5 MÉTODOS DE RESOLUCIÓN DE LOS MODELOS FDH CENTRALIZADOS

En los modelos desarrollados en el capítulo anterior aparecen variables enteras, por lo que los métodos de resolución de programación lineal continua no son aplicables en este caso.

Por tanto, es necesaria la introducción de métodos como la Exploración Dirigida para resolver los modelos FDH centralizados.

Sin embargo, el problema que se nos presenta ahora, es el elevado tiempo de cálculo que se requiere para resolver problemas de gran tamaño. Por ello, se ha implementado además una metaheurística, que resolverá el problema de forma aproximada. Esta técnica presenta tiempos de resolución mucho menores al método exacto, y no importa si el tamaño del problema es elevado. La metaheurística elegida es el Algoritmo Genético.

La resolución del problema FDH centralizado la vamos a realizar desde dos puntos de vista. El primero consistirá en no permitir un empeoramiento de las unidades productivas en beneficio del conjunto y en el segundo caso tanto las entradas como las salidas podrán empeorar para conseguir un mejor resultado global.

El software utilizado para resolver los modelos de programación diseñados en el proyecto ha sido el XA.

Para la implementación del Algoritmo Genético y de su interfaz gráfica con el usuario, se han empleado dos lenguajes de programación, el Borland C++ y el Visual Basic.

1 RESOLUCIÓN DE MODELOS PUROS

Con este modelo, buscamos el beneficio del conjunto, sin tener en cuenta las restricciones individuales. Para ello hemos reformulado el modelo, ya que se ha comprobado que esto permite reducir notablemente los tiempos de resolución del problema.

Anteriormente, en los resultados que obteníamos, se nos indicaba dónde debía proyectarse cada DMU para lograr minimizar los recursos de entrada y maximizar los productos de salida.

Sin embargo, para identificar una solución óptima sólo es necesario saber el número de DMUs que se proyectan sobre cada unidad eficiente.

Por tanto, nos bastaría con tener un vector con tantas componentes como unidades eficientes existan en el problema. En cada una de sus componentes almacenaremos el número de unidades productivas que se proyectan sobre dicha unidad eficiente. Vemos que la suma de todas las componentes del vector, nos daría como resultado el número de unidades productivas total del problema.

Matemáticamente:

$$[n_1, n_2, \ldots, n_E]$$

$$\text{Con } \sum_{e=1}^{E} n_e = n$$

Donde E sería el número de unidades eficientes, n_e sería el número de DMUs que se proyectan sobre cada DMU eficiente y 'n' es el número total de unidades productivas que tenemos.

Los subapartados que se incluyen a continuación, parten de esta reflexión para desarrollar los modelos de programación que resuelven el caso de entradas y salidas puras en los escenarios de orientación de entrada y de salida.

1.1 Orientación de entrada

Sea:

$$D(r) = \{j : \exists i' : x_{i'j} < x_{i'r} \ con \ x_{ij} \le x_{ir} \ \forall i \ne i' \ y \ y_{kj} \ge y_{kr} \ \forall k \ y/ó$$
$$\exists k' : y_{k'j} > y_{k'r} \ con \ y_{kj} \ge y_{kr} \ \forall k \ne k' \ y \ x_{ij} \le x_{ir} \ \forall i\}$$

el conjunto de unidades que domina a DMU_r. Aquellas DMUs que presenten un conjunto D(r) vacío, serán identificadas con unidades eficientes.

Nos basaremos en el modelo FDH Centralizado puro con orientación de entrada, que repetiremos aquí por comodidad:

$$Minimizar \quad \theta$$

$$s.a. \qquad \sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} x_{ij} \le \theta \sum_{r=1}^{n} x_{ir} \quad \forall i$$

$$\sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} y_{kj} \ge \sum_{r=1}^{n} y_{kr} \quad \forall k$$

$$\sum_{j=1}^{n} \lambda_{jr} = 1 \quad \forall r$$

$$\lambda_{jr} \in \{0,1\}$$

$$\theta \quad libre$$

Los términos λ_{jr} sólo podrán valer uno cuando estemos sobre una DMU eficiente, por lo que podemos simplificar la primera restricción dejándola como sigue:

$$\sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} x_{ij} \quad \text{-----> } \quad \sum_{r=1}^{n} \sum_{e=1}^{E} \lambda_{er} x_{ie}$$

