Práctica 3a

Técnicas de búsqueda: Vuelta atrás

Algoritmos Avanzados

Víctor Fernández Fernández
Mikayel Mardanyan Petrosyan

20 de Octubre de 2019

0 Problema

El problema de esta práctica es el planteado en la práctica 2. 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.

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, en este caso, vamos a desarrollar de forma sistemática un algoritmo de vuelta atrás que resuelva el problema y comprobar experimentalmente su optimalidad.

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.

1 Técnica de vuelta atrás

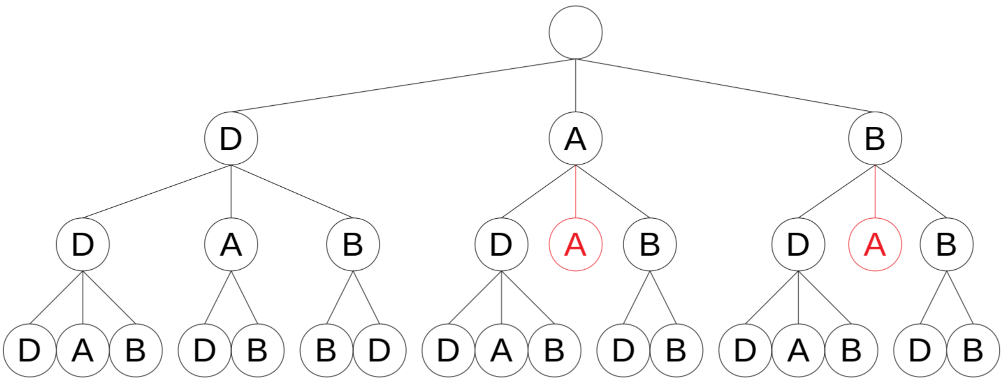
Para desarrollar sistemáticamente un algoritmo de vuelta atrás debemos diseñar un árbol de búsqueda, especificar la condición de validez que se re-

alizará en cada nodo del árbol y por último implementar el algoritmo acorde al diseño anterior.

1.1 Árbol de búsqueda

A continuación, se muestra un diagrama que corresponde con un árbol de búsqueda genérico para nuestro problema. En el nivel 0 vemos un nodo vacío que corresponde con el nodo raíz, cuando aún no hay ningún trabajo contratado. Desde este nodo tenemos 3 posibles elecciones: descansar (D), contratar un trabajo de alta tensión (A) o contratar un trabajo de baja tensión (B). El nodo raíz no tiene restricciones por lo que tiene los 3 posibles nodos hijo. En el nivel 2 observamos que hay dos nodos de color rojo, esto lo explicaremos en el siguiente apartado.

Cada nivel corresponde con una semana. En el nivel i se decide qué trabajo se contrata o si se descansa, por lo que **si en nuestro plan hay n semanas, el árbol tendra n niveles**. El número de nodos hijo corresponde con el número de posibilidades, **si hay m posibilidades, habrá m candidatos para cada nodo**, 3 en este caso.



1.2 Comprobación de validez

La comprobación de validez se realiza en cada nodo para descartar aquellos que no cumplan las restricciones del problema. La restricción del problema es que no se puede contratar un trabajo de alta tensión sin haber descansado la semana anterior, excepto en la primera semana, en la que esta restricción no se aplica. Por tanto, los nodos válidos serán, en el nivel 1, todos y, en los niveles siguientes, aquellos que no contraten un trabajo de alta tensión después de haber trabajado la semana anterior.

Los nodos válidos aparecen en color negro y los inválidos en color rojo. Estos últimos no cumplen la restricción y no tendrán nodos hijos. Solo se

han marcado en el nivel 2, en los niveles posteriores no se han representado.

1.3 Implementación

La cabecera del método principal del algoritmo es:

```
public static int trabajosVA (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.

```
 public static int trabajosVA (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 = contratar(k, n, a, b, solActual,
solFinal, bFinal);
     imprimir(a, b, solFinal);

     return calcularBeneficio(a, b, solFinal);
 }

 private static int[] contratar(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) {
             solActual[k] = i;
             if (k < n - 1)  // nodos intermedios
                 solFinal = contratar(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 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 creados para la práctica anterior 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.
- Algortimo de vuelta atrás (trabajosVA).

