Práctica 3b

Técnicas de búsqueda: Ramificación y poda

Algoritmos Avanzados

Víctor Fernández Fernández
Mikayel Mardanyan Petrosyan

27 de Octubre de 2019

0 Problema

Esta práctica es una continuación de la práctica 3a y trata el mismo *problema del plan de trabajos estresantes de beneficio máximo* [1].

Para lograr el objetivo del problema vamos a desarrollar de forma sistemática un algoritmo de ramificación y poda que resuelva el problema y comprobar experimentalmente no solo su optimalidad sino también su eficiencia. Esta última comprobación es nueva y se realiza debido a que el algoritmo de ramificación y poda debe obtener el mismo resultado que el de vuelta atrás pero de forma más eficiente.

[1] En el *problema del plan de trabajos estresantes de beneficio máximo* 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 ($0 \leq i \leq n-1$) 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.
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. 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.

1 Técnica de ramificación y poda

En la práctica, un algoritmo de ramificación y poda es un algoritmo de vuelta atrás con el añadido de una función de cota. Por ello, para desarrollar sistemáticamente un algoritmo de ramificación y poda debemos realizar los pasos llevados a cabo en la práctica 3a (diseñar un árbol de búsqueda y especificar la condición de validez que se realizará en cada nodo del árbol) y, además, diseñar la función de cota mencionada que podará los nodos que no sean prometedores [2], reduciendo el número de nodos generados y el tiempo de ejecución. Por último, se debe implementar el algoritmo acorde al diseño anterior.

Ya que esta es una segunda parte de la práctica 3, no realizaremos los pasos propios de la vuelta atrás y pasaremos a los no realizados.

1.1 Función de cota

Para cada una de las funciones propuestas debemos indicar: la definición de la función de cota en términos de los parámetros del problema, el valor inicial de la cota y cómo se actualiza su valor en cada nodo del árbol de búsqueda. Al ser un problema de maximización tenemos que buscar una cota superior. La cota inicial puede ser infinito o un valor alto calculado en función de los datos de entrada que solo se pueda alcanzar en el caso ideal.

1.1.1 Función de cota 1: Suma de los beneficios

Esta función de cota es optimista en cuanto al beneficio máximo, ya que supone que podemos realizar en una misma semana tanto el trabajo de alta tensión como el de baja. No es realista.

Se define, para cada nodo del árbol de búsqueda, como la suma de los beneficios de los trabajos contratados más la suma de todos los trabajos de las semanas que aún no han sido planificadas.

$$f(i) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + \sum_{j=i}^{n-1} (a_j + b_j) \tag{1}$$

donde i es el índice del nodo actual, n es el número de semanas, a_j y b_j son los costes de los trabajos de alta y baja tensión, respectivamente, de la semana j; y c_j y d_j valen 1 cuando se contratan los trabajos de alta y baja tensión, respectivamente, en la semana j, o 0 en caso contrario.

[2] Llamamos nodos prometedores a aquellos que pueden ofrecer una solución mejor que alguna ya obtenida.

El valor inicial de la cota será la suma de todos los beneficios de alta y baja tensión y su valor se actualizará restando de la cota el beneficio del trabajo descartado.

$$f(0) \equiv \sum_{j=0}^{n-1} (a_j + b_j) \tag{2}$$

$$f(i+1) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + (a_i \cdot c_i + b_i \cdot d_i) + \sum_{j=i+1}^{n-1} (a_j + b_j) \tag{3}$$

1.1.2 Función de cota 2: Máximo de los beneficios

Esta función de cota también es optimista en cuanto al beneficio máximo, aunque menos que la primera, ya que supone que siempre podemos realizar el trabajo de mayor beneficio independientemente de si se ha descansado. Por ello, no es realista.

Se define, para cada nodo del árbol de búsqueda, como la suma de los beneficios de los trabajos contratados más la suma de los trabajos que mayor beneficio dan en cada semana de las restantes.

$$f(i) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + \sum_{j=i}^{n-1} (max\,(a_j, b_j)) \tag{4}$$

donde i es el índice del nodo actual, n es el número de semanas, a_j y b_j son los costes de los trabajos de alta y baja tensión, respectivamente, de la semana j; y c_j y d_j valen 1 cuando se contratan los trabajos de alta y baja tensión, respectivamente, en la semana j, o 0 en caso contrario.