Esto se corresponde con la suma de las entradas de las proyecciones correspondientes al recurso i. Por ello podemos reescribirla como la suma de los recursos de cada entrada i utilizado por cada unidad eficiente, multiplicado por el número de veces que el resto de unidades se proyecta sobre dicha unidad eficiente.

$$\sum_{e=1}^{E} x_{ie} n_e$$

De forma similar, se hacen las mismas consideraciones para la segunda restricción, con lo que resulta:

$$\sum_{e=1}^{E} y_{ke} n_e$$

Como restricción adicional, añadimos que la suma de todos los elementos del vector resultado, ha de ser igual al número total de DMUs existentes:

$$\sum_{e=1}^{E} n_e = n$$

Por tanto el modelo quedaría como sigue:

$$Minimizar \quad \theta$$

$$s.a. \qquad \sum_{e=1}^{E} x_{ie} n_e \leq \theta \sum_{r=1}^{n} x_{ir} \qquad \forall i$$

$$\sum_{e=1}^{E} y_{ke} n_e \geq \sum_{r=1}^{n} y_{kr} \qquad \forall k$$

$$\sum_{e=1}^{E} n_e = n$$

$$n_e \quad enteras$$

$$\theta \quad libre$$

Vemos que este modelo presenta m+p+1 restricciones, que es menor o igual que las m+p+n restricciones que teníamos anteriormente. También ha disminuido el número de variables, de las n^2 que teníamos anteriormente, a las E+1 que tenemos ahora.

El obstáculo que podemos tener a la hora de resolver el problema, es que las variables n_e son enteras, y que directamente no pueden ser procesadas por el XA. Para solucionar esto, hemos introducido variables binarias auxiliares que nos permitirán tratar a los componentes del vector solución como variables libres en el proceso de resolución, aunque en la solución final, sólo tomarán valores enteros.

Matemáticamente podemos expresar estas variables binarias designadas como X_{SE} como sigue:

$$n_e = \sum_{s} 2^{s-1} \cdot X_{se}$$

Con todas estas simplificaciones realizadas, podemos lograr con el XA un resultado en un tiempo razonable y mucho más inferior al del modelo inicial, aún cuando el problema de entrada sea de tamaño elevado.

1.2 Orientación de salida

Comenzamos de nuevo calculando las unidades eficientes a partir del conjunto siguiente:

$$D(r) = \{j : \exists i' : x_{i'j} < x_{i'r} \ con \ x_{ij} \le x_{ir} \ \forall i \ne i' \ y \ y_{kj} \ge y_{kr} \ \forall k \ y/ó$$
$$\exists k' : y_{k'j} > y_{k'r} \ con \ y_{kj} \ge y_{kr} \ \forall k \ne k' \ y \ x_{ij} \le x_{ir} \ \forall i\}$$

A partir del modelo centralizado puro con orientación de salida, que exponemos a continuación:

$$Maximizar \quad \gamma$$

$$s.a. \qquad \sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} x_{ij} \le \sum_{r=1}^{n} x_{ir} \qquad \forall i$$

$$\sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} y_{kj} \ge \gamma \sum_{r=1}^{n} y_{kr} \qquad \forall k$$

$$\sum_{j=1}^{n} \lambda_{jr} = 1 \quad \forall r$$

$$\lambda_{jr} \in \{0,1\}$$

$$\gamma \quad libre$$

Nuevamente, los términos λ_{jr} sólo podrán valer uno cuando estemos sobre una DMU eficiente, por lo que podemos simplificar la primera restricción dejándola como sigue:

$$\sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} x_{ij} \quad \text{--------} > \quad \sum_{r=1}^{n} \sum_{e=1}^{E} \lambda_{er} x_{ie}$$

$$\sum_{r=1}^{n} \sum_{j=1}^{n} \lambda_{jr} y_{kj} \quad \text{--------} > \quad \sum_{r=1}^{n} \sum_{e=1}^{E} \lambda_{er} y_{ke}$$

Podemos reescribir esto sustituyendo la primera restricción por la suma de los recursos de cada entrada i utilizado por cada unidad eficiente, multiplicado por el número de veces que nos proyectamos sobre dicha unidad eficiente. La segunda restricción sería igual a la suma de los productos de la salida k generada por cada DMU eficiente multiplicado por el número de veces que nos proyectamos sobre ella.

