Universidad de Valladolid
Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Estadística

ALGORÍTMOS HEURÍSTICOS PARA EL PROBLEMA DEL SET COVERING: MEJORA MEDIANTE ALEATORIZACIÓN

Autora:
Dª. Rosa González Silos

Tutor:
D. Jesús Sáez Aguado

Índice

RESUMEN

Los métodos heurísticos pretenden resolver problemas de optimización proporcionando soluciones factibles que, aunque no optimicen la función objetivo, se acercan mucho al valor óptimo empleando un tiempo más que razonable.

En esta memoria se va a resolver el problema del Set Covering y para conseguirlo usamos algoritmos heurísticos que consisten en la combinación de un método greedy y un algoritmo de mejora, basados en dos ideas principalmente, la diversificación y la intensificación, la primera se lleva a cabo mediante la aleatorización y la segunda mediante la búsqueda local.

Se estudia cómo la introducción de aleatorización, de diferentes formas, en los métodos greedy hace que mejore sustancialmente los resultados y se emplee un menor tiempo que el de los algoritmos exactos.

ABSTRACT

Heuristic methods have been used successfully to solve optimization problems by providing feasible solutions. This type of algorithm not only gives a good approximated solution to the problem but it does it by saving analysis time.

In this project, we attempt to solve the Set Covering problem by using a combination of a greedy method and a search algorithm. We arrived to this idea based on two concepts, diversification and intensification, the first one is performed using randomization and the second one by using a local search.

In this work we studied how the introduction of randomization in greedy methods gives a very good approximated solution and spends less time than using exact algorithms.

INTRODUCCIÓN A LAS METAHEURÍSTICAS

En los problemas de decisión que normalmente se presentan en la vida real, se tiene el objetivo de encontrar la solución que optimiza algún tipo de criterio sujeto a algunas restricciones. Para ello hay que basarse en diferentes herramientas que han sido desarrolladas con el objetivo de ayudar en la resolución de muchos tipos de problemas.

En el caso particular en el que la función objetivo y las restricciones son lineales y las varaibles son continuas, apareció una nueva disciplina, y un algoritmo eficiente, como era él método simplex.

Otro tipo concreto de estos problemas son los de optimización combinatoria, que se caracterizan por tener variables de decisión enteras y porque el espacio de soluciones está formado por subconjuntos de números naturales.

Para resolver este tipo de problemas existe un procedimiento que consiste en explorar todas las soluciones factibles, calcular para cada una el coste asociado, y elegir de entre ellas la que diera el menos coste. Este método se denomina la enumeración total. Sin embargo, este método no es eficiente debido al enorme tiempo empleado en el cálculo.

En los años 70 y 80 se desarrollaron los algoritmos de enumeración parcial, fundamentalmente los algoritmos Brach&Bound y la versión moderna Brach&Cut. Aunque estos métodos han ido ganando en eficiencia computacional, y en muchos casos, resuelven problemas reales de forma exacta, en muchos casos, se requieren grandes tiempos de cálculo.

Ante todas estas dificultades de resoluciones exactas, aparecieron toda una serie de importantes algoritmos, llamados heurísticas, que pretendían resolver estos problemas, proporcionando soluciones factibles que aunque no optimicen la función objetivo se acercan mucho al valor óptimo empleando un tiempo más que razonable.

Estas heurísticas son importantes aparte de por lo flexibles que son y por el poco tiempo de cálculo y memoria requeridos, porque no pretenden encontrar la solución óptima exacta. Son muy útiles cuando se tratan datos poco fiables y son muy usadas como punto de partida de algoritmos exactos de tipo iterativo.

Los principales inconvenientes en el uso de métodos heurísticos son la imposibilidad de conocer la calidad de la solución y no conseguir una solución exacta.

Existen diferentes tipos de heurísticas, las basadas en métodos constructivos (que consisten en añadir componentes individuales a la solución hasta obtener la solución factible, como los algoritmos greedy), las basadas en métodos de descomposición (consiste en dividir el problema en subproblemas de tal manera que la solución del

primero sea la entrada del siguiente), las basadas en métodos de reducción (consiste en identificar una característica que va a tener la solución óptima y así poder fijar el valor de una variable), las basadas en manipulación del modelo (modifican la estructura del modelo para hacer la resolución más sencilla y deduce a partir de la solución de este problema la del problema original), las basadas en métodos de búsqueda por entornos (parten de una solución factible inicial y mediante alteraciones de esta solución van pasando, a otras soluciones factibles mientras no se cumpla el criterio de parada, a otras factibles de su entorno, almacenando finalmente como óptima la mejor de las soluciones encontradas).

Dentro de este último tipo de heurísticas, se ubican las metaheuristicas más conocidas.
Una metaheurística es un método heurístico para resolver un tipo de problema computacional general, normalmente se aplican a problemas que no tienen un algoritmo que obtenga una solución satisfactoria, o que no se puede implementar.

Una clase especial dentro de las heurísticas basadas en métodos de búsqueda por entornos es el método de búsqueda local, o descenso. En este problema, en cada iteración el movimiento se produce desde la solución actual hasta una de su entorno que sea mejor que ella, parando la búsqueda cuando todas las soluciones de su entorno sean peor que ella, es decir, la solución final siempre será un óptimo local. El inconveniente principal de este método es que la solución es un óptimo local, pero no sabemos si absoluto. Para solucionarlo se han planteado dos técnicas, tabu search (mantiene una memoria de la ruta seguida para así poder identificar los óptimos) y simulated annealing (que permite con cierta probabilidad la aceptación de soluciones peores).

Un algoritmo heurístico consiste en la combinación de un método greedy y un algoritmo de búsqueda local a esto se le llama también método heurístico completo. El principal inconveniente es como evitar salir de los óptimos locales, para eso se usan las metaheurísticas, basadas en dos ideas principalmente, la diversificación y la intensificación, la primera se lleva a cabo mediante la aleatorización y la segunda mediante la búsqueda local. Las metaheurísticas más conocidas son el Simulated Annealing (SA), Tabu search (TS), algoritmos genéticos (GA), colonias de hormigas (Ant colony), búsqueda de entorno variable, búsqueda local guiada, GRASP, multi-arranque, redes neuronales.

PARTE 1: ELEMENTOS BÁSICOS EN OPTIMIZACIÓN HEURÍSTICA. USO DE ALEATORIZACIÓN.

1.1 Heurísticas de construcción

Las heurísticas de construcción son las que construyen una solución factible inicial que normalmente se usa para dar la primera solución inicial, tienen que ser sencillas pero con soluciones aceptables.

Los métodos greedy son los representantes más conocidos de este tipo de heurística, su nombre viene del inglés, en castellano, sería voraz, glotón, porque son métodos que toman lo que pueden sin analizar las consecuencias, también se les llama algoritmos miopes, porque no ven más allá, son cortos de vista en el sentido de que no piensan en el futuro, no se detienen a pensar, toman la solución y no vuelven atrás.

Son métodos que se utilizan generalmente para resolver problemas de optimización, incorporan cada vez un elemento a la solución en función de la información disponible en cada momento; una vez elegido el elemento que va a pertenecer a la solución, el método no se replantea sacarlo, ni se hace ningún tipo de reoptimización. Son rápidos y fáciles de implementar pero no siempre garantizan alcanzar la solución óptima.

El funcionamiento es muy sencillo, se define un criterio greedy, normalmente el criterio es una función del costo. El algoritmo tratará de encontrar un subconjunto de candidatos tales que, cumpliendo las restricciones del problema, constituyan la solución final. Para esto, en cada etapa se tomará la decisión que se considera mejor en ese momento, sin considerar las consecuencias futuras, es decir, escogerá entre todos los candidatos el que produce una mejora respecto a la etapa anterior. Antes de añadir un nuevo candidato a la solución que se está construyendo se comprobará si la solución mejora al añadirlo, en caso afirmativo se incluirá en ella y en caso contrario se descartará ese candidato para su inclusión. El siguiente paso es redefinir el criterio y repetirlo hasta obtener la solución completa.

El pseudocódigo, denominando S como la solución factible y C como el listado de candidatos es:

```
S = Ø
while (S no sea una solución y C ≠ Ø) {
        x = selección(C)
        C = C – {x}
        if (S∪{x} es factible)
        S = S∪{x}
}
if (S es una solución
return S;
```

El inconveniente de este método, a parte que la solución no tiene por qué optimizar la función objetivo, sino que es aproximada, es que podemos estar ante un óptimo local y no global como sería el objetivo.

Para cada problema de optimización existe un método greedy, los más conocidos el problema del Set Covering, de cubrimiento máximo, el problema de la p mediana, el problema del viajante y cualquier problema de optimización conbinatorial.

Resumiendo, el método greedy es una herramienta muy elemental usada generalmente para proponer buenas soluciones iniciales y que se puede mejorar después con búsqueda local y con introducción de aleatorización, como veremos después con los métodos GRASP, greedy aleatorizado y meta RaPS.

1.2 Búsqueda local

La búsqueda local consiste en ir pasando de una solución a otra de su entorno siguiendo unas reglas previamente definidas. Se realiza mediante algoritmos de búsqueda local, que se mueven, como se ha dicho antes, de solución a solución en el espacio de soluciones de candidatos, mediante la aplicación de cambios locales, hasta encontrar la solución óptima o hasta que transcurre un tiempo determinado.

Se considera el problema de minimizar la función F(x) en un número finito de puntos X.
Se empieza la búsqueda partiendo de una solución arbitraria $x_1 \in X$ y en cada paso n se elige una nueva solución x_{n+1} dentro del entorno $V(x_n)$ de la actual solución X_n.
Se define un entorno de X como cada $x \in X$ se asocia a un subconjunto $V(x) \subseteq X$. Se supone que una solución no pertenece a su propio entorno, $x \notin V(x)\ \forall\ x \in X$ y que los puntos de su entorno son soluciones obtenidas a partir de x por medio de un movimiento elemental. Se aprecia que la evolución de la solución actual x_n con n= 1,2,… sigue una trayectoria en el espacio de búsqueda de x.
El criterio más común para la selección de la solución siguiente, x_{n+1}, es encontrar el mejor entorno de x_n, es decir, una solución $x_{n+1} \in V(x_n)$ con $F(x_{n+1}) \leq F(x)\ \forall\ x \in V(x_n)$.

Entonces x_{n+1} se convierte en la solución actual, sabiendo que no es peor x_n, de este modo, $F(x_{n+1}) \leq F(x_n)$, de lo contrario, se detiene la búsqueda, pues se ha llegado a un mínimo local. Esta estrategia se conoce como descenso o estrategia del máximo descenso (steepest descent). F_n^* indica el mejor valor de F hasta n y x_n^* es entonces $F(x_n^*)=F_n^*$.

Algoritmo Descenso

- Inicialización: se tiene $x_1 \in X$.
- Con n = 1, 2. . . .; x_n denota la solución actual.
  - Se localiza la mejor $\overline{x}$ en el entorno V (x_n).
  - Si F($\overline{x}$) ≤ F(x_n), entonces $\overline{x}$ se convierte en la actual solución x_{n+1} en n+1 y F_n^* en el mejor valor de F, y x_n^* se actualiza.
  - Sino: parar.

Cuando el entorno V(x) es muy grande, en vez de usar el algoritmo del máximo descenso, se suele usar la estrategia del *first improvement*, que consiste en partir de una solución factible examinar el entorno y seleccionar el primero movimiento que produce una mejora de la solución actual.

Hay que tener en cuenta que la elección de un buen entorno es importante para la eficacia del proceso, de hecho el principal inconveniente de este algoritmo es el no poder saber si se está ante un mínimo local o absoluto como vemos en la figura 1 (fig1).
Todas las soluciones del entorno $V(x_n)$ son mayores que x_n, pero puede existir otro entorno en el que se encuentre una solución menor que $F(x_n)$, y este algoritmo no lo encontraría.

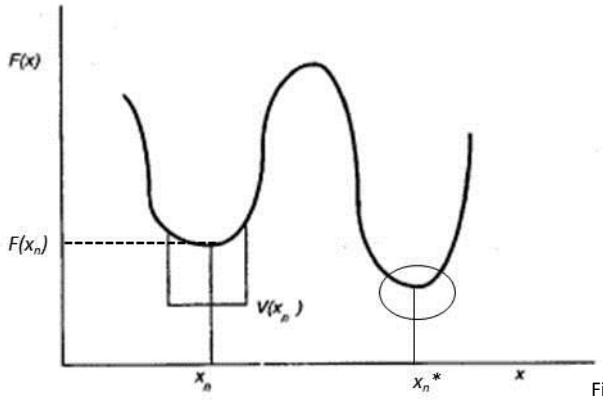

Fig. 1

Prácticamente todos los algoritmos heurísticos y metaheurísticos utilizan búsqueda local. Las metaheurísticas como el Simulated Annealing y el Tabu Search son algoritmos de búsqueda local que para evitar quedar atrapado en un óptimo local permite que la función objetivo también empeore en algún caso.

1.3 GRASP

El nombre son las siglas en inglés de Greedy Randomized Adaptative Search Procedures, procedimiento de búsqueda greedy aleatorizado y adaptativo, es una metaheurística para encontrar soluciones aproximadas cercanas a la óptima en problemas de optimización combinatoria.

Es un procedimiento multi-arranque que es muy sencillo de implementar ya que cada iteración GRASP consiste en una solución greedy aleatorizada (algoritmo de construcción) y una búsqueda local partiendo de la solución obtenida con el método greedy. Se hacen varias iteraciones y de entre todas las soluciones se elige la mejor. En este método cobran gran importancia las soluciones iniciales, cuanto mejores sean, más efectivo es el procedimiento.

Se tiene un problema de optimización combinatoria definido por un conjunto base finito $E = \{1, \dots, n\}$, un conjunto de soluciones factibles $F \subseteq 2^E$ y la función objetivo $f: 2^E \to R$, el método GRASP consiste en buscar una solución óptima $S^* \in F$ tal que $f(S^*) \leq f(S)$, $\forall S \in F$. El conjunto base y las soluciones factibles son propias de cada problema.

