UNIVERSIDAD DE CÓRDOBA

GRADO DE INGENIERÍA INFORMÁTICA

METAHEURISTICA

MEMORIA DE LA PRÁCTICA 1.

Codificación y evaluación de soluciones.

Grupo:
NP-Group

Miembros:
Cristian García Jiménez
Daniel Carmona Martínez
Guillermo Sugráñez Pérez
Antonio Jesús Jiménez Urbano

1. Introducción.

Las tareas a realizar consistirán en la programación clases y métodos que puedan leer instancias de estos problemas desde un archivo, generar soluciones aleatorias y evaluarlas.

Los problemas que vamos a evaluar son los siguientes:

- Problema de la mochila.

Supongamos que tenemos *N* objetos *{o1,...,oN}*. Cada objeto tiene un beneficio *pi* y un peso *wi*, asociados, no negativos. Por otro lado, se tiene una mochila donde se pueden introducir los objetos, que soporta un peso máximo *W*. El problema consiste en meter en la mochila la combinación de objetos que maximice la suma de sus beneficios y que no supere el peso máximo que puede soportar la misma.

- Problema del viajante de comercio.

Dado un grafo no dirigido con pesos *G=(V, E, w)*, *V={v1,...,vN}* son sus nodos, *E={eij=(vi,vj)* | *vi,vj* ∈ *V}* sus arcos y *w: E->R* una función que asigna un peso a los arcos del grafo, el objetivo es encontrar el camino que recorre todos los nodos del grafo y cuya suma de pesos es mínimo.

2. Fases de codificación.

Para ambos problemas, hemos diseñado un esquema inicial donde organizamos las clases con sus diferentes atributos y métodos, así como las relaciones entre sí. En esta fase inicial nos aseguramos de que toda la funcionalidad del problema queda cubierta y que la solución es consistente.

Para la codificación de las soluciones de estos problemas hemos decidido usar el lenguaje de programación C++, con el que nos encontramos más familiarizados.

2.1. Knapsack Problem (KP).

Para la resolución de este problema se ha diseñado una clase que genera soluciones las cuales deben satisfacer la regla de no superar la capacidad máxima de la mochila con la siguiente restricción:

$$\sum_{i=1}^{n} w_i \cdot x_i \leq W$$
$$x_i \in \{0,1\}$$

Siendo:

W: capacidad máxima

X: material parte de la solución (1) o no (0)

w: peso del material x

Y obtener el máximo beneficio cumpliendo:

$$\max \sum_{i=1}^{n} p_i \cdot x_i$$

p: beneficio del material.

2.2.

2.3. Travel Salesman Problem (TSP).

En este problema se ha considerado el caso más simple, asumiendo que todos los nodos de la instancia se encuentran conectados entre sí.

Debido a esto la clase implementada que genera las soluciones solo debe obtener la suma de los caminos de una permutación de los nodos anteriormente mencionados que minimice la siguiente condición:

$$F(x) = \sum_{i=1}^{N} d_i * x_i$$

3. Evaluación.

Para la evaluación de las diferentes instancias, en ambos problemas se han realizado 5 repeticiones con 1000 ejecuciones por repetición.

Los resultados obtenidos se representarán en gráficas junto a tablas con diferentes valores de aptitud obtenidos.

3.1. Mochila

Para la evaluación de este problema se ha usado el valor óptimo de cada instancia proporcionado en los ficheros y en cada iteración se ha calculado la diferencia de este valor respecto al coste de la solución óptima obtenida en el momento actual.