Por lo tanto quedaría como:

$$\sum_{e=1}^{E} x_{ie} n_e$$

$$\sum_{e=1}^{E} y_{ke} n_e$$

Por último añadiríamos la restricción de que la suma de todos los elementos del vector resultado, ha de ser igual al número total de DMUs que tengo:

$$\sum_{e=1}^{E} n_e = n$$

El modelo final quedaría como:

$$Maximizar \quad \gamma$$

$$s.a. \qquad \sum_{e=1}^{E} x_{ie} n_e \le \sum_{r=1}^{n} x_{ir} \qquad \forall i$$

$$\sum_{e=1}^{E} y_{ke} n_e \ge \gamma \sum_{r=1}^{n} y_{kr} \qquad \forall k$$

$$\sum_{e=1}^{E} n_e = n$$

$$n_e \quad enteras$$

$$\gamma \quad libre$$

Al igual que antes, tenemos m+p+1 restricciones, que es menor o igual que las m+p+n restricciones que teníamos anteriormente. También ha disminuido el número de variables.

Otra vez definiremos las variables binarias auxiliares como en el caso de orientación de entrada:

$$n_e = \sum_{s} 2^{s-1} \cdot X_{se}$$

2 RESOLUCIÓN DE MODELOS HÍBRIDOS

Anteriormente hemos formulado el modelo matemático de estos modelos híbridos.

En este apartado, se explicarán las dos metodologías que se han desarrollado para resolverlos.

2.1 Método exacto

En el modelo FDH Centralizado, la reasignación de los recursos utilizados para la mejora total del sistema, convierte el problema en combinatorio, ya que se

tendrán que explorar todas las posibles proyecciones que pueden realizarse, con el fin de minimizar la suma de recursos consumidos o bien de maximizar la suma de los productos obtenidos.

En la aplicación que hemos desarrollado, se implementan los modelos tanto de orientación de entrada como de salida, así como todos los datos de entrada que el XA necesita y en el formato adecuado. El fichero generado será enviado a la librería XA, que resolverá nuestro problema mediante Exploración Dirigida.

Este método de resolución nos brinda buenos resultados en problemas pequeños, ya que emplea un tiempo razonable en su resolución. Sin embargo para problemas de mayor tamaño, aumenta la complejidad computacional y los tiempos empleados son demasiado elevados.

2.2 Método aproximado

Dada la complejidad de ciertos problemas de elevado tamaño, es inviable resolverlos con el método exacto, debido a los elevados tiempos de resolución que ello conllevaría, como ya se ha comentado anteriormente.

Por ello, se ha implementado un Algoritmo Genético, que permite llegar a una buena solución a nuestro problema, en un tiempo de resolución razonable.

En el apartado siguiente, haremos una introducción general del Algoritmo Genético, y posteriormente, explicaremos su particularización al problema FDH Centralizado.

2.2.1 Introducción al Algoritmo Genético

John Holland fue el precursor de los Algoritmos Genéticos. Su observación de la naturaleza, le llevó a preguntarse como ésta logra "crear" seres cada vez más adaptados al entorno. Esto se consigue, a base de interacciones locales entre individuos, y entre éstos y el espacio que les rodea.

Holland aprendió, que la evolución es una forma de adaptación mucho más potente que el simple aprendizaje, y así fue como decidió basarse en la naturaleza, para aplicar este método a otros procesos de optimización.

Holland centró sus investigaciones en dos campos:

- Estudiar y tratar de imitar los procesos adaptativos de la naturaleza
- Diseñar sistemas artificiales que funcionasen del mismo modo que los sistemas naturales.

Quince años después, David Goldberg, ingeniero industrial y alumno de Holland, consiguió aplicar los algoritmos genéticos, en la resolución de complejos problemas industriales.

Un Algoritmo Genético es un método sistemático, para la resolución de problemas de búsqueda y optimización, que aplica a éstos, los mismos métodos de la evolución biológica.

Estos métodos son:

- La selección basada en la población
- El cruce o reproducción sexual
- La mutación

Los algoritmos genéticos, son métodos de optimización para la resolución de problemas, es decir, hallan (x1,...,xn), tales que F(x1,...,xn) que será la función objetivo, sea máxima.