El pseudocódigo para este método es:

```
Require: i_max
        f* ← ∞
        for i ≤ i_max do
        x ← GreedyRandomized()
        x ← LocalSearch(x)
        if f(x) < f* then
        f* ← f(x)
        x* ← x
        end if
        end for
return x*
```

La mayoría de los GRASP construyen una solución incorporando un elemento cada vez. En cada paso del proceso de construcción, se obtiene una solución parcial que se elige al azar entre una serie de elementos candidatos. Para determinar qué elemento candidato debe incluirse en la solución generalmente se hace uso del criterio greedy que mide la contribución local de cada elemento a la solución parcial.

Uno de los defectos de este método es la falta de memoria del algoritmo de construcción, es decir, no se tiene en cuenta la solución encontrada en la iteración anterior para la elección de la solución en la iteración actual.

Existen varias maneras para incorporar aleatoriedad, como la creación de una lista restringida de candidatos (RCL), mezclar la construcción al azar y un método greedy y otra manera es mediante la incorporación de una perturbación en los costos.

Como adelantábamos antes, uno de los pasos interesantes a seguir en este método es la creación de una lista restringida de candidatos buena (RCL), para ello se han desarrollado mecanismos que según el caso convendrá usar uno u otro. Hay dos tipos de criterios, los basados en la cardinalidad y en el valor.

Los criterios basados en la cardinalidad son los que forman la lista por un número fijo de elementos, su pseudocódigo es:

Require: k, E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Calcular costo greedy $c(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    RCL $\leftarrow \{k$ elementos $e \in C$ con el menor $c(e)\}$;
    Seleccionar un elemento s de RCL al azar;
    $x \leftarrow x \cup \{s\}$;
    Actualizar el conjunto candidato C;
    Calcular el costo greedy $c(e)$, $\forall\, e \in C$;
  end while

Y los criterios basados en el valor donde los elementos que la forman tienen un valor dentro de un rango dado, por ejemplo, si c* y c_* son los valores mayor y menor respectivamente, y α un valor entre 0 y 1, la RCL basada en el valor consiste en todos los elementos candidatos "e" cuyo valor de la función greedy, c(e), es tal que:

$$c(e) \leq c_* + \alpha\,(c^* - c_*)$$

Se ve fácilmente que si α =0 el esquema de selección es un greedy normal (determinístico), y mientras que si α =1 la selección es completamente aleatoria. El pseudocódigo es:

Require: α, E, $c(\cdot)$;
$x \leftarrow \emptyset$;
$C \leftarrow E$;
Calcular costo greedy $c(e)$, $\forall\, e \in C$;
while $C \neq \emptyset$ **do**
$c^* \leftarrow mín\{c(e) \mid e \in C\}$;
$c^* \leftarrow$ máx $\{c(e) \mid e \in C\}$;
RCL $\leftarrow \{e \in C \mid c(e) \leq c^* + \alpha(\, c^* - c^*)\}$;
Seleccionar un elemento s de RCL al azar;
$x \leftarrow x \cup \{s\}$;
Actualizar conjunto de candidatos C;
Calcular el costo greedy $c(e)$, $\forall\, e \in C$
end while
return x;

Otro enfoque de una RCL basada en el valor es el procedimiento llamado función del sesgo, donde en vez de seleccionar los elementos de la lista con las mismas probabilidades se le asignan diferentes probabilidades a cada elemento, para favorecer la elección de elementos bien evaluados, es decir, que posean mejor valor de la función greedy.

La probabilidad de seleccionar un elemento es $\pi\left(r(e)\right) = \frac{sesgo(r(e))}{\sum_{e' \in RCL} sesgo(r(e'))}$, donde r(e) es la posición del elemento e en la RCL, también se proponen tres alternativas para asignar sesgos a los elementos, sesgo aleatorio (sesgo(r)=1) , sesgo lineal (sesgo(r)=1/r) , y sesgo exponencial (sesgo(r)=exp(-r)).

El pseudocódigo basado en la función de sesgo es:

Require: α, E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$;
  Calcular costo greedy $c(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    $c^* \leftarrow \text{mín}\{c(e) \mid e \in C\}$;
    $c^* \leftarrow \text{máx}\ \{c(e) \mid e \in C\}$;
    $\text{RCL} \leftarrow \{e \in C \mid c(e) \leq c^* + \alpha(\, c^* - c^*)\}$;
    Asignar rango $r(e)$, $\forall\, e \in \text{RCL}$;
    Asignar una probabilidad $\pi(r(e))$ de elegir el elemento e en RCL que favorezca a los candidatos con mejores rangos;
    Elegir el elemento s de RCL al azar con probabilidad $\pi(r(s))$;
    $x \leftarrow x \cup \{s\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo greedy $c(e)$, $\forall\, e \in C$
  end while
return x;

Otra forma de introducir aleatorización es mezclar una construcción al azar con una construcción greedy, eligiendo secuencialmente un conjunto parcial de elementos candidatos al azar, y después completar la solución con un greedy como se ve en el pseudocódigo siguiente:

Require: k, E, $c(\cdot)$;
  $x \leftarrow \emptyset$
  $C \leftarrow E$;
  Calcular el costo greedy $c(e)$, $\forall\, e \in C$;
  for $I = 1,2,\ldots, k$ **do**
    if $C \neq \emptyset$ **then**
      Elegir el elemento e al azar de C;
      $x \leftarrow x \cup \{e\}$;
      Actualizar el conjunto de candidatos C;
      Calcular el costo greedy $c(e)$, $\forall\, e \in C$;
    end if
  end for
  while $C \neq \emptyset$ **do**
    $e^* \leftarrow \text{argmin}\ \{c(e) \mid e \in C$;
    $x \leftarrow x \cup \{e^*\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo greedy $c(e)$, $\forall\, e \in C$;
  end while

Por último se puede introducir aleatorización mediante la perturbación de los costos, es decir, introducimos una perturbación aleatoriamente en los costos y aplicamos el algoritmo greedy de la manera explicada a continuación:

Require: E, $c(\cdot)$;
  $x \leftarrow \emptyset$;
  $C \leftarrow E$
  Perturbar alatoriamente los datos del problema;
  Calcular el costo greedy perturbado $\tilde{c}(e)$, $\forall\, e \in C$;
  while $C \neq \emptyset$ **do**
    $e^* \leftarrow \text{argmin}\ \{\tilde{c}(e) \mid e \in C\}$;
    $x \leftarrow x \cup \{e^*\}$;
    Actualizar el conjunto de candidatos C;
    Calcular el costo greedy perturbado $\tilde{c}(e)$, $\forall\, e \in C$
  end while
return x;

Después de los algoritmos de construcción en los métodos GRASP se usa la búsqueda local, para mejorar el resultado, juega un papel importante, porque busca soluciones óptimas en regiones prometedoras del espacio de soluciones.

La búsqueda local puede ser implementada utilizando la estrategia del *best improving* o la del *first improving* , esta última se detiene tan pronto como encuentres un elemento en el entorno que mejore la solución, mientras que la primeria estrategia consiste en evaluar todos los elementos del entorno y elegir el mejor de ellos como solución actual. El inconveniente de estos métodos es el tiempo que pueden emplear. Si el punto de partida es elegido aleatoriamente este método mejora significativamente la estrategia.

El pseudocódigo de la búsqueda local sería:

Require: x^0, $N(\cdot)$, $f(\cdot)$;
  $x \leftarrow x^0$
  while x no es localmente óptimo con respecto a $N(x)$ **do**
    Sea $y \in N$ tal que $f(x) < f(x)$;
    $x \leftarrow y$;
  end while
return x;

1.4 Simulated Annealing

1.4.1 Descripción

Simulated Annealing, traducido como Recocido Simulado, es el nombre que se le da a la estrategia heurística, que anteriormente nombramos, surgida en la década de los ochenta para resolver complejos problemas combinatorios. Toma este nombre debido a su analogía con el proceso físico del recocido con sólidos, en el cual un sólido cristalino es calentado y luego enfriado lentamente hasta que se convierta en una red lo más regular posible de estructura cristalina y sin defectos cristalinos. Este algoritmo establece la conexión entre este tipo de comportamiento termodinámico y la búsqueda de un mínimo global en un problema de optimización

La idea original surgió con el algoritmo de Metrópolis basándose en el método Montecarlo y la manera de relacionar los parámetros que intervienen en la simulación termodinámica y los parámetros de los métodos de optimización local es:

TERMODINÁMICA		**OPTIMIZACIÓN**
Configuración	$\Longleftrightarrow$	Solución Factible
Configuración Fundamental	$\Longleftrightarrow$	Solución Óptima
Energía de la Configuración	$\Longleftrightarrow$	Coste de la Solución
Temperatura	$\Longleftrightarrow$	(?)

El principal problema de esta analogía es la temperatura, que no tiene un significado claro en el campo de la optimización y por eso debemos considerarlo como un parámetro T que iremos ajustando.

De esta manera se podrían imaginar similares los procesos que ocurren cuando las moléculas de una sustancia van colocándose en los diferentes niveles energéticos buscando el equilibrio a una determinada temperatura, y los que ocurren en los procesos de minimización en optimización local, en el primer paso, fijada la temperatura, la distribución de las partículas en los diferentes niveles sigue la distribución Boltzmann, por lo que cuando una molécula se mueve, ese movimiento será aceptado en la simulación si la energía disminuye o bien con una probabilidad proporcional al factor de Boltzmann en caso contrario; al hablar de optimización, fijado el parámetro T, se produce una perturbación, aceptando directamente la nueva solución cuando su coste disminuye, o bien con una probabilidad proporcional al “factor de boltzmann” en caso contrario.

Este último detalle de la probabilidad proporcional es esencial en el algoritmo, ya que para evitar caer en un óptimo local permite con cierta probabilidad el paso a soluciones peores.

Se observa que el comportamiento del factor Boltzmann disminuye rápidamente conforme disminuye la temperatura, es decir, disminuye la probabilidad de que aceptemos una solución peor que la actual.

Entonces la técnica a seguir sería a partir de una temperatura alta que se irá reduciendo y disminuyendo la posibilidad de cambios a soluciones peores cuando ya se haya cercano al óptimo buscado.

El algoritmo sigue los siguientes pasos:

Inicialización:

Se elige una solución inicial x_1 en X.

Se inicializa el mejor valor de F de F y su correspondiente solución x*,*

De tal manera que se tiene: F<- $F(x_1)$*

x^*<- x_1

Se toma n=1,2,… de tal manera que x_n denota la actual solución

Obtenemos una x aleatoria en un entorno de x_n ,$V(x_n)$.

Si $F(x) \leq F(x_n)$ entonces x_{n+1} <-x

Si F(x)<F entonces F*<- F(x) y x*<- x*

Si no se asigna a p un valor aleatorio entre [0,1]

Si $p \leq p(n)$ entonces x_{n+1}<-x

Acaba cuando la condición de parar se cumple.

Se necesitan unas decisiones previas para llevarlo a cabo.

Por un lado la probabilidad usada p(n), que como se ha dicho antes, usaba la misma que el factor Boltzmann:

$$p(n) = e^{-\frac{1}{T(n)}\Delta F_n}$$

Donde $\Delta F_n = F(x) - F(x_n)$ y T(n) se denomina temperatura en n.

Por otro lado la elección de la temperatura de enfriamiento se obtiene a partir de la fórmula $T(kL) = T_k = \alpha^k T_0$ con T_0 como la temperatura inicial, α la velocidad de enfriamiento y L número de iteración mantenida cada vez que una T es elegida. Elegir estos parámetros correctamente es muy importante porque de ellos depende la eficacia del método. A continuación podemos ver una representación gráfica (fig2) de cómo funciona la temperatura.

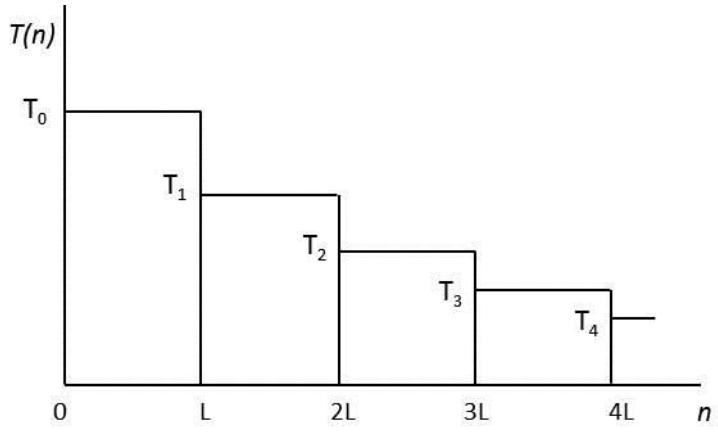

Fig. 2

Y por último la decisión de una condición de parar, la primera opción decide que se para si F* no mejora más del ϵ_1% durante las K_1 series consecutivas de L iteraciones y la segunda opción de parada se produce si el número de movimientos aceptados es menor que el ϵ_2 % durante K_2 series consecutivas de L iteraciones.

1.4.2 La justificación estadística

Para describir varios aspectos del S.A. en problemas discretos se necesitan algunas definiciones,

Se denota Ω como el espacio de soluciones, entonces, se tiene f: Ω → R, que es la función objetico definida en el espacio de soluciones.

El objetivo es identificar un mínimo global x* donde x* ∈ Ω y F(x) ≥ F*(x*) ∀ x ∈ Ω
La función objetivo debe de ser capaz de garantizar que x* existe.

Se define V(x) como un entorno de x ∈ Ω, donde cada solución tiene un entorno asociado V(x) que puede ser alcanzado con una iteración de un algoritmo de búsqueda local.

