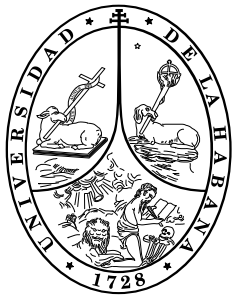
"Algoritmo Genético para un problema de cubrimiento"

Dayron Garcia Perez

Facultad de Matemática y Computación
Universidad de La Habana



Resumen:

En el presente trabajo pretendemos mostrar al lector un problema de cómo cubrir una ciudad con routers inalámbricos de modo que todos sus pobladores tengan acceso a la red de ordenadores que se desea montar. Para dar solución al problema de cubrir la ciudad con el menor costo posible se emplea un algoritmo genético en el cual puede cambiar los parámetros de entrada si así lo desea. También se obtiene un resultado aplicando programación dinámica y se hace una comparación de los resultados.

Índice

1. Introducción

El *Set Covering Problem* forma parte de unos problemas clásicos que pertenecen a la clase NP-Completo y en donde la entrada esta dada por varios conjuntos de elementos o datos que tienen algún elemento en común. En general, estos problemas consisten en encontrar el número mínimo de conjuntos que contengan cierto número de elementos de todos los conjuntos. En otras palabras, consiste en encontrar un conjunto de soluciones que permitan cubrir un conjunto de necesidades al menor costo posible. Un conjunto de necesidades corresponde a las filas, y el conjunto solución es la selección de columnas que cubren en forma óptima al conjunto de filas.

2. Definición del Problema

Como dijimos en el punto anterior, el problema de Set-Covering consiste en encontrar el número mínimo de conjuntos que contengan todos los elementos de todos los conjuntos dados. Existen muchas aplicaciones para este tipo de problemas, siendo las principales la localización de servicios, selección de archivos de un banco de datos, simplificación de expresiones booleanas, balanceo de líneas de producción, entre otros.

Una definición del problema en general es el siguiente:

Sea:

$A = (a_{ij})$ una matriz binaria $(0, 1)$ de dimensiones m x n
$C = (c_j)$ un vector n dimensional de enteros
$M = \{1, ..., m\}$
$N = \{1, ..., n\}$

El valor $c_j (j \in N)$ representa el costo de la columna j, y podemos asumir, sin pérdida de generalidad, $c_j > 0$ para $j \in N$. Así, decimos que una columna $j \in N$ *cubre* una fila $i \in M$ si $a_{ij} = 1$. El SCP (*Set Covering Problem*) busca un subconjunto de columnas de costo mínimo $S \subseteq N$, tal que cada fila $i \in M$ esta cubierta por al menos una columna $j \in S$.

Así, el problema de Set-Covering se puede plantear de la siguiente manera:

$$\text{mín}(z) = \sum_{j=1}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in M$$

$$x_j = \begin{cases} 1 & \text{si} \quad j \in S \\ 0 & \text{si no} \end{cases}$$

Para nuestro trabajo hemos seleccionado el siguiente problema de cubrimiento al cual le aplicaremos un algoritmo genético diseñado para dar solución al mismo.

2.1. Planteamiento del problema

Se desea establecer una red de ordenadores en cierta localidad con determinada distribución geográfica. Para ello se realiza un estudio de los posibles lugares donde se pueden instalar los routers

y su correspondiente área de cubrimiento dada por los distintos sectores de la localidad. A continuación mostramos la distribución geográfica de nuestra localidad y los posibles lugares donde se pueden instalar los servicios. Cada lugar puede dar cobertura a los sectores adyacentes. Se pide dónde ubicar los routers de forma tal de minimizar los costos, pero asegurar que se cubran todos los sectores.

Para nuestro trabajo tenemos la siguiente distribución geográfica según el siguiente esquema donde se representan los diversos sectores y los respectivos lugares donde se pueden instalar los routers. Se consideran que todos los routers son iguales y que su costo de instalación varía según el lugar.

