METAHEURÍSTICAS

Memoria de la **PRÁCTICA 1**

Grado en Ingeniería Informática
Universidad de Córdoba
Teófilo Rojas Mata

1. Objetivo

El objetivo de la práctica será el de resolver los problemas clásicos de la mochila (KP) y el del viajante de comercio (TSP) por la metodología del tratamiento de datos aleatorio.

Para la resolución se utilizará los archivos suministrado de nombre:
knapPI_1_200_10000.csv.
berlin52.tsp

2. Problema de la mochila.

2.1. Descripción

El problema de la mochila consiste en introducir en una capacidad C una serie de objetos con beneficio p_i y de peso w_i, de forma que el sumatorio de los pesos de los objetos introducidos en mochila, no supere la capacidad de la mochila. Además se busca maximizar el beneficio total de los objetos introducidos en mochila.

Esta práctica consiste en el tratamiento aleatorio puro de la solución, para ello se utilizará un nuevo concepto añadido llamado Fitness. Teniendo en cuenta que al ser introducción aleatoria pura en mochila, cada objeto tiene un 50% de posibilidad de estar en mochila, por ello se pueden tener dos tipos de soluciones:

- La suma de los pesos es menor que la capacidad de la mochila:
  Fitness = suma de **beneficios** de los objetos.
- La suma de los pesos es mayor que la capacidad de la mochila:
  Fitness = Capacidad de la mochila - suma de los **pesos** de los objetos.