El S.A. empieza con una solución inicial x ∈ Ω donde se genera una solución vecina x*∈V(x), se basa en el criterio de aceptación de Metrópolis (Metropolis et al. , 1953) donde los modelos, como los de un sistema termodinámico, se mueven desde la solución actual (estado) x ∈ Ω, hacia una solución candidata que contenga energía menor x*∈ V(x).
Esta solución candidata x*, es aceptada como la actual solución basándose en el criterio de probabilidad de aceptación siguiente:

$$P(aceptar\ x^{*} como\ siguiente\ solución) = \begin{cases} e^{-\frac{F(x^{*})-F(x)}{T_k}} & si\ F(x^{*}) - F(x) > 0 \\ 1 & si\ F(x^{*}) - F(x) \leq 0 \end{cases}$$

Se define Tk como un parámetro de temperatura en la iteración k sabiendo que:

$$T_k > 0 \; para \; toda \; k \; y \; \lim_{k \to +\infty} T_k = 0$$

Esta probabilidad de aceptación es el elemento básico del mecanismo de búsqueda en la S.A. Si la temperatura se reduce a una velocidad suficientemente lenta, entonces el sistema puede llegar a un equilibrio en cada iteración k. Se tienen $F(x)\; y \; F(x^*)$ que denotan las energías asociadas con soluciones x ∈ Ω y x*∈ Ω respectivamente. Este equilibrio sigue la distribubción de Boltzmann, que puede ser descrito como la probabilidad del sistema de estar en estado x ∈ Ω con energía F(x) y temperatura T:

$$P\{el \; sistema \; está \; en \; estado \; x \; y \; temperatura \; T\} = \frac{e^{-\frac{F(x)}{T_k}}}{\sum_{x^{**} \in \Omega} e^{-\frac{F(x^{**})}{T_k}}}$$

Si la probabilidad de generar una solución candidata x* del entorno de solución x∈Ω es $G_k(x,x^*)$, donde:

$$\sum_{x^* \in V(x)} G_k(x, x^*) = 1 \;\; \forall \, x \;\in\; \Omega, \qquad k = 1,2, \ldots,$$

Y la matriz estocástica cuadrada no negativa P_x se define con las siguientes probabilidades de transición:

$$P_x(x, x^*) = \begin{cases} G_k(x, x^*) e^{-\frac{\Delta_{x,x^*}}{T_k}} & x^* \in V(x), \quad x^* \neq x \\ 0 & x^* \notin V(x), \quad x^* \neq x \\ 1 - \sum_{\substack{x^{**} \in V(x) \\ x^{**} \neq x^*}} P_k(x, x^{**}) & x^* = x \end{cases}$$

Para cada solución i∈Ω y todas las iteraciones k = 1,2, … , y $\Delta_{x,x^*} \equiv F(x^*) - F(x)$.
Estas probabilidades de transición definen una secuencia de soluciones generadas a partir de una cadena de Markov no homogénea (Romeo y Sangiovanni - Vincentelli , 1991).

PARTE 2: PROBLEMA DEL SET COVERING

2.1 Introducción al SCP

El problema del Set Covering (SCP) es un problema clásico, ha llevado a desarrollo de técnicas fundamentales en el campo de los algoritmos de aproximación. Consiste en encontrar un conjunto de soluciones que permita cubrir unas necesidades al menor costo posible. Un conjunto de necesidades, también llamado demanda, que corresponde a las filas y el conjunto solución es la selección de columnas que cubren de forma óptima el conjunto de filas.

Forma parte de uno de los problemas de la lista de 21 problemas NP-completos de Karp, una lista de 21 problemas computacionales famosos acabada en 1972 por el informático Richard Karp. Estos problemas tratan sobre combinatoria y teoría de grafos y que cumplen la característica común de que todos ellos pertenecen a la clase de complejidad de os NP-completos.

En este modelo se minimiza el coste total de apertura de instalaciones mientras se exige que todo punto de demanda quede cubierto por al menos una instalación.

Cuando el coste es igual a uno para todas las instalaciones se denomina unicost y el problema es el de encontrar el mínimo número de instalaciones que cubren todos los puntos de demanda.

El problema de Set Covering puede aparecer en varios formatos, o con la matriz, o con los conjuntos de cubrimiento para cada fila o columna, o con la matriz y la distancia de recubrimiento

2.2 Descripción

El problema del Set Covering (SCP) se describe como, dado un conjunto S, de m objetos, y una familia F={S_1,…,S_n} donde $S_j \subseteq S$ (j=1,…,n) y con un coste no negativo c_j definido para cada subconjunto $S_j \in F$, una cubierta es una familia $C \subseteq F$ tal que esa $S = \bigcup_{k:S_k \in C} S_k$ el coste de la cubierta es $\sum_{k:S_k \in C} c_k$. El SCP consiste en encontrar una cubierta C* minimizando el coste.

El SCP se modela en programación entera definiéndolo para cada índice j (j=1,…,n), siendo x una variable binaria que vale si $S_j \in C^*$ y cero en el resto de casos y a_{ij} es una constante binaria que vale si el objeto i$\in S_j$ con j=1,…,n, y cero en otro caso.

La formulación matemática es:

$$\min \sum_{j=}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, con\ i = 1, \dots, n$$

$$x_j \in \{0,1\}\ con\ j = 1, \dots, n$$

Esta formulación nos sugiere que el SCP se puede ver como un problema de selección de un subconjunto con mínimo coste de columnas de una matriz 0-1 tal que para cada fila i (i=0,…,m) existe al menos una columna seleccionada de tal manera que a_{ij}=1.

A pesar de su aparente simplicidad, el SCP que como hemos visto antes pertenece a la clase de problemas NP-difíciles, y es considerado como uno de los problemas fundamentales de optimización combinatorial. Su importancia teórica se debe a su versatilidad ya que generaliza los problemas más importantes de optimización combinatorial, como node covering, set partitioning, set packing y mini-cut.

2.3 Aplicaciones

El problema del SCP tiene infinidad de aplicaciones, entre las más comunes están, las de localización de servicios, que consiste en encontrar la ubicación más adecuada donde establecer uno o más servicios, de forma que se optimice algún criterio específico.

Otra aplicación es el balance de líneas de montaje que consiste en la agrupación de tareas secuenciales de trabajo en centros de trabajo, con el fin de lograr el máximo aprovechamiento de la mano de obra y equipo y de esa manera reducir o eliminar el tiempo ocioso.

En la recuperación de información también se usa el problema del Set Covering, en la búsqueda de metadatos que describan documentos o en la búsqueda de bases de datos relacionales, para recuperar en textos, imágenes, sonidos o datos de manera pertinente y relevante.

Scheduling de camiones, trenes, barcos etc es otra aplicación, dado un conjunto de máquinas y un conjunto de trayectos, se trata de conseguir minimizar el tiempo y cubrir todas las necesidades.

Otra aplicación es el Airline crew scheduling, consiste en que dentro de un conjunto de restricciones y reglas hay que colocar a un conjunto de personas con ciertos requisitos, de lugar en lugar, con la menor cantidad de personal y material de transporte en el menor tiempo posible, en otras palabras, el menor coste posible.

El cutting stock consiste en aprovechar al máximo el material (papel, tela…) para hacer rectángulos de diferentes tamaños con él y que los sobrante sea mínimo.

Y por último, otra de las aplicaciones más usadas del SCP es en los distritos políticos, se trata de agrupar un territorio en zonas electorales para garantizar la democracia tratando de dificultar la manipulación de procesos electorales, mediante la construcción de distritos conexos que tengan equilibrio poblacional, y compacidad geométrica, existen muchas más, aunque las más usadas son las que se han explicado en este apartado.

2.4 Métodos de Resolución

Se han desarrollado muchos algoritmos para resolver el problema, los métodos exactos requieren un esfuerzo computacional considerable para resolver esos casos. Sin embargo los algoritmos heurísticos constituyen una estrategia de solución alternativa viable, no pueden garantizar una solución óptima pero sí una aproximada.

2.4.1 Métodos exactos

Los algoritmos exactos más utilizados son el Branch and Bound y el Branch and Cut.

El método de branch and bound (ramificación y poda) para resolver problemas de optimización y consiste en tener un árbol de soluciones donde cada rama nos lleva a una posible solución posterior a la actual. El algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas para “podar” esa rama y no continuar gastando recursos y tiempo en procesos que se alejan de la solución óptima.

En este método los nodos del espacio de búsqueda pueden ser de tres tipos, vivos, muertos, o en expansión. Un nodo vivo es un nodo factible y prometedor del que no se han generado todos sus hijos. Con prometedor nos referimos a un nodo que tiene información en expansión para así conseguir una solución mejor que la que está en curso. Un nodo muerto es un nodo del que no van a generarse más hijos o bien porque ya se han generado todos o porque no es factible o porque no es prometedor. Y un nodo en expansión es el que está generando hijos en ese instante.

Este método tiene un recorrido informado, porque fijando un nodo del espacio de búsqueda el siguiente nodo es el más prometedor entre todos los nodos vivos, y es el que se va a convertir en el siguiente nodo en expansión.

Para recorrer el espacio de búsqueda se usan principalmente dos técnicas de ramificación, la estrategia FIFO, la LIFO y la de menor coste. En estrategia FIFO (First In First Out, primero en entrar, primero en salir) se usa la lista de nodos vivos como si fuera una cola y que recorre la anchura del árbol. En la estrategia LIFO (Last In First Out, último en entrar, primero en salir) se usa la lista de nodos vivos como si fuera una cola. Ambas estrategias son búsquedas a ciegas, ya que se expanden sin cuestionarse los beneficios o no de cada nodo. Para mejorar esto surge la estrategia de menor coste, que aporta una visión de futuro a la búsqueda, selecciona de la lista de nodos vivos al que tenga menos coste. Esto puede derivar a la situación de que varios nodos puedan ser expandidos al mismo tiempo, para evitar esto tenemos la estrategia LC-LIFO en la que en caso de empate entre nodos escoge el primero que se introdujo, y la estrategia LC-FIFO, que en caso de empate se queda con el último en ser introducido.

Existen también dos estrategias de poda para conseguir una buena eliminación de nodos que no lleven a soluciones buenas. La primera estrategia consiste en podar un nodo si se puede obtener una solución válida si la cota superior es menor o igual que la cota inferior para algún nodo generado en el árbol. Y la segunda estrategia de poda sería podar aquellos nodos cuya cota superior es menor o igual que el beneficio que se puede obtener si es que se obtiene una posible solución válida para el problema con un beneficio concreto.

El pseudocódigo del algoritmo de Branch and Bound teniendo en cuenta que G(x) es la función de estimación del algoritmo que P es la pila con las posibles soluciones y que usamos "<" en un problema de minimización y ">" en uno de maximización, se tiene:

```
Funcion B&B{
        P= Hijos(x,k)
        Mientras (no vacio (P))
                x(k) = extraer(P)
                if esFactible (x,k) y G(x,k) < optimo
                        si esSolucion (x)
                                Almacenar(x)
                        Else
                                B&B (x,k+1)
                Fin if
        Fin mientras
}
```

La eficiencia de este método depende del procedimiento por el cual los nodos se expanden y de la estimación de los nodos padres e hijos. Lo ideal sería elegir un método de expansión en el que no se solapen los subconjuntos.

El algoritmo de Branch and Cut es un método de optimización combinatorial para resolver problemas enteros lineales, son problemas donde algunas o todas las incógnitas están restringidas a valores enteros. Se trata de una mezcla entre el Branch and Bound explicado anteriormente con métodos de plano de corte.

Este método resuelve programas lineales sin restricciones enteras. Cuando se obtiene una solución óptima que tiene un valor no entero para una variable que ha de ser entera, el algoritmo de planos de corte se usa más adelante para encontrar una restricción lineal que sea satisfecha por todos los puntos factibles enteros pero no cumplida por la solución fraccional actual, si se encuentra esa desigualdad se añade al problema lineal, y se repite hasta que o no encontramos solución entera, demostramos que es óptima, o bien no se encuentran más planos de corte.

El algoritmo sigue los siguientes pasos:

1. Agregar el problema lineal entero inicial L a la lista de problemas activos
2. Establecer x*= null y y_v*=-infinito
3. Mientras L no está vacío
   Selección y eliminación de un problema de L
   Resuelve la relajación PL del problema
   Si la solución no es factible volver a 3, sino x es un valor objetivo v
   Si v≤v* regresar a 3
   Si x es entero ajustar v*<-v y x*<-x y volver a 3
   Buscar planos de corte y si se encuentra añadirlos a la relajación PL y regresar a 3 y a 2
   Ramificar para dividir el problema en nuevos problemas. Añadir estos problemas a L y volver a 3.
4. Devolver x*

En este método el paso más importante es el de la ramificación, existe una variablidad enorme de heurísticas de ramificación que se pueden utilizar. Las más importantes son tres. Most Infeasible Branching, Pseudo Cost Branching y Strong Branching y están basadas en la ramificación de una variable, que consiste en elegir una variable x_i con un valor fraccional x'_i y con la restricción:

$$x_i \leq \lceil x'_i \rceil \; y \; x_i \geq \lfloor x'_i \rfloor$$

El primer método elije la variable con la parte fraccionaria más cercana a 0.5. En el segundo se hace un seguimiento de cada variable x_i y se analiza el cambio en la función objetivo cuando se ha elegido previamente la ramificación. Y en el tercer método se trata

de encontrar la mejora de la función objetivo antes de la ramificación haciendo test a cada una de las variables, por lo que el principal inconveniente es el coste computacional muy elevado.

Los métodos exactos que se usan en esta memoria para el Set Covering están implementados genéricamente en el programa Xpress.

2.4.2 Métodos heurísticos

Para evitar toda la carga computacional explicada anteriormente, se echa mano de los métodos heurísticos. Se van a analizar algunos de los algoritmos más usados para el SCP como son el método greedy clásico, la eliminación de columnas redundantes, el Simulated Annealing y diferentes formas aleatorización en el método greedy como son, el greedy probabilístico el random greedy), introducción de perturbación en el random greedy, Meta-RaPS.