Figura 1: Sectores de la ciudad y posibles lugares de instalación

2.2. Modelo Matemático

Para poder dar solución al problema primeramente debemos realizar un modelo matemático que nos permita interactuar con los datos y poder hacer inferencia sobre las variables que debemos determinar. Por ello planteamos el siguiente modelo:

Datos: Los datos conocidos son:

- Número de sectores: $M = \{1, ..., 22\}$
- Número de routers: $N = \{1, ..., 14\}$
- Costo de los routers: $C = \{c_1, ..., c_{14}\}$
- Información acerca de que si un router j cubre al sector i :
  $a_{ji} = \begin{cases} 1 & \text{si el router } j \text{ cubre al sector } i \\ 0 & \text{si no} \end{cases}$

Ejemplo de ello lo es: la columna $j = 2$ cubre a las filas $i = 3$, $i = 4$, $i = 7$ e $i = 8$; mientras que la columna $j = 14$ cubre a $i = 19$, $i = 20$, $i = 21$ e $i = 22$.

Variables: Las variables necesarias para modelar el problema son:

- $x_j = \begin{cases} 1 & \text{si se utiliza el router en la posición } j \\ 0 & \text{si no} \end{cases}$

Así, podemos plantear la función objetivo como:

$$\text{mín}(z) = \sum_{j=1}^{14} c_j x_j$$

Sujeto a la restricción:

$$\sum_{j=1}^{14} a_{ij} x_j \geq 1 \quad \forall i \in M$$

Dada la matriz:

$$\mathbf{A} = \begin{pmatrix} 00010000000000 \\ 00110000000000 \\ 01100000000000 \\ 11000000000000 \\ 10000000000000 \\ 10000000000000 \\ 11001000000000 \\ 01101100000000 \\ 00110100000000 \\ 00010010000000 \\ 00000110100000 \\ 00001101000000 \\ 00001000000000 \\ 00000001010000 \\ 00000001111000 \\ 00000001100000 \\ 00000000011100 \\ 00000000001110 \\ 00000000000111 \\ 00000000000101 \\ 00000000000001 \\ 00000000000011 \end{pmatrix}$$

2.3. Representación

Al tratar de resolver un problema, lo primero que hay que definir es la representación a utilizar. A los efectos de nuestro problema se escogió la representación binaria (el dominio de todas la variables es {0,1}). Así tenemos un vector de n bits (con n el número de columnas del problema) en donde un 1 en el bit i indica que esa columna (i) es parte de la solución.

Por definición, los AG utilizan representación binaria, sin embargo, esta representación tiene el problema que al aplicarle los operadores genéticos a los individuos, los resultados no siempre serán

soluciones factibles. Esto obliga a aplicar una función de penalización en la función de evaluación para que quede representado el grado de no satisfacción de la solución.

3. Descripción del Algoritmo Genético

Una vez definida la representación, se puede comenzar el desarrollo del algoritmo que resolverá el problema. Para entender mejor el trabajo realizado, primero se explicará, en forma breve, las características principales de un algoritmo genético (AG), para luego explicar cómo se aplican dichas características al problema en cuestión.

Los AG copian el proceso evolucionario mediante el procesamiento simultáneo de una población de soluciones. Por lo general, se comienza con una población generada de manera aleatoria, cuyos individuos se van recombinando de acuerdo al valor que tengan en la función de evaluación. Así mientras mejor sea el valor obtenido por el individuo, más posibilidades tiene de ser seleccionado para generar nuevas soluciones. De esta forma, las nuevas generaciones de soluciones conservan lo bueno de las generaciones anteriores permitiendo, después de un proceso iterativo, llegar a un valor que eventualmente podría ser el óptimo.

Para hacer las combinaciones, el AG estándar posee dos operadores genéticos que son: el cruzamiento y la mutación. El primero toma dos individuos y forma un tercero mediante la combinación de los componentes de los individuos padres, permitiendo de esta forma que el algoritmo efectúe una explotación del espacio de búsqueda. Por otra parte, la mutación toma un individuo y por medio de un proceso genera otro individuo para la nueva población. Esto permite que el algoritmo efectúe una exploración del espacio de búsqueda.

