Inteligencia Artificial

Tarea 4

Docente: Juan Pablo Rosas Baldazo

1735839 Bryan Aguirre Tudon

Alexis Emiliano Eguia Sifuentes

Leonardo Martin Vázquez Cruz

Instrucciones

Programar un algoritmo genético para el problema de mochila. Recuerden que el problema de mochila es aquel donde tratan de saber que productos deben de llevar en una mochila con una capacidad limitada, maximizando el beneficio de los elementos agregados.

Me deberán entregar un reporte con el pseudocódigo de su algoritmo, donde expliquen:

- La búsqueda local que realizaron
- Los porcentajes de mutación y cruce que consideraron explicando por que esos valores.
- El tamaño de la población inicial
- La cantidad de generaciones
- Deberán incluir en su reporte de igual manera gráficas comparativas entre el valor óptimo y el valor encontrado con su algoritmo, explicando porque creen que obtuvieron esos resultados, si tienen alguna idea de como mejorar la solución alcanzada.

El reporte normalmente debe de tener una introducción breve del problema, donde se describe en que consiste.
Una explicación de las instancias que van a utilizar, de donde las obtuvieron y que tan diversas son.
Un resumen de los parámetros de su algoritmo, así como un pseudocódigo, nada de pegar código ni capturas de pantalla. Y una conclusión breve.

0-1 Knapsack Problem

El problema de la mochila (0 – 1 Knapsack Problem) es un problema clásico que consiste en un excursionista que debe preparar su mochila, la cual tiene una capacidad **limitada** y por tanto no le permite llevar todos los artículos que quisiera tener en la excursión. Cada articulo que el excursionista puede incluir en la mochila le reporta una determinada **utilidad**. Luego el problema consiste en seleccionar un subconjunto de objetos de forma tal que se **maximice la utilidad** que el excursionista obtiene, pero **sin sobrepasar la capacidad** de acarrear objetos.

- Se tiene n objetos y una mochila con una capacidad.
- El objeto i tiene un peso w_i y la inclusión de este objeto i en la mochila produce un beneficio v_i.
- El problema de la mochila tiene como objetivo es llenar en si la mochila que tiene una capacidad k, de manera que se maximice el beneficio.

$$Max \sum_{i=0}^{n-1} v_i x_i$$

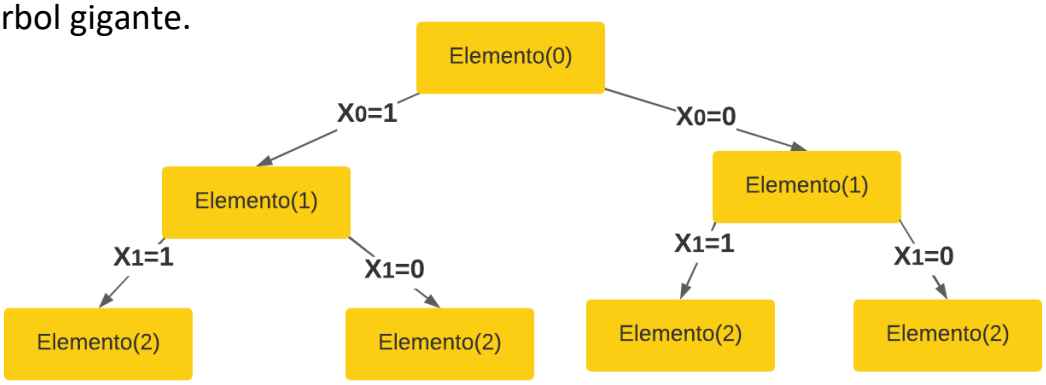
$$|v| = |w|$$
$$k \geq 0$$
$$v_i \geq 0$$
$$w_i > 0$$

$$sujeto\ a \sum_{i=0}^{n-1} w_i\, x_i \leq k\ y\ x_i \in \{1,0\}$$

Solución de Knapsack Problem

- Tenemos para calcular la cantidad de modos de asignación de si podemos usarlo **1** o no podemos usarlo **0** lo cual esto se hace de la forma 2^n donde n es la cantidad de objetos que queremos colcar dentro de la mochila (elementos) los cuales estarán representados por x_i.
- Con consecuencia de esto tenemos una tupla de tamaño fijo la cual los elementos tendrán dos estados de **0** no se introduce y **1** si se introduce.

Esto nos da un árbol de decisión que si el primer elemento que será nuestro nodo tendrá dos ramas las cuales serán dos estados los cuales será entrar o no entrar a la mochila, en este caso no sabemos la opción optima ya que tenemos que consultar los demás elementos para ver si es buen opción, eso nos crea un árbol gigante.