2.4.2.1 Gredy clásico y extensiones

El método greedy clásico (Chvátal, 1979), consiste en inicializar una solución S como vacía e igualar el conjunto de filas no cubiertas M' con M. Después, de forma iterativa, la columna con el mínimo valor de c_j/α_j es añadida a S y M' se actualiza. c_j es el costo original y α_j las filas en M' cubiertas por la columna j.

El pseudocódigo de éste método greedy es:

Paso 1: Inicialización
C= ∅, U=I, $\beta_i=0 \ \forall \ i \in I$

Paso 2 : Selección de columna
Calcular para todo $A_j \in J$ la relación $f_j= c_j/\alpha_j$
Establecer $C=C \cup \{A_k\}$ donde $A_k= \arg(\min_{j:Aj \in J} f_j)$
Establecer $\beta_i = \beta_i+1$ para todo $i \in I_k$
Establecer $U=U \setminus I_k$

Paso 3: Terminación del Test
Si ($U \neq \emptyset$) ir al paso 2 sino devolver C como la solución

Para entender el pseudocódigo anterior, tendremos en cuenta que:

$A=(a_{ij})$ es una matriz con columnas $A_1, \ldots, A_n$
I es el conjunto de filas de A
J el conjunto de columnas de a
I_j el conjunto de filas tal que $a_{ij}=1$
C el conjunto de columnas en una solución
U el conjunto de fijas no cubiertas
α_j el número de filas de U tal que $a_{ij}=1$
β_i el número de columnas de C que cubren la fila i.

La mayor parte de la carga computacional se usa en la ordenación de las columnas, con lo cual es muy rápido.

Con el fin de favorecer la generación de soluciones mejores se consideraran varios métodos de introducción de aleatorización como son la asignación al azar(random greedy), el greedy probabilístico y la perturbación y el meta-RaPS que veremos más adelante. Al ejecutarse de forma multi-arranque, estos métodos introducen diversificación. Constituyen unas metaheurísticas.

Eliminación de columnas

En la práctica se ha observado que los métodos greedy proporcionan unas soluciones en las que alguna de las filas está sobrecubiertas. De foma que se puede eliminar alguna columna y que la solución siga siendo factible. La eliminación de columnas redundantes mejora la eficacia de los métodos heurísticos

Esta herramienta consiste en que para cada columna j se calcula el número de filas que cubre, el objetivo es identificar las columnas que se pueden quitar y que todas las filas sigan estando cubiertas.

Sea $S \subseteq N$
Siendo ω_i el número de columnas de la solución S que cubren la fila i
$\omega_i \geq 1 \quad \forall i \in I$
$\forall k \in S$ **si** se cumple $\omega_i >= 1+a_{ik}$
entonces se puede eliminar la columna k de S (S=s-{k}) factible

Greedy probabilístico

Como adelantábamos anteriormente, el algoritmo greedy puede ser aleatorizado de la siguiente manera: en lugar de seleccionar en cada iteración de la columna el f_j menor se puede hacer en dos etapas. Primero establecer que $B=\{A_{(k)}: k=1,\ldots,s\}$ estará formado por las s mejores columnas candidatas y en una segunda etapa seleccionar una columna B al azar y anexada a C.

De tal manera que la selección de probabilidad de una columna $A_{(k)} \in B$ sea:

$$\pi_{(k)} = \frac{c_{max} - c_{(k)} + 1}{\sum_{j=1,s}(c_{max} - c_{(k)} + 1)}$$

donde $c_{max} = \max_{j=1,\dots,s} c_j$

Esta distribución de probabilidad disminuye con el costo de las columnas y de esta manera, las columnas con costos más pequeños quedan favorecidas. Por tanto la diversificación se introduce a través de la asignación al azar del proceso de selección. El valor del parámetro s debe fijarse con un valor estrictamente mayor que 1. Un valor grande de s tiende a producir una solución al azar pura.

El pseudocódigo del greedy probabilístico es el siguiente:

Probabilistic_greedy (s,C,U)

Paso 1: inicialización

C=conjunto de columnas que cubren las filas en I\U

$$\beta_i = \sum_{j:A_j \in C} a_{ij} \quad \forall i \in I$$

Paso 2: selección de columnas

-Calcular para todos los A_j la relación $f_j = c_j/\alpha_j$

-Establecer B= conjunto de las s mejores columnas y calcular π_j para cada $A_j \in I$

-Establecer $C=C \cup \{A_k\}$ donde A_k se selecciona aleatoriamente de acuerdo con la distribución (π_j)

-Se toma $\beta_i = \beta_i + 1$ para $i \in I_k$

-Establecer $U=U \setminus I_k$

Paso 3: Eliminación de columnas redundantes

-Examinar todas las columnas de C por orden decreciente de los costes

-Para todo $A_j \in C$ hacer (si $\beta_i \geq 2$ para todo $i \in I_k$ entonces borrar A_j de C)

Paso 4: Terminación del Test

Si ($U \neq \emptyset$) ir al paso 2 sino devolver C como la solución

Random greedy

Para conseguir mejores resultados del método greedy se echa mano de la aleatorización. Es un método multi-arranque que es idéntico al greedy salvo que elije aleatoriamente un elemento de la lista de candidatos, que en este caso, son todas las columnas que alcanzan el mínimo del criterio greedy (c_j/α_j). Este método tiene excelentes resultados en el caso unicost, como demostró Tal Grossman y Avishai Wool en su trabajo Computational Experience with Approximation Algorithms for the Set Covering Problem.

Random greedy con perturbación

Una manera de mejorar el random greedy es introducir una perturbación en el coste. Para ello solo es necesario sustituir el coste c_j de cada columna $A_j \in J$ por $\tilde{c_j} = (1+\xi_j)\ c_j$ donde ξ_j es un pequeño número aleatorio tal que $E(\xi_j)=0$. La idea de la perturbación supone implícitamente una solución óptima que no es sensible a una variación pequeña de los datos, sin embargo la iteración del algoritmo greedy con diferentes costos perturbados puede dar lugar a diferentes soluciones casi óptimas. Por lo tanto se explorarán nuevas regiones del espacio de búsqueda.

2.4.2.2 Métodos que usan búsqueda local

Simulated Annealing

El Simulated Annealing es un método heurístico que como se explica al principio introduce aleatorización en la parte de búsqueda local. Para el problema del Set Covering, la heurística de búsqueda local requiere una solución inicial factible S, referida a las columnas que cubren todas las filas. Denotamos Z(S) como el coste de S.

El pseudocódigo del Simulated Annealing (teniendo en cuenta que MT es el máximo tiempo permitido, T la temperatura inicial, TL el número de iteraciones para cada temperatura y CF el factor de enfriamiento) es:

Establecer T, TL y CF
Establecer máximo tiempo de trabajo MT
Establecer la hora actual ctime
Do while ctime<MT
    Do i=1 TL
        Buscar un entorno S' de S
        Tomar Z(S') como costo de S'
        Tomar ∂=Z(S')-Z(S)
        If $\partial \leq 0$
            S=S'
            S*=S'
        Else
            S=S' con probabilidad $e^{-\partial/T}$
        End if
    Continue
    Tomar T= T x CF
    Leer ctime
Continue
Return S* y Z(S*)
End

Método Meta-RaPS

Otro de los métodos que nos ayudan a resolver el problema del Set Covering es el MetaRaPS (Metaheuristic for Randomized Priority Search) desarrollado por Guanghui Lan y otros (2007), en el trabajo "*An effective and simple heuristic for the set covering problema"*. Los pasos esenciales para aplicar con éxito este método son diseñar heurísticos de construcción y mejoras efectivas. Los métodos GRASP son un caso particular de Meta-RaPS.

En la parte de heurística de construcción consiste en modificar el algoritmo greedy, de tal manera que, si el greedy clásico evalúa cada columna j por la función $f(c_j, \alpha_j)= c_j/ \alpha_j$ donde c_j es el coste de la columna j y α_j es el número de filas descubiertas que podrían ser cubiertas por la columna j, es decir $\alpha_j = \{ i : i \in I_j \setminus U_{n \in x} I_n\}$ y siempre añade a un conjunto de soluciones X la columna con el mínimo valor de c_j/ α_j, en el Meta-RaPS se escoge $f(c_j, \alpha_j)= c_j/ \alpha_j$ como la regla de prioridad, por lo que el resultado del greedy para cada columna j(P_j) es c_j/ α_j.

Hay que tener en cuenta que aquí cuanto más bajo sea el resultado del greedy más fácil es de elegir la columna. Solo durante un % de tiempo (%priority) la columna con el mínimo valor de c_j/ α_j será seleccionada, mientras que el tiempo restante la columna se elegirá al azar. Después se construye la solución factible.

Todas las columnas redundantes serán eliminadas de la solución, la columna redundante con más costo se eliminará primero.

El pseudocódigo del método de construcción será:

Meta-RaPS_SCP_construction(I,J,%prioridad, %restricción)
X=∅
I*=I
While I*≠ ∅
Seleccionar $f(c_j, \alpha_j)= c_j / \alpha_j$ como regla de prioridad y encontrar los mejores candidatos ω, que cumplan $f(c_\omega, \alpha_\omega)=\min_{j\epsilon J\setminus X} f(c_j, \alpha_j)$
P=random(1,100)
If P>%priority **then**
CL={j:j ∈ J\X **and** c_ω / α_ω x (1+%restricción/100) }
Aleatoriamente se selecciona un elemento ω_1 de la CL y se establece $\omega= \omega_1$
End if
Añadir el elemento ω a la solución X, es decir, X=XU{ ω }; I*=I*\I_ω
End while
Eliminar columnas redundantes
Return X

Para mejorar la calidad de esta solución, se aplica un procedimiento de búsqueda local, comenzando en la solución obtenida. Este método está basado en la siguiente idea de vecindad: si dos soluciones comparten al menos una columna esas dos soluciones disjuntas se denominarán soluciones vecinas.

Una solución vecina es obtenida a través de un procedimiento en el que en primer lugar un número de columnas se elimina al azar de la solución factible, por lo que la solución se volverá infactible, al haber alguna fila descubierta. Y en segundo lugar, para completar la solución parcial hasta una solución completa se debería resolver un problema de Set Covering de tamaño reducido, que se compone de las filas no cubiertas, y de las columnas que podrían cubrir esas filas. Este problema es resuelto de forma heurística mediante un método greedy, en este caso, el Meta RaPS.

El pseudocódigo de la parte de la búsqueda local es:

NeighborSearch (I,J,X,%priority,%restriction, search_magnitude,imp_iteration)

For iter= 1, …, imp_iteration

Aleatoriamente borramos columnas de Xm el máximo número de columnas que se pueden borrar es: |X| * search_magnitude

Formular un SCP reducido:

$$I' = I \bigcup_{m \epsilon X} I_m \qquad J' = J \bigcup_{i \epsilon I'} J_i$$

Resolver el SCP reducido

X'= Meta_RaPS_SCP_construction (I',J',%priority, %restriction)

Construir las soluciones vecinas X'=X' U X

Eliminar las columnas redundantes de X'

If el valor de la función objetivo, X', es menor que X **then**

X=X'

end if

end for

return X

Y ahora el pseudocódigo del algoritmo Meta-RaPS es:

Meta_RaPS_SCP (I_0,J_0, %priority, %restriction, %improvement, max_iteration, search_magnitude, imp_iteration)

Realizar el reprocesamiento

obtener el conjunto de filas reducido $I \subseteq I_0$

obtener el conjunto de columnas reducido $J \subseteq J_0$

Establecer el conjunto de soluciones vacío $X^*=X=\emptyset$

$Z^*=Z_{\text{antes de la mejora}}$ = LARGE_NUMBER {establecer un número grande}

Establecer el problema inicial básico vacío $J_c=\emptyset$

For it= 1, …, max_iteration

{fase de construción}

Meta_RaPS_SCP_construction(I,J,%priority,%restriction)

Actualizar J_c

$Z = \sum_{j \epsilon X} c_j$ donde X es la solución de la fase de contrucción

If $Z<Z^*_{\text{antes de la mejora}}$ **then** $Z^*_{\text{antes de la mejora}}=Z$

{fase de mejora}

If $Z \leq (1+\%\text{improvement}) \times Z^*_{\text{antes de la mejora}}$ **then** NeighborSearch(I,J_c,X, %priority, %restriction,search_magnitude, imp_iteration)

Penaliza los peores elementos de X

{actualización del a mejor solución}

If (Z<Z*)**then**

Z*=Z

X*=X

End if

End for

Return X*

PARTE 3: RESULTADOS COMPUTACIONALES

Se van a analizar los diferentes bancos de datos, aplicando lo visto anteriormente y analizando la eficacia de cada método.
Los siguientes archivos contienen datos reales sobre cierto servicio sanitario, el formato de los datos .cyl es el siguiente:

m el número de puntos de demanda
n el número de puntos de servicio
dem(i) para i=1, …, m las demandas en una línea
dist(i,j) para i=1, …, m y j=1, …, n matriz mxn de tiempos de desplazamiento del servicio sanitario de la demanda al servicio

Los resultados obtenidos del método exacto Branch and Cut, del greedy, del greedy aleatorizado, del greedy aleatorizado con perturbación y del Meta-RaPS y sus respectivos tiempos de ejecución para una distancia de cubrimiento de 35 y para el caso unicost son:

	B&C	t	Gr	t	R-Gr	t	RGr+pert	t	Meta-RaPS	t
a1.cyl	4	0.01	5	0.014	4	0.35	4	0.516	4	0.26
a2.cyl	9	0.032	9	0.078	9	0.94	9	1.532	7	0.61
a3.cyl	7	0.016	10	0.032	7	1.11	7	1.703	7	0.62
a4.cyl	9	0.031	14	0.063	9	1.65	9	2.656	8	0.91
a5.cyl	3	0.01	3	0.032	3	0.31	3	0.453	3	0.212
a6.cyl	4	0.06	4	0.07	4	0.13	4	0.203	2	0.07
a7.cyl	8	0.172	9	0.188	8	0.71	8	1.14	6	0.39
a8.cyl	4	0.09	4	0.011	4	0.24	4	0.344	4	0.18
a9.cyl	7	0.012	9	0.016	7	0.3	7	0.453	5	0.21
a10.cyl	2	0.007	2	0.008	2	0.11	2	0.172	2	0.15
a11.cyl	2	0.008	3	0.01	3	0.17	3	0.25	3	0.12
a12.cyl	4	0.031	4	0.031	4	0.52	4	0.828	4	0.35
a13.cyl	6	0.031	8	0.031	6	0.522	6	0.813	6	0.31

Se puede ver que el método random greedy consigue llegar siempre al resultado óptimo en poco tiempo, en este conjunto de datos no se ve muy bien la diferencia de carga computacional del método greedy, pero en los siguientes se verá mejor. Hay que destacar la rapidez y eficacia también del Met-RaPS.