Partiremos de un conjunto de individuos; cada uno de ellos contiene una posible solución de nuestro problema. Al conjunto de todos los individuos se les llama población.

Una vez que hemos parametrizado el problema en una serie de variables (x1,...,xn), las codificamos en un cromosoma. Por tanto cada individuo estará representado por un cromosoma.

Dicho cromosoma a su vez está compuesto de distintos genes, que serán los distintos parámetros considerados en el problema.

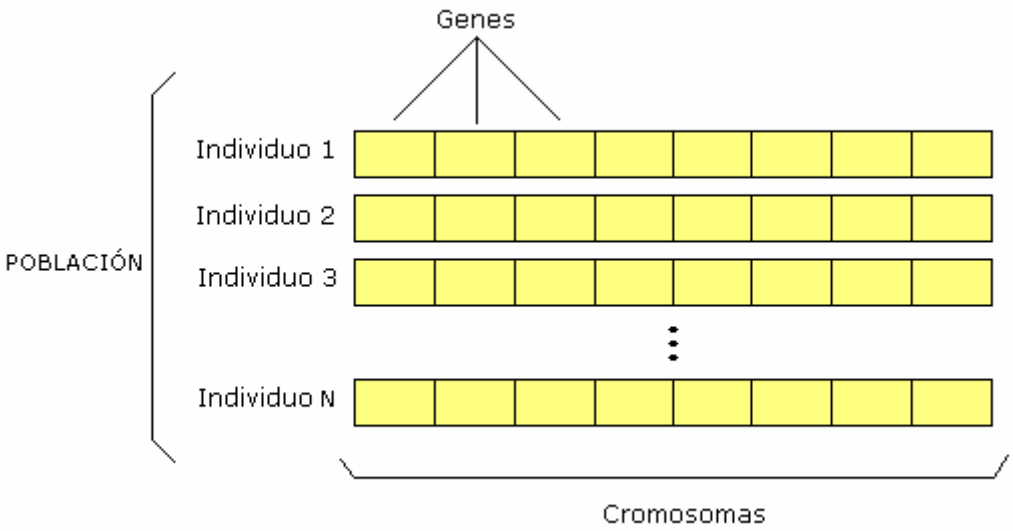


Ilustración 3: Esquema de la codificación del Algoritmo Genético

Para poder trabajar con estos genes en el ordenador, codificaremos cada parámetro como un número entero de bits.

Por tanto, partimos de un conjunto de posibles soluciones (cromosomas), y aplicando ciertos operadores sobre ellas, pretendemos encontrar la mejor de todas las soluciones posibles (que no tiene por qué ser necesariamente la mejor de todas las que existen).

Es decir, las soluciones codificadas en un cromosoma, "compiten" para ver cual de ellas es la mejor. El "entorno" formado por el resto del conjunto de posibles soluciones, ejercerá una presión selectiva sobre la población, de forma que sólo los mejor adaptados (aquéllos que resuelvan mejor el problema), conseguirán sobrevivir y legar su material genético a la siguiente generación. Esto mismo ocurre en la evolución de las especies.

Para conseguir la diversidad genética, emplearemos ciertos operadores sobre los cromosomas, como la mutación o el cruce. Existen otros operadores, pero nos limitaremos a usar estos dos que son los más importantes.

Por tanto, podemos enumerar los pasos básicos de cualquier algoritmo genético, que son:

- Ver de qué parámetros depende el problema
- Codificar estos parámetros en un cromosoma
- Aplicar los operadores genéticos a mis cromosomas, a través del intercambio de información (cruce), y de alteraciones que generen diversidad (mutación)
- Observando los resultados, seleccionar el individuo que mejores características presente

Los operadores genéticos que emplearemos son:

- Cruce: consiste en tomar a dos individuos de la población y reproducirlos. Con ello conseguimos que ambos "padres", leguen su material genético. Como resultado obtendremos un "hijo".
- Mutación: consiste en tomar uno de los genes de un individuo cualquiera y mutarlo, es decir, cambiar su valor aleatoriamente por otro.

En la naturaleza, sabemos que la mutación es un suceso poco común (sucede aproximadamente en una de cada mil replicaciones) y que en la mayoría de los casos es letal.

