Práctica 1.

Codificación y evaluación de soluciones

Metaheurísticas – Grado de Ingeniería Informática

Universidad de Córdoba

2018 / 2019

El objetivo de esta práctica es el de introducir al alumnado en los primeros pasos para la resolución de problemas NP mediante metaheurísticas. Para ello, se presentarán el problema de la mochila múltiple cuadrática, una descripción de un código esqueleto a completar, y las tareas a realizar sobre dicho código.

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

1 Descripción del problema de la mochila múltiple cuadrática

Supongamos que tenemos *n* objetos $\{o_1,...,o_n\}$. Cada objeto tiene un peso w_i, un beneficio individual por ser introducido en alguna mochila p_i y un beneficio conjunto al ser introducido en la misma mochila que otro objeto p_{ij}. Todas estas cantidades serán no negativas. Por otro lado, se tiene un conjunto de K mochilas de igual capacidad W donde se pueden introducir los objetos. El problema consiste en decidir qué objetos meter en cada mochila de forma que: 1) se maximice la suma de beneficios individuales y conjuntos, y 2) no se supere la capacidad de ninguna mochila.

2 Código esqueleto

Los ficheros proporcionados en la página de la asignatura definen parcialmente las siguientes clases

- `MQKPInstance`: describe los objetos que tienen la información del problema, es decir, los objetos con sus beneficios individuales y conjuntos, sus pesos, las mochilas y sus capacidades. Esta clase, que debéis completar, debe definir las siguientes variables miembro y los siguientes métodos:
  - `numKnapsacks`: Entero que indica el número de mochilas que se va a considerar.
  - `numObjs`: Entero donde se almacenará el número de objetos del problema.

- `profits`: Matriz donde se almacenarán los beneficios de los objetos. La celda `profits[i][j]` deberá tener el beneficio cuadrático de meter los objetos *i* y *j* en la misma mochila. La celda `profits[i][i]` tendrá el beneficio individual de meter el objeto *i* en alguna mochila. La matriz debe ser completa (`profits[numObjs][numObjs]`) y simétrica (`profits[i][j] = profits[j][i]`).
- `weights`: Vector con los pesos de los objetos.
- `capacities`: Vector con las capacidades de las mochilas.
- `MQKPInstance()`: Constructor por defecto.
- `void readInstance(char *filename, int numKnapsacks)`: Función que lee la información de un fichero instancia del problema e inicializa el objeto.
- Funciones que devuelven la información que se consulte acerca de la instancia del problema (número de mochilas, número de objetos, beneficio conjunto de los objetos i y j…).
- `double getMaxCapacityViolation(MQKPSolution &solution)`: Función que devuelve la máxima violación de la capacidad máxima de alguna de las mochilas según la solución candidata provista como argumento.
- `double getSumProfits(MQKPSolution &solution)`: Función que devuelve la suma de los beneficios de los objetos incluidos en las mochilas, tanto los individuales como los conjuntos, según la solución candidata provista como argumento.

- `MQKPSolution`: describe los objetos que representan una solución candidata al problema. La representación interna de las soluciones será un vector de número enteros: de 1 a K para objetos que están en alguna de las K mochilas y 0 para objetos que no están en ninguna mochilas. Esta clase, que debéis completar, debe definir las siguientes variables miembro y los siguientes métodos:
  - `sol`: vector de enteros que será la representación interna de la solución al problema.
  - `numObjs`: Entero donde se almacenará el número de objetos del problema.
  - `fitness`: valor double que almacena la calidad de la solución.
  - `void putObjectIn(int object, int knapsack)`: Función que asigna un objeto a una mochila.
  - `int whereIsObject(int object)`: Función que devuelve la mochila en la que está insertado un objeto.
- `MKQPSolGenerator`: Clase con una función estática para generar una solución aleatoria. Define el siguiente método:
  - `static void genRandomSol(MQKPInstance &instance, MQKPSolution &solution)`: Función que genera una solución aleatoria para el problema de las múltiples mochilas cuadráticas. Importante: el objeto solution debe estar correctamente inicializado, en particular, su vector interno debe haber sido reservado previamente.
- `MQKPEvaluator`: Clase con una función estática para evaluar una solución. Define el siguiente método:
  - `static double computeFitness(MQKPInstance &instance, MQKPSolution &solution)`: Función que calcula el fitness de una solución para

el problema de las múltiples mochilas cuadráticas.

