IVector ;
      ListaEntra,ItemEntra,UltimoEntra : VertPter ;
      Count : Integer ;

  BEGIN (* SEMI-PRIM *)
  For i := 1 to nNodos                  { Inicializaciones }
  do Begin
     Level[i] := MaxInt;
     Dentro[i] := False ;
     NumEnlaces[i] := 0 ;
     End ;

  s := 1 ;                              { La raiz es 1 }
  Level[s] := 0;
  Padre[s] := s ;
  New(ListaEntra) ;                     { Lista de vertices a entrar }
  With ListaEntra^
  do Begin
     Vertice := s ;
     Next := Nil ;
     End ;
  UltimoEntra := ListaEntra ;
  For k := 1 to nNodos
  do Begin (* ITERATION *)
```

```
    i := ListaEntra^.Vertice ; { Entra el primer vertice de la lista }
    Dentro[i] := True ;
    Edge := Grafo [i];
    While Edge <> Nil
    do Begin
       j := Edge^.Vertice ; { Con los adyacentes al vertice entrante }
       If Dentro[j] = False Then   { que no hayan entrado }
          Begin { IF-1 }
          If Level[j] = MaxInt Then
             Begin        { Si no hay ningun enlace se conecta a i }
             Level[j] := Edge^.Long ;
             Padre[j] := i ;
             NumEnlaces[j] := 1 ;
             New(ItemEntra) ;
             With ItemEntra^
             do Begin
                Vertice := j ;
                Next := Nil ;
                End ;
             UltimoEntra^.Next := ItemEntra ;
             UltimoEntra := ItemEntra ;
             End
          Else
             Begin
             NumEnlaces[j] := NumEnlaces[j] + 1 ;
                 If Random < 1.0/NumEnlaces[j]
                             { Si hay ya hay enlaces }
                 Then Beginm { con esta probabilidad el enlace es i }}
                      Level[j] := Edge^.Long ;
                      Padre[j] := i ;
                      End ;
             End ;
          End ; { IF-1 }
       Edge := Edge^.Next;
       End;     { ENDWHILE }
    ItemEntra := ListaEntra ;
    ListaEntra := ListaEntra^.Next ;
    Dispose(ItemEntra) ;
    End  { Iteracion };
                 { Creacion de las listas de adyacencia del arbol }

For i := 1 to nNodos
do Arbol[i] := nil;             { Inicializa las listas }
```

```
  For i := 1 to nNodos
  do For j := 1 to nNodos
     do If (i <> j) Then
           If Padre[j] = i Then
              Begin
              New(Nuevo);              {Listas de adyacencia}
              Nuevo^.Vertice := j;
              Nuevo^.Long := Level[j];
              Nuevo^.Next := Arbol[i];
              Arbol[i] := Nuevo ;

              New(Nuevo);              {Listas de adyacencia}
              Nuevo^.Vertice := i;
              Nuevo^.Long := Level[j];
              Nuevo^.Next := Arbol[j];
              Arbol[j] := Nuevo ;
              End; {for}
  Count := 0 ;
       { lista de las aristas del grafo que quedan fuera del arbol }
  For i := 1 to nNodos
  do Begin
     Edge := Grafo [i];
     While Edge <> Nil
     do Begin
        j := Edge^.Vertice ;
        If (Padre[i] <> j) and (Padre[j] <> i) and (i < j)
        Then Begin
             count := count+1 ;
             With Fuera[count]
             do Begin
                e1 := i ;
                e2 := j ;
                Long12 := Edge^.long ;
                End ;
             End ;
        Edge := Edge^.Next ;
        End ;
     End ;

END ; { SEMI-PRIM }

PROCEDURE EscribeGrafo( Grafo : ListaAdy ) ;
```

```
var  e : NodoPter ;
     i : Integer ;


BEGIN
   Writeln('Grafo:');
   For i := 1 to nNodos
   do Begin
      Write(i:3,': ');
      e := Grafo[i] ;
      Repeat
        Write(e^.Vertice:3);
        e := e^.Next ;
      Until e = Nil ;
      Writeln;
      End ;
END ;

BEGIN   {PROGRAMA PRINCIPAL}

  Writeln(' Nombre del fichero (sin extensin) que contiene el grafo');
  Readln (NomFich) ;

  Assign ( Datos, 'c:\otros\Dioni\tesis\'+Nomfich+'.dat') ;
  Reset (Datos);

  Assign (Salida, 'c:\otros\Dioni\tesis\'+Nomfich+'.sal') ;
  Rewrite (Salida);
  Writeln(Salida,'Problemas de ',Nomfich);

   LeerDatos( Grafo,Pesos) ;
   Lambda := 0.8 ;

   For k := 1 to nNodos do
       pesos[k] := 1;

   randomize ;

   Tiempo := cputime;

   TotalMejorSol.Funcion := MaxLongInt ;
   MejorSol.Funcion := MaxLongInt ;
   Repeat
```

```
   SemiPrim (Grafo, Arbol, Fuera ) ;
   DOSCDARBOL ( Arbol, Pesos , MejorSol );
   Sol := MejorSol ;
   Repeat
      Mueve(Arbol,Fuera,Sol) ;
      If Sol.Funcion < MejorSol.Funcion Then
         Begin
         Mejora := True ;
         MejorSol := Sol ;
         Mejortiempo := cputime - tiempo ;
         Writeln ('MEJOR SOLUCION :');
         EscribeSolucion(Sol) ;
         Writeln(' Mejor Tiempo: ',Mejortiempo:8:2,' segundos');
         Writeln('Pulsa Return'); Readln ;
         End
      Else Mejora := False ;
   Until Not Mejora ;
   If Sol.Funcion < TotalMejorSol.Funcion Then
      Begin
      TotalMejorSol := Sol ;
      Mejortiempo := cputime - tiempo ;
      Writeln ('MEJOR SOLUCION :');
      EscribeSolucion(Sol) ;
      Writeln(' Mejor Tiempo: ',Mejortiempo:8:2,' segundos');
      Writeln('Pulsa Return'); Readln ;
      End ;

Until (cputime - tiempo > 180);

Tiempo := cputime - tiempo;
With MejorSol
do Begin
      Writeln (Salida,'MEJOR SOLUCION :');
      Writeln (Salida,' radio: ', Radio:6:2);
      Writeln (Salida,' suma : ', Suma:6:2);
      Writeln (Salida,' funcion lambda-cent-dian: ',funcion:6:2);
      With Punto1
      do Writeln (Salida,' punto1: p([',vi,',',vj,'],',ti:4:2,').');
      With Punto2
      do Writeln (Salida,' punto2: p([',vi,',',vj,'],',ti:4:2,').');
      end;
writeln(Salida);
writeln(Salida,' Mejor Tiempo: ',Mejortiempo:8:2,' segundos');
```

```
   Writeln(Salida,MejorSol.funcion:6:2,Mejortiempo:6:2);
   Writeln(MejorSol.funcion:6:2,Mejortiempo:6:2);

   Close (Datos);

   Close (Salida) ;

END.
```

Apéndice B

CÓDIGO DE LA HEURÍSTICA VNDS.