Sin embargo, en promedio, contribuye a la diversidad de la especie, por lo que, en los algoritmos genéticos, este operador tendrá también una probabilidad de ocurrencia, a pesar de que ésta será muy baja.

Estos operadores se aplicarán sucesivamente sobre nuestros individuos, es decir sobre los bits que representan a nuestros genes.

Además de estos operadores, existen otros, pero no los consideraremos en nuestro caso.

Es necesario además definir otros dos parámetros:

- Tamaño de la población: ésta debe de tener un tamaño adecuado para garantizar la diversidad de las soluciones.

- Condición de terminación: Estableceremos un número de iteraciones, tras las cuales consideraremos que hemos obtenido una buena solución para nuestro problema. Otra opción posible, sería iterar hasta llegar a la convergencia del algoritmo. En nuestro caso estableceremos un límite de iteraciones, llegado el cual, obtendremos el resultado a nuestro problema.

Cada individuo de la población, tendrá unas características distintas del resto. En función de la bondad o no de dichas características, este individuo será mejor o peor. Para expresar numéricamente esta situación, surge el concepto del fitness.

El fitness podríamos decir que es una medida de lo bueno o malo que es ese individuo. Es la "puntuación" de cada individuo. Indudablemente, para mejorar la especie, interesa que los individuos mejores permanezcan, y se eliminen aquéllos que presenten peores características.

Por tanto, para cada uno de los individuos que conforman nuestra población, hallaremos su fitness. Ésta será la función objetivo a maximizar. Por tanto, a mayor puntuación, mayor posibilidad de permanencia de ese individuo en la población, ya que se encontrará más cerca de la solución del problema.

El fitness va a determinar qué individuos van a reproducirse, y qué otros van a ser eliminados.

Queremos que los "buenos" rasgos de nuestra población, se sigan transmitiendo. Habría muchas formas de hacer esto. A continuación enumeramos las principales:

- Basado en el rango: en este caso se mantiene un porcentaje de la población, generalmente la mayoría, para la siguiente generación. Se ordenará toda la población por orden de fitness, y los M con menor puntuación, serán eliminados y sustituidos por la descendencia de alguno de los N mejores.

- Una variante de esto sería, que al efectuar un cruce, sustituyamos este descendiente por el individuo más parecido al mismo de entre los perdedores. A esto se le denomina Crowding.

- Otra variante sería el muestreado estocástico universal, que intenta evitar que los individuos con mayor fitness copen la población.

- Selección de torneo: se escogen aleatoriamente un número T de individuos de la población y se reproducen. La descendencia de éstos, sustituirá al individuo que menor fitness tenga de la población.

Existen muchas otras posibilidades, según las condiciones por las que nos rijamos a la hora de elegir a los individuos que deberán mutar o cruzarse. Y también según que criterio seleccionemos a la hora de elegir un individuo para ser eliminado de nuestra población y sustituido por otro.

2.2.2 Caso particular de estudio: los modelos FDH Centralizados

A continuación, se explica el programa implementado para resolver el problema de forma aproximada mediante el diseño de un Algoritmo Genético.

En primer lugar, se halla la solución tradicional al problema aplicando el modelo FDH. Éste será el primer individuo de la población de la primera generación. Una vez realizado esto, se generan el resto de individuos de la población aleatoriamente.

La solución tradicional se caracteriza porque siempre tendremos que mantener o disminuir los recursos disponibles de entrada (en orientación de entrada), o que mantener o aumentar la producción de salida (en orientación de salida).

En el caso aquí tratado, la solución tradicional que hallamos, considerará todas las entradas y todas las salidas tradicionales. Por tanto, en ninguno de los

dos casos podremos empeorar. Se sabe que esta es una buena solución, así que hemos decidido tomarla como solución de partida. Con esto garantizamos que vamos a mejorar necesariamente, ya que partimos de una buena solución inicial.

Por tanto, se halla la solución tradicional y posteriormente se genera aleatoriamente el resto de individuos que conformarán la población.

Para ello, lo primero que haremos, es analizar qué tipo de entradas o de salidas tiene el problema a resolver, es decir, ver si éstas son tradicionales o centralizadas. En función de dicho estudio, generaremos una matriz proyección, que indicará sobre qué DMUs eficientes puede proyectarse la DMU original.