Ahora se analizan 87 ficheros de datos de la librería ORLIB, 50 de estos son el conjunto de problemas de 4 a 6 y de la A a la E de J.E.Beasley “An algorithm for Set Covering problems" European Journal of Operational Research 31 (1987) 85-93. Los problemas 4, 5 y 6 son los originales del artículo. La siguiente tabla muestra la relación entre estas pruebas de grupos de problemas y los archivos correspondientes:

Problema	Archivo
4	scp41, …, scp410
5	scp51,…, scp510
6	Scp61,…, scp610
A	scpa1,…, scpa5
B	scpb1,…, scpb5
C	scpc1,…, scpc5
D	scpd1,…, scpd5
E	scpe1,…, scpe5

20 de estos archivos de datos son el conjunto de problemas de la E a la H de J.E.Beasley "A lagrangian heuristic for Set Covering problems" Naval Research Logistics 37 (1990) 151-164. La siguiente tabla muestra la relación entre estas pruebas grupos de problemas y los archivos correspondientes:

Problema	Archivo
E	scpnre1, …, scpnre5
F	scpnrf1, …, scpnrf5
T	scpnrt1, …, scpnrt5
H	scpnrh1, …, scpnrh5

10 de estos archivos de datos son problemas unicost del artículo "Computational Experience with Approximation Algorithms for the Set Covering Problem", by T. Grossman and A. Wool, que se public en European Journal of Operational Research.

El formato de todos estos archivos de datos 80 es :

el número de filas (m), el número de columnas (n), el costo de cada columna c (j) con j = 1 , ... , n, para cada fila i (i = 1 , ..., m): el número de columnas que cubren la fila i seguido de una lista de las columnas que cubren la fila i.

Para los archivos del European Journal of Operational Research paper by Beasley se sabe que el valor de la solución óptima para cada uno de estos archivos de datos se da en el documento, que el archivo más grande es scpd5.txt de tamaño y que el conjunto de los archivos es de tamaño 5800Kb.

Los archivos asociados con el artículo del Naval Research Logistics tienen en el archivo de datos el valor de la solución heurística y el archivo más grande es scpnrh5txt de tamaño 2600Kb.

Los resultados obtenidos para el caso unicost del método exacto Branch and Cut, del greedy, del greedy aleatorizado con 200 iteraciones, del greedy aleatorizado con perturbación y del Meta-RaPS y sus respectivos tiempos de ejecución son:

	$B\&C_{20}$	t	Gr	t	R-Gr	t	RGr+p	t	Meta-RaPS	t
scp41.txt	40	19.8	41	0.7	39	7.1	41	10.2	41	3.6
scp42.txt	37	20.0	41	0.6	38	6.9	38	9.9	38	3.5
scp43.txt	39	19.6	43	0.7	40	7.3	40	10.1	40	3.6
scp44.txt	40	19.5	44	0.7	42	7.5	42	10.5	41	3.7
scp45.txt	39	19.2	44	0.7	39	7.2	39	10.1	40	3.5
scp46.txt	39	20.1	43	0.7	40	7.2	39	10.4	40	3.7
scp47.txt	39	20.1	43	0.6	41	7.3	41	10.3	41	3.6
scp48.txt	38	19.5	42	0.6	40	7.2	39	10.3	40	3.6
scp49.txt	39	19.2	42	0.7	40	7.3	40	10.5	41	3.6
scp51.txt	35	20.1	37	1.2	36	13.2	36	18.9	36	6.6
scp52.txt	36	19.7	38	1.2	35	13.1	36	18.6	36	6.6
scp53.txt	36	20.2	37	1.3	36	13.2	36	18.8	36	6.6
scp54.txt	37	20.2	39	1.2	36	13.1	35	18.4	35	6.5
scp55.txt	35	20.2	37	1.2	36	13.0	36	18.5	36	6.5
scp56.txt	36	20.0	40	1.2	36	13.2	36	18.8	37	6.6
scp57.txt	35	19.7	38	1.3	35	13.1	35	18.7	35	6.5
scp58.txt	37	20.4	39	1.3	37	13.7	37	19.4	37	6.8
scp59.txt	37	19.5	38	1.2	37	13.4	37	19.1	37	6.6
scp61.txt	24	19.9	23	0.5	21	7. 1	21	10.6	22	3.6
scp62.txt	23	19.8	22	0.4	21	6.9	21	10.9	21	3.6
scp63.txt	26	29.9	23	0.5	21	7.2	22	11.3	21	3.6
scp64.txt	23	19.7	22	0.5	21	7.2	21	11.3	22	3.6
scp65.txt	23	19.4	23	0.5	22	7.4	22	11.5	22	3.7
scp410.txt	39	19.8	43	0.7	40	7.2	41	10.5	41	3.6
scp510.txt	37	19.8	39	1.3	36	13.3	36	18.8	36	6.6
scpa1.txt	42	20.1	42	3.2	41	28.1	40	41.4	41	14.1
scpa2.txt	43	20.5	42	3.2	40	27.9	40	41.4	41	13.9
scpa3.txt	44	20.4	43	3.2	40	27.9	40	41.4	41	13.9
scpa4.txt	42	19.8	41	3.1	39	27.2	39	40.0	40	13.5
scpa5.txt	41	19.8	43	3.2	40	27.4	40	40.6	40	13.5
scpb1.txt	28	20.9	24	2.2	22	30.6	22	48.1	23	15.3

scpb2.txt	28	20.9	23	2.1	22	29.9	22	46.8	23	15.1
scpb3.txt	29	20.9	23	2.2	23	30.5	22	47.6	23	15.5
scpb4.txt	29	20.6	24	2.2	23	30.7	23	48.1	23	25.7
scpb5.txt	28	20.7	25	2.2	23	30.5	23	47.7	23	15.5
scpc1.txt	49	21.1	47	6.5	45	49.0	44	72.9	45	25.4
scpc2.txt	48	20.9	47	6.5	45	49.1	45	77.4	45	25.3
scpc3.txt	49	20.8	47	6.5	45	49.8	45	73.1	46	25.4
scpc4.txt	47	21.6	46	6.5	45	49.7	45	73.1	46	24.3
scpc5.txt	49	21.5	47	6.5	45	49.3	45	72.5	45	24.3
scpclr10.txt	26	19.5	32	0.3	28	7.7	28	12.2	29	3.8
scpclr11.txt	31	20.9	30	0.9	27	23.9	27	38.9	28	11.6
scpclr12.txt	34	27.2	31	2.9	23	72.2	26	119.2	29	35.2
scpclr13.txt	35	170.2	32	8.8	31	221.5	31	363.2	30	105.8
scpd1.txt	30	21.9	27	4.3	25	57.6	25	87.8	25	29.1
scpd2.txt	31	22.9	26	4.3	26	57.8	25	88.0	25	29.0
scpd3.txt	32	22.6	27	4.3	25	57.1	25	88.1	26	29.7
scpd4.txt	32	21.9	26	4.3	26	59.1	26	89.1	25	30.1
scpd5.txt	31	22.01	27	4.4	26	57.9	25	89.0	26	30.0
scpe1.txt	5	1.9	5.	0.0	5	11	5	1.6	5	0.6
scpe2.txt	5	0.9	5	0.0	5	1.1	5	1.6	5	0.6
scpe3.txt	5	1.7	5	0.0	5	1	5	1.5	5	0.5
scpe4.txt	5	0.6	6	0.0	5	1.1	5	1.6	5	0.6
scpe5.txt	5	0.8	5	0.0	5	1	5	1.5	5	0.6
scpnre1.txt	22	29.1	18	5.1	17	111.4	17	173.4	17	57.8
scpnre3.txt	21	28.3	18	5.0	17	129.7	17	176.5	17	57.6
scpnre4.txt	21	30.2	18	5.1	17	128.9	17	171.9	18	58.1
scpnre5.txt	20	29.1	18	5.1	17	129.2	17	172.6	17	57.9

El cálculo del óptimo exacto para estos problemas requiere mucho tiempo de ejecución, en general alrededor de una hora, por lo que se ha reducido el tiempo a 20 segundos, y el que aparece como óptimo exacto, es la mejor solución obtenida en esos 20 segundos.

Se observa que el método random greedy alcanza el óptimo la mayoría de las veces con tiempos muy cortos, y que el Meta-RaPS llega muy rápidos a soluciones muy aproximadas.

El código en Xpress, para cada método se encuentra en los Anexos 1, 2, 3, 4, 5 y 6. Los métodos exactos son los ya implementados en el programa Xpress y los métodos heurísticos están programados en el lenguaje mosel, pero sin usar Xpress como optimizador. Se podrían programar en cualquierl lenguaje como C, Java, R, etc…

CONCLUSIONES

Después de este estudio se puede asegurar que la introducción de aleatorización en los métodos greedy mejora los resultados. En los casos exactos la carga computacional es muy grande y debido a que el algortimo es muy elaborado tiene una implementación comercial poco económica, a parte, de emplear mucho tiempo en encontrar el óptimo. Sin embargo, los algoritmos heurísticos propuestos funcionan muy bien, disminuyen el tiempo de ejecución considerablemente y los resultados se acercan mucho a los resultados óptimos exactos. El método meta-RaPS, en concreto, es el que mejores soluciones encuentra, las que se acercan más al valor óptimo exacto y se nota mucho que el tiempo empleado es menor que en los otros algoritmos.

En los problemas de optimización de los archivos de formato .cyl que son los que contienen datos reales sobre cierto servicio sanitario, se alcanza el óptimo exacto, muy rápidamente, y con el greedy, random greedy, random greedy más perturbación y el meta-RaPS, se resuelven empleando más tiempo, pero siempre en tiempos muy cortos.

Sin embargo con los problemas de la ORLIB, que son problemas mucho más grandes, con más número de celdas. Se nota mucho la diferencia de carga operativa entre los métodos exactos y los algoritmos heurísticos. Estos últimos son mucho más rápidos y ofrecen una solución idéntica a la exacta en muchos de los casos, y muy aproximada en otros.

BIBLIOGRAFÍA

Adenso-Díaz , B. (y otros) (ed.) (1996), *Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería,* Paraninfo, Madrid.

Henderson (y otros) (2003), *The Theory and Practice of Simulated Annealing, Handbook of metaheurstics,.*

Paola Festa (2002), *Greedy Randomized Adaptative Search Procedures,* AIROnews.

Mauricio G.C. Resende y José Luis González Velarde (2003), *GRASP: Greedy Randomized Adaptative Search Procedures,* Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial No. 19

Mauricio G.C. Resende y Celso C. Ribeiro (2002), *Greedy Randomized Adaptative Search Procedure.*

Leonidas S. Pitsoulis, Mauricio G.C. Resende , *Greedy Randomized Adaptative Search Procedures,*

Thomas A. Feo, Mauricio G.C. Resende (1995), *Greedy Randomized Adaptative Search Procedures,* Journal od Golbal Optimitation.

Thomas A. Feo, Mauricio G.C. Resende (1989), *A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem,* Operations Research Letters.

Rafael Martí (2003), *Algoritmos Heurísticos.*

Guanghui Lan (y otros) (2007), *An effective and simple heuristic for the set covering problem,* Science Direct, European Journal of Operational Reseach 176.

Elena Marchiori y Adri Steenbeek (1998), *An Iterated Heuristic Algorithm for the Set Covering Problem,* Proceedings WAE'98.

Larry W. Jacobs, Michael J. Brusco (1995), *Note: A Local-Search Heuristic for Large Set-Covering Problems,* Naval Research Logistics Vol 42.

Joaquín Bautista, Jordi Pereira (2007), *A GRASP Algorithm to solve the Unicost Set-Covering Problem,* Knowledge and Decision Technologies.

Tal Grossman, Avishai Wool (1996), *Computational experience with approximation algorithms for the set covering problem,* El Servier, European Journal of Operational Reseach 101.

M. Haouari, J.S. Chaouachi (2002), *A probabilistic greedy search algorithm for combinatorial optimization with application to the set covering problem*, Journal of Operational Reseach Society.

