Universidad Rey Juan Carlos

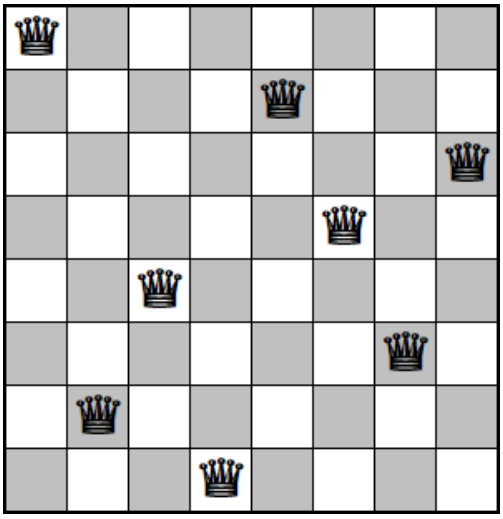
Universidad Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería Informática

Arquitecturas Avanzadas de Computadores

Práctica 3.a - Técnicas de búsqueda - Vuelta atrás



Curso Académico 2021/2022

Gustavo Andrés Marrero Tovar
Milagros Mouriño Ursul

Índice

Índice de imágenes

1. Introducción

La profundización de la técnica de vuelta atrás para el diseño de un algoritmo aplicado al problema de optimización de la asignación maximal de calificaciones.

El principal objetivo de la práctica es la comprensión de la técnica de *backtracking* que permite la implementación de algoritmos exactos, es decir, aquellos que calculan la solución óptima el 100% de los casos. Asimismo, se persigue la comparación de la optimalidad del algoritmo de vuelta atrás respecto a los algoritmos heurísticos diseñados en la segunda práctica de la asignatura utilizando la herramienta de AlgorEx.

La memoria consta de un apartado de propuesta en el que se explica cómo se ha procedido para realizar la práctica desde la elaboración del árbol de búsqueda por el algoritmo de vuelta atrás hasta la comparación entre algoritmos exactos e inexactos con AlgorEx. Por otra parte, se exponen las conclusiones extraídas tras la elaboración de la práctica y la bibliografía consultada.

2. Propuesta

2.1. Técnica de vuelta atrás

2.1.1. Diseño de un árbol de búsqueda

El problema de la asignación maximal de calificaciones busca la maximización de la suma de las calificaciones obtenidas por un alumno según el número de días que estudia sus asignaturas (función objetivo). La precondición tiene en cuenta varios puntos. En primer lugar, las celdas de la matriz de entrada deben contener un número entero positivo comprendido en el intervalo [1,10]. Asimismo, el número de asignaturas que cursa el alumno debe ser un entero positivo y el número de filas de la tabla de entrada tiene que estar incluido en el intervalo [1, a). También, ha de cumplirse que el alumno dedique como mínimo un día por asignatura, así como que el número de días dedicados a estudiar no supere el doble del número de asignaturas. En nuestro ejemplo de cabecera, cs es {{4,3,5,2}, {4,5,6,4}, {5,6,8,7}}. A partir de la tabla, se deduce que el número de asignaturas se corresponde con el número de columnas (a=4). A partir de este dato y el número de filas (f=3), se deduce que el número de días de estudio es a+f-1, que es lo mismo que 6 (4+3-1). Se cumple la precondición en el ejemplo, ya que el seis está incluido en el intervalo [4,8). El último componente de la precondición es que el número de días extra dedicados por asignatura esté comprendido entre 0 y a-1, que en nuestro ejemplo es [0,4). Por otra parte, la poscondición comprende la función objetivo ya mencionada y la condición de validez. En la segunda práctica de la asignatura se determinó que la condición de validez es que el resultado obtenido por la función objetivo devuelva un número entero positivo teniendo en consideración que no se puede superar el número de días de estudio (d). Sin embargo, en el apartado 2.1.2, se especifica la condición de validez parcial que permite verificar si una solución parcial es válida o no resolviendo el problema utilizando la técnica de *backtracking*.

Tras haber recapitulado lo aprendido en la práctica dos, se muestra el árbol de búsqueda para nuestro ejemplo de cabecera en la figura 1. Este permite ver cómo se consigue la exactitud en los algoritmos de vuelta atrás. Se obtiene la solución optimal el 100% de las veces, ya que se exploran todas las soluciones para el problema dado.

