Declarative programming project :

Objects Stacking

Realised by : Hélain Schoonjans

Table des matières

1 Introduction

Solving the blocks stacking problem using an optimal best-first algorithm, is a way of applying well-known techniques to a new problem.

1.1 Block stacking as a graph search problem

This implementation design is heavily inspired from the optimal best-first algorithms. Stacking blocks can indeed be seen as the act of searching goal nodes through a graph, or in this case, a tree. We can observe an example on figure 1. The algorithm will search for a solution from the

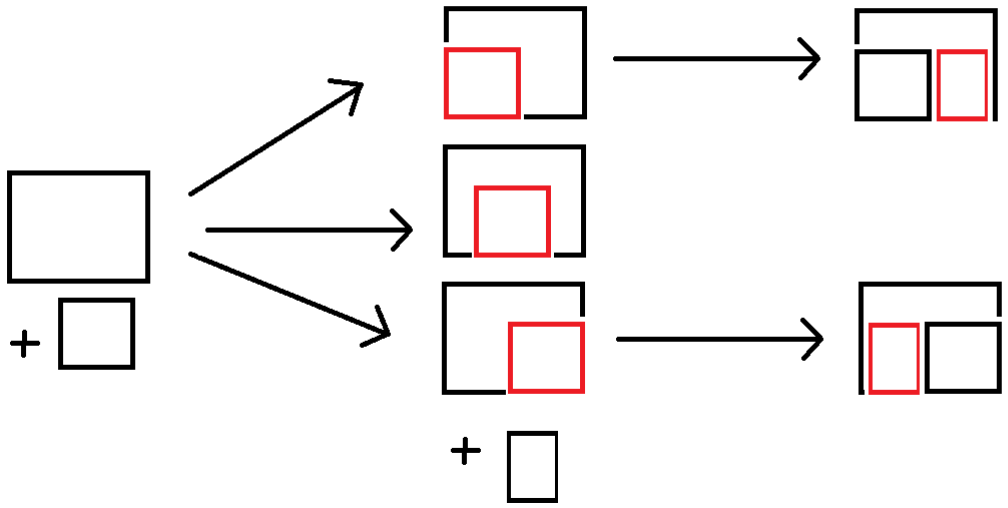

FIGURE 1 – Possible states of the algorithm, applied on a single box, seen as a tree.

starting node, the state with both boxes empty. The neighbours or children of a node or state will be the states that can be reached by placing one more block.
The *search space* is thus the set of possible placements of objects in two boxes. The *starting node* is the state with two empty boxes, and a list of items to place. The *solution* is a "path"; a list of valid placements of objects in the boxes. The program will look for the solution with the minimal cost, the *optimal solution*, using an A algorithm. The *cost functions* are defined by heuristics, described below. A *solution* of the block stacking problem is a state where there is no more objects the place in the boxes; they all have been put in or discarded.
Trying each possible node will however be quite long. The use of pertinent heuristics will allow the algorithm to search for the best solution first. How to express the estimated "cost" and "distance" to the goal node? Counting the remaining empty spaces in a box and the volume of the objects yet to place will be the key of good heuristics, as explained below.

2 Design

2.1 Algorithm

2.1.1 The placing order

In reality, the blocks could be placed in any order in the box, if they respect the specifications. Generating all the permutations of the block's ids and trying put them in that order would however take too much time and memory.
A first step for pruning some combinations is to try to put the blocks by order of volume. Sorting them by weight to generate the placing order is a way to benefit to the maximum of the fact that a bigger block can't be placed on a smaller one. A subtlety is to allow permutations in the order of blocks of identical volume, since they can be put on top of each other in any order (it is done when generating the children of a state).
If an empty space remains under an object, an object can fill this gap later (it represents the fact that the smaller object could have been placed earlier, but hadn't because of the block sorting), but it musn't touch the object above or it would violate the rules.