El valor inicial de la cota será la suma de los beneficios máximos de cada semana y su valor se actualizará restando de la cota el beneficio del trabajo descartado.

$$f(0) \equiv \sum_{j=0}^{n-1} (max\,(a_j, b_j))$$

$$f(i+1) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + (a_i \cdot c_i + b_i \cdot d_i) + \sum_{j=i+1}^{n-1} (max\,(a_j, b_j))$$

1.1.3 Función de cota 3: Solución del algoritmo voraz

Si buscáramos una cota menos optimista aún, podríamos usar alguna de las heurísticas diseñadas en la práctica 2. No vamos a detenernos a explicar cuál

de las heurísticas es mejor, todas valdrían, pero si quisiéramos elegir una solo deberíamos buscar en la memoria de la práctica 2 cuál ofrece mejores resultados para los datos dados.

Se define, para cada nodo del árbol de búsqueda, como la suma de los beneficios de los trabajos contratados más el beneficio estimado por la heurística para las semanas restantes.

$$f(i) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + heuristica(a', b') \tag{5}$$

donde i es el índice del nodo actual, a' y b' son los costes de alta y baja tensión, respectivamente, de las semanas restantes; a_j y b_j son los costes de los trabajos de alta y baja tensión, respectivamente, de la semana j; y c_j y d_j valen 1 cuando se contratan los trabajos de alta y baja tensión, respectivamente, en la semana j, o 0 en caso contrario.

El valor inicial de la cota será la estimación inicial de la heurística y su valor se actualizará recalculando la estimación de la heurística para las semanas restantes y sumándole el beneficio de los trabajos ya contratados.

$$f(0) \equiv heuristica(a, b)$$

$$f(i+1) \equiv \sum_{j=0}^{i-1} (a_j \cdot c_j + b_j \cdot d_j) + (a_i \cdot c_i + b_i \cdot d_i) + heuristica(a', b')$$

Es importante que la heurística calcule una estimación solo para las semanas restantes y que tenga en cuenta si se ha trabajado en la semana actual para saber si se puede contratar un trabajo de alta tensión en la primera semana de las restantes.

1.2 Implementación

La cabecera del método principal del algoritmo es:

```
public static int trabajosRyP (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. El algoritmo devuelve el plan óptimo de trabajos y el beneficio producido.

De las cotas anteriores hemos seleccionado la segunda porque queríamos una cota que no fuera muy optimista, para que el tiempo de ejecución fuese menor, y que no podara ramas prometedoras.

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

     int k = 0;  // nivel del arbol
     int n = a.length;  // numero de semanas
     int[] solActual = new int[n];  // array de
soluciones intermedias
     int[] solFinal = new int[n];  // array de solucion
final
     int bFinal = 0;  // beneficio final

     solFinal = contratarRyP(k, n, a, b, solActual,
solFinal, bFinal);
     imprimir(a, b, solFinal);

     return calcularBeneficio(a, b, solFinal);
 }

 private static int[] contratarRyP(int k, int n, int[]
a, int[] b, int[] solActual, int[] solFinal, int
bFinal) {

     /*exploracion de todos los nodos hijos posibles*/
     for (int i = 0; i < 3; i++) {

         /*condicion de validez*/
         if (k == 0 || i == 1 && solActual[k-1] == 0 ||
i != 1) {
             int cota = cotaSuperior(k, a, b,
solActual);
             if (cota > calcularBeneficio(a, b,
solFinal)) {
                 solActual[k] = i;
                 if (k < n - 1)  // nodos intermedios
                     solFinal = contratarRyP(k + 1, n,
a, b, solActual, solFinal, bFinal);
                 else {  // nodos hoja
                     int bActual = calcularBeneficio(a,
b, solActual);
                     bFinal = calcularBeneficio(a, b,
solFinal);
                     if (bActual > bFinal) {
                         solFinal = solActual.clone();
                         bFinal = bActual;
```

```
                    }
                }
            }
        }
    }

    return solFinal;
}

 private static int cotaSuperior(int k, int[] a, int[]
b, int[] solActual) {
    int cota = 0, i;
    for (i = 0; i < k; i++)
        cota += solActual[i] == 1 ? a[i] :
solActual[i] == 2 ? b[i] : 0;
    for (; i < a.length; i++)
        cota += Math.max(a[i], b[i]);

    return cota;
}

 private static int calcularBeneficio(int[] a, int[] b,
int[] solActual) {

    int beneficio = 0;
    for (int i = 0; i < solActual.length; i++) {
        if (solActual[i] == 1)
            beneficio += a[i];
        else if (solActual[i] == 2)
            beneficio += b[i];
    }

    return beneficio;
}
```

