Práctica 2

Algoritmos heurísticos

Algoritmos Avanzados

Víctor Fernández Fernández
Mikayel Mardanyan Petrosyan

6 de Octubre de 2019

1 Especificación del problema

Un equipo de informáticos contrata y realiza tareas semanalmente durante un periodo de n semanas. Las tareas pueden ser de alta tensión o de baja tensión. Una tarea de baja tensión en la semana i-ésima da un beneficio de $b_i > 0$ (euros) y puede contratarse siempre, mientras que una de alta tensión da un beneficio de $a_i > 0$ y solo puede contratarse si se ha descansado la semana anterior (excepto en la primera semana, en la cual se puede contratar cualquier tarea). Las tareas de alta tensión suelen dar mayor beneficio, pero esto no es así necesariamente.

El objetivo del *problema del plan de trabajos estresantes de beneficio máximo* es decidir qué secuencia de trabajos deben contratarse en n semanas para obtener un beneficio máximo. Para ello, vamos a implementar y analizar una serie de algoritmos heurísticos.

Los datos de entrada del algoritmo serán dos vectores llamados a y b que contendrán los beneficios de las actividades de alto nivel y de bajo nivel de cada semana, respectivamente. Los componentes de la especificación son los siguientes:

- Precondición:

  El tamaño de a y de b es n y todos los valores son mayores que cero:

  $a_i > 0, \quad b_i > 0, \quad 0 \leq i \leq n-1$

- Poscondición:

  - Condición de validez:

    $c_i \in \{0, 1\}$ vale 1 cuando contratamos la actividad de alta tensión en la semana i, y 0 en caso contrario.

    $d_i \in \{0, 1\}$ vale 1 cuando contratamos la actividad de baja tensión en la semana i, y 0 en caso contrario.

Solo se puede elegir una tarea como máximo cada semana:
$c_i + d_i \leq 1, \quad 0 \leq i \leq n-1$
Si contratamos una actividad de alta tensión tenemos que descansar la semana anterior (excepto la primera semana):
$c_i = 1 \implies c_{i-1} + d_{i-1} = 0, \quad 1 \leq i \leq n-1$

– Función objetivo:

$$max \sum_{i=0}^{n} c_i \cdot a_i + d_i \cdot b_i > 0 \tag{1}$$

2 Implementación de los algoritmos

La cabecera del método será:

```
public static int trabajosH (int[] a, int[] b)
```

donde $a[i]$ es el precio del trabajo de alta tensión de la semana i-ésima, $0 \leq i \leq n-1$, y $b[i]$ es el precio del trabajo de baja tensión de la misma semana.

2.1 Algoritmo heurístico del ejemplo

El algoritmo compara las semanas de dos en dos. Compara la suma de los beneficios de las tareas de baja tensión de ambas semanas con el beneficio de la tarea de alta tensión de la segunda semana. De este modo, nunca tiene en cuenta las tareas de alta tensión de las semanas impares (contando las semanas desde 1) y se ahorra la comprobación de si se ha descansado la semana anterior a coger una tarea de alta tensión, ya que las tareas de alta tensión que se contratan caen en semanas pares precedidas de una semana impar de descanso. En caso de que el número de semanas sea impar, siempre contrataremos en la última semana una tarea de bajo coste debido a que en la semana anterior (par) nunca se descansa.

```
public static int trabajosH1 (int[] a, int[] b) {

     int beneficio = 0;
     int i;
     for (i = 0; i < a.length-1; i+=2)
         beneficio += Math.max(b[i]+b[i+1], a[i+1]);
     if (a.length % 2 != 0)
         beneficio += b[i];
     return beneficio;
}
```

Este criterio conduce a soluciones optimales porque toma el valor máximo del trabajo de dos semanas, buscando el beneficio máximo.

2.2 Algoritmo heurístico 2

El criterio de este algoritmo consiste en tomar la tarea que mayor beneficio da en la primera semana porque al ser la primera no es necesario prescindir de trabajar la semana anterior (porque no hay semana antes de la primera). A continuación, elegimos siempre las tareas de baja tensión porque estas no requieren tiempo de descanso, por lo que podemos realizar más tareas, y no dan necesariamente menos beneficio que las de alta tensión.

En conclusión, trabajar más y realizar aproximadamente el doble de tareas con una recompensa algo menor da mayor beneficio y conduce a soluciones óptimas.

```
public static int trabajosH2 (int[] a, int[] b) {

     int beneficio = Math.max(b[0], a[0]);
     for (int i = 1; i < a.length; i++)
         beneficio += b[i];
     return beneficio;
 }
```