Chvátal (1979), A *Greedy heuristic for the set covering problem.* Mathematics of Operations Research. vol4, 233-235

ANEXO 1

Método exacto para los ficheros CYL

```
model "Modelo scp_dist_1"
  uses "mmxprs"
  uses "mmsystem"


declarations
   m, n: integer
   archivo_datos="a1.cyl"
end-declarations

!========= LECTURA DE DATOS ========================
fopen(archivo_datos,F_INPUT)
readln(m)
readln(n)

declarations
   pdemanda = 1..m
   pservicio = 1..n
   dem:array(pdemanda)of real
   dist:array(pdemanda,pservicio)of real
   x:array(pservicio)of mpvar

   dc = 35
   daux, dmm: real
   lambda:real

   !añadimos las declaraciones propias del método greedy
   d:array(pservicio)of integer
   r:array(pservicio) of real
   solu:array(pservicio)of real
   varFijada:array(pservicio)of integer
   filaCubierta:array(pdemanda)of integer
   jaux:integer
   aux:real
   nfilCub:integer
end-declarations

forall(i in pdemanda)do
   readln(dem(i))
```

```
  dem_total+=dem(i)
end-do
forall(i in pdemanda)forall(j in pservicio)read(dist(i,j))
fclose(F_INPUT)

!=================== MODELO ============================
writeln("\nModelo de cubrimiento total (Set Covering)\n")
writeln("Fichero de datos: ",archivo_datos)
writeln("Parametros fundamentales: m=",m,", n= ",n,", dc=",dc)

!writeln("Calculos previos: dem_total = ",dem_total)

! Calculo la distancia mínima máxima dmm de un p. de demanda a uno de servicio
! y tiene que ser DM > dmm para que haya soluciones

dmm:=-999999.9
forall(i in pdemanda)do
  daux:= 999999.9
  forall(j in pservicio)do
          if(dist(i,j)<daux)then daux:=dist(i,j)
          end-if
  end-do

  if(daux > dmm)then dmm:=daux
  end-if
end-do
writeln("Minima distancia maxima = ",dmm)

!=========================================================================
=
starttime:= gettime

forall(j in pservicio)x(j) is_binary


obj:=sum(j in pservicio)x(j)


forall(i in pdemanda)res_dem(i):=
        sum(j in pservicio|dist(i,j)<=dc)x(j)>=1

!====================== RESOLUCION ========================
!setcallback(XPRS_CB_INTSOL, "printsol")
```

```
minimize(obj)
if(getprobstat = XPRS_OPT)then

        writeln("Para dc = ",dc," -> Minimo numero de puntos de servicio: ",getobjval)

elif(getprobstat = XPRS_INF)then

        writeln("Para dc = ",dc," -> Problema  no factible")
        exit(0)

end-if

finaltime:=gettime

writeln("\nTiempo total = ",finaltime-starttime," segundos\n")
```

ANEXO 2

Método exacto para los ficheros de la librería ORLIB

```
model "Set covering 1"
uses "mmxprs";
uses "mmsystem"

parameters
        archivo_datos = "D:/datos/Scp/Orlib/scp51.txt"
end-parameters

declarations
  n:integer
  m:integer
end-declarations

fopen(archivo_datos,F_INPUT)
  readln(m,n)


declarations
  columnas = 1..n
  filas = 1..m

  a:array(columnas,filas)of integer
  costo:array(columnas)of integer
  nc:array(filas)of integer
  cub:array(filas,columnas)of integer
  x:array(columnas)of mpvar

  tiempo_calculo= 20
end-declarations

forall(j in columnas)read(costo(j))

forall(i in filas)do
  read(nc(i))
  forall(j in 1..nc(i))do
    read(cub(i,j))
    !a(i,cub(i,j)):=1
  end-do
end-do
```

```
fclose(F_INPUT)

forall(j in columnas)do
  x(j) is_binary
  costo(j):=1
end-do
costo_total:=sum(j in columnas)costo(j)*x(j)


forall(i in filas)res_cub(i):=sum(j in 1..nc(i))x(cub(i,j))>=1

!exportprob(EP_MIN,"",costo_total)
starttime:= gettime
setparam("XPRS_MAXTIME",tiempo_calculo)

writeln("\nRelajación Lineal. Solución óptima:")
minimize(XPRS_LIN,costo_total)
writeln("Costo total = ",getobjval)
!forall(j in columnas|x(j).sol>=0.99)writeln("  x(",j,") = ",x(j).sol)

writeln("\nSolución óptima entera:")
minimize(costo_total)
writeln("Costo total = ",getobjval)
!forall(j in columnas|x(j).sol>=0.99)writeln("  x(",j,") = ",x(j).sol)

finaltime:=gettime

writeln("\nTiempo total = ",finaltime-starttime," segundos\n")

end-model
```

ANEXO 3

Método greedy par alas librerías CYL y ORLOB

```
model "Modelo scp_formatos_2"
  uses "mmxprs"
  uses "mmsystem"


declarations
   m, n: integer
   archivo_datos="D:/Datos/SCP/ORLIB/scp43.txt"
   formato_datos = 1

   !archivo_datos="D:/Datos/SCP/CYL/a1.cyl"
   !formato_datos = 2

   resolver_bc =0
end-declarations

!================== LECTURA DE DATOS ========================
fopen(archivo_datos,F_INPUT)
if(formato_datos=1)then
    readln(m,n)
elif(formato_datos=2)then
    readln(m)
    readln(n)
end-if

declarations
   filas = 1..m
   columnas = 1..n
   dem:array(filas)of real
   dist:array(filas,columnas)of real
   a:array(filas,columnas)of integer
   costo:array(columnas)of integer

   x:array(columnas)of mpvar

   dc = 38
   daux, dmm: real
   dmax:integer

   nfilcub:integer
```

```
  varfijada, solu,d:array(columnas)of integer
  filacubierta, w:array(filas)of integer

  nc:array(filas)of integer
  cub:array(filas,columnas)of integer

  nonul:integer
  nfil:array(filas)of integer              ! número de columnas que cubren cada fila
  indfil:array(range)of integer
  inifil:array(filas)of integer
  finfil:array(filas)of integer

  ncol:array(columnas)of integer           ! número de filas que cubren cada columna
  indcol:array(range)of integer
  inicol:array(columnas)of integer
  fincol:array(columnas)of integer

  tiempo_calculo = 30
end-declarations
!================ LECTURA DE DATOS ======================
if(formato_datos=1)then                                  ! Formato ORLIB
  forall(j in columnas)read(costo(j))

  forall(i in filas)do
    read(nc(i))
    forall(j in 1..nc(i))do
      read(cub(i,j))
    end-do
  end-do

  forall(i in filas,j in 1..nc(i))a(i,cub(i,j)):=1
end-if

if(formato_datos=2)then                                  ! Formato CyL
  forall(j in columnas)costo(j):=1
  forall(i in filas)do
     readln(dem(i))
     dem_total+=dem(i)
  end-do
  forall(i in filas,j in columnas)read(dist(i,j))

  forall(i in filas,j in columnas)
    if(dist(i,j)<=dc)then a(i,j):=1
    else a(i,j):=0
```

```
    end-if
end-if

fclose(F_INPUT)

!=================== Calculos previos ============================
writeln("\nModelo Set Covering, costos unitarios\n")
writeln("Fichero de datos: ",archivo_datos)
writeln("Parametros fundamentales: m=",m,", n= ",n)
if(formato_datos=2)then
  writeln("dc = ",dc,", dem_total = ",dem_total)

  ! Calculo la distancia mínima máxima dmm de un p. de demanda a uno de servicio
  ! y tiene que ser DM > dmm para que haya soluciones

  dmm:=-999999.9
  forall(i in filas)do
    daux:= 999999.9
    forall(j in columnas)do
            if(dist(i,j)<daux)then daux:=dist(i,j)
            end-if
    end-do

    if(daux > dmm)then dmm:=daux
    end-if
  end-do
  writeln("Minima distancia maxima = ",dmm)
end-if

!================  Indices por filas y por columnas =================

nonul:=0

forall(i in filas)do
 inifil(i):=nonul+1
 forall(j in columnas)do
   if(a(i,j)=1)then
     nonul:=nonul+1
     indfil(nonul):=j
   end-if
 end-do
 finfil(i):=nonul
end-do
(!
```

```
writeln("\nnonul = ",nonul,", indfil = \n")
forall(k in 1.. nonul)write(indfil(k)," ")
writeln
writeln("inifil = ",inifil)
writeln("finfil = ",finfil)
!)
nonul:=0

forall(j in columnas)do
  inicol(j):=nonul+1
  forall(i in filas)do
    if(a(i,j)=1)then
      nonul:=nonul+1
      indcol(nonul):=i
    end-if
  end-do
  fincol(j):=nonul
end-do
(!
writeln("\n\nnonul = ",nonul,", indcol = \n")
forall(k in 1.. nonul)write(indcol(k)," ")
writeln
writeln("inicol = ",inicol)
writeln("fincol = ",fincol)
!)
!========================== MODELO ==============================
starttime:= gettime

forall(j in columnas)x(j) is_binary


obj:=sum(j in columnas)x(j)

!forall(i in filas)res_dem(i):=
!            sum(j in columnas|dist(i,j)<=dc)x(j)>=1

forall(i in filas)res_cub(i):=
                    sum(j in columnas)a(i,j)*x(j)>=1

!exportprob(EP_MIN,"a1_28_2",obj)
!====================== SOLUCION ÓPTIMA CON
XPRESS=========================

if(resolver_bc=1)then
```

```
 writeln("\n\nSoluciones:\n")

 setparam("XPRS_MAXTIME",tiempo_calculo)

 minimize(obj)

 if(getprobstat = XPRS_OPT or getprobstat = XPRS_UNF)then
  writeln("Con Xpress\t\t\tCosto = ",getobjval,", tiempo = ",gettime-starttime)
 elif(getprobstat = XPRS_INF)then
        writeln(dc,"\t\tProblema no factible")
 end-if

 finaltime:=gettime

 forall(i in filas)do
  w(i):=0
  forall(j in columnas|x(j).sol>=0.99 and a(i,j)=1)w(i):=w(i)+1
 end-do

 !writeln("w óptimo = ",w)
end-if
!================= HEURÍSTICA GREDDY CLÁSICA============================

nfilcub:=0
while(nfilcub < m)do
  !calculo d(j) = las filas q cubre cada columna
  forall(j in columnas|varfijada(j)=0)do
          d(j):=0
    forall(i in filas|filacubierta(i)=0)do
      if(a(i,j)=1)then d(j):=d(j)+1
      end-if
    end-do
    if(d(j)=0) then
      varfijada(j):=1
      solu(j):=0
    end-if
  end-do
  !indice que alcanza el máxiimo de d(j)
  dmax:=-999999
  forall(j in columnas|varfijada(j)=0)do
    if(d(j)> dmax)then
      dmax:=d(j)
      jaux:=j
    end-if
```

```
  end-do
  !dmax:=max(j in columnas)d(j)
  varfijada(jaux):=1
  solu(jaux):=1
  forall(i in filas|filacubierta(i)=0)do
    if(a(i,jaux)=1)then
      filacubierta(i):=1
      nfilcub:=nfilcub+1
    end-if
  end-do
end-do
zp1:=sum(j in columnas)solu(j)
writeln("\nCon el método Greedy\t\tCosto = ",zp1)

!========      Eliminación de columnas redundantes ======================

forall(i in filas)do
  w(i):=0
  forall(j in columnas)
    if(solu(j)=1 and a(i,j)=1)then w(i):=w(i)+1
    end-if
end-do
!writeln("w = ",w)
forall(k in columnas|solu(k)=1)do
  elimin:=1
  forall(i in filas)
    if(w(i)<1+a(i,k))then elimin:=0
    end-if
  if(elimin=1)then
    solu(k):=0
    forall(i in filas)w(i):=w(i)-a(i,k)
    writeln("eliminada la columna ",k)
  end-if
end-do

zp2:=sum(j in columnas)solu(j)
writeln("\nDespués de eliminación\t\tCosto = ",zp2)
 finaltime:=gettime

writeln("\nTiempo total = ",finaltime-starttime," segundos\n")


end-model
```