Para implementar un AG es necesario definir los siguientes puntos:

Representación: Este es el punto de partida en la definición de un AG. De esta definición va a depender gran parte de la estructura final de AG. En nuestro caso se eligió la representación binaria.

Inicialización: Se refiere a cómo se va a generar la población inicial. Generalmente, dicha población se genera de manera aleatoria.

Condición de término: Se debe establecer hasta cuando va a recorrer el algoritmo. Generalmente se utiliza un número determinado de generaciones (iteraciones) o una medida de convergencia.

Funciones de Evaluación o Fitness: Es la función que va a determinar que tan buena es una solución. La función de evaluación puede ser igual o diferente a la función objetivo.

Operadores Genéticos: Los operadores genéticos son los encargados de efectuar la reproducción de la población, siendo los más comunes el cruzamiento y la mutación.

Selección: Se refiere a cómo se va a seleccionar los individuos para aplicarles los operadores genéticos. La selección debe dirigir el proceso de búsqueda en favor de los miembros más aptos.

Parámetros de funcionamiento: Un AG necesita que se le proporcionen ciertos parámetros de funcionamiento, tales como el tamaño de la población, la probabilidad de aplicación de los operadores genéticos y la cantidad de generaciones, entre otros.

Una vez definidos los puntos anteriores, se tiene una estructura adecuada para comenzar el desarrollo y la implementación del AG.

3.1. Algoritmo Propuesto

A continuación se expondrá el algoritmo genético propuesto para resolver el problema planteado anteriormente y así como los resultados obtenidos tras varios experimentos realizados. El algoritmo fue diseñado en **MATLAB** en su versión **R2013a** y la función que debe implementarse es la llamada *genetico.m*. A continuación se expondrán las diferentes funciones adicionales que conforman el algoritmo y que de ellas depende en su gran importancia el funcionamiento del mismo en general.

3.1.1. Representación

Como se indicó anteriormente, lo primero que se debe definir es la representación que se va a utilizar. En nuestro caso utilizamos la representación binaria ya que es la representación más natural para el problema de set covering.

3.1.2. Inicialización

El tamaño de la población, así como la población inicial son elegidos de manera tal que el dominio de las soluciones asociadas con la población este cubierto adecuadamente. En nuestro caso se implementó una función que genera aleatoriamente una población de individuos de manera tal que se cumplan las condiciones iniciales de factibilidad iniciales. Una inicialización no aleatoria puede acelerar la convergencia al óptimo pero podemos incurrir en la desventaja de la prematura convergencia hacía óptimos locales.

3.1.3. Condición de término

Los AG poseen básicamente dos condiciones de término: número máximo de generaciones y convergencia en solución. En el primer caso, el algoritmo itera un número determinado de veces, luego se detiene y entrega el mejor valor que tenga hasta ese momento la función objetivo. En nuestro algoritmo el número de iteraciones es la condición de término.

3.1.4. Función de Evaluación

La función de evaluación de nuestro algoritmo coincide con la planteada inicialmente en el modelo, la cual queda como sigue:

$$\sum_{j=1}^{14} c_j * x_j$$

Como se dijo anteriormente, para trabajar con soluciones no factibles se utilizó la estrategia de utilizar una función de penalización a la función objetivo, la cual consiste en agregar como sumando a un número superior al mayor de los costos de los routers (M).

$$\sum_{j=1}^{14} (c_j * x_j + (M * \text{cantidad de routers empleados}))$$

Dicho factor de penalización debe tener un valor adecuado de forma tal que afecte al resultado final.