En cuanto a los niveles del árbol, el nivel 0 se corresponde con la raíz, el cual contiene la solución vacía. Más adelante, a partir de cada rama se elige la nota correspondiente a cada asignatura según los días que la hayamos estudiado. Por lo tanto, podemos concluir que el número de niveles del árbol es igual al número de asignaturas (a) + 1 correspondiente al nivel del nodo raíz. Respecto a las ramas, el número de candidatos por cada nodo del árbol es igual al número de días extra (e) + 1, ya que mínimamente se elegirá 1 día y, como máximo, se estudiará el número de días extra + 1 correspondiente al día con el que contamos.

Para diseñar el árbol, se pensaron las acciones que se debían tomar en cada nivel del árbol. Es decir,

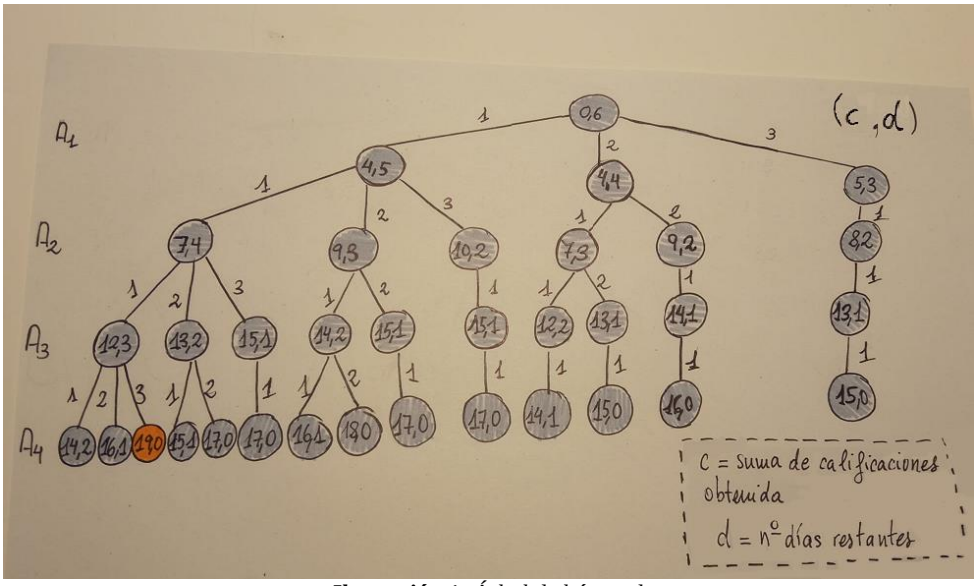


***Ilustración 1**: Árbol de búsqueda*

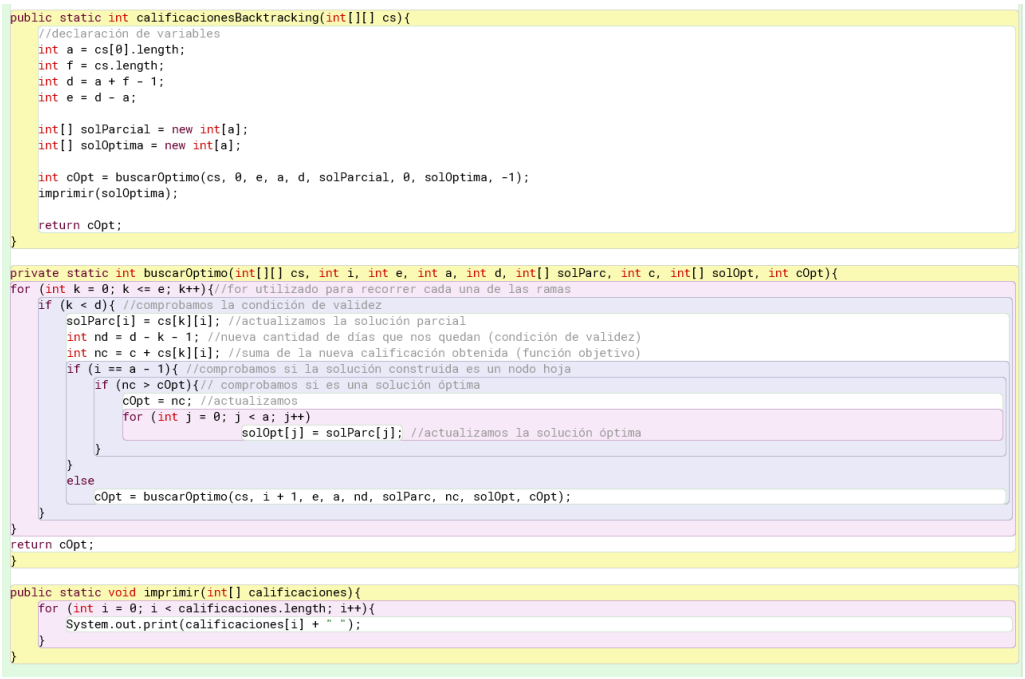
se plantearon las alternativas que se presentaban a partir de la asignatura 1 y se dedujo que éstas se correspondían a las notas obtenidas según los días que se dedique a la materia. Las tuplas que se ilustran en la imagen 1 están compuestas de una variable que contabiliza la suma de las calificaciones (c) obtenida en cada decisión tomada en los distintos niveles del árbol. Esta variable sirve para construir la función objetivo. Luego, hay otra que indica el número de días restantes que quedan por asignar al total de asignaturas (d). Esto último permite controlar la condición de validez del problema.