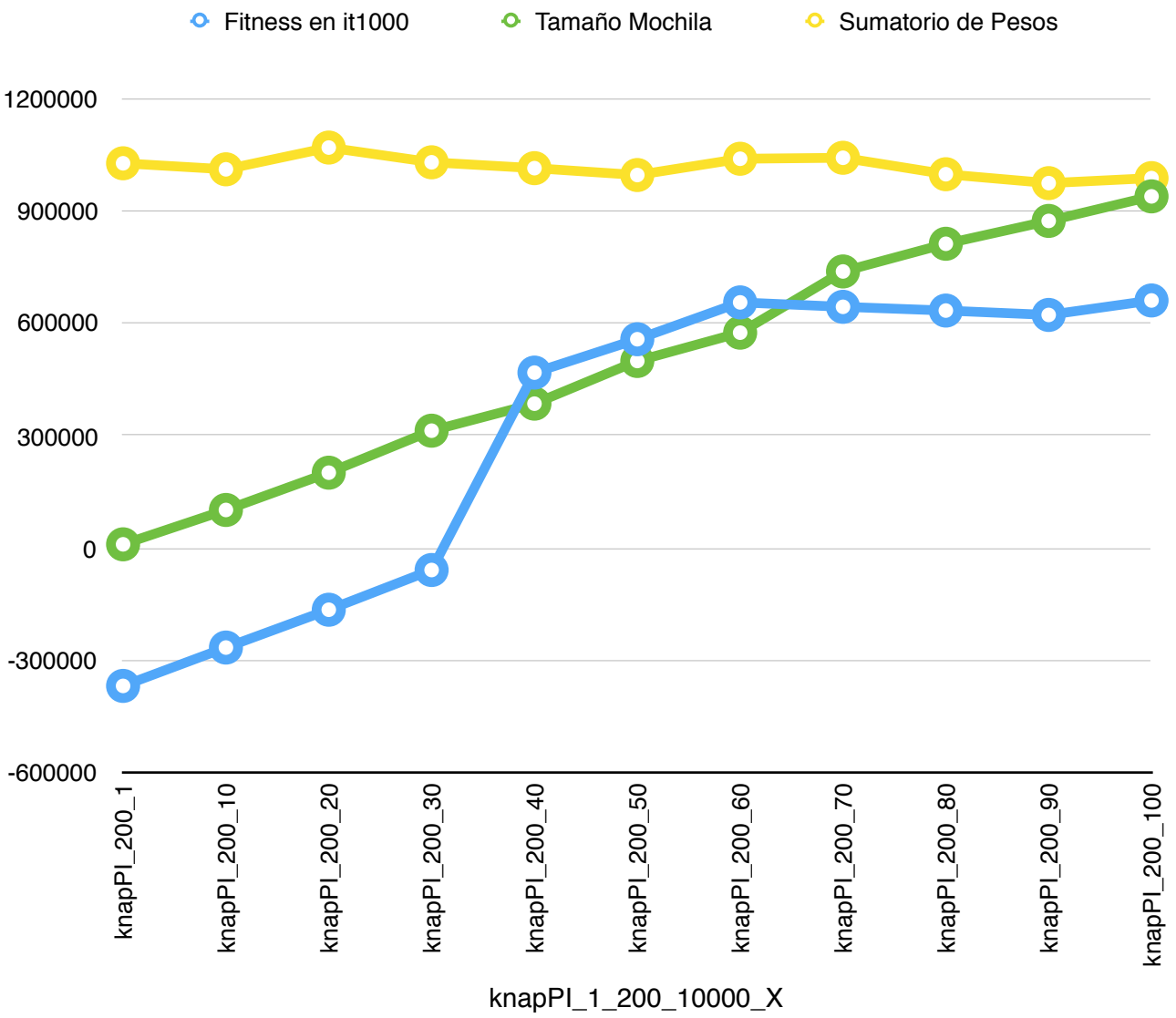
2.2. Análisis

El archivo knapPI_1_200_10000.csv tiene cien grupos de doscientos datos cada uno. Por tanto se tratarán 100 mochilas con 200 objetos con una probabilidad para cada objeto entrar en mochila de un 50%.

Cada mochila tiene una capacidad diferente, de manera que ésta va creciendo linealmente relacionada con el número de la mochila a tratar.

Para tratar el problema, se tomarán cada una de las mochilas, iterando sobre ellas 1000 veces en busca de la mejor solución Fitness.

Con este método será muy probable que para las mochilas más pequeñas, se obtenga un Fitness negativo y para mochilas muy grandes un Fitness positivo, debido a que



al ser un método de búsqueda puro es obvio que en la mochila estarán aproximadamente el 50% de los objetos (alrededor de 100).

Se observa además que el beneficio total de cada grupo de datos, es relativamente constante.

2.3. Desarrollo

Para la resolución del problema se implementan las siguientes clases con el objetivo de representar la información fielmente y llegar a una solución correcta.

- Clase básica de representación de objeto:

Objeto: Representa a cada objeto que podrá estar o no en mochila, cada objeto tendrá 4 variables, peso del objeto, beneficio del objeto, posición del objeto en la lista, bandera de aviso si está o no en la mochila.

- Clase de lectura de datos:

InstanceKP: clase encargada de leer los datos de cada mochila, para ello irá rellenando un vector con todos los objetos de la lista, además del peso máximo de la mochila y el número de elementos en lista.

- Clase contenedora de solución:

solutionKP: clase encargada de contener la mejor solución tras el tratamiento de los datos, para ello tendrá un vector solución con los elementos que entraron en la mochila, el beneficio total de los elementos en mochila, el peso total de los elementos en mochila y su fitness correspondiente.

- Clase generadora de soluciones:

solGeneratorKP: clase encargada de generar soluciones iterando sobre la InstanceKP de datos, almacenar en una lista todas las soluciones, además de la mejor solución encontrada hasta el momento.

Finalmente, a través del programa principal main.cpp, se realizan el número de iteraciones correspondientes sobre cada mochila, llamando en este caso 1000 veces a solGeneratorKP sobre la InstanceKP.

2.4. Interpretación de los resultados

Se realizan 1000 iteraciones en busca del mejor Fitness para cada una de las 100 mochilas existentes en el archivo indicado en el apartado 3.1.

Teniendo en cuenta lo comentado en el apartado 3.2 (sumatorio de beneficios prácticamente constante, tamaño de mochila muy pequeño en la mochila 1 y muy grande en la mochila 100 y probabilidad de que en la mochila entren alrededor de 50 objetos), se tiene como resultados el resumen mostrado en la siguiente gráfica:
Como se observan los resultados son bastante lógicos, teniendo en cuenta que para sumatorio de pesos prácticamente constantes (≈ 1.000.000) y para probabilidad de que entren en la mochila más o menos la misma cantidad de objetos, se distinguen tres casuísticas:

- Para mochilas demasiado pequeñas:

Mientras que la capacidad de la mochila se mantiene demasiado pequeña, al tener en cuenta que los pesos del 50% de los objetos van a ser siempre mayor, se obtiene un Fitness claramente negativo.
Fitness = Capacidad de mochila - Sumatorio de Pesos.

- Para mochilas suficientemente grandes:

El caso más interesante quizás sea a partir de la mochila número 40, donde la mochila tiene una capacidad suficientemente grande como para que el 50% de los datos sumen sus pesos sin sobrepasar dicha capacidad.

Se observa como un crecimiento en la capacidad de la mochila entre 300.000 y 600.000 genera una relación directa de mejoría en el Fitness, se presupone correcto debido a que a mayor tamaño de la capacidad de la mochila, se pueden introducir los mejor valores correspondientes a ese 50% de objetos que aproximadamente entrarán en la mochila.

- Para mochilas demasiado grandes:

A partir de una capacidad de mochila de 600.000, se observa como el Fitness se mantiene constante, hecho que se presupone correcto, debido a que el método de búsqueda es aleatoriamente puro, de manera que a partir de ese tamaño de mochila, siempre van a entrar aproximadamente el 50% de los objetos y entre éstos estarán los de mejores características.