3.1.5. Selección

Los operadores genéticos necesitan que se les entregue uno o dos individuos para procesarlos. Esto hace necesario que se defina un criterio de selección de los individuos. En nuestro caso, la selección se efectuó de acuerdo al método del torneo. Reporta un valor computacional muy bajo debido a su sencillez. Se selecciona un grupo de 2 individuos y se genera un número aleatorio entre 0 y 1. Si este número es menor que un cierto umbral K (usualmente 0.75 que es usado en el algoritmo), se selecciona para reproducirse al individuo con mejor adaptación, y si este número es menor que K, se selecciona, por el contrario, al individuo con peor adaptación. También cabe destacar que el individuo con mejor adaptación se define por la siguiente fórmula:

$$p_i = \frac{f_i}{\sum_{i=1}^{m} f_i}$$

Esta técnica tiene la ventaja de que permite un cierto grado de elitismo ya que el mejor nunca va a morir, y los mejores tienen más probabilidad de reproducirse y de emigrar que los peores pero sin producir una convergencia genética prematura cuando la población es suficientemente amplia.

3.1.6. Operadores Genéticos

Los operadores genéticos utilizados en el desarrollo del algoritmo fueron el cruzamiento y la mutación.

El operador de cruce se refiere a realizar una exploración de toda la información almacenada hasta el momento en la población y combinarla para crear mejores individuos de acuerdo a la siguiente regla propuesta según a como se plantea en la literatura. Una vez seleccionados los individuos mediante un proceso de selección previamente ejecutado se cruzan los individuos de acuerdo a la siguiente regla: se genera un número aleatorio entre 1 y la cantidad de routers que se disponen, luego una vez seleccionada una posición en las cadenas de los progenitores se intercambian los genes a la izquierda de esta posición. Es importante destacar que a dicho operador se le introduce una probabilidad de cruzamiento y si es mayor que cierto número generado aleatoriamente entre 0 y 1 se ejecuta el operador, si ocurre lo contrario no se ejecuta el operador y los padres pasan como los hijos para la siguiente generación.

La mutación es un operador básico, que proporciona un pequeño elemento de aleatoriedad en el entorno de los individuos de la población (vecindad). El objetivo del operador de mutación es producir nuevas soluciones a partir de la modificación de un cierto número de genes de una solución existente, con la intención de fomentar la variabilidad dentro de la población. En nuestro algoritmo se utiliza el método del intercambio, el cual se ejecuta de acuerdo a la siguiente regla: se genera un número aleatorio entre 1 y la cantidad de routers que se disponen y posteriormente se cambia el valor del gen en dicha posición (si es 0, se cambia por 1 y viceversa). El algoritmo se ejecuta bajo condiciones iguales a la del cruzamiento puesto que también se le debe introducir una probabilidad de mutación.

3.1.7. Parámetros de funcionamiento

El AG posee varios parámetros de funcionamiento, los cuales son:

- Tamaño de la población (TAMPOB)
- Vector de costos de los routers (COSTOS)
- Número máximo de generaciones (MAXGEN)
- Probabilidad de aplicación del operador de cruzamiento (PCRUZA)

- Probabilidad de aplicación del operador de mutación (PMUTA)
- Factor de penalización (PENA)

3.1.8. Funcionamiento general del algoritmo

En líneas generales el algoritmo desarrollado tiene el siguiente funcionamiento:

Procedimiento del Algoritmo
begin
Generar una población inicial de N soluciones aleatorias.
Se genera un vector con las evaluaciones de cada individuo de la población.
repetir hasta MAXGEN

Seleccionar los padres de la población inicial por el método del torneo
Aplicar la función de cruzamiento entre los individuos seleccionados en el paso anterior
Aplicar la función de mutación entre los hijos por cruzamiento
Se actualiza el vector de evaluación con los hijos seleccionados
Los individuos que no resultaron factibles se les aplica la respectiva factor de penalización
Se selecciona al mejor individuo de la generación

Seleccionar al mejor individuo de entre todos los obtenidos en cada una de las generaciones.
end

3.2. Resultados Experimentales

En esta sección mostramos los resultados obtenidos en la ejecución del algoritmo bajo ciertas condiciones constantes y otras variables. Para ejecutar el algoritmo debemos inicializar el programa **MatLab** y ejecutar el archivo **genetico.m** en el cual se deben introducir los valores correspondientes. Como constantes tomamos lo siguiente:

TAMPOB: 110
COSTOS: (28 29 21 30 26 21 23 26 30 30 21 30 30 25)
PCRUZA: 0.87
PMUTA: 0.09
PENA: 30

A continuación se presenta un gráfico de cómo se comporta el algoritmo a medida de que se aumenta el número de generaciones en relación con la evaluación en la función objetivo.