ANEXO 4

Método Random Greedy para CYL y ORLIB

```
model "Modelo scp_formatos_2"
  uses "mmxprs"
  uses "mmsystem"


declarations
   m, n: integer
  ! archivo_datos="scpnre5.txt"
  ! formato_datos = 1

   archivo_datos="a7.cyl"
   formato_datos = 2
end-declarations

!================== DECLARACIONES ========================

fopen(archivo_datos,F_INPUT)
if(formato_datos=1)then
    readln(m,n)
elif(formato_datos=2)then
    readln(m)
    readln(n)
end-if

declarations
   filas = 1..m
   columnas = 1..n
   dem:array(filas)of real
   dist:array(filas,columnas)of real
   a:array(filas,columnas)of integer
   costo:array(columnas)of integer

   x:array(columnas)of mpvar

   dc = 35
   daux, dmm: real

   nc:array(filas)of integer
   cub:array(filas,columnas)of integer
```

```
  nonul:integer
  nfil:array(filas)of integer              ! número de columnas que cubren cada fila
  indfil:array(range)of integer
  inifil:array(filas)of integer
  finfil:array(filas)of integer

  ncol:array(columnas)of integer           ! número de filas que cubren cada columna
  indcol:array(range)of integer
  inicol:array(columnas)of integer
  fincol:array(columnas)of integer

  nfilcub, zmin:integer
  varfijada, solu,d, solufin:array(columnas)of integer
  filacubierta, w, wfin:array(filas)of integer
  zp, zp2, zfin:integer
  dmax, nmax, kmax:integer
  indmax:array(columnas)of integer
  Niter = 200

  busqueda_local = 0
  iter_max_bl = 5

  resolucion_xpress =0
  tiempo_calculo = 30

end-declarations
!================ LECTURA DE DATOS ======================
if(formato_datos=1)then                                    ! Formato ORLIB
  forall(j in columnas)read(costo(j))

  forall(i in filas)do
    read(nc(i))
    forall(j in 1..nc(i))do
      read(cub(i,j))
    end-do
  end-do

  forall(i in filas,j in 1..nc(i))a(i,cub(i,j)):=1
end-if

if(formato_datos=2)then                                    ! Formato CyL
  forall(j in columnas)costo(j):=1
  forall(i in filas)do
```

```
    readln(dem(i))
    dem_total+=dem(i)
  end-do
  forall(i in filas,j in columnas)read(dist(i,j))

  forall(i in filas,j in columnas)
   if(dist(i,j)<=dc)then a(i,j):=1
   else a(i,j):=0
   end-if
end-if

fclose(F_INPUT)

!==================== Calculos previos ============================
writeln("\nModelo Set Covering, costos unitarios\n")
writeln("Fichero de datos: ",archivo_datos)
writeln("Parametros fundamentales: m=",m,", n= ",n)
if(formato_datos=2)then
  writeln("dc = ",dc,", dem_total = ",dem_total)

  ! Calculo la distancia mínima máxima dmm de un p. de demanda a uno de servicio
  ! y tiene que ser DM > dmm para que haya soluciones

  dmm:=-999999.9
  forall(i in filas)do
    daux:= 999999.9
    forall(j in columnas)do
          if(dist(i,j)<daux)then daux:=dist(i,j)
          end-if
    end-do

    if(daux > dmm)then dmm:=daux
    end-if
  end-do
  writeln("Minima distancia maxima = ",dmm)
end-if

!================  Indices por filas y por columnas ==================

nonul:=0

forall(i in filas)do
 inifil(i):=nonul+1
 forall(j in columnas)do
```

```
    if(a(i,j)=1)then
      nonul:=nonul+1
      indfil(nonul):=j
    end-if
  end-do
  finfil(i):=nonul
end-do
(!
writeln("\nnonul = ",nonul,", indfil = \n")
forall(k in 1.. nonul)write(indfil(k)," ")
writeln
writeln("inifil = ",inifil)
writeln("finfil = ",finfil)
!)
nonul:=0

forall(j in columnas)do
  inicol(j):=nonul+1
  forall(i in filas)do
    if(a(i,j)=1)then
      nonul:=nonul+1
      indcol(nonul):=i
    end-if
  end-do
  fincol(j):=nonul
end-do
writeln("\n\nnonul = ",nonul)
(!
writeln("\n\nnonul = ",nonul,", indcol = \n")
forall(k in 1.. nonul)write(indcol(k)," ")
writeln
writeln("inicol = ",inicol)
writeln("fincol = ",fincol)
!)
!========================== MODELO ============================
if(resolucion_xpress)=1 then
  starttime:= gettime

  forall(j in columnas)x(j) is_binary


  obj:=sum(j in columnas)x(j)

  !forall(i in filas)res_dem(i):=
```

```
  !         sum(j in columnas|dist(i,j)<=dc)x(j)>=1

  forall(i in filas)res_cub(i):=
                    sum(j in columnas)a(i,j)*x(j)>=1

  !exportprob(EP_MIN,"a1_28_2",obj)

  !======================= SOLUCION ÓPTIMA CON
XPRESS=========================
  writeln("\n\nSoluciones:\n")

  setparam("XPRS_MAXTIME",tiempo_calculo)

  minimize(obj)

  if(getprobstat = XPRS_OPT or getprobstat = XPRS_UNF)then
    writeln("Con Xpress\t\t\tCosto = ",getobjval,", tiempo = ",gettime-starttime)
  elif(getprobstat = XPRS_INF)then
          writeln(dc,"\t\tProblema no factible")
  end-if


  !forall(i in filas)do
  !   w(i):=0
  !   forall(j in columnas|x(j).sol>=0.99 and a(i,j)=1)w(i):=w(i)+1
  !end-do
  !writeln("w óptimo = ",w)
end-if
!================= HEURÍSTICA GREDDY
ALEATORIZADA============================
starttime:= gettime
zfin:= 9999

forall(t in 1..Niter)do
  ! inicializaciones para cada iteración:
  forall(j in columnas)do
    varfijada(j):=0
    solu(j):=0
  end-do
  forall(i in filas)filacubierta(i):=0
  nfilcub:=0

  while(nfilcub < m)do
    !calculo d(j) = las filas q cubre cada columna
```

```
    forall(j in columnas|varfijada(j)=0)do
            d(j):=0
      forall(k in inicol(j)..fincol(j))do
       if(filacubierta(indcol(k))=0)then d(j):=d(j)+1
       end-if
      end-do
      if(d(j)=0) then
        varfijada(j):=1
        solu(j):=0
      end-if
    end-do
    !indice que alcanza el máximo de d(j)
    dmax:=-999999
    forall(j in columnas|varfijada(j)=0)do
      if(d(j)> dmax)then
        dmax:=d(j)
        jaux:=j
      end-if
    end-do
    !dmax:=max(j in columnas)d(j)
    nmax:=0
    forall(j in columnas|varfijada(j)=0)do
      if(d(j)=dmax)then
        nmax:=nmax+1
        indmax(nmax):=j
      end-if
    end-do
    kmax:=ceil(random*nmax)
    jaux:=indmax(kmax)
    varfijada(jaux):=1
    solu(jaux):=1
    forall(k in inicol(jaux)..fincol(jaux))do
      if(filacubierta(indcol(k))=0)then
        filacubierta(indcol(k)):=1
        nfilcub:=nfilcub+1
      end-if
    end-do
  end-do
  zp1:=sum(j in columnas)solu(j)
  !writeln("\nCon el método Greedy\t\tCosto = ",zp1)

!========      Eliminación de columnas redundantes ======================

  forall(i in filas)do
```

```
    w(i):=0
    forall(k in inifil(i)..finfil(i))
      if(solu(indfil(k))=1)then w(i):=w(i)+1
      end-if
  end-do
  !writeln("w = ",w)
  forall(j in columnas|solu(j)=1)do
    wmin:=999
    forall(k in inicol(j)..fincol(j))
      if(w(indcol(k))<wmin)then wmin:=w(indcol(k))
      end-if
    if(wmin>=2)then
      solu(j):=0
      forall(k in inicol(j)..fincol(j))w(indcol(k)):=w(indcol(k))-1
      !writeln("eliminada la columna ",k)
    end-if
  end-do

  zp2:=sum(j in columnas)solu(j)
  !writeln("\nDespués de eliminación\t\tCosto = ",zp2)

  if(zp2<zfin)then
    zfin:=zp2
    forall(j in columnas)solufin(j):=solu(j)
    forall( i in filas)wfin(i):=w(i)
  end-if

end-do! final del bucle forall(t in 1..Niter)

writeln("\nSolución final R-Gr, con ",Niter," iteracioes: ",zfin,", tiempo = ",gettime-
starttime)

!============= BÚSQUEDA LOCAL SIMPLE (intercambio de 2 por 1)
==========================
if(busqueda_local=1)then
  starttime:=gettime
  writeln("Búsqueda local con intercambios (2 por 1)")
  zp:=zfin
  forall(j in columnas)solu(j):=solufin(j)
  forall( i in filas)w(i):=wfin(i)
  iter:=0
  final:=1

  repeat
```

```
   iter:=iter+1
   writeln("iteración ",iter,", valor actual ",zp)
   forall(j,k in columnas|solu(j)=1 and solu(k)=1)do
     forall(i in filas)w(i):=w(i)-a(i,j)-a(i,k)
     forall(l in columnas|solu(l)=0)do
       if(min(i in filas)(w(i)+a(i,l))>=1)then
         solu(j):=0
         solu(k):=0
         solu(l):=1
         forall(i in filas)w(i):=w(i)+a(i,l)
         zp:=sum(f in columnas)solu(f)
               writeln("\nDespués de intercambio\t\tCosto = ",zp)
         final:=0
         break
       end-if
     end-do
     forall(i in filas)w(i):=w(i)+a(i,j)+a(i,k)
     if final=0 then break;end-if
   end-do
   if final=0 then break;end-if
  until (final=1 or iter>iter_max_bl)
  writeln("Final de búsqueda local, tiempo = ",gettime-starttime)
end-if

end-model
```

ANEXO 5

Método Random Greedy con perturbación para los ficheros con formato CYL y ORLIB

```
model "Modelo scp_formatos_2"
  uses "mmxprs"
  uses "mmsystem"


declarations
   m, n: integer
  archivo_datos="scp41.txt"
   formato_datos = 1

  ! archivo_datos="a7.cyl"
   !formato_datos = 2
end-declarations

!================== DECLARACIONES ========================

fopen(archivo_datos,F_INPUT)
if(formato_datos=1)then
    readln(m,n)
elif(formato_datos=2)then
    readln(m)
    readln(n)
end-if

declarations
   filas = 1..m
   columnas = 1..n
   dem:array(filas)of real
   dist:array(filas,columnas)of real
   a:array(filas,columnas)of integer
   costo:array(columnas)of real

   x:array(columnas)of mpvar

   dc = 35
   daux, dmm: real

   nc:array(filas)of integer
   cub:array(filas,columnas)of integer
```

```
  nonul:integer
  nfil:array(filas)of integer              ! número de columnas que cubren cada fila
  indfil:array(range)of integer
  inifil:array(filas)of integer
  finfil:array(filas)of integer

  ncol:array(columnas)of integer           ! número de filas que cubren cada columna
  indcol:array(range)of integer
  inicol:array(columnas)of integer
  fincol:array(columnas)of integer

  nfilcub, zmin:integer
  varfijada, solu,d, solufin:array(columnas)of integer
  filacubierta, w, wfin:array(filas)of integer
  zp, zp2, zfin:integer
  dmax, nmax, kmax:integer
  indmax:array(columnas)of integer
  Niter = 200

  busqueda_local = 0
  iter_max_bl = 5

  resolucion_xpress =0
  tiempo_calculo = 30

end-declarations
!================ LECTURA DE DATOS ======================
if(formato_datos=1)then                                   ! Formato ORLIB
  forall(j in columnas)read(costo(j))

  forall(i in filas)do
    read(nc(i))
    forall(j in 1..nc(i))do
      read(cub(i,j))
    end-do
  end-do

  forall(i in filas,j in 1..nc(i))a(i,cub(i,j)):=1
end-if

if(formato_datos=2)then                                   ! Formato CyL
  forall(j in columnas)costo(j):=1
  forall(i in filas)do
```

```
     readln(dem(i))
     dem_total+=dem(i)
   end-do
   forall(i in filas,j in columnas)read(dist(i,j))

   forall(i in filas,j in columnas)
    if(dist(i,j)<=dc)then a(i,j):=1
    else a(i,j):=0
    end-if
end-if

fclose(F_INPUT)

!==================== Calculos previos ============================
writeln("\nModelo Set Covering, costos unitarios\n")
writeln("Fichero de datos: ",archivo_datos)
writeln("Parametros fundamentales: m=",m,", n= ",n)
if(formato_datos=2)then
  writeln("dc = ",dc,", dem_total = ",dem_total)

  ! Calculo la distancia mínima máxima dmm de un p. de demanda a uno de servicio
  ! y tiene que ser DM > dmm para que haya soluciones

  dmm:=-999999.9
  forall(i in filas)do
    daux:= 999999.9
    forall(j in columnas)do
          if(dist(i,j)<daux)then daux:=dist(i,j)
          end-if
    end-do

    if(daux > dmm)then dmm:=daux
    end-if
  end-do
  writeln("Minima distancia maxima = ",dmm)
end-if

!================  Indices por filas y por columnas ==================

nonul:=0

forall(i in filas)do
 inifil(i):=nonul+1
 forall(j in columnas)do
```

```
    if(a(i,j)=1)then
      nonul:=nonul+1
      indfil(nonul):=j
    end-if
  end-do
  finfil(i):=nonul
end-do
(!
writeln("\nnonul = ",nonul,", indfil = \n")
forall(k in 1.. nonul)write(indfil(k)," ")
writeln
writeln("inifil = ",inifil)
writeln("finfil = ",finfil)
!)
nonul:=0

forall(j in columnas)do
  inicol(j):=nonul+1
  forall(i in filas)do
    if(a(i,j)=1)then
      nonul:=nonul+1
      indcol(nonul):=i
    end-if
  end-do
  fincol(j):=nonul
end-do
writeln("\n\nnonul = ",nonul)
(!
writeln("\n\nnonul = ",nonul,", indcol = \n")
forall(k in 1.. nonul)write(indcol(k)," ")
writeln
writeln("inicol = ",inicol)
writeln("fincol = ",fincol)
!)

!añadir perturbación
forall(j in columnas)do
!writeln("costo", costo(j))
costo(j):=costo(j)*(1+(random/10-0.5))
!writeln("costo con perturbacion", costo(j))
end-do
!========================== MODELO ============================
if(resolucion_xpress)=1 then
  starttime:= gettime
```

```
  forall(j in columnas)x(j) is_binary


  obj:=sum(j in columnas)x(j)

  !forall(i in filas)res_dem(i):=
  !         sum(j in columnas|dist(i,j)<=dc)x(j)>=1

  forall(i in filas)res_cub(i):=
                  sum(j in columnas)a(i,j)*x(j)>=1

  !exportprob(EP_MIN,"a1_28_2",obj)

  !====================== SOLUCION ÓPTIMA CON
XPRESS=========================
  writeln("\n\nSoluciones:\n")

  setparam("XPRS_MAXTIME",tiempo_calculo)

  minimize(obj)

  if(getprobstat = XPRS_OPT or getprobstat = XPRS_UNF)then
    writeln("Con Xpress\t\t\tCosto = ",getobjval,", tiempo = ",gettime-starttime)
  elif(getprobstat = XPRS_INF)then
          writeln(dc,"\t\tProblema no factible")
  end-if


  !forall(i in filas)do
  !   w(i):=0
  !   forall(j in columnas|x(j).sol>=0.99 and a(i,j)=1)w(i):=w(i)+1
  !end-do
  !writeln("w óptimo = ",w)
end-if
!================= HEURÍSTICA GREDDY
ALEATORIZADA============================
starttime:= gettime
zfin:= 9999

forall(t in 1..Niter)do
  ! inicializaciones para cada iteración:
  forall(j in columnas)do
    varfijada(j):=0
```

```
  solu(j):=0
end-do
forall(i in filas)filacubierta(i):=0
nfilcub:=0

while(nfilcub < m)do
  !calculo d(j) = las filas q cubre cada columna
  forall(j in columnas|varfijada(j)=0)do
          d(j):=0
    forall(k in inicol(j)..fincol(j))do
      if(filacubierta(indcol(k))=0)then d(j):=d(j)+1
      end-if
    end-do
    if(d(j)=0) then
      varfijada(j):=1
      solu(j):=0
    end-if
  end-do
  !indice que alcanza el máximo de d(j)
  dmax:=-999999
  forall(j in columnas|varfijada(j)=0)do
    if(d(j)> dmax)then
      dmax:=d(j)
      jaux:=j
    end-if
  end-do
  !dmax:=max(j in columnas)d(j)
  nmax:=0
  forall(j in columnas|varfijada(j)=0)do
    if(d(j)=dmax)then
      nmax:=nmax+1
      indmax(nmax):=j
    end-if
  end-do
  kmax:=ceil(random*nmax)
  jaux:=indmax(kmax)
  varfijada(jaux):=1
  solu(jaux):=1
  forall(k in inicol(jaux)..fincol(jaux))do
    if(filacubierta(indcol(k))=0)then
      filacubierta(indcol(k)):=1
      nfilcub:=nfilcub+1
    end-if
  end-do
```

```
  end-do
  zp1:=sum(j in columnas)solu(j)
  !writeln("\nCon el método Greedy\t\tCosto = ",zp1)

!========      Eliminación de columnas redundantes ======================

  forall(i in filas)do
    w(i):=0
    forall(k in inifil(i)..finfil(i))
      if(solu(indfil(k))=1)then w(i):=w(i)+1
      end-if
  end-do
  !writeln("w = ",w)
  forall(j in columnas|solu(j)=1)do
    wmin:=999
    forall(k in inicol(j)..fincol(j))
      if(w(indcol(k))<wmin)then wmin:=w(indcol(k))
      end-if
    if(wmin>=2)then
      solu(j):=0
      forall(k in inicol(j)..fincol(j))w(indcol(k)):=w(indcol(k))-1
      !writeln("eliminada la columna ",k)
    end-if
  end-do

  zp2:=sum(j in columnas)solu(j)
  !writeln("\nDespués de eliminación\t\tCosto = ",zp2)

  if(zp2<zfin)then
    zfin:=zp2
    forall(j in columnas)solufin(j):=solu(j)
    forall( i in filas)wfin(i):=w(i)
  end-if

end-do! final del bucle forall(t in 1..Niter)

writeln("\nSolución final R-Gr con perturbación, con ",Niter," iteracioes: ",zfin,", tiempo =
",gettime-starttime)

!============= BÚSQUEDA LOCAL SIMPLE (intercambio de 2 por 1)
==========================
if(busqueda_local=1)then
  starttime:=gettime
  writeln("Búsqueda local con intercambios (2 por 1)")
```

```
  zp:=zfin
  forall(j in columnas)solu(j):=solufin(j)
  forall( i in filas)w(i):=wfin(i)
  iter:=0
  final:=1

  repeat
   iter:=iter+1
   writeln("iteración ",iter,", valor actual ",zp)
   forall(j,k in columnas|solu(j)=1 and solu(k)=1)do
     forall(i in filas)w(i):=w(i)-a(i,j)-a(i,k)
     forall(l in columnas|solu(l)=0)do
       if(min(i in filas)(w(i)+a(i,l))>=1)then
         solu(j):=0
         solu(k):=0
         solu(l):=1
         forall(i in filas)w(i):=w(i)+a(i,l)
         zp:=sum(f in columnas)solu(f)
               writeln("\nDespués de intercambio\t\tCosto = ",zp)
         final:=0
         break
       end-if
     end-do
     forall(i in filas)w(i):=w(i)+a(i,j)+a(i,k)
     if final=0 then break;end-if
   end-do
   if final=0 then break;end-if
  until (final=1 or iter>iter_max_bl)
  writeln("Final de búsqueda local, tiempo = ",gettime-starttime)
end-if

end-model
```