2.4.1 Conclusión

En búsqueda aleatoria sobredimensionar la capacidad de la mochila no es interesante, puesto que no van a entrar más objetos por ello, hay que recordar que el método de búsqueda escogido es el aleatorio puro, y por ello como no se van a introducir en la mochila muchos más objetos que el 50% de la lista.

Para este método sí resultaría interesante investigar la relación coste/beneficio en los rangos de mochila mayor de ≈ 300.000 y menor de ≈ 600.000.

3. Problema del viajante de comercio.

3.1. Descripción

El problema del viajante de comercio consiste en recorrer un grafo totalmente unido (conexo) con una cantidad de nodos N, de manera que se recorran todos los nodos de forma cíclica volviendo al punto de partida.

Se considera que la distancia entre nodos es de d_{ij}. La búsqueda de la solución consiste en minimizar la distancia total para recorrer todos los nodos del grafo.

3.2. Análisis

Para realizar este método de una forma aleatoria pura, se recorrerán en orden todos los nodos, de manera que su posición se intercambie por otra de otro nodo escogido de forma al azar, ejemplo de actuación:

Se toma el primer nodo de la lista, se obtiene un número aleatorio dentro la lista, y el primer nodo se cambia por el de la posición que corresponde al número aleatorio.
Se toma el segundo nodo de la lista, se obtiene un número aleatorio dentro de la lista, y el segundo nodo se cambia por el de la posición que corresponde al número aleatorio.

Se repite el método hasta realizarlo con todos los nodos. Al terminar se suman las distancias.

Este método se realizará mil veces y se irá quedando con la mejor de las soluciones que no será otra que la que haya obtenido la menor distancia total.

3.3. Desarrollo

Para la resolución del problema se implementan las siguientes clases con el objetivo de representar la información fielmente y llegar a una solución óptima.

- Clase básica de representación de objeto:

Punto: Representa a cada punto o nodo del mapa, para ello se generan 3 variables, 2 float para sus coordenadas X e Y y una entera para guardar su posición inicial.

- Clase de lectura de datos:

InstanceTSP: clase encargada de leer los datos de cada mochila, para ello irá rellenando un vector con todos los Puntos de la lista, además del número de elementos en lista.

- Clase contenedora de solución:

solutionTSP: clase encargada de contener la mejor solución tras el tratamiento de los datos, para ello tendrá un vector solución con los nodos y la suma total de sus distancias.

- Clase generadora de soluciones:

solGeneratorTSP: clase encargada de generar soluciones iterando sobre la InstanceKP de datos, almacenar en una lista todas las soluciones, además de la mejor solución encontrada hasta el momento.

Finalmente, a través del programa principal main.cpp, se realizan el número de iteraciones correspondientes sobre cada mochila, llamando en este caso 1000 veces a solGeneratorTSP sobre la InstanceTSP.

3.4. Interpretación de los resultados

El problema se aborda realizando 5 ejecuciones del código para interpretar los datos correspondientes.

Se obtienen los siguientes resultados:

	1 ejecución	2 ejecución	3 ejecución	4 ejecución	5 ejecución
Iteración 100	24947.9	26065.6	25570.3	26077.6	25398.3
Iteración 200	24947.9	25080.8	25557.6	24436.5	25398.3
Iteración 300	23973.7	25080.8	24551.3	24436.5	24764.8
Iteración 400	23973.7	24549.6	24551.3	24436.5	24764.8
Iteración 500	23973.7	24549.6	24551.3	24436.5	24764.8
Iteración 600	23001.1	24549.6	24551.3	24436.5	24764.8
Iteración 700	23001.1	24549.6	24271.6	23367	24764.8
Iteración 800	23001.1	24170.3	24271.6	23367	24764.8
Iteración 900	23001.1	24170.3	24271.6	23367	24764.8
Iteración 1000	23001.1	23826.7	**22147.6**	23367	24764.8

- Vector solución 1ª ejecución:

[40 38 25 14 10 29 50 17 21 20 18 7 30 31 16 1 39 15 5 12 42 2 46 33 28 47 34 45 8 4 43 22 24 48 6 13 49 35 26 27 11 51 19 3 9 32 44 37 52 23 36 41]

- Vector solución 2ª ejecución:

[20 7 6 36 12 11 27 29 5 24 19 25 10 38 51 45 48 49 28 52 33 23 15 42 50 40 44 31 37 13 14 47 26 3 2 1 16 46 9 43 30 34 39 41 8 18 22 4 32 35 21 17]

- Vector solución 3ª ejecución:

