Cómputo Evolutivo
Proyecto Intermedio

Adrián García Pérez
315131224

1 Problema de la mochila

El problema de la mochila se trata de un problema optimización combinatoria, donde buscamos encontrar la mejor solución entre un conjunto de soluciones posibles al problema. El problema consiste en que, dado un conjunto de objetos con un peso y un valor, buscamos maximizar la suma de los valores de los objetos que metemos a la mochila, con la restricción de que la mochila tiene una capacidad finita que no debe ser superada.

Existen diversas variantes del problema. La variante general considera un conjunto de objetos con valor y peso, donde hay diferentes cantidades de cada tipo de objeto, es decir, que se puede considerar meter varias veces un tipo de objeto, con mismo valor y peso.

1.1 Variante 0-1

La variante con la que trabajaremos es el problema de la mochila 0-1, donde se restringe el número de copias de cada tipo de objeto a 1, es decir, nuestra solución solo puede considerar si el objeto está o no está en la mochila, pero no varias copias del mismo.

Así, si existen n distintos objetos numerados de 1 a n, cada uno con un valor v_i y un peso w_i, y nuestra mochila tiene una capacidad máximade W, entonces buscamos maximizar el valor de los objetos contenidos en la mochila sin superar la capacidad de la misma, es decir maximizar:

$$\sum_{i=1}^{n} v_i x_i \text{ tal que } \sum_{i=1}^{n} w_i x_i \leq W \text{ y } x_i \in \{0, 1\}$$

De esta manera, nuestro objetivo será encontrar soluciones que maximicen el valor de los objetos que lleva la mochila sin exceder el peso máximo W, lo cual explicaremos cómo más adelante.

2 Representación de las soluciones

Necesitamos encontrar una representación de las soluciones que nos permita generarlas fácilmente de manera aleatoria y además, generar a nuevos miembros de la población con base en las probabilidades definidas en nuestro modelo.

El acercamiento que tendremos será que las soluciones son un vector binario, donde la longitud del vector es la cantidad de diferentes objetos que pueden meterse en la mochila.

2.1 Archivos de entrada

Nuestros archivos de entrada están definidos como:

$$n\ wmax$$
$$v_1\ w_1$$
$$v_2\ w_2$$
$$.$$
$$.$$
$$v_n\ w_n$$

donde n es el número de objetos de la instancia, $wmax$ el peso máximo que la mochila puede llevar, y a continuación se listan las parejas v_i, w_i, siendo el valor y el peso del objeto i-ésimo.

Por lo que un vector solución tiene la forma:

$$[b_0, b_1, ..., b_{n-1}] \text{ con } b_i = \{0, 1\}$$

Por ejemplo, si nuestro archivo de entrada es:

5 8
5 1
10 2
15 3
25 4
30 5
35 6
40 7
45 8

Entonces algunas posibles soluciones son:

$$[0, 0, 0, 0, 0, 0, 0, 1]$$
$$[1, 0, 0, 0, 0, 0, 1, 0]$$
$$[0, 1, 0, 0, 0, 1, 0, 0]$$

donde un 1 en la posición v_i nos dice que el i-ésimo objeto se mete en la mochila, y por el contrario un 0 indica que el objeto no se mete en la mochila.

Como tenemos un vector de n elementos, donde cada posición puede tomar dos posibles valores, entonces el espacio de búsqueda es igual a 2^n; todas las posibles combinaciones de n bits que podemos formar.

Finalmente, la implementación de nuestra representación de soluciones es manejada como una lista en Python, que contiene n elementos que pueden tomar el valor 1 o 0.

3 Función de evaluación

Para evaluar a los individuos, tomaremos el vector que representa a su solución y haremos la suma de los valores de cada elemento que el vector considera dentro de la mochila. En los casos en los que la capacidad de la mochila se vea rebasada, consideraremos que la evaluación de la solución es 0, ya que la solución no es viable y penalizamos su valor fitness.