2.1.2 Optimal best first algorithm

The A algorithm will explore first the most promising nodes, having some characteristics of both Dijkstra and depth-first algorithms. Two heuristics have been implemented, it is however an informed search strategy.
This program has the properties of the optimal best-first algorithms : using the power of his heuristics, it is optimal and complete.

2.1.3 The box-filling heuristic

This heuristic gives lower values to the nodes filling better the boxes.
To do that, it calculates the total cost of filling the box : $f(n) = g(n) + h(n)$, where g(n) is the cost of reaching the current state from the starting node, and h(n) is the estimated distance to the goal.
The best goal would be the state where both boxes are filled, it is logical then that the heuristic h(n) is the number of empty spaces remaining.
g(n) however must reflect the space sacrifices, "costs", made when putting the blocks in the boxes : it is the number of spaces that won't be filled, because there isn't enough blocks left to fill them all, or because they are empty but located under a bigger block, and placements there would violate the rules.
Empty spaces located under a bigger block, that cant' be filled despite being free are called "crushed spaces".
In short :

- g(n) = max(number of empty spaces - remaining object's weigth, number of crushed spaces)
- h(n) = volume of empty spaces + abs(weigth of box1 - weight of box2)

2.1.4 The box-balancing heuristic

The second algorithm must take into account the fact that the boxes must be as full as possible, while trying to have the lowest difference in empty (or full) volume between the boxes.
The cost g(n) is the weigth difference between the boxes that isn't correctable; it can happen that the objects yet to place have a total volume inferior to the boxes' weigth difference. When it happens, we known for sure that the imbalance won't be repaired later.
The distance h(n) represents the estimated distance to the goal. Since filling the boxes is one of the

objectives, counting the empty spaces left in this heuristic is logical. There is however a difference with the distance of the other heuristic : h(n) of the box-balancing heuristic also adds the weigth difference between the boxes in the distance. A good goal must indeed have his boxes filled AND balanced.
In short :

- g(n) = max(abs(weigth of box1 - weight of box2) - remaining object's weigth, 0)
- h(n) = volume of empty spaces + abs(weigth of box1 - weight of box2)

In case of ties when comparing nodes, the box-filling heuristic is used.

2.2 Data structures

2.2.1 $state(Boxes, Objects\_to\_place)$

The state functor represents a possible state of the box filling procedure, or a node in the search algorithm. The first argument the list of boxes, with the placements of the objects inside the box, and their coordinate, while the second one is a list of the objects yet to place.

2.2.2 $box([Placement|Placements])$

The box data structures contains a list of placements of items.

2.2.3 $placement(Id, coordinates(X, Y, Z))$

A placement describes at which position an object of ID Id is located. X is the vertical distance, Y is the horizontal distance, Z is depth. A coordinate of (1,1,1) would describe the bottom left of the box, as seen on figure 2.