```
*     Variable Neighbourhood Decomposition Search (VNDS)
*     for p-median problem of the vertex set of a graph.
* ------------------------------------------------------------------
*     Authors: Pierre Hansen, Nenad Mladenovic and Dionisio Perez Brito
* ------------------------------------------------------------------
*
*     Data:
*     -----
*     n - number of vertices
*     p - number of medians
*     d - distance matrix
*
*     Variables:
*     ----------
*     m(j) - median where vertex j is assigned (solution)
*     f - solution value obtained by this heuristic
*
*     Auxiliary Variables:
*     --------------------
*     x(j) - solution
*     m(j) - median
*     s(j) - second median
*
```

```
*     ---------------------------------------------------------------
      integer     n,p
      real        d(6000,6000)
      real*8      seed
*     ---------------------------------------------------------------
*
      write(*,*)'Enter: "1" for solving SINGLE problem'
      write(*,*)'       "2" for ORLIB test problems'
      read(*,*)input
      if(input.eq.1)then
         read(*,*) n,p,d
      else
         write(*,*)' Enter file '
         read(*,*) n,p,d
      endif
*
*     CONSTANTS:
*    ---------------------------------- seed for pseudo-random numbers
      seed=21.d+00
      big=1.e20
*
*     Stop parameters:
*     ---------------
      MaxNoImpIt =    1000
*    -------------------------- Max. Number of No Improvement Iterations
      MaxItInt   =   10000
*    ---------------------------- Max. Number of Interchange Iterations
      maxitVNS   = 1000000
*    ------------------------------------ Max. Number of VNS Iterations
      maxitVNDS  =  100000
*    ----------------------------------- Max. Number of VNDS Iterations
      kmaxVNS    = p
*    ---------------------------------------- Max. value of k for VNS
      kmaxVNDS   = p
*    --------------------------------------- Max. value of k for VNDS
      kmaxMC-VNS = 2
*    ---------------------------------------- Max. value of k for MC-VNS
*
      call VNDS(f,x,m,s,d,n,p,MaxTimeVNDS,MaxTimeVNS,TotalVNDS)
      stop
      end
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
      subroutine InitSol(f,x,m,s,d,n,p)
*     ------------------------------------------------ Random Solution
      real       d(6000,6000)
      integer    m(6000),s(6000),x(6000),p,n
*     ---------------------------------------
      do j=1,n
         x(j)=j
      enddo
      do j=1,p
         i=j+(n-j)*GGUBFS(seed)
         xj=x(j)
         x(j)=x(i)
         x(i)=xj
      enddo
      call FindMedian(x,m,d,n,p)
*     ------------------------------------- Compute the objective value
      f=0.
      do i=1,n
         f=f+d(x(i),m(x(i)))
      enddo
      call FindSecond(x,m,s,d,n,p)
      return
      end
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine FindMedian(x,m,d,n,p)
      integer    m(6000),x(6000),p,n
      real       d(6000,6000)
*     ---------------------------------------- Find the median for users
      do i=1,n
         xi=x(i)
         dmin=1.e20
         do j=1,p
            xj=x(j)
            if(d(xi,xj).lt.dmin)then
               dmin=d(xi,xj)
               m(xi)=xj
            endif
         enddo
      enddo
      return
      end
*
```

```
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine FindSecond(x,m,s,d,n,p)
      integer    m(6000),s(6000),x(6000),p,n
      real       d(6000,6000)
*   ------------------------------------ Find second medians of users
      do i=1,n
         xi=x(i)
         dmin=1.e20
         do j=1,p
            xj=x(j)
            if(xj.ne.m(xi))then
               if(d(xi,xj).lt.dmin)then
                  dmin=d(xi,xj)
                  s(xi)=xj
               endif
            endif
         enddo
      enddo
      return
      end
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine MedianOut(dif,x,m,s,d,n,p,xin,jout)
      integer      m(6000),s(6000),x(6000),p,n
      real         d(6000,6000),v(6000)
*   ---------------------------------- Find the best going-out median
      dif=0.
      do j=1,p
         v(x(j))=0.
*    ------------------------------- v(x) the difference if x goes out
      enddo
      do i=1,n
         xi=x(i)
         if(d(xi,xin).lt.d(xi,m(xi)))then
            dif=dif+d(xi,xin)-d(xi,m(xi))
         else
            mxi=m(xi)
            if(d(xi,xin).lt.d(xi,s(xi))then
               v(mxi)=v(mxi)+d(xi,xin)-d(xi,mxi)
            else
               v(mxi)=v(mxi)+d(xi,s(xi))-d(xi,mxi)
            endif
         endif
```

```
      enddo
      difmin=1.e20
      do j=1,p
         xj=x(j)
         if(v(xj).lt.difmin)then
            difmin=v(xj)
            jout=j
         endif
      enddo
      dif=dif+difmin
      return
      end
*
*  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine Update(x,m,s,d,n,p,xin,jout)
      real         d(6000,6000)
      integer      m(6000),s(6000),x(6000),p,n
* ---------------------------------- Update medians and second medians
      xout=x(jout)
      do i=1,n
         xi=x(i)
         if(m(xi).eq.xout)then
*          ---------------------------------------- x(i) loses its median
            if(d(xi,xin).lt.d(xi,s(xi)))then
*           ------------------------------- x_in will be the its median
               m(xi)=xin
            else
*           ---------------------- the second median will be its median
               m(xi)=s(xi)
*              --------------------------------- find the second median
               s(xi)=xin
               dmin=d(x(i),xin)
               do j=1,p
                  xj=x(j)
                  if(xj.ne.m(xi))then
                     if(xj.ne.xout)then
                        if(d(xj,xi).lt.dmin)then
                           dmin=d(xj,xi)
                           s(xi)=xj
                        endif
                     endif
                  endif
               enddo
```