2.1.2. Comprobación de validez

Tal y como se razonaba en el apartado anterior, se intuyó que la comprobación de validez para cada nodo del árbol aplicando la técnica de vuelta atrás es que el número de días aplicado en una decisión determinada no debe superar el número de días general para estudiar todas las asignaturas del alumno. Por ejemplo, en el nodo raíz de la figura 1 partimos de los valores (0,6) dado que, en principio, no se ha sumado ninguna calificación y contamos con los 6 días de estudio. Sin embargo, al elegir la calificación que se obtiene dedicando 1 día a la asignatura 1, observamos que los valores son (4,5). Esto se debe a que se suma la calificación obtenida (4) y se resta el día que se dedica a la asignatura. Este valor d que se resta como mínimo puede ser 0, es decir, que se aprovechen todos los días de estudio de los que cuenta el alumno. Se extrajo que si en el nodo anterior al hoja se dedican k días a la última asignatura, entonces ese valor tiene que ser menor o igual a los días restantes (d). En ningún caso debe ser superior, ya que no se cumpliría la condición de validez y se contemplaría una solución incorrecta.

2.1.3. Código del algoritmo de vuelta atrás

En la ilustración 2, se observa el código extraído a partir de los razonamientos anteriores y la aplicación del esquema de vuelta atrás. En primer lugar, al igual que los códigos de los algoritmos heurísticos de la segunda práctica de la asignatura se declararon las variables necesarias para la resolución del problema: el número de asignaturas (a), el número



```
public static int calificacionesBacktracking(int[][] cs){
    //declaración de variables
    int a = cs[0].length;
    int f = cs.length;
    int d = a + f - 1;
    int e = d - a;

    int[] solParcial = new int[a];
    int[] solOptima = new int[a];

    int cOpt = buscarOptimo(cs, 0, e, a, d, solParcial, 0, solOptima, -1);
    imprimir(solOptima);

    return cOpt;
}

private static int buscarOptimo(int[][] cs, int i, int e, int a, int d, int[] solParc, int c, int[] solOpt, int cOpt){
for (int k = 0; k <= e; k++){//for utilizado para recorrer cada una de las ramas
    if (k < d){ //comprobamos la condición de validez
        solParc[i] = cs[k][i]; //actualizamos la solución parcial
        int nd = d - k - 1; //nueva cantidad de días que nos quedan (condición de validez)
        int nc = c + cs[k][i]; //suma de la nueva calificación obtenida (función objetivo)
        if (i == a - 1){ //comprobamos si la solución construida es un nodo hoja
            if (nc > cOpt){// comprobamos si es una solución óptima
                cOpt = nc; //actualizamos
                for (int j = 0; j < a; j++)
                                solOpt[j] = solParc[j]; //actualizamos la solución óptima
            }
        }
        else
            cOpt = buscarOptimo(cs, i + 1, e, a, nd, solParc, nc, solOpt, cOpt);
    }
}
return cOpt;
}

public static void imprimir(int[] calificaciones){
    for (int i = 0; i < calificaciones.length; i++){
        System.out.print(calificaciones[i] + " ");
    }
}
```