3 Tareas a realizar

Se debe (todas las acciones sobre código tienen el comentario *TODO* en código, que indica que hay tareas por hacer):

1. Descargar e inspeccionar los ficheros de instancias de los problemas disponibles en la página de la asignatura. Entender qué significa cada número en dichos ficheros.
2. Completar las siguientes partes de `MQKPInstance`.
   a. Definir sus variables miembro, constructor y destructor.
   b. Definir su función `readInstance`.
   c. Definir funciones de consulta de la información que contiene.
3. Completar la clase `MKQPSolution`:
   a. Definir sus variables miembro, constructor y destructor.
   b. Definir las funciones `putObjectIn` y `whereIsObject`.
4. Repasar y **corregir** la clase `MQKPSolGenerator`.
5. Terminar la clase `MQKPInstance`.
   a. Completar la función `getMaxCapacityViolation` y `getSumProfits`.
6. Completar la clase `MQKPEvaluator`.
7. Implementar un programa que, recibiendo una lista de pares fichero instancia del problema y número de mochilas, realice la siguiente experimentación en cada uno de ellos (el programa ya está implementado en main.cpp):
   a. Repita 5 veces, con 5 semillas aleatorias para el generador de números aleatorios:
      1. Generar soluciones aleatorias durante 5 segundos (máximo 100.000 soluciones). Si todo es correcto, toda la experimentación debe tardar menos de 200 segundos.
   b. Calcule el valor medio de las 5 repeticiones por cada iteración de la generación de soluciones aleatorias y el mismo para los mejores valores encontrados hasta el momento, es decir: {(media-iteracion-1, media-mejor-hasta-iteracion-1), (media-iteracion-2 y media-mejor-hasta-iteracion-2)...}. Para las ejecuciones que realizaron menos iteraciones que las otras, se debe repetir el último valor producido.
8. Ejecuta el programa en las instancias provistas con 3 y 5 mochilas, guardando los resultados en uno o varios ficheros.
9. Dibujar dicha información en gráficas de convergencia para cada instancia del problema y número de mochilas.
10. Crear un documento **PDF** describiendo la experimentación realizada, el código incluido en el orden indicado en esta sección (puntos 2 a 7), y analizando los resultados obtenidos.

11. Desarrollar los primeros pasos para la codificación y evaluación de soluciones del problema seleccionado, diferente al MQKP.
12. Incluir una descripción del paso anterior en el documento a entregar (A entregar en la práctica 2).
13. Subir el documento a moodle.
14. En la evaluación del las prácticas de otros compañeros, deberéis proveer una pequeña retroalimentación de los errores que encontréis y una calificación de 0 a 10. Para dicha calificación, podéis guiaros de la siguiente rúbrica.

	Elemento a considerar	Posibilidades			
		Peor ----------> Mejor			
Penalizaciones	¿Es un documento PDF?	Sí - *se corrige*		No - *se pone un 0*	
	¿Tiene faltas de ortografía o gramaticales?	Con faltas de ortografía - *La nota máxima será 6*	Con alguna falta - *Se penaliza un poco*		Sin faltas y bien estructurado y escrito
	¿El documento contiene lo que se pide en el apartado 11 y en el orden correcto?	No es completo - *La nota máxima que se puede obtener sería un 8*	Es completo y desordenado - *se quitan 2 puntos*		Completo y ordenado
A evaluar	¿El código del apartado 2 es correcto y legible?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Es correcto y legible el código del apartado 3?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Es correcto y legible el código del apartado 4?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Es correcto y legible el código del apartado 5?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Es correcto y legible el código del apartado 6?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Es correcto y legible el código del apartado 7?	Poco legible e incorrecto	Legible pero incorrecto	Correcto pero poco legible	Correcto y legible
	¿Son correctos los resultados presentados y el análisis de éstos?	Son incorrectos	Hay algo raro	Son correctos	Son correctos e interesan-tes
	¿Son correctos los pasos realizados para el apartado 12? (A entregar en la práctica 2)	Yo diría que no	Demasiado genérico y no entiendo si son correctos	Parece que sí, pero deja alguna duda	Está muy claro y concreto y yo diría que sí es correcto