```
            endif
         else
*        ------------------------------- x(i) does NOT lose its median
            if(d(xi,xin).lt.d(xi,m(xi)))then
*           ---------------------------------- x_in will be the median
               s(xi)=m(xi)
               m(xi)=xin
            else
               if(d(xi,xin).lt.d(xi,s(xi)))then
*              ------------------------ x_in will be the second median
                  s(xi)=xin
               else
*              -------------------------- x(i) loses its second median
                  if(s(xi).eq.xout)then
                     s(xi)=xin
                     dmin=d(x(i),xin)
                     do j=1,p
                        xj=x(j)
                        if(xj.ne.m(xi))then
                           if(xj.ne.xout)then
                              if(d(xj,xi).lt.dmin)then
                                 dmin=d(xj,xi)
                                 s(xi)=xj
                              endif
                           endif
                        endif
                     enddo
                  endif
               endif
            endif
         endif
      enddo
      return
      end
*
* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine Shake(f,x,m,s,d,n,p,k)
      real       d(6000,6000)
      integer    m(6000),s(6000),x(6000),p,n
*     ------------------------------------
      fold=f
      do k=1,k
         goin=p+1+(n-p)*GGUBFS(seed)
```

```
*        ------------------------------------ a random vertex goes in
         xin=x(goin)
         call MedianOut(dif,x,m,s,d,n,p,xin,jout)
*        ----------------------------------- the best median goes out
         call Update(x,m,s,d,n,p,xin,jout)
         x(goin)=x(jout)
         x(jout)=xin
         f=f+dif
         if(f.lt.fold)then
           return
      enddo
      return
      end

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine InterCh(f,x,m,s,d,n,p)
      real         d(6000,6000)
      integer      m(6000),s(6000),xcur(6000),p,n
*     ------------------------ Initialization with the input solution
      fcur=fopt
      ItInter=0
      bestiter=1
*     ---------------------------------------------------- Iterations
      do while(ItInter.le.maxitInt)
         ItInter=ItInter+1
*     ------------------- Get the best solution in the 1-neighbourhood
         difmin=1.e20
         do i=p+1,n
            xin=x(i)
            call MedianOut(dif,x,m,s,xin,jout,d,n,p)
            if(dif.lt.difmin)then
               difmin=dif
               goin=i
               goout=jout
            endif
         enddo
*        ---------------------------------- If no improvement, return
         if(Dif.ge.0)then
            bestiter=IterInter
            return
         endif
*        ---------------------------------------- If improved, update
         f=f+fdif
```

```
         xin=x(goin)
         call Update(x,m,s,xin,goout,d,n,p)
         x(goin)=x(goout)
         x(goout)=xin
      enddo
      bestiter=iterInter
      return
      end
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*     Variable Neighbourhood Search (VNS)
*     -----------------------------------
      subroutine VNS(fopt,xopt,mopt,sopt,d,n,p,time,MaxTimeVNS,se)
      real       d(6000,6000)
      integer    xopt(6000),xcur(6000),p,n
*     ----------------------------------- Initialization of counters
      iterVNS=0
      kfirst=1
      kstep=1
      bestiterVNS=1
      time=0.
*     --------------------------------- Start with the input solution
      fcur=fopt
      do i=1,n
         xi=xopt(i)
         mcur(xi)=mopt(xi)
         scur(xi)=sopt(xi)
         xcur(i)=xopt(i)
      enddo
*     ---------------------------------------------------- Main step
      do while(iterVNS.le.maxitVNS)
         kinter=kfirst
         do while(kinter.le.kmaxVNS)
            iterVNS=iterVNS+1
*           ---------------------------- Shake in the k-th neighbourhood
            call Shake(fcur,xcur,mcur,scur,d,n,p,kinter)
            call InterCh(fcur,xcur,mcur,scur,d,n,p)
            if(fcur.lt.fopt)then
*           ------------------ If improved start with first neighbourhood
               bestiterVNS=iterVNS
               fopt=fcur
               do i=1,n
                  xi=xcur(i)
```

```
                  mopt(xi)=mcur(xi)
                  sopt(xi)=scur(xi)
                  xopt(i)=xcur(i)
               enddo
               kinter=kfirst
            else
*           ------------------- If no improved increase the neighbourhood
               fcur=fopt
               do i=1,n
                  xi=xopt(i)
                  mcur(xi)=mopt(xi)
                  scur(xi)=sopt(xi)
                  xcur(i)=xopt(i)
               enddo
               kinter=kinter+kstep
            endif
* ------------------------------------------------- Stopping Criteria
            if(iterVNS-bestiterVNS.gt.MaxNoImpItVNS)return
            if(time-InitTimeVNS.gt.MaxTimeVNS)return
         enddo
      enddo
      return
      end

*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      subroutine Decompose(fd,xd,md,sd,d,nd,pd,merged
     *                     f, x, m, s, n, p)
      real       d(6000,6000)
      integer    x(6000),xd(6000),m(6000),md(6000),s(6000),sd(6000)
      integer    p,pd,n,nd,merged(6000)
      logical*1  merged(6000)
*     --------------------------- select "pd" indeces between 1 and p
      do i=1,p
         merged(i)=i
      enddo
      do i=1,pd
         j=i+(p-i)*GGUBFS(seed)
         xx=merged(i)
         merged(i)=merged(j)
         merged(j)=xx
      enddo
      do i=1,p
         merging(i)=.false.
```

```
      enddo
      do i=1,pd
         merging(merged(i))=.true.
      enddo
*     ---------------------- merging the "pd" clusters with true value
      k=0
      fd=0.
      do i=1,n
         xi=x(i)
         if(merging(m(xi))) then
            k=k+1
            fd=fd+d(m(xi),xi)
            xd(k)=xi
            md(xi)=m(xi)
            sd(xi)=s(xi)
         endif
      enddo
      nd=k
      call FindSecond(xd,md,sd,d,nd,pd)
      return
      end
*
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    subroutine VNDS(f,x,m,s,d,n,p,se,MaxTimeVNDS,MaxTimeVNS,TotalTime)
*  ------------------------------------------------------------------
      integer     m(6000),s(6000),x(6000),pos(6000),p,n
      real        d(6000,6000)
*     ----------------------- Initial solution obtained by INTERCHANGE
*                           (only to find total CPU time 'MaxTimeVNS')
      call INSOL(f,x,m,s,d,n,p)
      t0=time
      call InterCh(f,x,m,s,d,n,p)
      t0=time-t0
      MaxTimeVNDS=t0
      t0=t0/1.5
      MaxTimeVNS=t0
*   --------------------- Initial solution obtained by Monte-Carlo VNS
      call INSOL(f,x,m,s,d,n,p)
      call VNS(f,x,m,s,d,n,p,MaxTimeVNS)
*     ---------------------------------------- Save initial solution
      fcur=f
      fopt=f
      do i=1,n
```

