Práctica opcional de ramificación y poda: “Exámenes de Juanita”

Pedro Simarro Guerra

Índice

1. Introducción

En este proyecto se desarrolla la implementación del algoritmo de ramficación y poda que resuelve el problema de los exámenes de Juanita, el cual dadas unas asignaturas y sus beneficios, decide cuáles de ellas se deben estudiar para una ciertas fechas ordenadas ascendentemente, obteniendo el máximo beneficio posible.

2. Estructura del algoritmo

El algoritmo sigue la estructura típica de un algoritmo de ramificación y poda.

2.1 Árbol

El árbol de exploración del problema es binario pues para cada posible asignatura solo tenemos las opciones de estudiarla o no. Tiene *n* nives siendo *n* el número de asignaturas del problema.

2.2 Nodo

La estructura del nodo viene dada en la propia clase *Nodo* que recoge la solución parcial, el nivel k de estudio del árbol, los días acumulados de estudio hasta la asignatura k, el valor acumulado hasta la asignatura k, y el valor optimista para ese nodo.

2.3 Factibilidad

En primer lugar, las asignaturas están ordenadas por orden de fecha. Dada una asignatura *k*, esta es factible si cumple con la fecha *k;* es decir, si los días acumulados de estudio hasta *k-1* más los días de estudio de la asignatura *k* son menores que el valor de la fecha *k*.

2.3 Cotas

Las cotas para este algoritmo siguen dos estrategias, una lineal y otra más compleja voraz.

2.3.1 Cota lineal

Para la cota optimista se acumulan los valores de las asignaturas que quedan por procesar siempre y cuando sean factibles. La cota pesimista añade la restricción de que las asignaturas que queden por procesar no interfieran en las fechas de las demás; para ello, se van acumulando los días de la asignaturas que se cogen. El coste de cota es lineal en el peor de los casos.

2.3.2 Cota voraz

Las asignaturas que quedan por procesar se ordenan por densidad de valor descendente (*v[i]/d[i]*) en un árbol binario. De esta forma se cogen primero las asignaturas que más valen por el tiempo de estudio, consiguiendo, posiblemente, unas cotas más ajustadas. De una forma parecida a la cota lineal, la cota optimista va tomando las asignaturas ordenadas de esta forma mientras sean factibles; además, fracciona la el valor de la última asignatura tomada para “aprovechar” los días restantes de estudio. La cota pesisimista va acumulando los días de las asignaturas que se van cogiendo y elige si son factibles. El coste de esta cota es del orden $n \cdot log(n)$ pues el acceso al árbol de densidades está en $log(n)$ y vamos sacando una tupla cada vez que procesamos una asignatura.

3. Estructura del código

El código del proyecto se compone de los siguientas clases:

3.1 Main

Clase que contiene el método *main* del proyecto. Aquí se construyen los set de datos dados por fichero o aleatoramiente y se lanza el algoritmo. El set de datos se recoge en una clase estática privada *SetDatos*.

3.2 ExamJuanita

Clase principal del proyecto. Contiene el método *rp_examenes* que es el algoritmo de ramificación y poda en sí, además de los métodos que calculan las cotas.

3.3 Nodo

Clase que implementa el nodo. Tiene como atributos las variables descritas en el punto 2.2.

3.4 Solucion

Clase que recoge una solución del problema. Sus atributos se explican a continuación en el punto 4.

3.5 MyThread

Clase que implementa un thread simple que ejecuta el algoritmo. Además recoge la solución con el tiempo transcurrido y los nodos explorados.

4. Parámetros de entrada y salida

Los parámetros de entrada del algoritmo son:

- *Integer d[]* : vector de enteros tal que *d[i]* representa los días necesarios para estudiar la asignatura *i*.
- *Float v[]* : vector de floats tal que *v[i]* es el valor de la asignatura *i.*
- *Integer f[]* : *f[i]* marca la fecha de examen para la asignatura *i*.
- *TreeMap<Float, Integer> vi_di* : árbol binario ordenado ascendentemente que guarda tuplas clave-valor (*v[i]/d[i]*, *i*) siendo *i* el número de la asignatura. Este árbol básicamente ordena la asignaturas por densidad de valor y es necesario para el cálculo de la cota voraz.

Los parámetros de salida son:

- *Solucion s* que se compone de:
  - *boolean sol[]* : array de booleanos tal que *sol[i]* indica si se estudia o no la asignatura *i*.
  - *int nodosExplorados* : entero que guarada los nodos explorados del algoritmo.
  - *float benefMejor* : máximo beneficio alcanzado.

5. Uso

La aplicación utiliza dos opciones dependiendo de si se quiere ejeucutar el algortimo para un set de datos dado por fichero o aleatorio. La solución del algoritmo se guarda en un archivo *resul.txt* .

- *-e FILE* : se lee el fichero *FILE* que debe ser de formato *.csv*, se genera un set de datos a partir de él y se ejecuta el algoritmo para dicho set.
- *-r <n>* : se crea un set de datos aleatorio de tamaño *n* y se ejecuta el algoritmo. El set de datos se guarda en un arhivo *random.csv* .

6. Resultados y conclusiones

Si ejecutamos el algoritmo para los ejemplos propuestos (n=30), el uso de las cotas, tanto la de estrategia lineal como voraz, mejora drásticamente el número de nodos explorados y el tiempo de ejecución. Tiene sentido pues, sin el uso de alguna cota, se dejan muchos nodos como prometedores ya que la función de factibilidad por si sola no restringe demasiado.
En cuanto a la comparación de las cotas, la voraz, al ser más ajustada, hace un mejor uso de la memoria con respecto a la lineal pues se exploran menos nodos. Sin embargo, si nos fijamos en el tiempo de ejecución, esta se comporta más lenta dado el cálculo más complejo, como se puede ver tanto en el tiempo total transcurrido como el tiempo por nodo. Esto concuerda con el planteamiento inicial puesto que la complejidad del algoritmo de la cota voraz es mayor que el de la lineal.
Para casos en los que n es menor y dependiendo del caso, el uso de la cota voraz no mejora la cota lineal en ningún aspecto, por tanto su uso tiene más sentido cuando el tamaño del árbol de exploración crece significativamente.