ANEXO 6

Método Meta-RaPS para las librerías CYL y ORLIB

```
model "Modelo scp_metaraps2"
  uses "mmxprs"
  uses "mmsystem"


declarations
   m, n: integer

  ! archivo_datos="scpnre5.txt"
   !formato_datos = 1

   archivo_datos="a1.cyl"
   formato_datos = 2

end-declarations

!================== DECLARACIONES ========================

fopen(archivo_datos,F_INPUT)
if(formato_datos=1)then
    readln(m,n)
elif(formato_datos=2)then
    readln(m)
    readln(n)
end-if

declarations
   filas = 1..m
   columnas = 1..n
   dem:array(filas)of real
   dist:array(filas,columnas)of real
   a:array(filas,columnas)of integer
   costo:array(columnas)of integer

   x:array(columnas)of mpvar

   !dc:integer
   dcmin = 40
   dcmax = 40
```

```
  dc1= 35
  daux, dmm: real

  nelefil:array(filas)of integer                    ! número de columnas que cubren cada
fila
  cub:array(filas,columnas)of integer

  nonul:integer

  indfil:array(range)of integer
  inifil:array(filas)of integer
  finfil:array(filas)of integer

  indcol:array(range)of integer
  inicol:array(columnas)of integer
  fincol:array(columnas)of integer

  nfilcub, zmin:integer
  varfijada, solu,d, solufin:array(columnas)of integer
  filacubierta, w, wfin:array(filas)of integer
  zp, zp2, zfin:integer
  dmax, nlc, klc, jmax:integer
  indlc:array(columnas)of integer

  Niter = 100
  prioridad = 10
  restriccion = 10
  eliminacion = 1

  P:integer                                         ! entero para la prioridad
  resolucion_xpress =0
  tiempo_calculo_xpress = 30



  magnitud_busqueda = 0.3
  mejora = 15
  max_iteration = 100
  imp_iteration = 400
end-declarations

tcero:=gettime
!================ LECTURA DE DATOS ======================
if(formato_datos=1)then                             ! Formato ORLIB
```

```
  forall(j in columnas)read(costo(j))

  forall(i in filas)do
   read(nelefil(i))
   forall(j in 1..nelefil(i))do
     read(cub(i,j))
   end-do
  end-do

  forall(i in filas,j in 1..nelefil(i))a(i,cub(i,j)):=1
end-if

if(formato_datos=2)then                              ! Formato CyL
  forall(j in columnas)costo(j):=1
  forall(i in filas)do
    readln(dem(i))
    dem_total+=dem(i)
  end-do
  forall(i in filas,j in columnas)read(dist(i,j))
end-if

fclose(F_INPUT)

!=================== Calculos previos ============================
writeln("\nModelo Set Covering, costos unitarios\n")
writeln("Fichero de datos: ",archivo_datos)
writeln("Parametros fundamentales: m=",m,", n= ",n)
if(formato_datos=2)then
  writeln("dc = ",dc1,", dem_total = ",dem_total)

  ! Calculo la distancia mínima máxima dmm de un p. de demanda a uno de servicio
  ! y tiene que ser DM > dmm para que haya soluciones

  dmm:=-999999.9
  forall(i in filas)do
    daux:= 999999.9
    forall(j in columnas)do
          if(dist(i,j)<daux)then daux:=dist(i,j)
          end-if
    end-do

    if(daux > dmm)then dmm:=daux
    end-if
  end-do
```

```
  writeln("maxima  distancia minima = ",dmm)
end-if

!========================================== BUCLE PRINCIPAL
==================================
forall(dc in dcmin..dcmax)do

  if(formato_datos=2)then                              ! Formato CyL
    forall(i in filas,j in columnas)
      if(dist(i,j)<=dc)then a(i,j):=1
      else a(i,j):=0
      end-if
  end-if

!================ Indices por filas y por columnas ==================

nonul:=0

forall(i in filas)do
 inifil(i):=nonul+1
 forall(j in columnas)do
   if(a(i,j)=1)then
     nonul:=nonul+1
     indfil(nonul):=j
   end-if
 end-do
 finfil(i):=nonul
end-do

nonul:=0

forall(j in columnas)do
 inicol(j):=nonul+1
 forall(i in filas)do
   if(a(i,j)=1)then
     nonul:=nonul+1
     indcol(nonul):=i
   end-if
 end-do
 fincol(j):=nonul
end-do
writeln("\n\nnonul = ",nonul)

!========================== MODELO ============================
```

```
  starttime:= gettime
if(resolucion_xpress=1)then
  forall(j in columnas)x(j) is_binary


  obj:=sum(j in columnas)x(j)

  forall(i in filas)res_cub(i):=
                  sum(j in columnas)a(i,j)*x(j)>=1

  !forall(i in filas)res_dem(i):=
  !         sum(j in columnas|dist(i,j)<=dc)x(j)>=1

  !exportprob(EP_MIN,"a1_28_2",obj)

  !====================== SOLUCION ÓPTIMA CON
XPRESS=========================
  !writeln("\n\nSoluciones:\n")

  setparam("XPRS_MAXTIME",tiempo_calculo_xpress)


  minimize(obj)

  if(getprobstat = XPRS_OPT or getprobstat = XPRS_UNF)then
    writeln("\ndc = ",dc,", Xpress\t\t\tCosto = ",getobjval,", tiempo = ",gettime-starttime)
  elif(getprobstat = XPRS_INF)then
          writeln(dc,"\t\tProblema no factible")
  end-if

end-if
  !forall(i in filas)do
  !   w(i):=0
  !   forall(j in columnas|x(j).sol>=0.99 and a(i,j)=1)w(i):=w(i)+1
  !end-do
  !writeln("w óptimo = ",w)

!================ HEURÍSTICA Meta-RaPS ============================
starttime:= gettime
zfin:= 9999

forall(t in 1..Niter)do
  ! inicializaciones para cada iteración:
```

```
forall(j in columnas)do
  varfijada(j):=0
  solu(j):=0
end-do
forall(i in filas)filacubierta(i):=0
nfilcub:=0

while(nfilcub < m)do
  !calculo d(j) = las filas q cubre cada columna
  forall(j in columnas|varfijada(j)=0)do
          d(j):=0
    forall(k in inicol(j)..fincol(j))do
     if(filacubierta(indcol(k))=0)then d(j):=d(j)+1
     end-if
    end-do
    if(d(j)=0) then
      varfijada(j):=1
      solu(j):=0
    end-if
  end-do
  !indice que alcanza el máximo de d(j)
  dmax:=-999999
  forall(j in columnas|varfijada(j)=0)do
    if(d(j)> dmax)then
      dmax:=d(j)
      jmax:=j
    end-if
  end-do
  jaux:=jmax

  P:=ceil(random*100)
  if(P > prioridad)then                    ! aquí se define la Lista de Candidatos
    nlc:=0
    forall(j in columnas|varfijada(j)=0)do
      if(d(j)>=dmax*(1-restriccion/100))then
        nlc:=nlc+1
        indlc(nlc):=j
      end-if
    end-do
    klc:=ceil(random*nlc)
    jaux:=indlc(klc)
  end-if
```

```
    varfijada(jaux):=1
    solu(jaux):=1
    forall(k in inicol(jaux)..fincol(jaux))do
      if(filacubierta(indcol(k))=0)then
        filacubierta(indcol(k)):=1
        nfilcub:=nfilcub+1
      end-if
    end-do
  end-do
  zp1:=sum(j in columnas)solu(j)
  writeln("\nCon el método Meta-RaPS\t\tCosto = ",zp1)

!========      Eliminación de columnas redundantes ======================
if(eliminacion=1)then
  forall(i in filas)do
    w(i):=0
    forall(k in inifil(i)..finfil(i))
      if(solu(indfil(k))=1)then w(i):=w(i)+1
      end-if
  end-do
  !writeln("w = ",w)
  forall(j in columnas|solu(j)=1)do
    wmin:=999
    forall(k in inicol(j)..fincol(j))
      if(w(indcol(k))<wmin)then wmin:=w(indcol(k))
      end-if
    if(wmin>=2)then
      solu(j):=0
      forall(k in inicol(j)..fincol(j))w(indcol(k)):=w(indcol(k))-1
      !writeln("eliminada la columna ",k)
    end-if
  end-do

  z1:=sum(j in columnas)solu(j)
  writeln("    Después de eliminación\t\tCosto = ",zp1)
end-if
  if(zp1<zfin)then
    zfin:=zp1
    forall(j in columnas)solufin(j):=solu(j)
    forall( i in filas)wfin(i):=w(i)
  end-if

end-do! final del bucle forall(t in 1..Niter)
```

```
writeln("Meta-RaPS, con ",Niter," iteraciones, costo = ",zfin,", tiempo = ",gettime-
starttime)

end-do
writeln("tiempo total = ",gettime-tcero)
!writeln("wfin = ",wfin)
(!============= BÚSQUEDA LOCAL SIMPLE (intercambio de 2 por 1)
==========================
starttime:=gettime
writeln("Búsqueda local con intercambios (2 por 1)")
zp:=zfin
forall(j in columnas)solu(j):=solufin(j)
forall( i in filas)w(i):=wfin(i)
iter:=0
final:=1

repeat
   iter:=iter+1
   writeln("iteración ",iter,", valor actual ",zp)
   forall(j,k in columnas|solu(j)=1 and solu(k)=1)do
     forall(i in filas)w(i):=w(i)-a(i,j)-a(i,k)
     forall(l in columnas|solu(l)=0)do
       if(min(i in filas)(w(i)+a(i,l))>=1)then
         solu(j):=0
         solu(k):=0
         solu(l):=1
         forall(i in filas)w(i):=w(i)+a(i,l)
         zp:=sum(f in columnas)solu(f)
               writeln("\nDespués de intercambio\t\tCosto = ",zp)
         final:=0
         break
       end-if
     end-do
     forall(i in filas)w(i):=w(i)+a(i,j)+a(i,k)
     if final=0 then break;end-if
   end-do
   if final=0 then break;end-if
until (final=1 or iter>iter_max_bl)
writeln("Final de búsqueda local, tiempo = ",gettime-starttime)
!)

end-model
```