```
         xi=x(i)
         mopt(xi)=m(xi)
         sopt(xi)=s(xi)
         xopt(i)=x(i)
      enddo
*     ------------------------------------ Constants and paremeters
      iterVNDS = 0
      kstep = 1
      kfirst = 1
*     ------------------------------------------------- Main loop
      do while(iterVNDS.le.maxitVNDS)
         kdec=kfirst
*        --------------------------- Merging and solving subproblems
         do while(kdec.lt.kmaxVNDS)
            iterVNDS=iterVNDS+1
            TimeInitVNDS=time
            pd=kdec
            call Decompose(fd,xd,md,sd,d,pd,nd,merged,x,m,s,f,n,p)
            fold=fd
            do i=1,n
               pos(x(i))=i
            enddo
            call VNS(fd,xd,md,sd,d,nd,kdec,t0)
            fcur=fopt+fd-fold
            do j=1,pd
               call Update(x,m,s,xd(j),merged(j),d,n,p)
               x(pos(xd(j))=x(merged(j))
               x(merged(j))=xd(j)
            enddo
            if(fd.lt.fold)then
               kdec=kfirst
               fopt=f
               do i=1,n
                  xi=x(i)
                  mopt(xi)=m(xi)
                  sopt(xi)=s(xi)
                  xopt(i)=x(i)
               enddo
            else
               kdec=kdec+kstep
               f=fopt
               do i=1,n
                  xi=x(i)
```

```
                  m(xi)=mopt(xi)
                  s(xi)=sopt(xi)
                  x(i)=xopt(i)
               enddo
            endif
            if(time.gt.,MaxTimeVNDS)then
               fopt=0.
               do j=p+1,n
                  fopt=fopt+d(x(j),m(x(j)))
               enddo
               return
            endif
         enddo
      enddo
      f=fopt
      return
      end
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*                  IMSL random number generator
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      real function GGUBFS(dseed)
      real*8 dseed,d2p31m,d2p31
      data d2p31m/2147483647.d0/,d2p31/2147483648.d0/
      dseed=dmod(16807.d0*dseed,d2p31m)
      ggubfs=dseed/d2p31
      return
      end
*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*    Input disimilarities
*
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Apéndice C

LISTA DE SÍMBOLOS.

A continuación se encuentra una lista de los principales símbolos que aparecen en el texto. En la primera de las columnas se muestra el símbolo y en la segunda una pequeña descripción del mismo.

G	grafo o red general
T	árbol o red árbol
V	conjunto de vértices de G
U	subconjunto de vértices de V
E	conjunto de arista de G
m	número de aristas de G
n	número de vértices de V
p	número de servicios
X	conjunto solución
f	función objetivo
f_c	función Centro
f_m	función Mediana
f_λ	función Centdian

f_T	función objetivo en el árbol T
$P(G)$	conjunto continuo de puntos de G
$P(T)$	conjunto continuo de puntos de T
$d(x,y)$	distancia entre el punto x e y
$d(x,Y)$	distancia entre el punto x y el conjunto Y
$B_a(r,v)$	conjunto de puntos cuello de botella de rango r asociado al vértice v
$E_p(r,v)$	conjunto de puntos extremos de rango r asociado al vértice v
$EP(r)$	conjunto de puntos extremos de rango r asociado a todos los vértices de U
EP	conjunto de puntos extremos de rango todos los r de R asociado a todos los vértices de U
$B_c(r:u,v)$	conjunto de puntos centro local con rango r asociado a los vértices u y v.
$LC(r)$	conjunto de puntos centro local con rango r asociado a todos los vértices u y v de U
LC	conjunto de puntos centro local de rango todos los r de R asociado a todos los vértices u y v de U
Y_j	conjunto de semivértices del camino que conecta al vértice $v_j \in U$ al vértice raíz v_1
V_j	conjunto de semivértices descendiente del vértice v_j
T_j	subárbol inducido por V_j, v_j será su raíz.
$Y_i(r_{ij})$	conjunto de semivértices que están a distancia r_{ij} del vértice v_i.
${r_j}^i$	distancia entre el vértice v_j al semivértice $y_k \in Y_i(r_{ij}), i = 1..m$
L_j	conjunto de distancias desde el vértice v_j a todos los semivértice de Y_j
$N(X)$	entorno de la solución X

N_j	conjunto de vértices que permanecen conectados al vértice v_j si eliminamos del árbol las aristas $[v_{j-1}, v_j]$, $[v_j, v_{j+1}]$
D_m	pendiente de la función Mediana.
D_c	pendiente de la función Centro.
D_λ	pendiente de la función Centdian.

Referencias

[1] **A.V.Aho, J.E.Hopcroft, J.D.Ullman**. *Estructura de Datos y Algoritmos*, Addison-Wesley Iberoamericana.

[2] **Y.P.Aneja, R.Chandrasekaran and K.P.K.Nair**. A Note on the M-Center problem with Rectilinear Distances, *European Journal Of Operational Research* 35 (1988) 118-123.

[3] **E.H.L. Aarts, P.J.M. van Laarhoven**. Simulated Annealing: An Introduction, *Statica Neerlandica* 43 (1988).

[4] **R.S. Anderssen**. Global Optimization. En R.S. Anderssen, L.S. Jennings, D.M.Ryan, editores, *Optimization* (1972), University of Queensland Press.

[5] **I. Averbakh and O. Berman.** Parallel NC-algorithms for multi-facility location problems on a tree and their applications, *Working Paper, University of Toronto, Faculty of Management*, (1993).

[6] **P.Avella, L.Canovas, k.Dalby, D.Girolamo, B.Dimitrijevic, G. Ghiani, I. Giannikos, N.Guttmann, T. Hultberg, J. Fliege, A. Marin, M.Munos, M.Ndiaye, S.Nickel, P.Peeters, D.Pérez-Brito, S.Policastro, F.Saldanha and P.Zidda.** Some Personal Views on the Current State and Future of Location Analysis, *European Journal of Operation Research* 104-2 (1998) 269-288.

[7] **J.E.Beasley**. A note on solving large p-median problems, *European Journal of Operation Research* 21 (1985) 270-273.

[8] **N.Biggs, E.Lloyd and R.Wilson**. *Graph Theory*, Clarendon Press, Oxford, (1976).

[9] **M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E.Tarjan**. Time bounds for selection, *J. Comp. Sys. Sci.* 7 (1972) 448-461.

[10] **C.G.E.Boender, A.H.G.Rinnooy Kay**. Bayesian Stopping Rules for Multistart Global Optimization. *Mathematical Programming* 37 (1987) 59-80.

[11] **D.K.Boese, B.A.Kahng, S.Muddu**. A New Adaptive Multi-Start technique for Combinatorial Global Optimization. *Operational Research Letters* 16 (1994) 101-113.