2.3 Algoritmo heurístico 3

En este algoritmo aprovechamos lo mejor de los dos algoritmos anteriores: la primera heurística da un buen balance independientemente de la diferencia entre el beneficio de las tareas de alta y de baja tensión, y la segunda aprovecha que la primera semana se puede elegir cualquiera de las dos tareas, pues no hay que tener en cuenta la semana anterior. El resultado es un algoritmo cuyo criterio es elegir la tarea con más recompensa en la primera semana y posteriormente escoger las tareas con mayor beneficio acumulado cada dos semanas.

```
public static int trabajosH3 (int[] a, int[] b) {

     int beneficio = Math.max(b[0], a[0]);
     int i = 1;
     for (i = 0; i < a.length-1; i+=2)
         beneficio += Math.max(b[i]+b[i+1], a[i+1]);
     if (a.length % 2 == 0)
         beneficio += b[i];
     return beneficio;
 }
```

2.4 Algoritmo heurístico 4

Esta vez presentamos un algoritmo con un criterio de elección diferente al resto. El criterio voraz que sigue este algoritmo es elegir siempre la tarea de alta tensión y en el caso de que el número de semanas sea par, en la última semana elegir

la tarea de baja tensión. Este criterio puede conducir a soluciones optimales porque las tareas de alta tensión en muchos casos producen mayor beneficio.

```
public static int trabajosH4 (int[] a, int[] b) {

     int beneficio = 0;
     int i;
     for (i = 0; i < a.length; i+=2)
         beneficio += a[i];
     if (i == a.length)
         beneficio += b[i-1];
     return beneficio;
 }
```

Esta heurística conducirá a soluciones óptimas cuando la diferencia entre los beneficios de las tareas de alta y baja tensión sea alta.

3 Experimentación con la optimalidad de los algoritmos

En este apartado vamos a comparar con OptimEx la optimalidad de los algoritmos desarrollados. Para ello hemos realizado lo siguiente:

1. Cargamos la clase Java y la compilamos. Si tenemos problemas debemos revisar la ruta de la máquina virtual de Java en la sección de configuración.

2. Seleccionamos el problema. Indicamos que el criterio es la optimalidad y que el objetivo es la maximización, ya que queremos maximizar el beneficio. Dejamos las dos últimas casillas en blanco.

3. A continuación, podemos seleccionar los métodos con los que queremos experimentar y marcar si alguno de estos es exacto. Esto es opcional y solo debe hacerse si se está seguro de que es exacto. En nuestro caso, no realizamos este paso porque queremos experimentar con todos los algoritmos y no sabemos si son exactos.

4. Generamos los datos de entrada aleatoriamente y especificamos las restricciones de los valores:

   - Número de juegos de datos: 100
   - Tamaño de los vectores de entrada (el número de semanas): 8
   - Rango de beneficio de tareas de alta tensión: 10-30
   - Rango de beneficio de tareas de baja tensión: 5-20

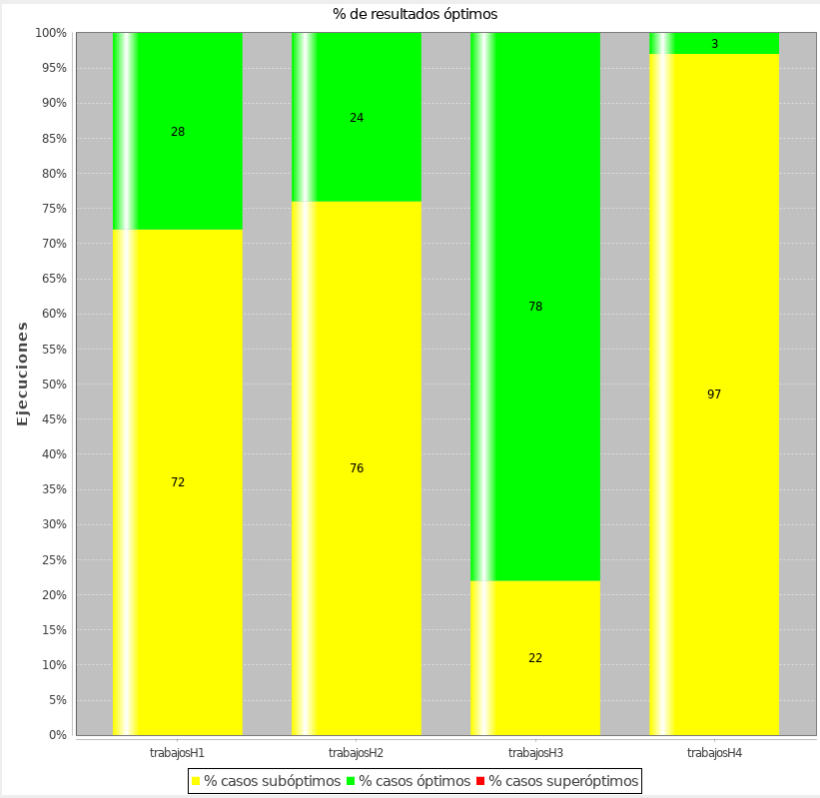
   Generamos un número alto de juegos para que los datos de salida sean lo

más realista posibles [1] . Si nos fijamos, los rangos especificados coinciden con la especificación del problema: no hay tareas con beneficio negativo y los vectores no están vacíos y tienen la misma longitud. El rango de beneficios no está especificado explícitamente, hay que intuirlo a partir de los datos del ejemplo.