***Ilustración 2**: Código del algoritmo de vuelta atrás*

de filas (f), el número de días (d) y el número de días extra (e). Asimismo, se declararon otras dos variables: solParcial y solOptima que se corresponden con dos arrays de enteros. Sirven para proporcionar feedback acerca de las calificaciones elegidas para componer la suma de calificaciones óptima. En la siguiente línea, se observa la llamada a la función que verdaderamente realiza el trabajo de computar la solución: buscarOptimo. Esta recibe como parámetros la matriz de entrada cs, el nivel i, el número de días extra e, el número de asignaturas a, el número de días disponibles de estudio d, el vector de la solución parcial, la suma de las calificaciones (c) obtenida en el momento, el vector de la solución óptima y la suma de las calificaciones óptima (cOpt) que es el resultado de aplicar la función objetivo del problema de maximización. Por ello, la llamada a la función tiene como parámetros la matriz cs, el 0, ya que se parte del nivel 0, las variables locales e y a que no se modificarán (son meramente consultivas), d que controlará la función de validez, los vectores solParcial y solOptima, un 0, ya que la suma de las calificaciones inicial es nula al principio y -1 porque cOpt parte del valor más pequeño posible para que cuando se compute la primera rama se actualice dicho valor. Dentro del método, se visualiza el esquema de vuelta atrás con el bucle for, que contempla las diferentes decisiones que se pueden tomar en un nivel concreto. El número de ramas habíamos dicho que estaba comprendido entre 1 y e+1. Como se han de acceder a las celdas de la matriz cs, los límites del bucle van de 0 a e incluido. Más adelante, se pregunta por la condición de validez, que indica que una solución es válida cuando el número de días asociados a la calificación que se está tomando no supera el número de días que restan por asignar a las asignaturas controlado con el parámetro d.

Se añade la calificación obtenida en dicha rama a la solución parcial y se utilizan dos variables locales nd y nc para indicar la cantidad de días que quedan por asignar a las asignaturas y la suma de calificaciones obtenidas hasta el momento por dicha solución parcial respectivamente. Si estamos en un nodo hoja, tenemos que comprobar si la suma de calificaciones que se ha obtenido es superior a la óptima calculada hasta el momento. De ser así, se actualiza cOpt y se guarda en solOpt la solución óptima hallada. Si, por el contrario, no nos situamos en un nodo hoja debemos llamar recursivamente a la función para que se siga construyendo la solución parcial. Se debe tener en cuenta incrementar el nivel, ya que se computará la calificación obtenida para la siguiente asignatura. También, se deben pasar los valores actualizados de d y c (que se corresponden con las variables nd y nc). Por último, la función retorna el parámetro cOpt. Al retornarse el valor en la función calificacionesBacktracking, se imprimen por pantalla las calificaciones que se obtuvieron para obtener la solución óptima y se retorna el cOpt conseguido previamente.

2.2. Comparación de optimalidad

2.2.1. Material del experimento

Para esta prueba hemos comparado tres algoritmos. El método h1, cuya función de selección consiste en ir eligiendo la mayor calificación posible mientras queden días libres mientras que el método h2 asigna los valores de los días sobrantes de manera uniforme entre las distintas asignaturas. Hay que recalcar que estos dos métodos implementan los algoritmos heurísticos realizados en la segunda práctica de la asignatura.
Por otra parte, tenemos el algoritmo calificacionesBacktracking, que hemos implementado, tal y como se ha explicado en el apartado anterior, utilizando la técnica de vuelta atrás.
Al igual que en la segunda práctica, se han generado 2000 sets de datos con los rangos que pueden apreciarse en la ilustración 3, los cuales cumplen la precondición:
número de columnas = a = 5 > 0
$1 \leq f \leq a = 1 \leq f \leq 5 \Rightarrow f \in [1,5]$
Esto se satisface, ya que el número de filas = f = 4 y es un valor dentro del rango.
Además, los valores deben estar comprendidos entre el 1 y el 10, debido a que **∀i ∀j, 0 < cs[i][j] <= 10, 0<=i<=f, 0<=j<=a**

A continuación, se muestra la tabla con las primeras 24 ejecuciones y sus resultados para los tres algoritmos evaluados (ilustración 4):
Se puede apreciar que las celdas en verde se corresponden con las ejecuciones que han devuelto un mayor resultado entre los tres algoritmos, aunque necesariamente no implique que esas ejecuciones hayan devuelto soluciones óptimas.
Cada fila se corresponde con una ejecución del set de datos, y cada columna con uno de los algoritmos comparados (calificacionesBacktracking, h1 y h2).