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, el algoritmo de vuelta atrás es el único algoritmo exacto. Esto se debe a que este algoritmo explora el árbol de búsqueda por completo y, por lo tanto, siempre encuentra una solución optimal.

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 es el único exacto, el único que obtiene 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	trabajosVA
Núm. ejecuciones	1000	1000	1000	1000	1000
Núm. ejec. válidas del método	1000	1000	1000	1000	1000
Núm. ejec. válidas en total	1000	1000	1000	1000	1000
% soluciones subóptimas	86.40 %	81.40 %	50.30 %	98.60 %	0 %
% soluciones óptimas	13.60 %	18.60 %	49.70 %	1.40 %	100 %
% soluciones sobreóptimas	0 %	0 %	0 %	0 %	0 %
% valor medio subóptimo	90.71 %	91.93 %	94.84 %	80.04 %	0 %
% valor extremo subóptimo	64.86 %	68.93 %	74.49 %	52.94 %	0 %
% valor medio sobreoptimo	0 %	0 %	0 %	0 %	0 %
% valor extremo sobreóptimo	0 %	0 %	0 %	0 %	0 %

Figure 1: Tabla de resultados

En el primer gráfico de la figura 2 se puede apreciar que el algoritmo de vuelta atrás siempre ofrece 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 no tiene resultados subóptimos y por ello no se muestra información sobre él.

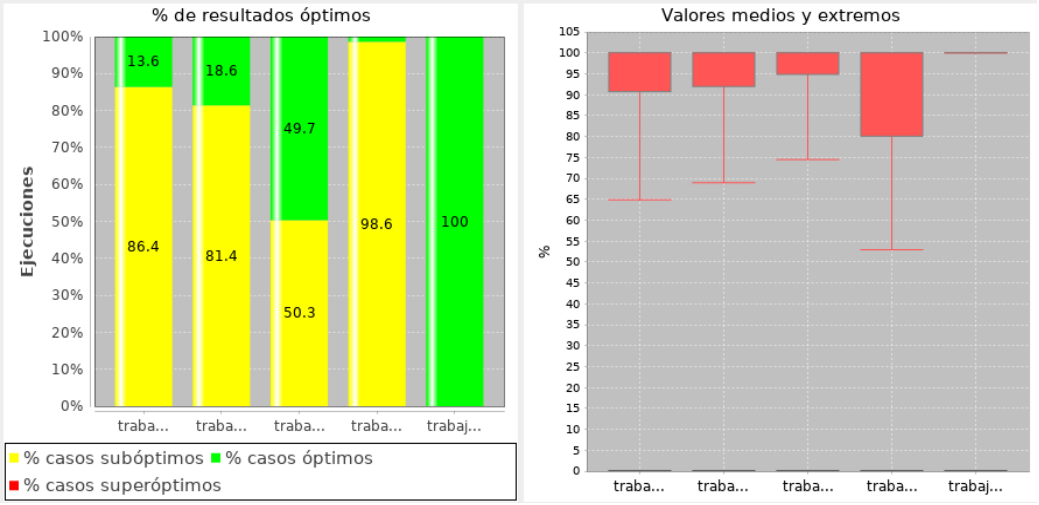


Figure 2: Gráficos

2.4 Incidencias

Encontramos una errata en el algoritmo heurístico 3. El bucle empezaba con `i = 0` cuando debía empezar con `i = 1`. De todas formas, los resultados no se ven afectados porque este error se introdujo en el informe pero no en la experimentación.

```
    public static int trabajosH3 (int[] a, int[] b) {
        int beneficio = Math.max(b[0], a[0]);
        int i = 1;
        for (; i < a.length-1; i+=2)  // en este for
antes se encontraba el i=0 que hacia que el algoritmo
fallara
            beneficio += Math.max(b[i]+b[i+1], a[i+1]);
        if (a.length % 2 == 0)
        beneficio += b[i];
        return beneficio;
    }
```

3 Conclusiones

Aunque los algoritmos heurísticos dan resultados aceptables con poca complejidad, si queremos obtener resultados optimales tenemos que recurrir a algoritmos como el de vuelta atrás para explorar todo el espacio de estados. La parte más complicada de realizar fue la implementación del algoritmo, aunque también fue la más interesante y la más educativa.