5. Ejecutamos el análisis. Los datos de entrada generados cumplen todas las restricciones y se seleccionan automáticamente.

6. Analizamos los datos obtenidos. En nuestro caso son los siguientes:

Medidas	trabajosH1	trabajosH2	trabajosH3	trabajosH4
Núm. ejecuciones	100	100	100	100
Núm. ejec. válidas del método	100	100	100	100
Núm. ejec. válidas en total	100	100	100	100
% soluciones subóptimas	72 %	76 %	22 %	97 %
% soluciones óptimas	28 %	24 %	78 %	3 %
% soluciones sobreóptimas	0 %	0 %	0 %	0 %
% valor medio subóptimo	-	-	-	-
% valor extremo subóptimo	-	-	-	-
% valor medio sobreoptimo	-	-	-	-
% valor extremo sobreóptimo	-	-	-	-



[1] Con juegos de datos pequeños los algoritmos inexactos pueden producir resultados que se acerquen mucho a los de algoritmos exactos, por lo que hay que usar un número de juegos mayor para no obtener falsos positivos.

El número de ejecuciones válidas es el esperado y no se han producido soluciones sobreóptimas, por lo que no ha habido fallos en la ejecución ni hay errores en el algoritmo. Ninguno de los algoritmos da un porcentaje de soluciones óptimas cercano al 100%, por lo que ninguno es exacto. Podemos ver en la tabla y también en el gráfico que el tercer algoritmo es el que más se acerca al exacto, seguido a bastante distancia por el primero y el segundo, y por último, también a distancia, el cuarto algoritmo.

Como ninguno es exacto procedemos a mostar los contraejemplos:

- Algoritmo 1

  Encontrado Alta tensión:{29,29,12,13,20,24,13,18}, baja tensión:{13,7,8,17,20,14,15,12}.

  Deducido Alta tensión:{30,30,20,25}, baja tensión:{15,20,10,10}.

  El fallo de este algoritmo es que no intenta asignar una tarea de alta tensión en la primera semana, y cuando la tarea de alta tensión de la primera semana es mayor que la de baja tensión, no se llega a la solución óptima. En nuestro ejemplo, se contratarían las tareas 1 y 2 de baja tensión y la 4 de alta tensión, dando un beneficio de 55. En cambio, si contratamos la tarea de alta tensión de la primera semana y el resto de tareas de baja tensión obtenemos un beneficio de 70.

- Algoritmo 2

  Encontrado Alta tensión:{11,11,23,29,25,14,30,23}, baja tensión:{9,8,13,9,14,7,18,13}

  Deducido Alta tensión:{30,20,30,20}, baja tensión:{10,5,10,10}

  En este caso, al ser mayor la diferencia entre los beneficios de las tareas de alta y baja tensión, merece la pena elegir las de alta tensión. En nuestro ejemplo, se contrataría la tarea de alta tensión de la primera semana y las de baja del resto, obteniendo 55 de beneficio. Habría sido mejor elegir las tareas de alta tensión de las semanas 1 y 3 y la tarea de baja tensión la última semana, obteniendo un beneficio de 70.

- Algoritmo 3

  Encontrado Alta tensión:{11,11,24,26,20,29,26,19}, baja tensión:{18,19,17,14,5,6,14,6}

  Deducido Alta tensión:{10,30,15,25,20}, baja tensión:{10,10,5,15,10}

  El motivo por el que este algoritmo no consigue siempre la solución óptima es que no tiene en cuenta los trabajos de alta tensión de las semanas pares, por lo que si las tareas de mayor beneficio se concentran en estas semanas, la solución dada será subóptima. En nuestro ejemplo escogería cualquier tarea de la primera semana y el resto de tareas de baja tensión, obteniendo 50 de beneficio. Sin embargo, como en las semanas pares hay trabajos con alto beneficio, si elegimos las de alta tensión las semanas 2 y 4 ya obtenemos 55.

- Algoritmo 4

  Encontrado Alta tensión:{12,24,22,11,17,20,14,18}, baja tensión:{11,9,19,12,20,19,10,17}

  Deducido Alta tensión:{10,10,10,10,10}, baja tensión:{10,10,10,10,10}

  El fallo de este algoritmo es que no tiene en cuenta las tareas de baja tensión porque se basa en la suposición de que estas producen siempre menor beneficio que las de alta tensión. En cuanto no se cumple esta premisa, los resultados están muy lejos de ser los óptimos. Si la diferencia de beneficio entre ambos tipos de tareas es cero, el algoritmo falla y, en nuestro caso, devuelve un beneficio de 30 en vez de 50.

4 Conclusión

Como hemos podido observar tras la experimentación con OptimEx, ninguno de los algoritmos es exacto. Sin embargo, podemos ver que una buena heurística ofrece buenos resultados con coste bajo de complejidad en tiempo, como la heurística del algoritmo 3.
La realización de la práctica ha sido en general sencilla, siendo la especificación del problema la parte más complicada de realizar. Aún así, esta y la implementación han sido las más interesantes.