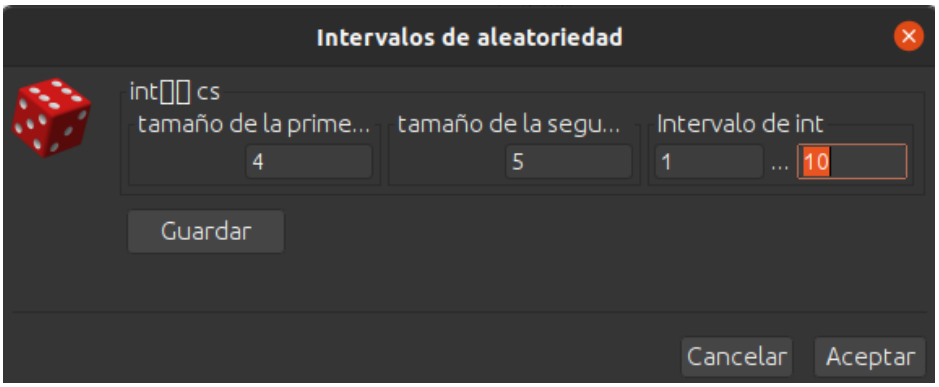


***Ilustración 3**: Rangos de valores*

Núm.	calificacionesBacktracking	h1	h2
1	33	33	27
2	46	46	41
3	33	27	22
4	39	36	27
5	34	30	25
6	39	32	29
7	32	29	21
8	39	38	29
9	26	26	23
10	39	37	35
11	41	41	33
12	45	40	45
13	41	41	41
14	32	29	27
15	32	28	27
16	42	42	41
17	30	30	21
18	39	39	24
19	28	18	17
20	34	34	27
21	36	36	32
22	35	35	20
23	34	29	21
24	41	41	41

***Ilustración 4**: Primeras 24 ejecuciones*

2.2.2. Conclusión

Luego de realizar la batería de pruebas y comparar los resultados para los tres algoritmos evaluados, hemos podido observar que el método calificacionesBacktracking, realizado con la técnica de vuelta atrás, siempre devuelve una solución igual o más alta respecto a la que proporcionan h1 y h2. Se ha devuelto un valor óptimo en el 100% de las ejecuciones para este método. Si bien no se puede asegurar matemáticamente que el algoritmo sea exacto, desde el punto de vista empírico podemos estar bastante seguros de que será así.

También, se concluye con que los métodos h1 y h2 son menos eficaces de lo que habíamos pensado. Esto se debe a que, al compararse con un algoritmo que siempre devuelve resultados iguales o mayores a ellos (el algoritmo de backtracking es exacto), se reducen los casos en los que se devuelven resultados que AlgorEx considera óptimos para h1 y h2.

Medidas	calificacionesBacktracking	h1	h2
Núm. ejecuciones	2000	2000	2000
Núm. ejec. válidas del método	2000	2000	2000
Núm. ejec. válidas en total	2000	2000	2000
% Soluciones subóptimas	0,00 %	59,95 %	95,70 %
% Soluciones óptimas	100,00 %	40,05 %	4,30 %
% Soluciones sobreóptimas	0,00 %	0,00 %	0,00 %
% Diferencia media subóptima	0.00 %	88,18 %	74,13 %
% Diferencia máxima subóptima	0,00 %	51,11 %	31,11 %
% Diferencia media sobreóptima	0,00 %	0,00 %	0,00 %
% Diferencia máxima sobreóptima	0,00 %	0,00 %	0,00 %

Ilustración 5*: Resumen numérico de ejecuciones*

2.2.3. Evidencias

La tabla de resumen numérico de la ilustración 5 realiza un análisis de la optimalidad de los algoritmos. Las primeras tres filas nos permiten verificar que la ejecución de los algoritmos ha tenido éxito. En este caso, se han ejecutado adecuadamente los tres algoritmos sobre los 2000 conjuntos de datos. Por otro lado, las siguientes tres filas presentan la optimalidad de los resultados obtenidos. Se puede apreciar que el algoritmo de backtracking ha obtenido un 100% de resultados óptimos, por lo que no ha devuelto resultados subóptimos ni tiene diferencias respecto al caso óptimo.
Los métodos heurísticos h1 y h2 han obtenido un 40,05 % y un 4,30 % de soluciones óptimas respectivamente, un decremento importante respecto a los resultados de la práctica 2 (con un porcentaje de casos óptimos de 92,45 % y 14,55 % para h1 y h2), que como se ha comentado antes, se debe a que el método de vuelta atrás siempre devuelve resultados con una calificación mayor o igual a estas dos heurísticas. Se puede ver la comparativa en las siguientes tablas de resumen gráfico, correspondientes a las ilustraciones 6 y 7.
Tomemos por ejemplo el caso de prueba que se ha estado manejando: cs = {{4,3,5,2}, {4,5,6,4}, {5,6,8,7}}. El algoritmo de backtracking devuelve 19 para este caso, que es la solución óptima. Sin embargo, en la práctica 2, la solución marcada como óptima venía dada por 16, que es el resultado obtenido por h1. Como en aquella práctica no había forma de indicarle a AlgorEx que no se trataba de una solución óptima (ninguno de los algoritmos comparados es exacto), se marcaba como tal. Esto desencadena que muchos casos considerados como óptimos en la práctica 2, dejen de serlo en la práctica 3, porque ahora hay un punto de referencia con el cual comparar algoritmos que no son exactos, lo cual se traduce en esa reducción de los porcentajes.