[12] **M.L.Brandeau and S.S.Chiu**. An overview of representative problems in location research, *Management Science* 35 (1989) 645-674.

[13] **B.Bruno, F.Choen**. Sequential Stopping Rules for the Multistart Algorithm in Global Optimization. *Mathematical Programming* 38 (1987) 271-186.

[14] **A.Cayley**. Collected papers, *Quart.JL. of Mathematics* 13, Cambridge (1897) 26.

[15] **E.J.Carrizosa, E.Conde, F.R.Fernandez and J.Puerto**. An axiomatic approach to the Centdian criterion, *Location Science* 3, (1994), 1-7.

[16] **M. Chams, A.Herts,D.De Werra**. Some Experiments with Simulated Annealing for Colouring Graphs, *European Journal of Operational Research* 32 (1987) 260-266.

[17] **R.Cole**. Slowing down sorting networks to obtain faster sorting algorithms, *Journal ACM* 34 (1987) 200-208.

[18] **R.Cole, J.Salowe, W.Steiger and E.Szemeredi**. Optimal slope selection, *SIAM Journal On Computing* 18 (1989) 792-810.

[19] **G.Cornuejols, M.L.Fisher, and G.L.Nemhauser**. Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximation Algorithms, *Managements science* 23 (1977) 789-810.

[20] **P.H.Chen, P.Hansen, B.Jaumard, H.Tuy**. Solution of the multisource Weber and conditional Weber problems by d.-c. programming. *GERAD report, G-92-35, Montreal, Canada.*

[21] **L.Cooper**. Heuristic methods for location-allocation problems, *SIAM Review 6* (1964) 37-53.

[22] **L.Cooper**. The transportation-location problem. *Operation Research* 20 (1972) 94-108.

[23] **C.Charalambous**. Extension of the Elzinga-Hearn Algorithm to the Weighted Case. *Operation Research* 30 (1982) 591-594.

[24] **I.Charon and O.Hudry**. The Noising Method: A New Method for Combinatorial Optimization. *Operation Research Letters* 14 (1993) 133-137.

[25] **R. Chandrasekaran and A. Tamir**, An $O((n \log p)^2)$ algorithm for the continuous p-Center problem on a tree, *SIAM Journal On Algebraic and Discrete Methods* 1 (1980) 370-375.

[26] **R. Chandrasekaran and A. Tamir**, Polynomially bounded algorithms for locating p-centers on a tree, *Mathematical Programming* 22 (1982) 304-315.

[27] **W.K.Chen**. *Applied Graph Theory*, North-Holland, Amsterdam. (1971).

[28] **N.Christofides and P.Viola**. The Optimum Location of Multicenters on a Graph, *Operational Research Quarterly* 145 (1971) 145.

[29] **N.Christofides**. *Graph Theory: An Algorithms Approach*, Academic Press, 1975.

[30] **N.Christofides and E.Beasley**. A tree Search Algorithm for the p-Median problem. *European Journal of Operational Research* 14 (1982) 196-204.

[31] **G.B. Dantzig**, *Linear Programming and Extensions*, Princeton University Press, Princeton, N.J. (1963).

[32] **P.M.Dearing and R.L.Francis**. A Network Flow Solution to a Multifacility Minimax Location Problem Involving Rectilinear Distances, *Transportation Science* 8 (1974) 126-141.

[33] **P.M.Dearing, R.L. Francis and T.J.Lowe**, Convex Location Problem on Tree Networks, *Operation Research* 24 (1976) 628-642.

[34] **E.W.Dijkstra**. A note on Two Problems in Connexion with Graphs, *Numer. Math* 1 (1959) 269-271.

[35] **Z.Drezner**. The p-Center Problem-Heuristic and Optimal Algorithms, *Journal of the Operational Research Society* 35 (1984) 741-748.

[36] **Z.Drezner**. On the Rectangular p-Center Problem. *Naval Research Logistics* 34 (1987) 229-234.

[37] **Z.Drezner**. *Facility Location*, Springer Series in Operation Research. (1995).

[38] **H.A.Eiselt and G.Charlesworth**. A note on p-Centre Problems in the Plane, *Transportation Science* 20 (1986) 130-133.

[39] **S.Eilon,C.D.T.Watson Gandy and N.Christofides**. Distribution Management. New York: Hafner.

[40] **A.M.El-Shaieb**. A New Algorithm for Locating Sources Among Destinations, *Management Science* 20 (1973) 221-231.

[41] **D.J.Elzinga and D.W.Hearn**. The Minimum Covering Sphere Problem, *Management Science* 19 (1972) 96-104.

[42] **D.Eppstein**, Sequence comparison with mixed convex and concave costs, *Journal Algorithms* 11 (1990) 85-101.

[43] **E.Erkut, R.L.Francis, T.J.Lowe and A.Tamir.** Equivalent Mathematical Programming Formulations of Monotonic Tree Network Location Problems, *Operation Research* 37 (3) (1989) 447-461.

[44] **E. Erkut, R.L. Francis and A. Tamir.** Distance constraints multifacility minimax location problems on tree networks, *Networks* 22 (1992).

[45] **D. Erlenkotter**. A Dual-Based Procedure for Uncapacitated Facility Location, *Operation Research* 26 (1978) 992-1009.

[46] **F.A.Feldman, F.A.Leher and T.L.Ray**. Warehouse locations under continous economies of scale, *Management Science* 12 (1966) 670-684.

[47] **G.N. Frederickson.** Optimal algorithms for partitioning trees and locating p-centers in trees, *Technical Report, Department of Computer Science, Purdue University*, (1990).

[48] **R.L.Francis and J.A.White**. *Facility Layout and Location: an Analytical Approach*, Prentice Hall. (1992).

[49] **R.L. Francis, T.J. Lowe and H.D. Ratliff.** Distance constraints for tree network location problems, *Operations Research* 26 (1978) 570-596.

[50] **R.L. Francis and P.B.Mirchandani.** *Discrete Location Theory*, John Wiley and Sons, New York. (1989).

[51] **G.N. Frederickson.** Optimal algorithms for partitioning trees and locating p-centers in trees, *Technical Report, Department of Computer Science, Purdue University*, (1990).

[52] **R.D.Galvao**. A Dual-Bounded Algorithm for the p-Median Problem, *Operations Research* 28 (1980) 1112-1121.

[53] **R.S.Garfinkel, A.W.Neebe and M.R.Rao**. An algorithm for the M-Median Plant Location Problem. *Transportation Science* 8 (1974) 217-236.

[54] **M.Garey and D.Johnson.** *Computer and Intractibility: a Guide to the Theory of NP-completeness*, Freeman, San Francisco, (1979).