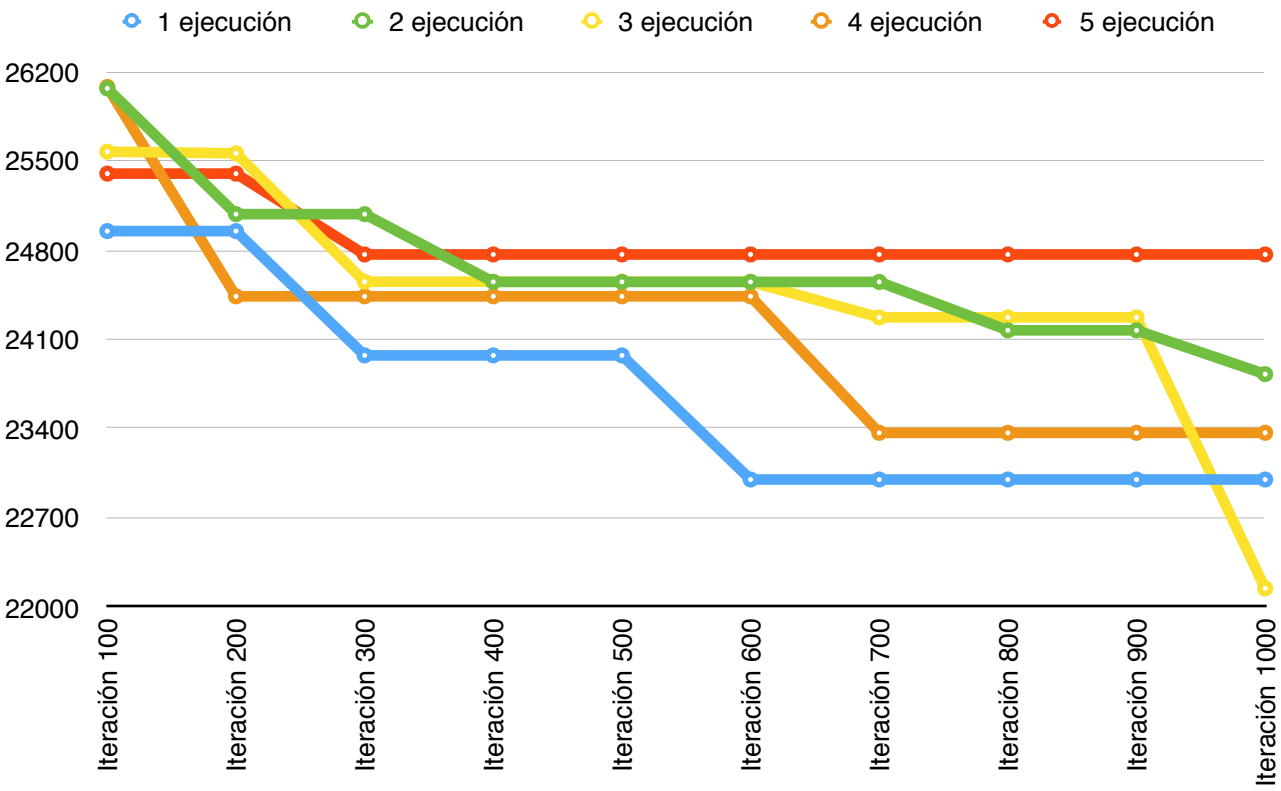
[10 45 43 35 28 50 32 20 37 3 44 46 38 16 31 26 11 52 30 29 1 42 17 7 41 19 9 15 12 24 6 33 51 14 2 21 8 23 18 47 13 5 48 36 39 34 25 4 27 40 49 22]

- Vector solución 4ª ejecución:

[47 5 39 48 14 52 27 25 3 4 32 31 22 24 36 43 33 37 46 34 6 8 16 29 26 9 19 17 11 35 28 41 13 44 40 45 10 21 30 51 12 15 50 1 42 7 2 23 49 18 20 38]

- Vector solución 5ª ejecución:

[40 45 18 9 19 36 23 27 52 43 17 42 25 49 33 5 22 21 44 4 37 11 14 50 15 32 10 16 38 8 31 7 3 20 2 30 35 34 41 51 48 6 46 26 47 39 1 28 13 24 29 12]



Con dicho método aleatorio de búsqueda, se observa la mejor solución en la 3ª ejecución del software. Teniendo una diferencia máxima con el peor resultado de 2617,2 (10,67 %) de diferencia.

3.4.1 Conclusión

La conclusión final que se obtiene respecto al método de búsqueda aleatoria para el problema del TSP, es que conforme avanzan las iteraciones se van produciendo menos cambios a mejor.

Parece ser que salvo para la ejecución número 3, a partir de 600-700 iteraciones no se producen cambios muy significativos. Esto hace pensar que para este método no es totalmente beneficioso realizar búsquedas con iteraciones grandes (del orden de millones o billones), puesto que el beneficio obtenido con respecto al coste computacional sería muy pequeño o nulo.