Todas las combinaciones posibles que están dadas por 2 elevado a la n-ésima potencia donde n es la cantidad total de elementos a trabajar.

Lo que hicimos fue hacer un algoritmo de fuerza bruta lo cual que por medio de la recursividad es hacer todas las combinaciones posibles para encontrar la solución óptima. Lo que hace es calcular el peso y el valor total de todos los subconjuntos y solo tomar los que son menores al peso total de la mochila (W).

Pseudocódigo

Defino **knapSack(** entero w, entero wt, entero val, entero n**)**

si n es idéntico a 0 o w es idéntico a 0

entonces retorna 0

si wt[n-1] es mayor que w

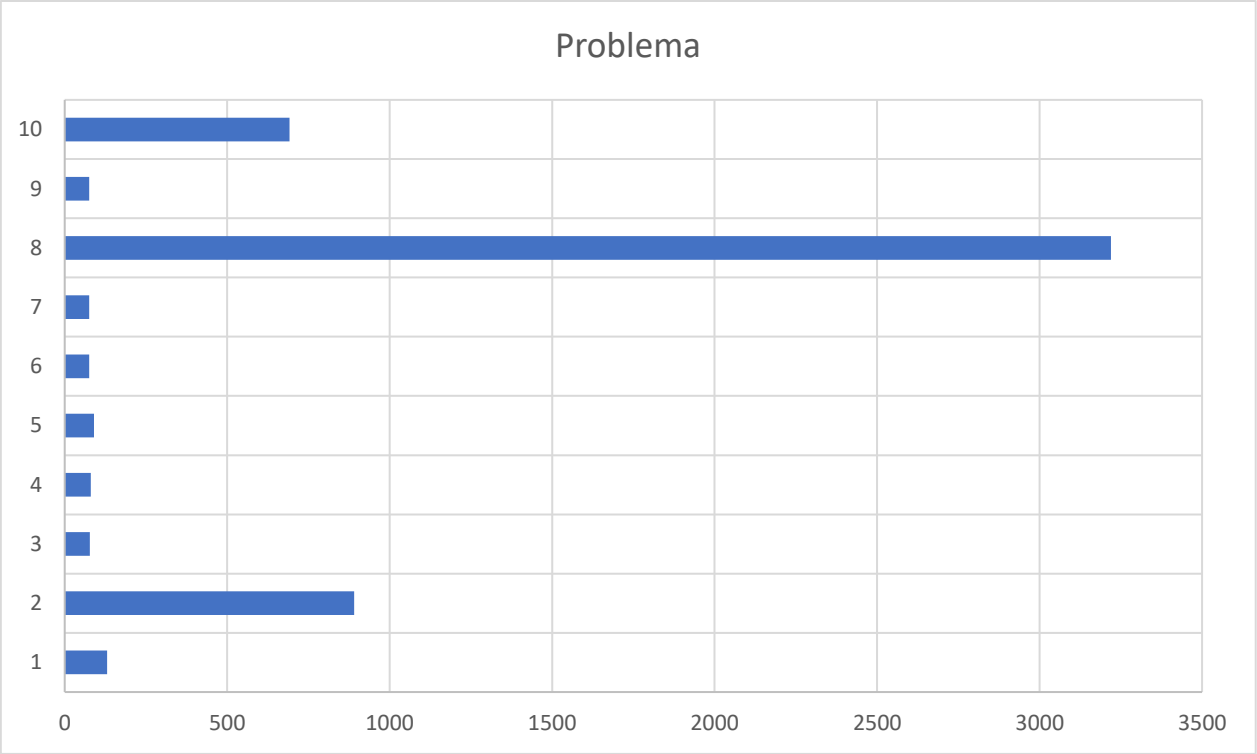
entonces retorna knapSack(w,wt,val,n-1)

si no

entonces retorna el máximo de val[n-1] + knapSack(w-wt[n-1],wt,val,n-1), knapSack(W, wtm val, n-1))

Este algoritmo tenemos los siguientes resultados los cuales un apartado es el tiempo de ejecución que tardo nuestro algoritmo en realizar los cálculos para la solución óptima.

Problema	Solucio Optima del problema	Tiempo	Solucio Optima
1	295	130	295
2	1024	891	1024
3	35	77	35
4	23	80	23
5	4,810,694	90	481.069368
6	52	75	52
7	107	75	107
8	9767	3220	9767
9	130	75	130
10	1025	692	1025


Problema
10
9
8
7
6
5
4
3
2
1
0
500
1000
1500
2000
2500
3000
3500