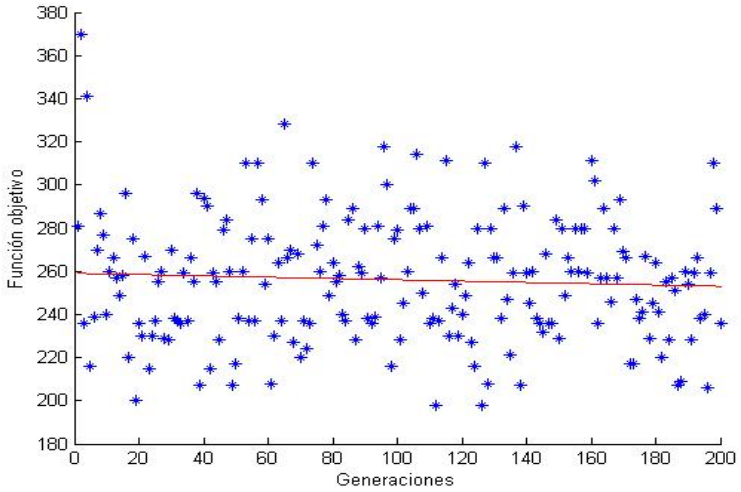


Figura 2: Resultados experimentales

Como información del gráfico se puede observar que el menor valor de la función objetivo es 198 en manos del vector solución (1 0 1 1 1 1 0 1 0 0 1 0 0 1). Realizando un análisis de los resultados obtenidos se tiene que el valor medio de los resultados obtenidos es 256.04, también se puede apreciar una linea de tendencia la cual nos indica que a medida que hagamos mayor cantidad de generaciones mayor probabilidad de obtener el valor óptimo. El vector solución nos indica de que son necesarios para cubrir la ciudad routers que se instalen en los nodos 1, 3, 4, 5, 6, 8, 11 y 14.

4. Programación Dinámica

"La programación dinámica no es un algoritmo. Es más bien un principio general aplicable a diversos problemas de optimización que verifican una cierta propiedad denominada descomponibilidad". Con esta frase queremos empezar a hablar sobre la programación dinámica que tanto se emplea en la rama de la optimización. Primeramente veremos las bases teóricas que sustentan el algoritmo así como una breve descripción se su funcionamiento.

Esta técnica de programación se emplea fundamentalmente en problemas que presentan características como que se puede dividir en **subproblemas optimales:** la solución óptima a un problema puede ser definida en función de soluciones óptimas a subproblemas de tamaño menor; y otra de las características es que permiten un **solapamiento entre subproblemas:** al plantear la solución recursiva del problema, un mismo problema se resuelve más de una vez.

La clave para resolver los diferentes problemas es la memorización que permite almacenar las soluciones de los subproblemas en alguna estructura de datos para reutilizarlas posteriormente. De esa forma, se consigue un algoritmo más eficiente que la fuerza bruta, que resuelve el mismo subproblema una y otra vez.

La programación toma normalmente uno de los dos siguientes enfoques:

- **Top-down:** El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursión.

- **Bottom-up:** Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado.