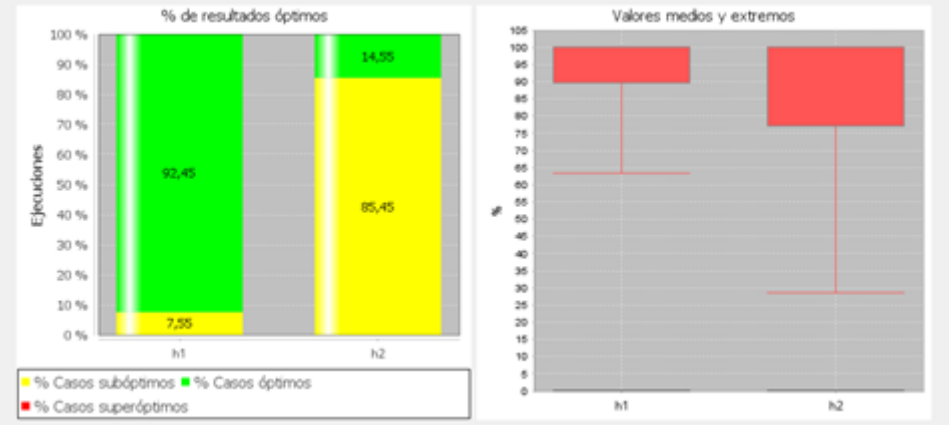


***Ilustración 6**: Resumen gráfico de las ejecuciones de la práctica 2*

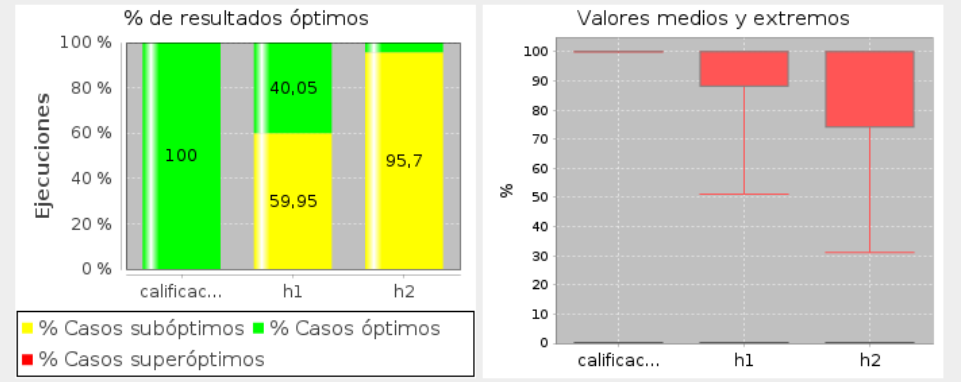


***Ilustración 7** Resumen gráfico de las ejecuciones de esta práctica*

3. Conclusiones

Se puede apreciar que, basados en los resultados obtenidos con la experimentación, podemos afirmar con cierto grado de seguridad que el algoritmo de *backtracking* implementado es exacto o, al menos, que siempre devuelve un resultado igual o mayor que el de los algoritmos heurísticos diseñados en la práctica anterior.

Ahora tenemos un mejor manejo de la herramienta AlgorEx, y hemos aprendido un poco más sobre su funcionamiento y sobre cómo determina la optimalidad de los algoritmos, que resulta ser de forma comparativa.

De igual modo, ahora tenemos conocimientos más sólidos sobre la implementación de algoritmos basados en vuelta atrás. Esta resulta una técnica muy poderosa que es capaz de construir algoritmos exactos. Sin embargo, aunque puede ser realmente eficaz, también puede ser costosa computacionalmente, pues se recorren muchos nodos en busca de soluciones.

4. Referencias

Transparencias: Repaso de fundamentos de la técnica de vuelta atrás

Transparencias: Técnicas de búsqueda para problemas de optimización

Programas: Sistema AlgorEx

Programas: Paquete Java con algoritmos de búsqueda