La implementación de este pseudocódigo se encuentra en el archivo `Instance.py`.

Algorithm 1: Fitness

Input: Instance: *instance*, Solution: *solution*
Output: Int: $fitness$

```
1  total_value ← 0
2  total_weight ← 0
3  For i, item in enumerate(instance.items):
4    if solution[i] == 1:
5      total_value ← total_value + item[0]
6      total_weight ← total_weight + item[1]
7  if total_weight > instance.wmax:
8    return 0
9  else:
10   return total_value
```

4 Algoritmos estimación de distribución

4.1 PBIL

El algoritmo PIBIL fue propuesto por Shumeet Baluja en 1994, y se basa en la evolución de un vector de probabilidad para toda la población en lugar de vectores individuales para cada individuo.

Los genes se representan como valores reales (valores de tipo float o double en su representación en computadora) en un rango de 0 a 1, en particular, indican la probabilidad de que un alelo aparezca en el gen. El algoritmo se basa en la repetición de 4 partes fundamentales:

1. Se genera una población a partir del vector de probabilidades (el inicial o el correspondiente a la iteración)
2. Se calcula el valor fitness de cada individuo y se clasifica con respecto a que tan bueno es a comparación del resto de la población.
3. Se actualiza el vector de probabilidades (genotipo) de la población, tomando como base el mejor individuo de la iteración.
4. Se realiza algún tipo de mutación (para que las soluciones tengan mayor diversidad)

4.2 MIMIC

Por su lado, los algoritmos tipo MIMIC se basan en factorizaciones bivariadas para la generación de sus modelos.

Se considera generar un modelo cadena, donde se calculan las dependencias 2 a 2 de las variables para determinar el mejor modelo. Esto se logra calculando la distribución de probabilidad conjunta para después obtener la información mutua de las variables.

El procedimiento del algoritmo es similar a PBIL, donde se obtiene una población inicial aleatoria y después se realizan iteraciones calculando los modelos de cada iteración con base en la información mutua de las variables para obtener nuevos miembros descartando a los que no cumplan con los requerimientos de su función fitness.

5 Experimentación

Se realizaron 10 repeticiones para 5 diferentes archivos. En la tabla se muestra el nombre de la instancia, el tamaño de la población, número de repeticiones de los experimentos, número máximo de iteraciones en cada repetición, el mejor, promedio y peor valor obtenido, así como el valor óptimo que se buscaba y finalmente un promedio del tiempo de ejecución de cada experimento.

Instancia	Tamaño	Repeticiones	Iteraciones	Mejor valor	Promedio	Peor valor	Óptimo	Tiempo prom.
23_10000	1000	10	1000	9740	9644	9223	9767	5.37 s
20_879	1000	10	1000	1019	975	924	1025	4.10 s
20_878	1000	10	1000	1024	985	909	1024	3.88 s
15_375	1000	10	1000	481.069368	425.93	350.218	481.0694	2.27 s
10_269	1000	10	1000	295	246.1	218	295	1.04 s

6 Conclusiones y comentarios

Como podemos observar, los valores obtenidos fueron favorables en la mayoría de los experimentos, incluso alcanzando el valor óptimo en algunos, y el tiempo de ejecución no fue mayor a 6 segundos en promedio. Sin embargo, algo que pude notar es que en algunas repeticiones, caemos en convergencia prematura al momento de generar nuevas soluciones, algo que podría verse fácilmente con una gráfica pero no tuve la oportunidad de generarlas.

En conclusión, la implementación me pareció bastante interesante debido a que ahora consideramos mucha información basada en probabilidades frecuentistas de nuestras soluciones. Me gustó bastante el como podemos elegir distintos modelos bivariados para generar diferentes soluciones y creo que sería buena idea investigar más el tema para comparar las posibles ventajas y desventajas entre modelos (como vimos en clase), algo que podría verse como proyecto final de la materia.