A continuación se mostrarán las gráficas beneficio / iteración obtenidas en las pruebas y su correspondiente fitness / iteración:

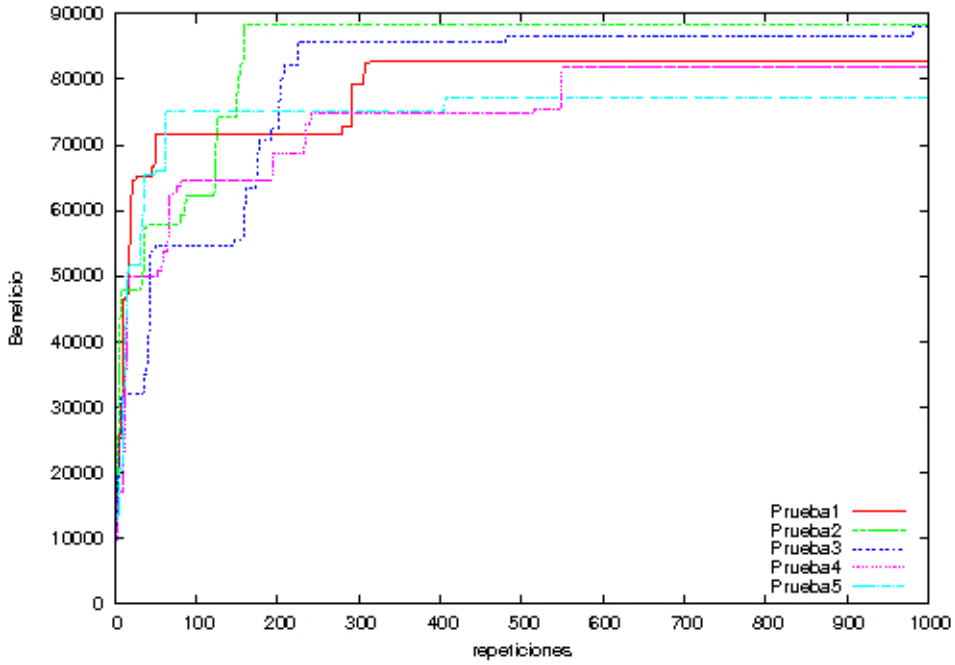

Figura 1: KnapPI_1_200

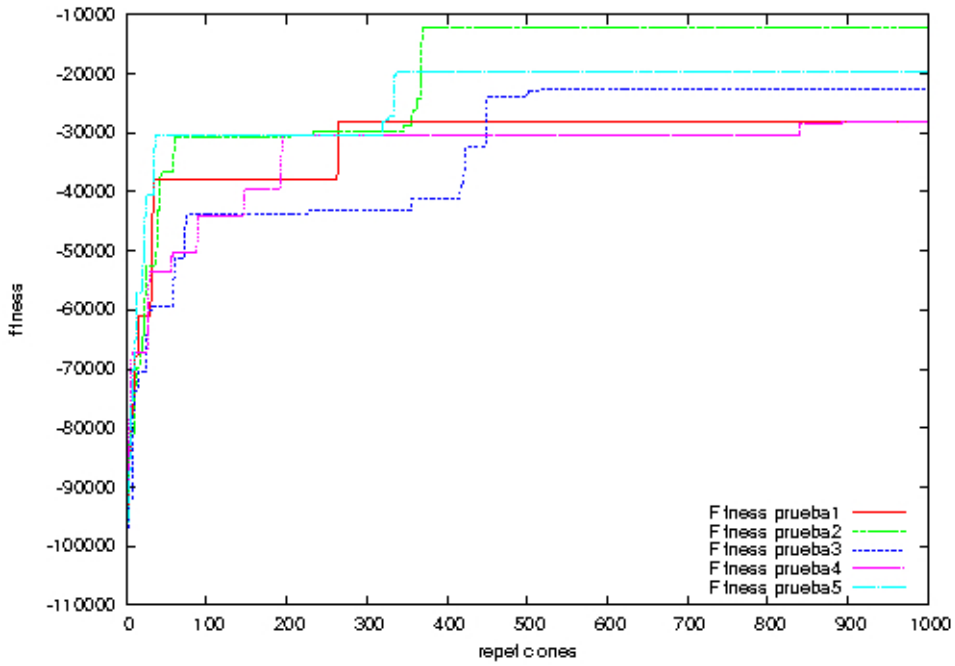

Figura 2: Fitness KnapPI_1_200

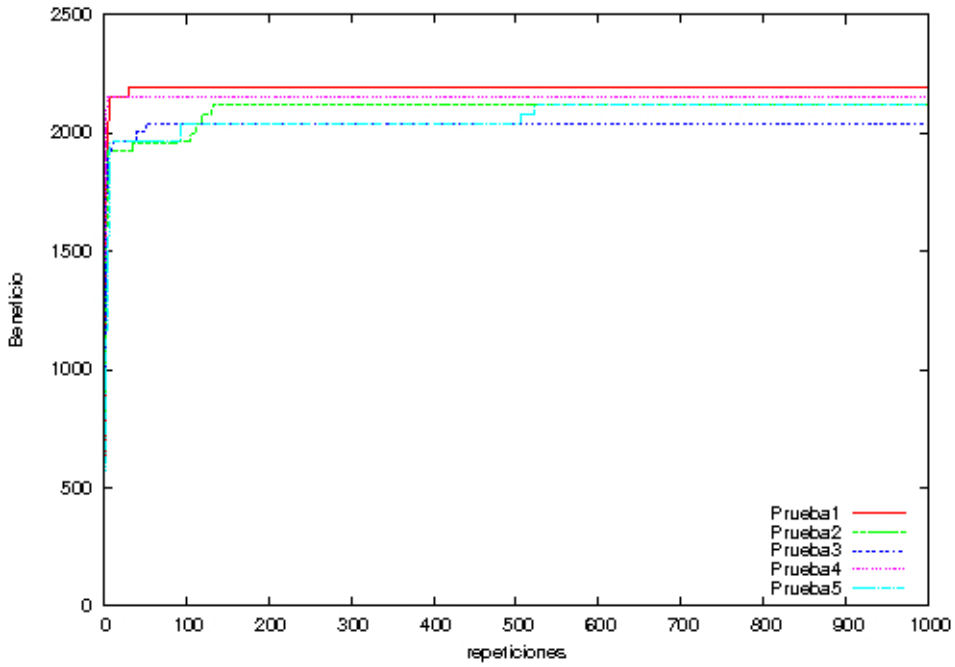

Figura 3: KnapPI_12_500_1000

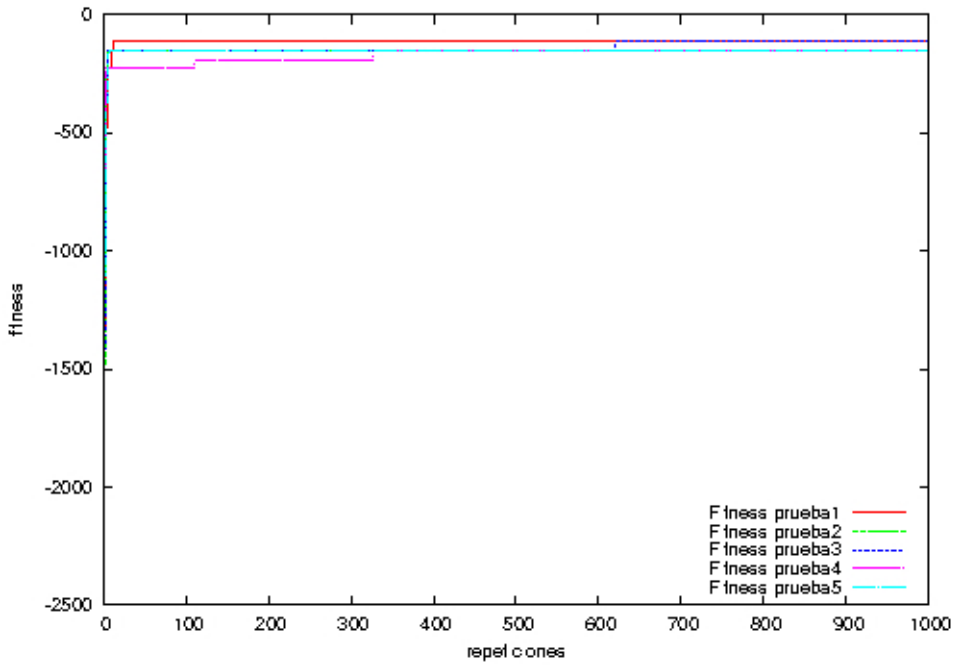

Figura 4: Fitness KnapPI_12_500_1000