Aplicando esta matriz, generaremos aleatoriamente una serie de individuos, que someteremos a dos comprobaciones previas. La primera de ellas será, que el individuo no esté repetido, y la segunda consistirá en verificar que cada individuo sea admisible. Es decir, debe verificarse que la suma parcial de entradas, considerando cada entrada por separado, debe ser menor o igual que la suma de entradas original, y que la suma parcial de salidas, considerando cada salida por separado, debe ser mayor o igual que la suma de las salidas de las que parto originalmente.

Si alguno de los individuos estuviera repetido o fuera no admisible, lo desecharíamos y generaríamos otro también de forma aleatoria. Con este nuevo individuo, volveríamos a verificar las condiciones de no repetibilidad de individuos, así como de admisibilidad.

Si no pudieran obtenerse individuos admisibles en un tiempo razonable, se le indica al usuario que la dimensión del problema es posiblemente muy reducida.

Una vez que se obtiene una población admisible, calculamos su fitness y ordenamos la población en función de este valor. Distinguiremos aquí dos casos, en función de si trabajamos con orientación de entrada o de salida.

- Orientación de entrada: para cada individuo, hallamos las reducciones parciales para cada una de sus entradas. La θ final será la mayor de estas reducciones parciales. A partir de las θ finales de todos los individuos, ordenaremos a la población de menor a mayor θ.

- Orientación de salida: para cada individuo, hallamos las amplificaciones parciales para cada una de sus salidas. La γ final será la menor de estas amplificaciones parciales. A partir de las γ finales de todos los individuos, ordenaremos a la población de mayor a menor γ.

En ambos casos, será el primer individuo el que presente mejores características.

Una vez hecho esto, son aplicados los operadores genéticos a la población. Por tanto, se realizarán cruces y mutaciones a los individuos, todo ello siguiendo un patrón aleatorio.

Como la probabilidad de mutación en la naturaleza es menor, hemos reflejado esto también en el algoritmo, otorgándole una probabilidad de ocurrencia del 20%, respecto al 80% que presenta el cruce.

Para nuestro caso particular, cuando realicemos un cruce, seleccionaremos aleatoriamente a dos individuos, y los cruzaremos creando un hijo. Seguidamente seleccionaremos aleatoriamente a otro individuo de la población y lo eliminaremos, poniendo en su lugar al hijo anteriormente generado.

Del mismo modo, cuando realicemos una mutación, elegiremos a un individuo al azar, y seleccionaremos a uno de sus genes al azar y lo mutaremos. Posteriormente, seleccionaremos aleatoriamente a otro individuo de la población, y lo eliminaremos, sustituyendo éste por el individuo que hemos mutado. Éste será en definitiva el procedimiento que seguiremos.

Por supuesto, todo esto estará sujeto a la comprobación previa de que ese individuo lleva codificada en sus genes una solución válida a nuestro problema, es decir, que esté contenida en nuestro espacio de soluciones. Por lo tanto, a cada nuevo individuo generado por cruce o mutación, se le vuelve a aplicar todo el proceso descrito anteriormente. Nuevamente, si el individuo generado no cumpliera con las condiciones establecidas, lo desecharíamos, y volveríamos a repetir el cruce o la mutación. Si el individuo cumple todas las condiciones, entonces pasaría a sustituir a otro individuo de la población que eliminaríamos aleatoriamente. Para concluir, ordenaríamos a los individuos según su fitness.