Para la entrada del ejemplo con $a = \{10, 15, 15, 15, 15\}$ y $b = \{10, 10, 10, 10, 10\}$ tenemos la siguiente salida:

```
El plan óptimo es:
Semana     |  0  |  1  |  2  |  3  |  4  |
Actividad  |  A  |  B  |  B  |  B  |  B  |
Beneficio  |  10 |  10 |  10 |  10 |  10 |

El beneficio del plan óptimo es 50.

Process finished with exit code 0
```

Donde para cada semana se indica si se descansa (D), se contrata un trabajo de alta tensión (A) o se contrata un trabajo de baja tensión (B) y el correspondiente beneficio.

2 Comparación de optimalidad

2.1 Material del experimento

Para experimentar con la optimalidad de distintos algoritmos sobre este problema hemos utilizado los algoritmos usados en la práctica 3a y el algoritmo presentado en el apartado anterior. Estos son:

- Algoritmo heurístico 1 (trabajosH1). 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.
- Algoritmo heurístico 2 (trabajosH2). En la primera semana toma la tarea de alta tensión y en las semanas restantes la tarea de baja tensión.
- Algoritmo heurístico 3 (trabajosH3). En la primera semana toma la tarea de alta tensión y en las restantes utiliza la función de selección del algoritmo heurístico 1.
- Algoritmo heurístico 4 (trabajosH4). Elige siempre la tarea de alta tensión y si queda una semana libre al final se toma la tarea de baja tensión.
- Algoritmo de vuelta atrás (trabajosVA). Explora el árbol de búsqueda completo y encuentra la solución óptima.
- Algoritmo de ramifica y poda (trabajosRyP). Explora el árbol de búsqueda completo podando las ramas no prometedoras y encuentra la solución óptima de forma más eficiente.

Los valores de experimentación han sido generados con las siguientes restricciones:

- Número de juegos de datos: 1000
- Tamaño de los vectores de entrada (número de semanas): 8
- Rango de beneficio de tareas de alta tensión: 10-30
- Rango de beneficio de tareas de baja tensión: 5-20

2.2 Conclusión

Como se puede ver en la evidencia presentada en el apartado siguiente y según lo esperado, el algoritmo de vuelta atrás y el algoritmo de ramifica y poda son exactos.

2.3 Evidencias

En las figuras 1 y 2 podemos ver los resultados de la experimentación con los algoritmos presentados antes. El algoritmo de vuelta atrás y el algoritmo de ramificación y poda son los únicos exactos, los únicos que obtienen un 100% de soluciones óptimas. Que no haya soluciones sobreóptimas es una señal de que los algoritmos no tienen fallos.

Medidas	trabajosH1	trabajosH2	trabajosH3	trabajosH4	trabajosRyP	trabajosVA
Núm. ejecuciones	1000	1000	1000	1000	1000	1000
Núm. ejec. válidas del método	1000	1000	1000	1000	1000	1000
Núm. ejec. válidas en total	1000	1000	1000	1000	1000	1000
% soluciones subóptimas	86 %	81,50 %	50 %	98,40 %	0 %	0 %
% soluciones óptimas	14 %	18,50 %	50 %	1,60 %	100 %	100 %
% soluciones sobreóptimas	0 %	0 %	0 %	0 %	0 %	0 %
% valor medio subóptimo	90,87 %	92,11 %	94,89 %	79,90 %	0 %	0 %
% valor extremo subóptimo	66,14 %	73,17 %	74,17 %	51,85 %	0 %	0 %
% valor medio sobreoptimo	0 %	0 %	0 %	0 %	0 %	0 %
% valor extremo sobreóptimo	0 %	0 %	0 %	0 %	0 %	0 %