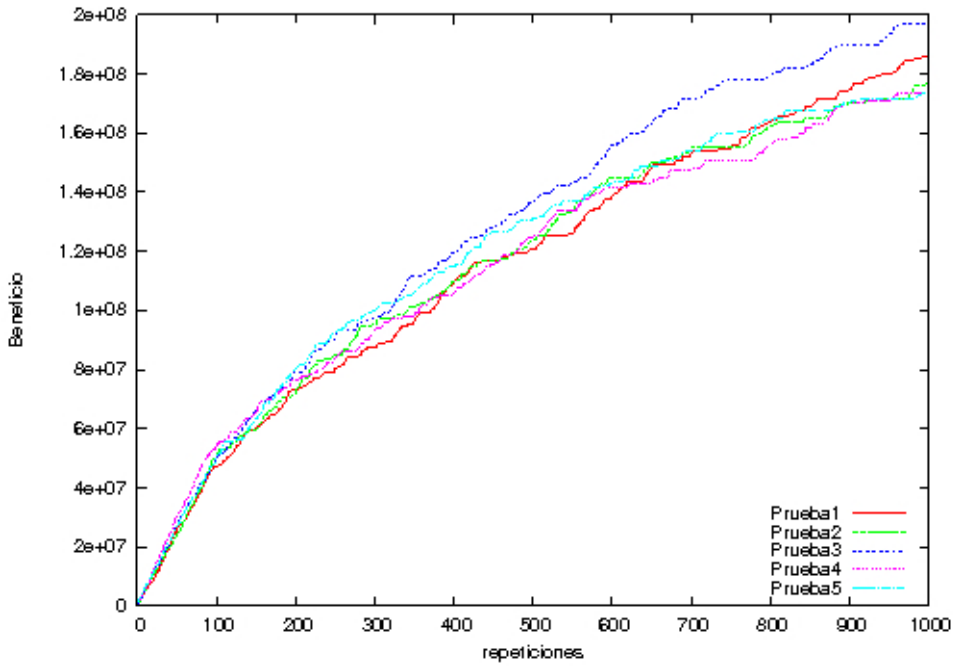

Figura 5: KnapPI_1_10000_1000000

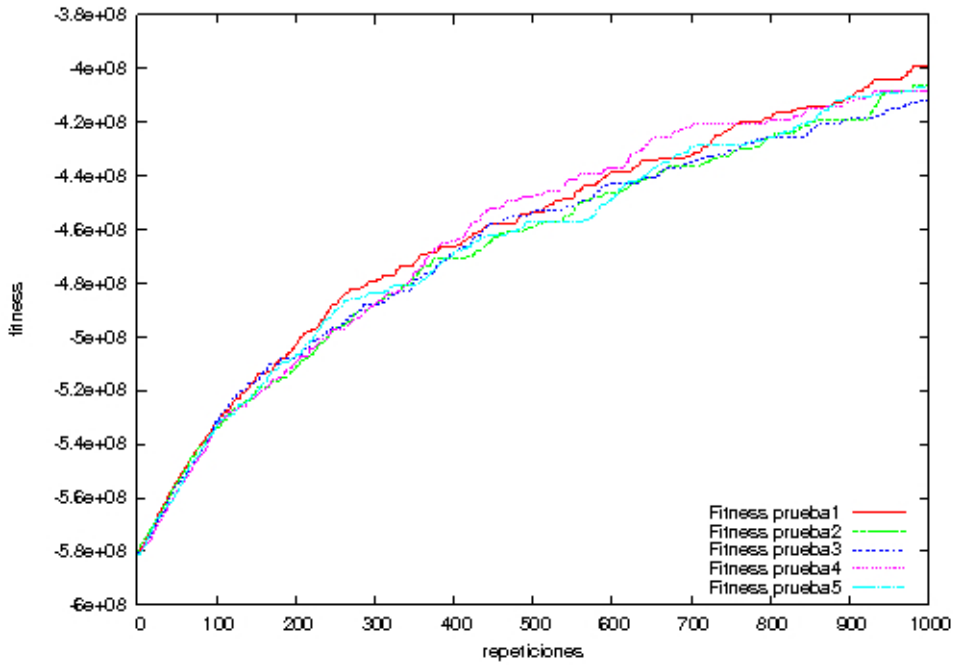

Figura 6: Fitness KnapPI_1_10000_1000000

3.2. Viajante

Tras las primeras pruebas preliminares constatamos que el orden en el que estaba el fichero de puntos era la solución más óptima, por tanto, hemos podido obtener el valor del camino mínimo calculando el coste de esta configuración. A partir de aquí hemos obtenido el valor de aptitud de las diferentes soluciones candidatas generadas en cada instante, reordenando de manera aleatoria el vector que contiene los índices ordenados que representan la unión de un nodo i con el siguiente i+1, y a continuación realizando permutaciones del mismo.

A continuación se mostrarán las gráficas coste / iteración obtenidas en las pruebas y su correspondiente fitness / iteración:

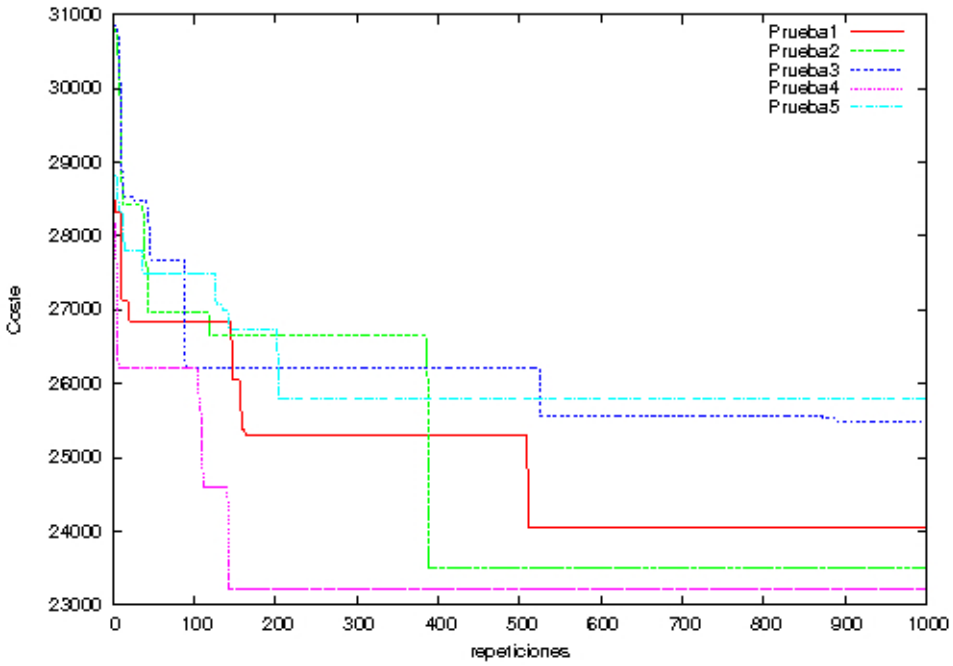

Figura 7: Berlin52

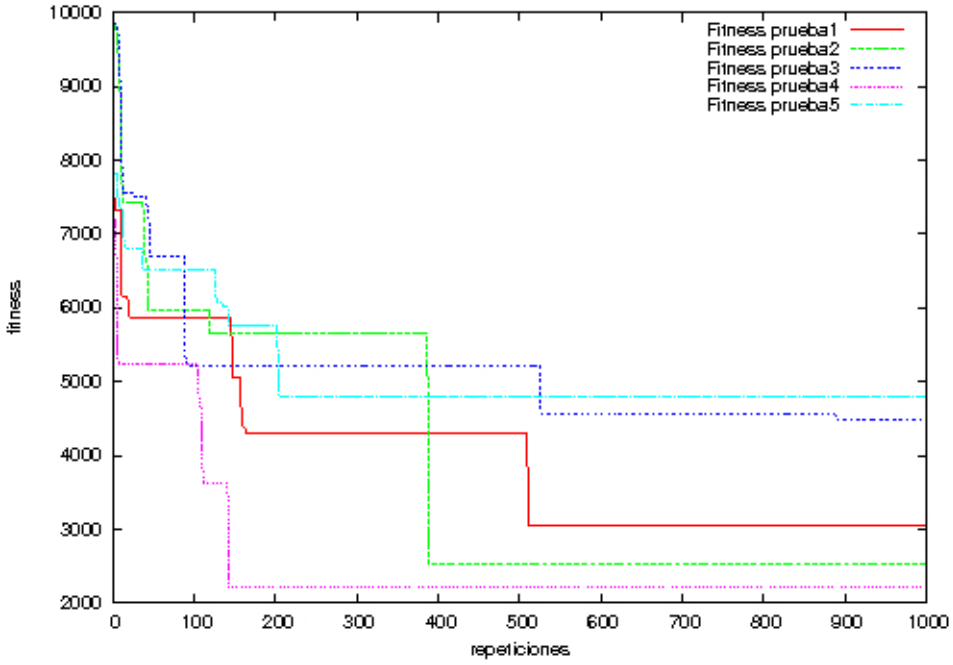

Figura 8: Fitness Berlin52

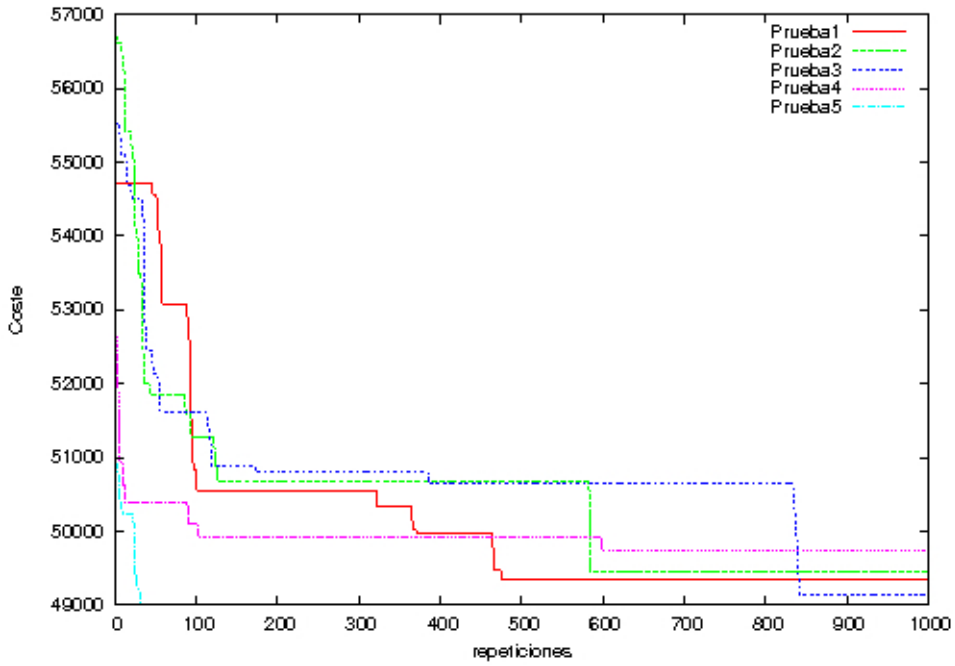

Figura 9: ch150

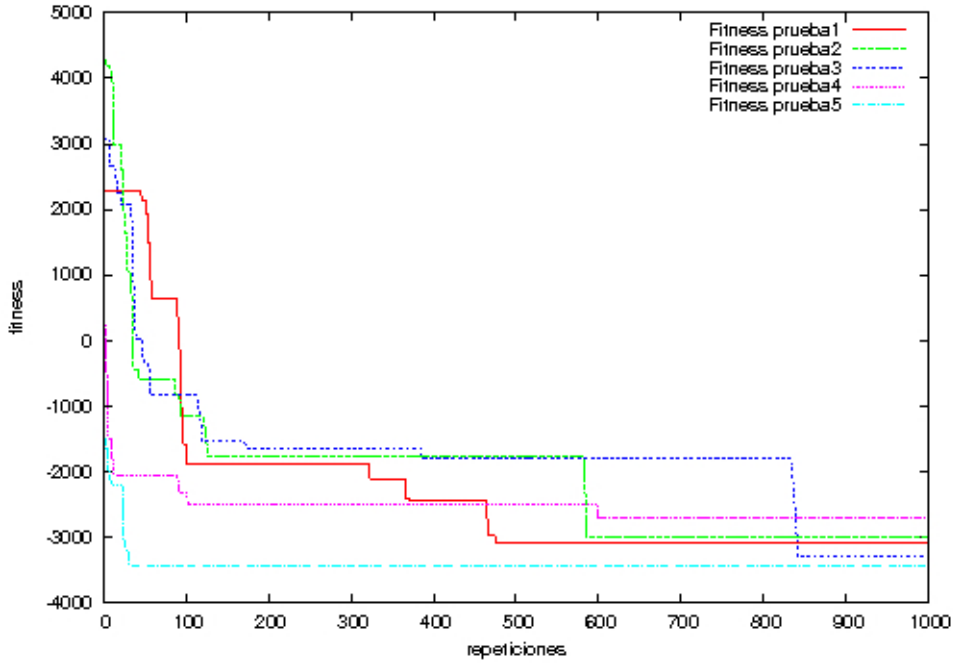

Figura 10: Fitness ch150

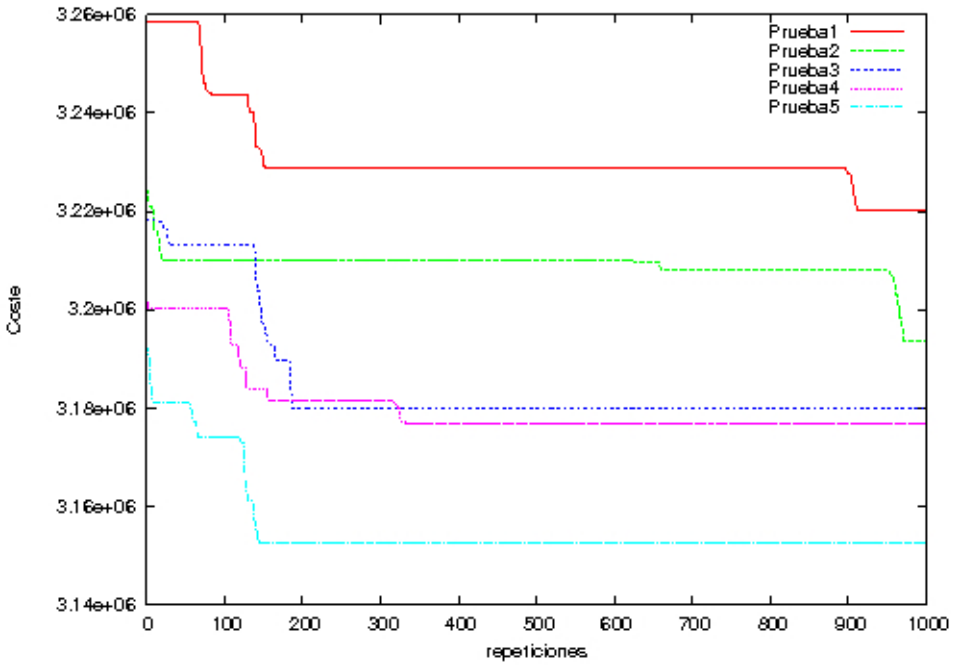

Figura 11: d2103

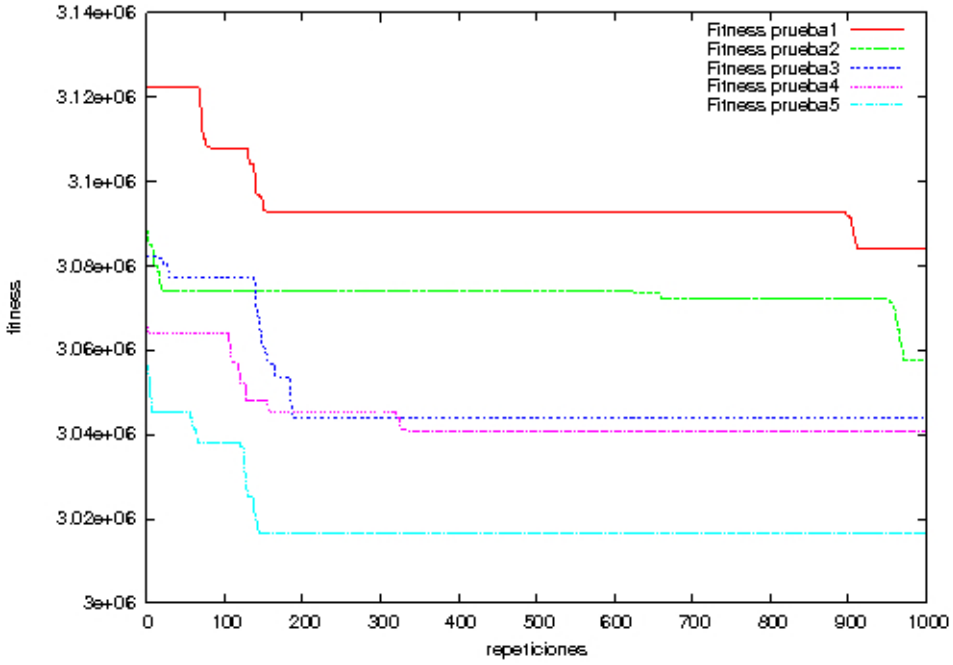

Figura 12: Fitness d2103

4. Conclusiones finales.

- **Knap:**

En función de las gráficas observadas obtenemos como conclusión que debido al peso de los materiales respecto a la capacidad máxima de la mochila es más fácil (se necesitan menos iteraciones) llenar ésta cuando el número de materiales disponibles aumenta y conseguir acercarse lo máximo posible a la solución más óptima o fitness.

- **TSP:**

Nuevamente, observando las gráficas obtenidas en este problema, podemos concluir que a mayor número de nodos en el problema, se necesita un mayor número de iteraciones, se empeora su valor de fitness y por tanto se aleja más de la solución óptima real.

5. Problema diversidad máxima.

El problema de la diversidad máxima (Maximum Diversity Problem, MDP) consiste en seleccionar un subconjunto de m elementos de un conjunto mayor de n elementos de tal forma que la suma de las distancias entre los elementos seleccionados sea máxima. En la mayoría de las aplicaciones, se asume que cada elemento puede representarse por un conjunto de atributos. Si definimos s_{ik} como el valor del atributo k-ésimo del elemento i, donde k=1, …, K.

Para la resolución de este problema se intentarán reutilizar algunos métodos y clases utilizados en esta práctica, como por ejemplo una clase que calcula la distancia euclídea entre dos puntos.