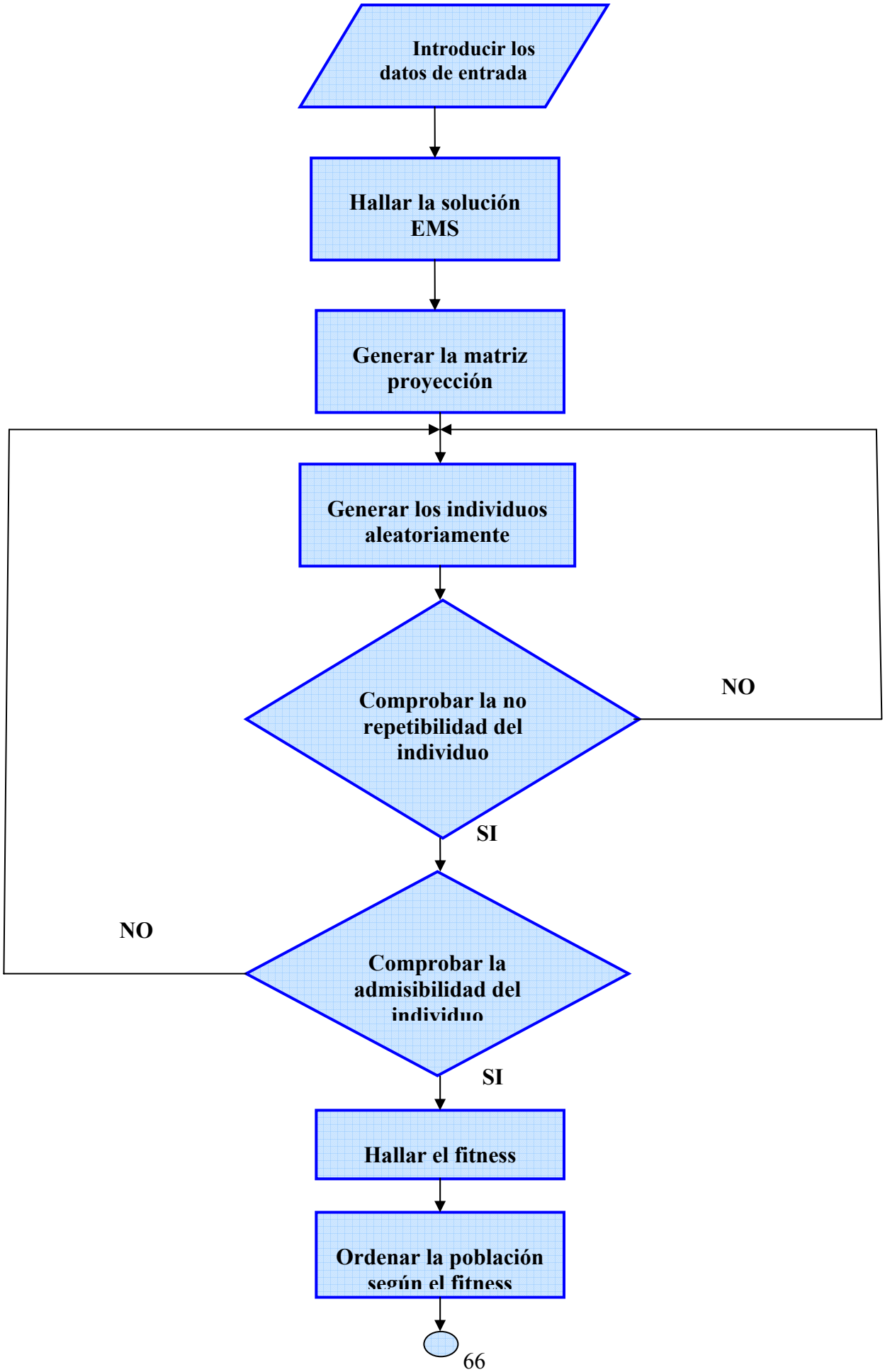
Hay que repetir el proceso hasta que se cumpla un criterio de parada. Para este proyecto se ha diseñado uno consistente en que cuando la solución óptima no es mejorada durante un cierto número de iteraciones (a fijar para cada problema), el algoritmo finaliza.

