Práctica 7

Eduardo Blanco Bielsa

UO285176

Implementación

Se ha diseñado un heurístico que asegura que ningún nodo es podado, de forma que proporciona exactamente el mismo resultado que Backtracking sin balanceo.

En primer lugar, se ha creado un método al que se llamará posteriormente tanto para la ejecución como para la medición de los tiempos. En este método de la clase PromediadoImagen se llamará al método branchAndBound de la clase BranchAndBound, pasándole como parámetro un AvgNode con el dataset, la profundidad 0 y un arraylist vacío que se corresponderá con las imágenes de cada nodo.

Una vez hecho esto se guardará en los atributos half1 y half2 la mejor solución y se actualizará el contador al número total de nodos recorridos.

```
public void branchAndBound() {
    this.counter = 1;
    this.max_zncc = -1;
    branchAndBound(new AvgNode(dataset, 0, new ArrayList<Integer>()));
    this.half1_img = ((AvgNode)bestNode).getHalf1_avg();
    this.half2_img = ((AvgNode)bestNode).getHalf2_avg();
    this.avg_img.addSignal(half1_img);
    this.avg_img.addSignal(half2_img);
    this.counter = super.counterBnB;
}
```

Se ha creado la clase **AvgNode** donde:

- El método **calculateHeuristicValue()** calcula en función de si el nodo es solución o no el valor del heurístico.
- El método **expand()** devuelve un arraylist de Node con las correspondientes soluciones.
- El método **isSolution()** devuelve si un nodo es solución (nodo hoja), comparando su profundidad con la longitud del array de imágenes (dataset).

```java
public class AvgNode extends Node {

    private Imagen[] dataset;
    private ArrayList<Integer> sol;
    private Imagen half1_avg, half2_avg;

    public AvgNode(Imagen[] imagenes, int grupo, ArrayList<Integer> imagenesNodo) {
        ID = UUID.randomUUID();
        this.dataset = imagenes;// copia de todas las imágenes
        this.depth = grupo;
        this.sol = new ArrayList<Integer>(imagenesNodo);
        this.half1_avg = new Imagen(dataset[0].getWidth(), dataset[0].getHeight());
        this.half2_avg = new Imagen(dataset[0].getWidth(), dataset[0].getHeight());
        calculateHeuristicValue();
    }

    @Override
    public void calculateHeuristicValue() {
        if (!isSolution()) {
            this.heuristicValue = -2;
        } else {
            Imagen group1_pd = new Imagen(dataset[0].getWidth(), dataset[0].getHeight());
            Imagen group2_pd = new Imagen(dataset[0].getWidth(), dataset[0].getHeight());
            for (int i = 0; i < this.sol.size(); i++) {
                if (sol.get(i) == 1)
                    group1_pd.addSignal(this.dataset[i]);
                if (sol.get(i) == 2)
                    group2_pd.addSignal(this.dataset[i]);
            }
            this.heuristicValue = (group2_pd.zncc(group1_pd)) * (-1);
            this.half1_avg = group1_pd.copy();
            this.half2_avg = group2_pd.copy();
        }

    }
```

```java
@Override
public ArrayList<Node> expand() {
    // x3 new AvgNode, que imagen va a que conjunto
    ArrayList<Node> res = new ArrayList<Node>();
    sol.add(0);
    res.add(new AvgNode(dataset, depth+1, sol));
    sol.set(depth, 1);
    res.add(new AvgNode(dataset, depth+1, sol));
    sol.set(depth, 2);
    res.add(new AvgNode(dataset, depth+1, sol));
    return res;
}
```

```java
@Override
public boolean isSolution() {
    // devuelve si es un nodo hoja o no
    return getDepth() == this.dataset.length;

}

public Imagen getHalf2_avg() {
    return half2_avg;
}

public Imagen getHalf1_avg() {
    return half1_avg;
}
```

Para conocer el valor del contador del branchAndBound se ha modificado ligeramente el método branchAndBound añadiendo un atributo contador que se modifica en cada llamada y en el bucle while:

```java
public void branchAndBound(Node rootNode) {
    this.counterBnB++;// contador
    ds.insert(rootNode); // First node to be explored

    pruneLimit = rootNode.initialValuePruneLimit();

    while (!ds.empty() && ds.estimateBest() < pruneLimit) {
        Node node = ds.extractBestNode();

        ArrayList<Node> children = node.expand();
        this.counterBnB += children.size();// contador
        for (Node child : children)
            if (child.isSolution()) {
                double cost = child.getHeuristicValue();
                if (cost < pruneLimit) {
                    pruneLimit = cost;
                    bestNode = child;
                }
            } else if (child.getHeuristicValue() < pruneLimit) {
                ds.insert(child);
            }
    } // while
}
```

Medidas

Para 6 imágenes se obtienen los siguientes resultados:

n	Tiempo_BT_sin_balanceo	Contador BT	Tiempo_BnB	Contador BnB
2	8	13	67	13
3	17	40	17	40
4	47	121	50	121
5	125	364	192	364
6	321	1093	675	1093
7	996	3280	2247	3280
8	3052	9841	5611	9841
9	9324	29524	**heap space**	**heap space**
10	29626	88573	**heap space**	**heap space**

