Problemas.

2

Problema de selección de conjuntos de tamaño m. *El problema de la selección de un conjunto de tamaño fijo, de m elementos, a partir de un subconjunto de tamaño mayor n, consiste en seleccionar el subconjunto de elementos que cumplan con el óptimo asociado a una función objetivo que nos permita conocer la sinergia positiva o beneficio entre los elementos seleccionados. Utilizamos la matriz B, tal que B(i,j) nos mide el beneficio de seleccionar los ejemplos i y j conjuntamente.*[1]

Lo primero que vamos a hacer es anotar los datos que se nos dan en el enunciado. Entonces tendríamos:

n elementos.
m seleccionados. (El conjunto de todos los elementos m seleccionados hacen M.
$B(i,j)$ beneficio $i,j \in M$.

Consejo: antes de comenzar a abarcar cualquiera de los apartados sería conveniente estudiar unas cuantas representaciones de modo que nos sea más fácil realizar el ejercicio entero –algunas representaciones nos pueden complicar mucho algunos apartados–. De modo que antes de empezar leer con detenimiento todos los apartados.

Vamos a anotar unas cuantas **representaciones**:

1. **Permutación**. $\{v_1, \dots, v_m, \dots, v_n\}$ Donde tenemos los elementos seleccionados desde v_1 hasta v_m.
2. **Vector binario**. $\{x_1, \dots, x_n\} \mid x_i \in \{0,1\}$
   sujeto a: $\sum x_i = m$
3. **Vector de tamaño m**. $\{y_i, \dots, y_m\} \mid y_j \in \{1, \dots, n\}$
   Teniendo en cuenta que tiene que haber m elementos y no pueden repetirse: $y_i \neq y_j \; \forall \; i,j \in \{1, \dots, m\} \; i \neq j$

Y ahora vamos a ver para cada representación la **función objetivo** asociada:

1. **Permutación**.
$$f(v) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} B(v_i, v_j) \mid v_i, v_j \in M$$
2. **Vector binario**.
$$f(x) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} B(x_i, x_j) \cdot x_i \cdot x_j - \boldsymbol{p\left(x, \sum x_i \neq m\right)}$$
   Aquí hemos de introducir una penalización, para que cumpla la restricción indicada de que necesitamos m elementos, sino podríamos tener una solución no valida.
3. **Vector de tamaño m**.
$$f(y) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} B(y_i, y_j)$$

[1] Una vez con esto anotado y en mente comenzamos a lidiar con los problemas, propiamente dichos. (Importante: en el examen **SOLO ES NECESARIO** 1 de las respuestas, 1 representación, 1 función objetivo,..., se exponen varias soluciones para ilustrar mejor la resolución de los mismos).

a) ¿Qué algoritmo entre los vistos en la asignatura crees que serían el más adecuado para resolver este problema?
Da la lista de algoritmos más adecuados y justifica la respuesta. Explica las ventajas e inconvenientes de su elección.
Dar los elementos básicos para resolverlo con un algoritmo que selecciones para ello (representación, evaluación, elementos necesarios).

De entre los algoritmos vistos en clase tenemos:

1. **Simulated Annealing**:
   Cualquiera de las tres representaciones sería válida y simple de aplicar.
2. **Tabú Search**:
   Tenemos también una buena opción para esta técnica cualquiera de las 3 representaciones.
3. **Algorítmos Genéticos**:
   El problema que encontramos en estos algoritmos son los operadores, puesto que serían más complejos de implementar porque las representaciones no dan mucha información; y no dan mucha información porque en este problema no importa el orden de los elementos. Por ejemplo para la permutación, da igual tener la solución $\{1,2,3,4_{\mathrm{m}},5,6,7,\} = \{2,1,3,4_{\mathrm{m}},6,7,5\} = \{2,4,3,1_{\mathrm{m}},5,6,7\}$ …
   Mientras tengamos a la izquierda $\{1,2,3,4\}$ son esos los que vamos a seleccionar, de modo que nos importa seleccionar primero el $1,2,3$ o 4 indistintamente.

En este problema, los algoritmos de trayectorias serían las mejores técnicas. Y por simplicidad la técnica **simulated annealing**, porque tenemos representaciones poco informativas. El problema de elegir **algoritmos genéticos** es que los operadores a realizar serían muy complejos y muy específicos.

Elementos básicos para **simulated annealing**.
En cuanto a los elementos necesarios a especificar en este problema tendríamos que definir muy pocos para **simulated annealing**, y unos cuantos más para **tabú search** y para **algoritmos genéticos**.
De hecho para **simulated annealing** tenemos solo que especificar el operador de vecino:

1. **Permutación**.
   Vecino {
   a = random(1, m)
   b = random(m+1, n)
   Intercambiar(aleatorio1, aleatorio2)
   }
2. **Vector binario**.
   Vecino {
   Hacer{
   a = random(1, m)
   b = random(m, n)
   }mientras($v_a == v_b$)
   Intercambiar(aleatorio1, aleatorio2)
   }
   Se hace la condición: $v_a == v_b$ para que se cumpla la restricción indicada. De modo que si $v_a = 0, v_b = 1 \rightarrow v_a = 1, v_b = 0$ así el número de 1 y 0 es constante.

3. **Vector de tamaño m**.
   Vecino {
   a = random(1, m)
   b = random(1, n-m)
   intercambiar($y_a, elemento_{n-m} de\ la\ lista\ de\ 'elmen'\ que\ no\ están\ en\ m$)
   }
   En esta representación se tendrían dos vectores un de tamaño m con los elementos seleccionados y otro de tamaño $\mathrm{n}-\mathrm{m}$ con los elementos no seleccionados.

b) Considera que se utiliza un algoritmo genético. Formula la representación de este problema, su función objetivo, y los operadores genéticos necesarios para utilizar un algoritmo genético sobre este problema. En cada caso, describe su formulación e indica un ejemplo de su funcionamiento.