[55] **G.V.Gens and E.V.Levner**. *Computational complexity of approximation algorithms for combinatorial problems*, Lecture Notes in Computer Science 74, Spring Verlag, Berlin (1979) 192-300.

[56] **F.Glover**. Tabu Search. Part I. *ORSA Journal Computing* 1 (1989) 190-206.

[57] **F.Glover and M.Laguna**. *Tabu Search.* En C.Reeves, editor, Modern Heuristic Techniques for Combinatorial Problems. Oxford, (1993), Blackwell.

[58] **A.J. Goldman.** Optimal center location in simple networks, *Transportation Science* 5 (1971), 212-221.

[59] **A.J. Goldman and P.M.Dearing**. Concepts of Optimal Location for Partially Noxious Facilities, *ORSA Bulletin*, 23, 1, B-31.

[60] **D. Gusfield and E. Tardos**. A faster parametric minimum cut problem, *Technical Report 926, School of Operations Research and Industrial Engineering, Cornell University, Itacha* (1990).

[61] **D. Gusfield and C. Martel.** A fast algorithm for the generalized parametric minimum cut problem and applications, *Algorithmica* 7 (1992) 499-519.

[62] **S.L.Hakimi**. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph. *Operation Research.* 12, (1964), 450-459.

[63] **S.L.Hakimi**. Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph Theoretic Problems, *Operations Research* 13 (1965) 462-475.

[64] **S.L.Hakimi, E.Shemeichel and M.Labbe.** On locating path-or tree shaped facilities on networks, *Networks* 23 (1993) 543-556.

[65] **J.Halpern.** The location of a cent-dian convex combination on an undirected tree, *Journal of Regional Science.* 16, (1976) 237-245.

[66] **J.Halpern**. Finding Minimal Center-Median Convex Combination Cent-Dian of a Graph, *Management Science* 24 (1978) 535-544.

[67] **J.Halpern**. Duality in the Cent-Diant of a Graph, *Operations Research* 28 (1980) 722-735.

[68] **G.Y. Handler**. Minimax location of a facility in an undirected tree graph, *Transportation Science* 7,(1973), 287-293.

[69] **G.Y.Handler.** Minimax network location theory and algorithms, *Technical Rep.No.107, Oper. Res. Center, Mass. Inst. of Tech., Cambridge, Mass.*, (1974).

[70] **G.Y. Handler**. Finding two-centers of a tree: The continuous case, *Transportation Science* 12 (1978) 93-106.

[71] **G.Y.Handler and P.B.Mirchandani**. Location on Networks: Theory and Algorithms, *the MIT Press, Cambridge, MA.* (1979).

[72] **G.Y.Handler**. Medi-centers of a tree, *Transportation Science* 19, (1985), 246-260.

[73] **P.Hanjoul and D.Peeters**. Comparison of two dual-based procedures for solving the p-median problem, *European Journal of Operational Research* 20 (1985) 387-396.

[74] **P.Hansen, M.Labbe and J.F.Thisse**. From the median to the generalized center, R*AIRO Recherche Operationnelle* 25, 1.(1991),

[75] **P.Hansen, B.Jaumard and S.Krau**. A column generation algorithm for the multisource Weber problem, *GERAD research report* (1996) (forthcoming).

[76] **P.Hansen, D.Peteers, D.Richard and J.F.Thise**. The Minisum and Minimax Location Problem Revisited, *Operation Research* 33 (1985)

[77] **P.Hansen, N.Mladenovic**. Variable Neighbourhood Search for the p-Median, *Les Cahiers du GERAD*, Montreal, Canada, (1996) (forthcoming).

[78] **P.Hansen, N.Mladenovic and D.Pérez-Brito**. Variable Neighbourhood Decomposition Search, *Les Cahiers du GERAD*, Montreal, Canada, (1997) (forthcoming).

[79] **D.Harvey.** *Social Justice and the City.* Baltimore, M.D.John Hopkins University Press. (1973).

[80] **R.Hassin and A.Tamir**, Improved complexity bounds for location problems on the real line, *Operations Research Letters* 10 (1991) 395-402.

[81] **S.V.Hoesel and A.Wagelmans.** On the p-coverage problem on the real line, *Working Paper OR 255-91, Operations Research Center, MIT,* (1991).

[82] **J.N.Hooker, R.S.Garfinkel and S.K.Chen**. Finite dominating sets for network location problems.*Operations Research*, 39, (1991), 100-118.

[83] **W.L.Hsu and G.L.Nemhauser**. Easy and Hard Bottleneck Location Problems, *Discrete Appl. Math.* 2, (1979) 77-91.

[84] **W.L.Hsu**. The distance-domination numbers of trees, *Operation Research Letter* 1, (1982), 96-100.

[85] **B.L.Hulme and P.J.Slater**. Minimean location of different facilities on a line network, *SIAM J Alg. Disc. Methods* 2 (1981) 411-415.

[86] **T.Ibaraki and N.Katoh.** *Resource Allocation Problems. Algorithmic Approaches*, MIT Press, Cambridge, MA, (1988).

[87] **P.Jarvinen, I.Rajala and H.Sinervo**. A Branch and Bound Algorithm for Seeking the p-Median, *Operations Research* 20 (1972) 173-178.

[88] **C.Jordan**. Sur Les Assamblages Des Lignes. *Zeitschrift Fur Die Reine Und Angewandte Mathematik* 70 (1869) 185-190.

[89] **O.Kariv and S.L.Hakimi**. An algorithmic Approach to Network Location Problems. I: The p-Centers. *SIAM J. Appl. Math* 37, 3 (1979), 513-538.

[90] **O.Kariv and S.L.Hakimi**. An algorithmic Approach to Network Location Problems. II: The p-Medians. *SIAM J. Appl. Math* 37, 3 (1979), 539-560.

[91] **M.J.Katz and M.Sharir**. Optimal slope selection via expanders, *Technical Report* (1993).

[92] **B.M.Khumawala**. An Efficient Branch and Bound Algorithm for the Warehouse Location Problem, *Management Science* 18 (1972) B718-B731.

[93] **R.E.Kuenne and R.M.Soland**. Exact and approximate solution to the multisource Weber problem, *Mathematical Programming* 3 (1972) 193-209.

[94] **T.U.Kim, T.J.Lowe, A.Tamir and J.E.Ward.** On the location of a tree-shaped facility, *Networks* 28 (1996), 167-175.

[95] **M.M. Klawe and D.J. Kleitman**. An almost linear time algorithm for generalized matrix searching, *SIAM J Disc. Math.* 3 (1990) 81-97.