ZNCC_Backtracking	ZNCC_ BnB
0,001746	0
0,029555	0,023124
0,021497	0,033691
0,038218	0,035221
0,051588	0,047109
0,059882	0,053151
0,054618	0,053285
0,070974	**heap space**
0,076252	**heap space**

```
TESTING BACKTRACKING SIN BALANCEO:
TAM     TIEMPO  ZNCC                    CONTADOR
2       8       0.0017463392578065395   13
3       17      0.02955571562051773     40
4       47      0.021497389301657677    121
5       125     0.03821875900030136     364
6       321     0.051588933914899826    1093
7       996     0.05988288298249245     3280
8       3052    0.054618459194898605    9841
9       9324    0.07097499817609787     29524
10      29626   0.07625273615121841     88573
TESTING BnB:
TAM     TIEMPO  ZNCC                    CONTADOR
2       67      0.0     13
3       17      0.02312426082789898     40
4       50      0.03369101509451866     121
5       192     0.035221584141254425    364
6       675     0.04710958153009415     1093
7       2247    0.05315165966749191     3280
8       5611    0.05328576639294624     9841
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.base/java.util.Arrays.copyOfRange(Arrays.java:3821)
        at java.base/java.lang.StringLatin1.newString(StringLatin1.java:764)
        at java.base/java.lang.StringBuilder.toString(StringBuilder.java:447)
        at java.base/java.lang.Object.toString(Object.java:239)
        at Node.equals(Node.java:45)
        at Heap.nodeRepeated(Heap.java:43)
        at Heap.insert(Heap.java:36)
        at BranchAndBound.branchAndBound(BranchAndBound.java:52)
        at PromediadoImagen.branchAndBound(PromediadoImagen.java:275)
        at PromediadoImagenBench.main(PromediadoImagenBench.java:100)
```

Para 8 imágenes se obtienen los siguientes resultados:

n	Tiempo_BT_sin_balanceo	Contador BT	Tiempo_BnB	Contador BnB
2	9	13	73	13
3	16	40	16	40
4	44	121	52	121
5	130	364	184	364
6	328	1093	559	1093
7	1017	3280	1919	3280
8	3123	9841	5515	9841
9	9493	29524	**heap space**	**heap space**
10	29320	88573	**heap space**	**heap space**

ZNCC_Backtracking	ZNCC_ BnB
0,004199	0,001799
0,028478	0,03692
0,027522	0,027824
0,034373	0,040192
0,047012	0,054616
0,054909	0,058058
0,060142	0,060269
0,074447	**heap space**
0,079416	**heap space**

```
TESTING BACKTRACKING SIN BALANCEO:
TAM     TIEMPO  ZNCC                    CONTADOR
2       9       0.004199767950922251    13
3       16      0.028478583320975304    40
4       44      0.02752210758626461     121
5       130     0.034373313188552856    364
6       328     0.047012388706207275    1093
7       1017    0.05490918084979057     3280
8       3123    0.060142528265714645    9841
9       9493    0.07444711774587631     29524
10      29320   0.07941687107086182     88573
TESTING BnB:
TAM     TIEMPO  ZNCC                    CONTADOR
2       73      0.0017990126507356763   13
3       16      0.036920420825481415    40
4       52      0.027824200689792633    121
5       184     0.040192365646362305    364
6       559     0.054616332054138184    1093
7       1919    0.05805838853120804     3280
8       5515    0.0602697879076004      9841
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.base/java.util.Arrays.copyOfRange(Arrays.java:3821)
        at java.base/java.lang.StringLatin1.newString(StringLatin1.java:764)
        at java.base/java.lang.StringBuilder.toString(StringBuilder.java:447)
        at java.base/java.lang.Object.toString(Object.java:239)
        at Node.equals(Node.java:45)
        at Heap.nodeRepeated(Heap.java:43)
        at Heap.insert(Heap.java:36)
        at BranchAndBound.branchAndBound(BranchAndBound.java:52)
        at PromediadoImagen.branchAndBound(PromediadoImagen.java:275)
        at PromediadoImagenBench.main(PromediadoImagenBench.java:101)
```

Respuestas a preguntas y conclusiones:

- En ambos casos se comprueba que el **contador** de Backtracking y BranchAndBound es el mismo, por tanto, nuestro heurístico es correcto.
- Al llegar a un determinado tamaño, se produce un "**heap space**" en el BranchAndBound, es decir, que Java no tiene suficiente memoria para seguir calculando datos del heurístico.
- Los **tiempos** son "similares", aunque el BranchAndBound tarda más tiempo, por tanto, posee un peor rendimiento que el backtracking normal.
- Ambos tienen un **Zncc** bastante similar, por tanto, deducimos que obtienen una solución muy similar o casi idéntica.