Figure 1: Tabla de resultados numéricos

En el primer gráfico de la figura 2 se puede apreciar que el algoritmo de vuelta atrás y el algoritmo de ramifica y poda siempre ofrecen resultados óptimos. En el segundo gráfico, el rectángulo ofrece información sobre el valor medio devuelto en los casos subóptimos y, la línea delgada, sobre la desviación máxima de los casos subóptimos. Este gráfico muestra que, aunque los algoritmos heurísticos no encuentren siempre la solución optimal, ofrecen una solución bastante cercana. El algoritmo de vuelta atrás y el algoritmo de ramifica y poda no tienen resultados subóptimos, ya que son exactos, y por ello no se muestra información sobre ellos.

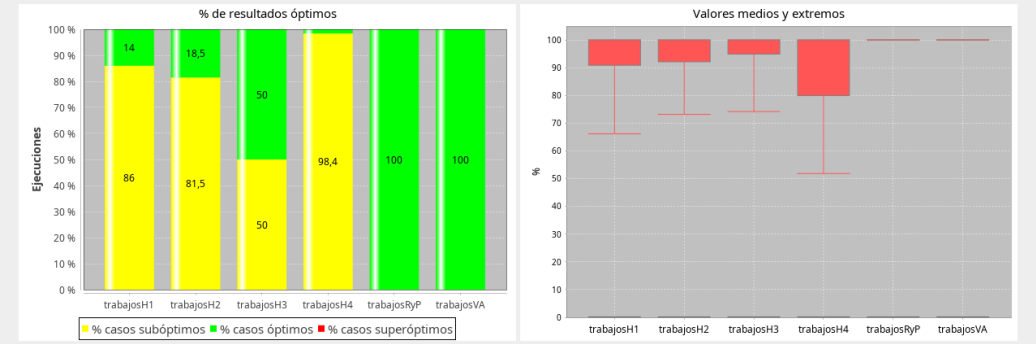

Figure 2: Gráficos

3 Comparación de eficiencia en tiempo

En las dos últimas prácticas hemos realizado únicamente comprobaciones de la optimalidad, esta vez vamos a experimentar también con los tiempos de ejecución de los algoritmos para comparar la eficiencia. Para ello, vamos a usar los mismos datos de entrada que en la sección anterior y vamos a seleccionar en Optimex 'Eficiencia' como criterio de experimentación en las opciones de selección de problema del menú 'Ejecutar'.

3.1 Conclusión

Los algoritmos de vuelta atrás y de ramifica y poda ofrecen las mismas soluciones a un determinado problema. La diferencia es que el algoritmo de ramifica y poda es una optimización del algoritmo de vuelta atrás. La función de cota detecta qué nodos no son prometedores y poda estas ramas, reduciendo el número de nodos a explorar y, con ello, el tiempo de ejecución. Esto da como resultado un aumento de la eficiencia en tiempo.

3.2 Evidencias

En la figura 3 podemos ver los resultados numéricos de la eficiencia de los algoritmos. Observamos que los tiempos de las heurísticas son mucho menores que los tiempos de los dos últimos algoritmos. Esto se debe a que los algoritmos heurísticos no exploran el árbol de búsqueda por completo. Sin embargo, encontramos la diferencia más notoria entre los algoritmos de vuelta atrás y de ramifica y poda. Como hemos comentado, este último explora un menor número de nodos, lo cual resulta en un menor tiempo de ejecución tanto en tiempos mínimos como medios y máximos. Cuanto más optimista sea la cota, menor será la diferencia de tiempos entre ambos algoritmos, y cuanto

menos optimista sea la cota, mayor será la diferencia. Sin embargo, una cota demasiado pesimista reducirá mucho el tiempo de ejecución a costa de podar las ramas que produzcan la solución óptima. De este modo, el algoritmo dejaría de ser exacto y es algo que debemos evitar.