[96] **M.T. Ko and Y.T. Ching.** Linear time algorithms for the weighted tailored 2-partition problem and the weighted 2-center problem under l_∞, Discrete Applied Mathematics 40 (1992) 397-410.

[97] **A.J.W. Kolen**. *Location problems on trees and in the rectilinear plane*, Stichting Mathematics Centrum, Amsterdam, The Netherlands, (1982).

[98] **A.Kolen and A.Tamir**. *Covering Problems, in Discrete Location Theory*, P.B. Mirchandani and R.L. Francis (eds.) John Wiley (1990).

[99] **J.Krarup, and P.M.Pruzan**. The Simplex Plant Location Problem: Survey and Synthesis, *European Journal of Operational Research* 12 (1983) 36-81.

[100] **A.A.Kuehn and M.J.Hamburger**. A Heuristic Program for Locating Wareshouse, *Management Science* 18 (1963) 643-666.

[101] **M.Kurt** *Graph Algorithms and NP-Completeness*, Springer Verlag, (1984).

[102] **P.J.M.Laarhoven and E.H.L.Aarts**. *Simulated Annealing: Theory and Applications*, Kluver, Dordcht, (1988).

[103] **E.Lawler.** Fast approximation algorithms for knapsack problems, *Mathematics of Operations Research* 4 (1979) 339-356.

[104] **S.Lin**. Computer Solutions of the Traveling Salesman Problem, *Bell System Tech. J.*, (1980) 2245-2269.

[105] **R.F.Love, J.G.Morris and G.O.Wesolowsky**. *Facilities Location: Models and Methods*, North Holland, NY.

[106] **R.F.Love.** One-dimensional facility location-allocation using dynamic programming, *Management Science* 22 (1976) 614-617.

[107] **R.F.Love and J.G.Morris**. A computational procedure for the exact solution of location-allocation problems with rectangular distances, *Naval Research Logistics Quarterly* 22 (1975) 441-453.

[108] **T.J.Lowe**. Efficient Solutions in Multiobjective Tree Network Location Problems, *Transportation Science* 12, 4 (1978) 298-316.

[109] **T.L.Magnanti and R.T.Wong**. Decomposition methods for facility location problem, *in: P.B.Mirchandani and R.L.Francis (eds.), Discrete Location Theory (Wiley, New York)* 209-262.

[110] **F.E.Maranzana.** On the Location of Supply Points to Minimize Transport Costs, *Operational Research Quartely* 15 (1964) 261-270.

[111] **W.Mayeda**. *Graph Theory*, Wiley-Interscience, New York. (1972).

[112] **N.Megiddo.** Linear-time algorithms for linear programming in R^3 and related problems, *SIAM Journal on Computing* 12, (1983), 751-758.

[113] **N.Megiddo.** The Weighted Euclidean 1-Center Problem. *Mathematics of Operation Research* 8 (1983) 498-504.

[114] **N.Megiddo**. Combinatorial optimization with rational objective function, *Mathematics of Operations Research* 4 (1979) 414-424.

[115] **N.Megiddo A. Tamir, E. Zemel and R.Chandrasekaran**, An $O(n \log^{2n})$ algorithm for the k-th longest path in a tree with applications to location problems, *SIAM Journual on Computing* 10 (1981) 328-337.

[116] **N.Megiddo.** Linear time algorithms for linear programming in R^3 and related problems, *SIAM Journal on Computing* 12 (1983) 759-776.

[117] **N.Megiddo.** Applying parallel algorithms in the deign of serial algorithms, *Journal ACM* 30 (1983) 852-865.

[118] **N.Megiddo and A.Tamir.** New results on p-center problems, *SIAM Journal on Computing* 12 (1983) 751-758.

[119] **N.Megiddo and A.Tamir**. Linear time algorithms for some separable quadratic programming problems, *Operation Research Letters* 13 (1993) 203-211.

[120] **E.Minieka**. The M-Center Problem, *SIAM Review* 12 (1970) 138-139.

[121] **E.Minieka**. The Centers and the Medians of a Graph, *Operation Research* 25 (1977) 641-650.

[122] **E.Minieka**. Conditional Centers and Medians of a Graph, *Networks* 10 (1980) 265-272.

[123] **E.Minieka.** The Optimal Location of a Path or Tree in a Tree Network, *Networks* 15 (1985) 309-321.

[124] **P.B.Mirchandani and R.L.Francis**. *Discrete Location Theory*, Wiley, New York. (1990).

[125] **N.Mladenovic**. Variable Neighbourhood Algorithm A New Metaheuristic For Combinatorial Optimization. *Abstracts of papers presented at Optimization Days. Montreal*, p.112 (1995).

[126] **N.Mladenovic, J.A.Moreno and J. Moreno-Vega**. A Chain-Interchange Heuristic Method, *Yogoslav Journal Operation Research* 6 (1) (1996) 41-54.

[127] **J.M.Moreno**. Metaheurísticas en Localización: Analisis Teórico y Experimental, *Tesis Doctoral. Univ. La Laguna.* (1996).

[128] **J.A.Moreno**. A Correction to the Definition of Local Center, *European Journal of Operation Research* 20 (1985) 382-386.

[129] **J.A.Moreno**. Localización Minimax en Grafos Mixtos, *Tesis Doctoral. Univ. Complutense Madrid.* (1986).

[130] **J.A.Moreno, C.Rodriguez and N.Jimenez**. Heuristic Cluster Algorithm for the Multiple Facility Location-Allocation Problem. *Operations Research* 25 (1) (1991) 97-107.

[131] **J.A.Moreno, J.L.Roda, J.M.Moreno**. A Parallel Genetic Algorithm for the Discrete p-Median Problem, *Studies in Locational Analysis* 7 (1995) 131-141.

[132] **S.C.Narula, U.I.Ogbu and H.M.Samuelson**. An Algorithm for the p-Median Problem. *Operations Research* 25 (1977) 709-712.

[133] **B. Olstad and F. Manne.** Efficient partitioning of sequences, *Technical Report, Norwegian Institute of Technology, Trondheim-NTH, Norway,* (1993).

[134] **B.Pelegrin**. The p-Center Problem under Bidirectional Polyhedral Norms. *Proceeding Meeting III of the EuroWorking Group on Locational Analysis*, 151-169. Sevilla.

[135] **J.C.Picard and H.D.Ratliff.** A cut approach to the rectilinear distance facility location problem, *Operation Research* 28 (1978) 422-433.

[136] **F.Plastria**. A Note on Fixed Point Optimality Criteria for the Location Problem with Arbitrary Norms, *Journal of the Operational Research Society* 34 (1983) 164-165.