Cuando hablamos de optimizar nos referimos a buscar alguna de las mejores soluciones de entre muchas alternativas posibles. Dicho proceso de optimización puede ser visto como una secuencia de decisiones que nos proporcionan la solución correcta. Si, dada una subsecuencia de decisiones, siempre se conoce cuál es la decisión que debe tomarse a continuación para obtener la secuencia óptima, el problema es elemental y se resuelve trivialmente tomando una decisión detrás de otra, lo que se conoce como estrategia voraz. En otros casos, aunque no sea posible aplicar la estrategia voraz, se cumple el **principio de optimalidad de Bellman** que dicta que: "dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima". En este caso sigue siendo posible el ir tomando decisiones elementales, en la confianza de que la combinación de ellas seguirá siendo óptima, pero será entonces necesario explorar muchas secuencias de decisiones para dar con la correcta, siendo aquí donde interviene la programación dinámica.

4.1. Problema de la Mochila 0-1

El problema de la mochila, comúnmente abreviado por KP es un problema de optimización combinatoria, es decir, que busca la mejor solución entre un conjunto de posibles soluciones a un problema. Modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo.

En nuestro trabajo tenemos el hecho de que las variables toman valores binarios 0 o 1 dependiendo de que si se emplea o no el router que responde a la variable en cuestión. De modo general se puede plantear el problema como sigue:

$$\text{(KP) Maximizar} \quad \sum_{j=1}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=1}^{n} a_j x_j \leq b$$
$$x_j \in \{0,1\} \qquad j = 1, ..., n$$

donde $a_j \in \mathbb{N} \quad \forall j = 1, ..., n; \quad b \in \mathbb{N}$ y $c_j \in \mathbb{R} \quad \forall j = 1, ..., n.$

Para nuestro trabajo si le realizamos la transformación a un problema de la mochila nos quedaría lo siguiente:
Función objetivo:

$$\max(z) = \sum_{j=1}^{14} -c_j x_j$$

Sujeto a la restricción:

$$-4x_1 - 4x_2 - 4x_3 - 4x_4 - 4x_5 - 4x_6 - 2x_7 - 4x_8 - 3x_9 - 3x_{10} - 3x_{11} - 4x_{12} - 3x_{13} - 4x_{14} \leq -22$$

La idea principal de la programación dinámica consiste en intentar reducir el problema (KP) de n variables en una secuencia de problemas en una sola variable.

En el caso del problema (KP) se considera la siguiente familia de problemas:

$$[KP_i(E)] \qquad \text{Maximizar} \quad \sum_{j=i}^{n} c_j x_j$$

Sujeto a:

$$\sum_{j=i}^{n} a_j x_j + E \leq b$$
$$x_j \in \{0,1\} \qquad j = 1, ..., n$$

con i variando entre 1 y n, y E entero variando entre 0 y b.

La cantidad E se denomina **variable de estado** (o vector de estado cuando la cantidad sea vectorial). El conjunto de los posibles valores de E se denomina **espacio de estados** y se representa por ξ. Se puede observar que el problema original [KP] es un miembro de la familia (es de hecho $[KP_1(0)]$). Se denotará por $F_i^*(E)$ al valor óptimo de $[KP_i(E)]$. El valor óptimo del problema original es $F^* = F_1^*(0)$.

4.2. Diseño del Algoritmo

Primeramente debemos buscar una relación recurrente que conecte los valores óptimos de los diferentes problemas de la familia. Para ello vamos a analizar la familia de problemas $[KP_i(E)]$. Supongamos que en la mochila, al decidir sobre los objetos $j = 1, ..., i-1$, se ha ocupado una capacidad E. Por tanto queda una capacidad $b - E$ para decidir sobre los objetos $j = i, ..., n$.

Etapa n. En $[KP_n(E)]$ se debe decidir si se introduce o no el objeto n. Si ha ocupado ya una capacidad $E > b - a_n$, no puede hacerlo, y por lo tanto $F_n^*(E) = 0$. En caso de que $E \leq b - a_n$, aún existe la posibilidad de introducir el objeto. Lo hará si su beneficio es positivo ($c_n \geq 0$) con lo cual $F_n^*(E) = c_n$, o lo desecharía en caso contrario ($c_n < 0$), con lo cual $F_n^*(E) = 0$.