Medidas	trabajosH1	trabajosH2	trabajosH3	trabajosH4	trabajosRyP	trabajosVA
Núm. ejecuciones	1000	1000	1000	1000	1000	1000
Núm. ejec. válidas del método	1000	1000	1000	1000	1000	1000
Núm. ejec. válidas en total	1000	1000	1000	1000	1000	1000
Tiempo máximo	0.313 ms	90.542 ms	0.303 ms	0.300 ms	487.485 ms	508.229 ms
Tiempo medio	0.044 ms	0.128 ms	0.041 ms	0.036 ms	316.550 ms	339.910 ms
Tiempo mínimo	0.029 ms	0.027 ms	0.029 ms	0.026 ms	227.670 ms	229.920 ms

Figure 3: Gráficos

En la figura 4 observamos gráficamente las diferencias comentadas anteriormente. Apreciamos la enorme diferencia en cuanto a los tiempos de los algoritmos heurísticos y los de vuelta atrás y ramifica y poda. Entre estos dos últimos, vemos que hay poca diferencia entre los tiempos mínimos, pero que los tiempos medios y máximos se ven reducidos.

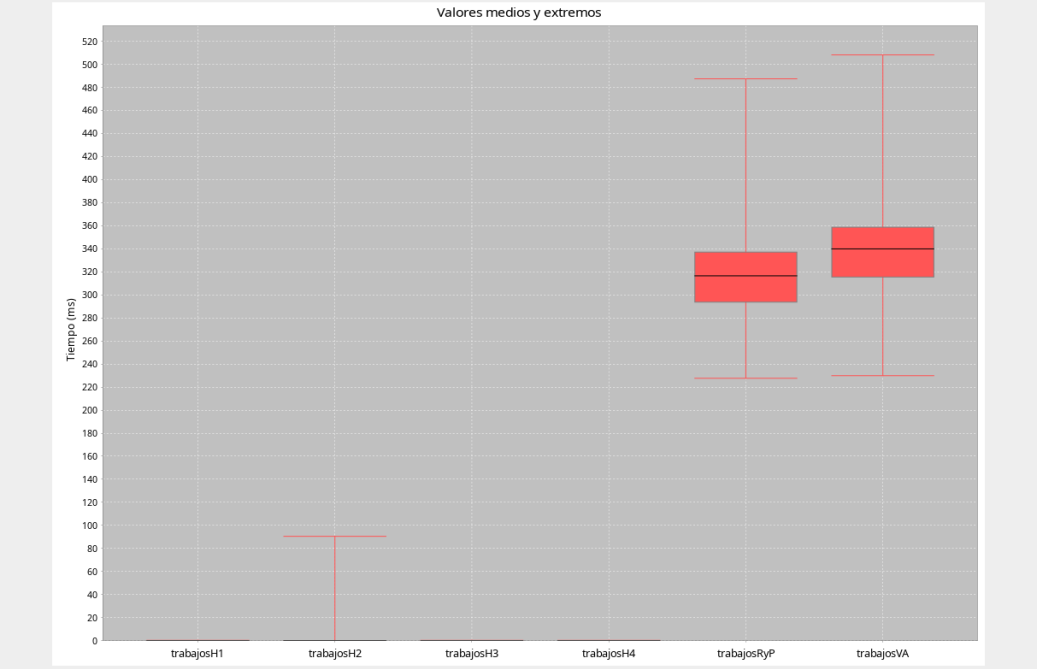

Figure 4: Gráficos

4 Conclusiones

La conclusión que obtenemos de la realización de esta práctica es que no siempre es necesario explorar el árbol de búsqueda por completo. Es cierto que hasta que no lleguemos hasta los nodos hijo no sabremos la solución de una rama, pero podemos realizar una estimación optimista de esta y podar la rama si dicha estimación es menor que alguna solución ya obtenida. Como resultado obtenemos un algoritmo más eficiente.

La parte más interesante de la práctica fue la implementación de la función de cota ya que esta pequeña modificación produce una reducción significante en el tiempo de ejecución.