[137] **D. Pérez-Brito, J.A. Moreno-Pérez and J.M. Moreno-Vega**. Soluciones heurísticas de problemas de localización-asignación múltiple, *I Congreso Iberoamericano y XX Reunión Nacional de Estadística e Investigación Operativa*, Cáceres, (1992).

[138] **D. Pérez-Brito, J.A. Moreno-Pérez and J.M. Moreno-Vega**. Heurística Basada en árboles para problema de localización, *XXI Congreso Nacional de Estadística e Investigación Operativa*, Calella, (1994).

[139] **D. Pérez-Brito, N. Mladenovic y J.A. Moreno-Pérez.** Problemas De Localización Por Árboles Generadores, *Acta del Seminario sobre Localización*, Sevilla, (1994).

[140] **D. Pérez-Brito, J.A. Moreno-Pérez and I.Rodriguez-Martín.** Problemas Clásicos de Localización, *25 aniversario de Matemáticas en la Universidad de La Laguna*, (1996). 463-474.

[141] **D. Pérez-Brito.** Spanning Trees For Solving Location Problems On General Graphs, *Twelfth EURO Summer Institute-ESI XII on Locational Analysis*, Tenerife, (1995).

[142] **D. Pérez-Brito, J.A. Moreno-Pérez.** Forest For Solving The p-Median Problem In General Graphs, *8th Meeting of the EURO Working Group On Locational Analysis.* Lambrecht, Germany, (1995).

[143] **D. Pérez-Brito, J.A. Moreno-Pérez and I. Rodriguez-Martín.** The 2- facility centdian network problem, *Location Science*, (1996) (To appear).

[144] **D. Pérez-Brito, J.A. Moreno-Pérez and I.Rodriguez-Martín**, The finite dominating set for the p-facility centdian network location problem, *Studies in Location Analysis*, ISSUE 11, (1997) 27-40.

[145] **D. Pérez-Brito, N.Mladenovic and J.A. Moreno-Pérez**, A Note on Spanning Trees for Network Location Problems, *Yugoslav Journal Operation Research*, (1997) (To appear).

[146] **D. Pérez-Brito and J.A. Moreno-Pérez**, The Voronoi Sets And the Multi-facility Max-Sum Location on Networks, *Studies in Location Analysis*, (1997) (To appear).

[147] **F.P. Preparata and M.I. Shamos**. *Computational Geometry*, Springer- Verlag, New York (1985).

[148] **R.C.Prim**. Shortest Connection Networks and Some Generalizations, *Bell System Technical Journal.* 36 (1957) 1389-1401.

[149] **R.Rabinovitch and A.Tamir.** On a Tree-Shaped Facility Location Problem of Minieka, *Networks* 22 (1992) 515-522.

[150] **Reinelt**. TSLIB - A traveling salesman problem library, *ORSA Journal on Computation* 3 (1991) 376-384.

[151] **C.ReVelle and R.W.Swain.** Central Facilities Location, *Geographical Analysis* 2 (1979) 30-42.

[152] **C.ReVelle, C.Toregas and L.Falkson**. Applications of the Location Set Covering Problem, *Geographical Analysis* 8 (1976) 65.

[153] **Rinnoy kan and G.T.Timmer**. Stochastic Global Optimization, *American Journal of Mathematical and Management Sciences* 4 (1984) 7-40.

[154] **R.T. Rockafellar.** *Network Flows and Monotropic Optimization*, Wiley, New York, 1984.

[155] **S.Sahni.** General techniques for combinatorial approximations, *Operation Research* 25 (1977) 920-936.

[156] **M. Sharir and E. Welzl.** Rectlinear and polygonal p-piercing and p-center problems, *Technical Report* (1996). (To be presented in the 12-th ACM Symp. on Computational Geometry 1996.)

[157] **R.Swain**. A Paramatric Decomposition Algorithm for the Solution of Uncapacitated Location Problems, *Management Science* 21 (1974) 189-198.

[158] **J.J.Sylvester**. A Question in the Geometry of Situation, *Quarterly Journal of Pure and Applied Mathematics* 1 (1857) 79.

[159] **A.Tamir.** Obnoxious facility location on graphs, *SIAM Journal on Discrete Mathematics* 4 (1991) 550-567.

[160] **A.Tamir.** The least element property of center location on tree networks with applications to distance and precedence constrained problems, *Mathematical Programming* 62 (1993) 475-496.

[161] **A.Tamir.** Complexity results for the p-median problem with mutual communication, *Operations Research Letters* 14 (1993) 79-84.

[162] **A.Tamir.** A distance constrained p-facility location problem on the real line, *Mathematical Programming*, 66 (1994) 201-204.

[163] **A.Tamir.** An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs, *Operations Research Letters* 19 (1996) 59-64.

[164] **A.Tamir.** Location problems in the Real line, *working paper*, 1997.

[165] **A.Tamir, D.Pérez-Brito and J.A.Moreno**. A Polynomial Algorithm for the p-Centdian Problem on a Tree, *Networks*, (1998) (To appear).

[166] **B.C.Tansel, R.L.Francis and T.J.Lowe**. Location on Networks: A Survey. Part I: The p-Center and p-Median Problems. *Management Science* 29 (4) (1983) 482-497.

[167] **B.C.Tansel, R.L.Francis and T.J.Lowe**. Location on Networks: A Survey. Part II: Exploiting Tree Network Structure. *Management Science* 29 (4) (1983) 498-511.

[168] **M.B.Teitz and P.Bart**. Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph, *Operation Reseach* 16 (1968) 955-961.

[169] **A.Tucker**. *Applied Combinatorics*, John Wiley and sons, (1980).

[170] **R.V.V.Vidal**. Applied Simulated Annealing. *Lecture Notes in Economics and Mathematical Systems* 396 (1993).

[171] **J.Vijay**. An Algorithm for the p-Center Problem in the Plane, *Transportation Science* 19 (1985) 235-245.

[172] **S.VoB**. A Reverse Elimination Approach for the p-Median Problem, *Studies in Location Analysis* 8 (1996) 49-58.

[173] **R.Wendell and P.H.Hunter**, Location Theory, Dominance and Convexity, *Operations Research*, 21 (1973), 314-320.

[174] **G.O.Wesolowsky**. Location Problems on a Sphere, *Regional Science and Urban Economics* 12 (1983) 495-508.

[175] **R. Wilber**, The concave least weight subsequence revisited, *Journal Algorithms* 9 (1988) 418-425.

[176] **R.A.Whitaker**, A fast algorithm for the greedy Interchange for large-scale clustering and median location problems, *INFOR* 21:2 (1983) 95-108.

[177] **E. Zemel**, An O(n) algorithm for the linear multiple choice knapsack problem and related problems, *Information Processing Letters* 18 (1984) 123-128.