Etapa n-1. En $[KP_{n-1}(E)]$ debe decidir si se introducen los objetos $n-1$ y n, pero con lo indicado en el apartado anterior se puede simplificar el problema a decidir solamente sobre el objeto $n-1$. Si se introduce dicho objeto, en la etapa n el problema a resolver será $[KP_n(E + a_{n-1})]$, por tanto se obtendrá un beneficio de $c_{n-1} + F_n^*(E + a_{n-1})$, y si no lo introducimos, en la etapa n el problema a resolver será $[KP_n(E)]$ y obtendremos un beneficio de $F_n^*(E)$. Por tanto la decisión se basa simplemente en la siguiente comparación.

$$F_{n-1}^*(E) = \text{Max}\{c_{n-1} + F_n^*(E + a_{n-1}), F_n^*(E)\}$$

Etapa i. Con un razonamiento análogo al de la etapa $n-1$, en una etapa intermedia $i, 1 \leq i \leq n$, tendremos la siguiente relación:

$$F_i^*(E) = \text{Max}\{c_i + F_{i+1}^*(E + a_i), F_{i+1}^*(E)\}$$

Obtención de la solución óptima. Mediante el algoritmo anterior se obtiene el valor óptimo de (KP) pero no la solución óptima x^* explícitamente. Para ello se procede del siguiente modo. Para $i = 1, ..., n-1$

$$x_i = \begin{cases} 0 & \text{si} \quad F_i^*(E) = F_{i+1}^*(E) \\ 1 & \text{si} \quad F_i^*(E) = c_i + F_{i+1}^*(E + a_i) \end{cases}$$

Siendo inicialmente $E = 0$ y actualizando en cada paso el valor de E a $E + a_i x_i^*$. Finalmente,

$$x_n = \begin{cases} 0 & \text{si} \quad F_n^*(E) = 0 \\ 1 & \text{si} \quad F_n^*(E) = c_n \end{cases}$$

4.3. Resultados Experimentales

Para ejecutar el algoritmo debemos inicializar el programa **MatLab** y ejecutar el archivo **dinamica.m** en el cual se deben introducir los valores correspondientes a los beneficios, que en nuestro caso serían los costos de los routers, el vector de pesos y la capacidad de la mochila. Nosotros para la ejecución introducimos los valores siguientes:

BENEFICIOS: (28 29 21 30 26 21 23 26 30 30 21 30 30 25)
PESOS: (4 4 4 4 4 4 2 4 3 3 3 4 3 4)
CAPACIDAD: 22

Una vez ejecutado el algoritmo se obtiene que el valor óptimo para el problema que se discute es 205 y la solución que genera dicho valor es (0 0 1 1 1 0 0 0 0 0 1 1 1 1). Debemos recordar que esta solución corresponde a un problema de la mochila asociado y no a uno de cubrimiento el cual debe cumplir con varias restricciones y en el de la mochila solamente contamos con una: la de la capacidad.

5. Conclusiones

De esta forma podemos plantear que el algoritmo genético no resuelve el problema de manera exacta pero en su ejecución si obtenemos resultados satisfactorios en un tiempo computacionalmente razonable. Los tiempos de ejecución no son muy elevados y en general dependen de la cantidad de generaciones y del número de individuos. Cabe además destacar que los parámetros de funcionamiento al ser variables podemos obtener soluciones para diversas conjugaciones de los mismos.

También no podemos dejar a un lado los resultados obtenidos mediante la aplicación de la programación dinámica como otra vía alternativa para la resolución de problemas de optimización. Con los resultados obtenidos podemos observar que los tiempos de ejecución no son tan elevados como en el algoritmo genético y sin embargo se obtiene un resultado bastante cercano al obtenido en el genético, podemos hacer referencia a la siguiente observación: el valor obtenido en la programación dinámica es mayor que el del genético una vez que se realizó su ejecución con varios parámetros, pero sin embargo es menor al promedio de valores del algoritmo genético. Como a modo de conclusión podemos plantear que tanto el algoritmo genético como el de programación dinámica son una propuesta válida para la resolución de problemas de cubrimiento como es nuestro caso.