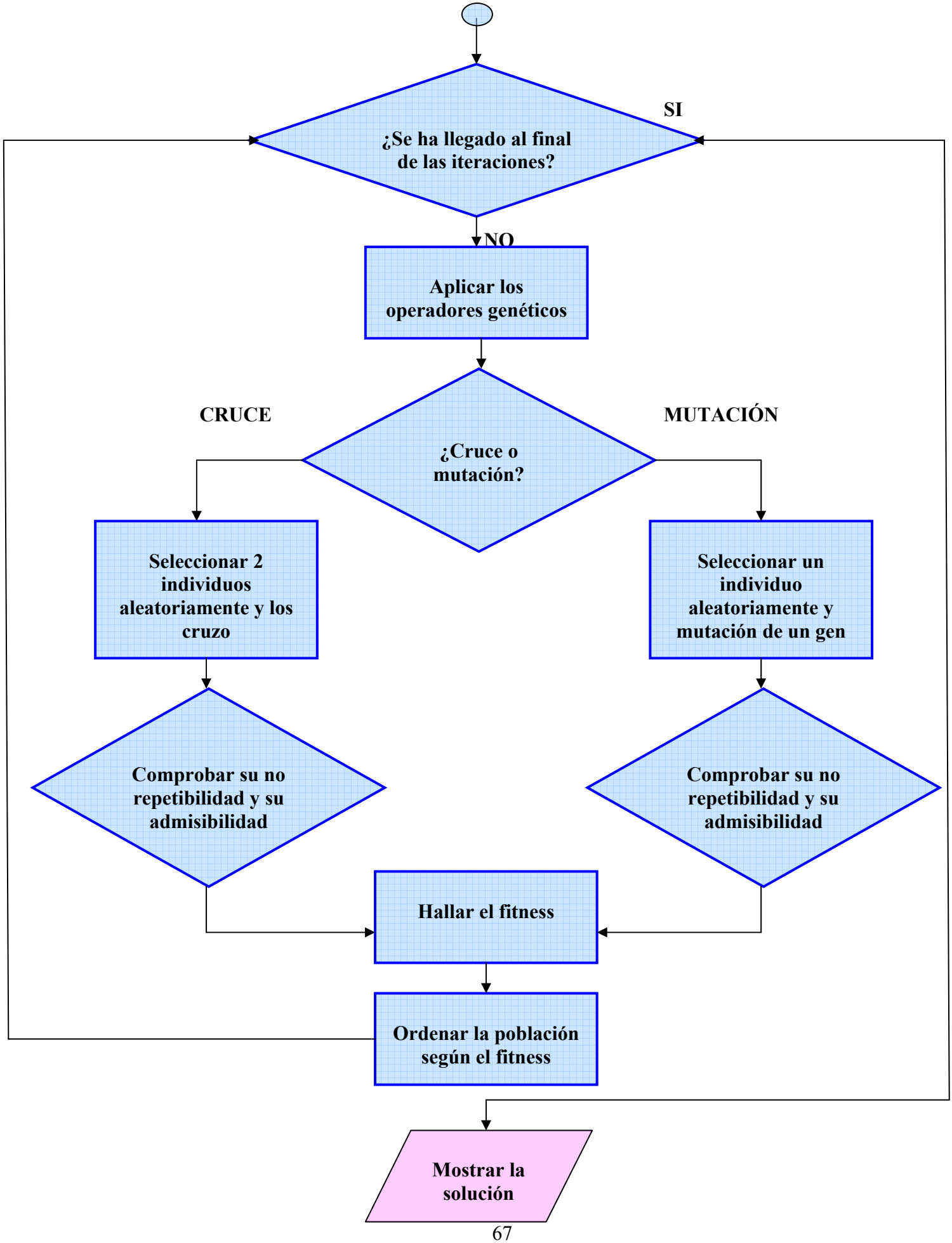
Al final de todo este proceso, la solución del problema será el primer individuo de la población final, que será aquel que presenta las mejores características. Esto es debido a que, en el proceso de selección del individuo que desaparece de una generación a otra, se ha excluido sistemáticamente al mejor individuo de cada iteración.

2.2.3 Diagrama de flujo

A continuación mostramos el diagrama de flujo del algoritmo implementado.

Ilustración 4: Diagrama de flujo del algoritmo implementado


Introducir los datos de entrada
Hallar la solución EMS
Generar la matriz proyección
Generar los individuos aleatoriamente
Comprobar la no repetibilidad del individuo
NO
SI
Comprobar la admisibilidad del individuo
NO
SI
Hallar el fitness
Ordenar la población según el fitness


¿Se ha llegado al final de las iteraciones?
SI
NO
Aplicar los operadores genéticos
¿Cruce o mutación?
CRUCE
MUTACIÓN
Seleccionar 2 individuos aleatoriamente y los cruzo
Seleccionar un individuo aleatoriamente y mutación de un gen
Comprobar su no repetibilidad y su admisibilidad
Comprobar su no repetibilidad y su admisibilidad
Hallar el fitness
Ordenar la población según el fitness
Mostrar la solución