En cuanto a la representación podríamos elegir dos representaciones:

1. **Permutación**.
   a. Y tendríamos el operador de cruce: el operador OX.
   b. Operador de mutación: Seleccionar dos elementos aleatorios, 1 elemento de los que están a la izquierda del elemento m y otro de los que están a la derecha (los no seleccionados) e intercambiar.
2. **Binaria**.
   a. Operador de cruce: operador HUX –una variante del operador del algoritmo CNC, fijando las posiciones comunes con el 1 y 0, y con probabilidad $1/2$ cada posición toma el valor 1 uno de los hijos. Cuando uno de ellos tenga m valores con 1, entonces el resto de valores 1 son asignados al otro hijo – teniendo cuidado con la restricción, (Hemos de tener cuidado con el elemento m).
   b. Operador de mutación: intercambiar dos valores que sean diferentes (0 y 1).

Nota: Quedaría dar el ejemplo.

5

***Problema de separación de una muestra en 2 subconjuntos**. Se dispone una balanza con dos platillos y de n objetos, cada uno de los cuales tiene un peso positivo. El objetivo es encontrar un reparto de los objetos entre los dos platillos de la balanza de forma que la diferencia entre los pesos de los objetos situados en cada platillo sea mínima.*

Aquí tenemos:
n objetos.
m elementos en una balanza.
$n - m$ elementos en la otra balanza.

Se nos pide:

$$\min \left| \sum_{i=1}^{m} p_i - \sum_{j=1}^{n-m} p_j \right|^2$$

a) ¿Qué algoritmo entre los vistos en la asignatura crees que serían el más adecuado para resolver este problema? Da la lista de algoritmos más adecuados y justifica la respuesta. Explica las ventajas e inconvenientes de su elección. Dar los elementos básicos para resolverlo con un algoritmo que selecciones para ello (representación, evaluación, elementos necesarios).

En esta ocasión, podríamos realizar el problema con cualquiera de los algoritmos vistos en clase. La respuesta aquí podría ser muy variada y todas ellas correctas, de modo que tanto **simulated annealing**, como **tabú search**, como **algoritmos genéticos** serían adecuados. De hecho, con una representación binaria, en algoritmos genéticos tendríamos mucho juego, y quizás podríamos decantarnos por esa, pero aun así, todas serían válidas.

Si tuviésemos que distinguir entre **simulated annealing** y **tabú search** por simplicidad podríamos escoger la primera opción, puesto que sería más difícil proponer una lista tabú en este caso de representación binaria.

Incluso una **ILS** con **simulated annealing** también sería una buena opción.

Para la representación como hemos dicho, podríamos elegir una representación binaria: $x = \{x_1, \dots, x_n\} \mid x_i \in \{0, 1\}$ / donde $0 \rightarrow balanza\ 1\ y\ 1 \rightarrow balanza\ 2$.
En cuanto a la función objetivo tendríamos:

$$f(x) = \left| \sum_{i=1}^{n} x_i p_i - \sum_{i=1}^{n} (1 - x_i) p_i \right|$$

Y tendríamos que minimizar esa expresión, donde se va contando el peso de cada elemento, si está en la balanza 2 cuenta la sumatoria de la izquierda, si está en la balanza 1 cuenta la sumatoria de la derecha.

b) Considera que se utiliza un algoritmo genético. Formula la representación de este problema, su función objetivo, y los operadores genéticos necesarios para utilizar un algoritmo genético sobre este problema. En cada caso, describe su formulación e indica un ejemplo de su funcionamiento.

Con algoritmos genéticos tendríamos una solución muy simple con la representación binaria:

1. Operador de cruce: cruzar por 1 punto.
2. Operador de mutación: Cambiar el valor en 1 punto.

Nota: Quedaría dar el ejemplo.

[2] Nota: Cada sumatoria estaría asociada a un platillo.

7

***Problema de la mochila**. Se dispone una mochila y un conjunto de n objetos, cada uno de los cuales tiene un peso positivo y un beneficio. El objetivo el conjunto de objetos con peso menor a la capacidad de la mochila y mayor beneficio.*

En este problema podríamos optar por dos representaciones:

1. **Vector binario**. La más común, solo que en esta representación tenemos que arrastrar la restricción en cada operador.
2. **Permutación**. Esta representación sería más sutil, de modo que tendríamos: $\{y_1, \dots, y_m, \dots, y_n\}$ donde los elementos que quedan a la izquierda de m son los cogido, y a la derecha de m los elementos que no están en la mochila.
   De modo que nos quedamos con los primeros m elementos que cumplan:

$$\sum_{i=1}^{m} p_i \leq M$$

$siendo\ M$ el peso de la mochila.

En cuanto al mejor algoritmo, podríamos también, al igual que en el ejercicio anterior, escoger cualquiera de los vistos en clase. Pero por simplicidad podríamos elegir **simulated annealing** porque solo necesitaríamos escoger el operador de vecino que intercambia un elemento que hay a la izquierda de m y otro a la derecha de m. Para la representación de orden sin embargo con los algoritmos genéticos, el operador de cruce: repite muchas soluciones *(Sería lo mismo* $\{1, \dots, 3_m, 6, 7\} = \{3, \dots, 1_m, 7, 6\}$. *Da igual coger el elemento 1 antes que el 3, puesto que si va en la mochila tal elemento, pesa lo mismo meterlo antes que después);* con la representación de orden los **algoritmos genéticos** pueden utilizar el operador de cruce OX y el operador de mutación sería intercambiar dos posiciones.