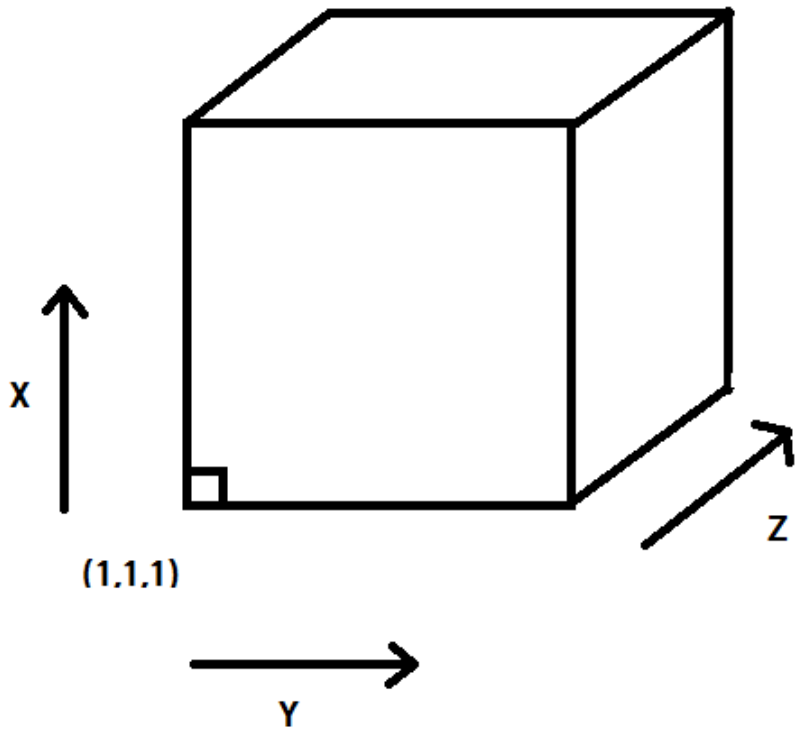

FIGURE 2 – The coordinates system.

2.3 Extensions

2.3.1 Visual display

The program give the choice to the user to have the results displayed visually. The boxes are then represented with ASCII characters like the ones of Figure 3.

```
Box:
---
|       9       9       9       9       9       9       9       4       18              |
|       9       9       9       9       9       9       9       8       8       8       |
|       9       9       9       9       9       9       9       8       8       8       |
|       9       9       9       9       9       9       9       8       8       8       |
|       14      14      14      14      14      14      14      8       8       8       |
|       14      14      14      14      14      14      14      16      16      16      |
|       14      14      14      14      14      14      14      16      16      16      |
|       14      14      14      14      14      14      14      16      16      16      |
|       14      14      14      14      14      14      14      16      16      16      |
|       14      14      14      14      14      14      14      16      16      16      |
\       -       -       -       -       -       -       -       -       -       -       /
Box:
---
|       11      11      11      11      11      11                              6       |
|       15      15      15      15      15      15      15      3               6       |
|       15      15      15      15      15      15      15      3               6       |
|       15      15      15      15      15      15      15      5       5       6       |
|       15      15      15      15      15      15      15      5       5       6       |
|       2       2       2       2       2       2       2       5       5       1       |
|       2       2       2       2       2       2       2       10      10      1       |
|       2       2       2       2       2       2       2       10      10      1       |
|       2       2       2       2       2       2       2       10      10      1       |
|       2       2       2       2       2       2       2       10      10      1       |
\       -       -       -       -       -       -       -       -       -       -       /
Continue?
Press any key...
```

FIGURE 3 – Example of the visual display of results.

The numbers found in the box are the IDs of the objects put inside. This allow to identify and distinguish the different blocks stacked.
When this feature is not enabled, the results are displayed textually as in Figure 4

```
---
Box
---
Object number 18 put at coordinates(10,7,1).
Object number 3 put at coordinates(8,8,1).
Object number 1 put at coordinates(5,10,1).
Object number 11 put at coordinates(10,1,1).
Object number 5 put at coordinates(5,8,1).
Object number 8 put at coordinates(1,8,1).
Object number 15 put at coordinates(6,1,1).
Object number 2 put at coordinates(1,1,1).

---
Box
---
Object number 4 put at coordinates(10,8,1).
Object number 6 put at coordinates(6,10,1).
Object number 10 put at coordinates(6,8,1).
Object number 16 put at coordinates(1,8,1).
Object number 9 put at coordinates(7,1,1).
Object number 14 put at coordinates(1,1,1).

Continue?
Press any key...
```

FIGURE 4 – Example of the textual display of results.

3 Manual

3.1 Datasets

All the files containing the objects must have the extension ".pl", and be in the same working directory. The must contains objects in the right format ; "object(Id, size(Height, Length, Width))". The program will ask at the beginning which dataset must be loaded, as shown in Figure 5, and the name of the file must be given without the extension ".pl".

```
Please enter the name the data file, without the extension.
|: data1.
Loading of file data1.pl
% data1.pl compiled 0.00 sec, 1 clauses
Done.
```

FIGURE 5 – Enter the name of the data file you want to use.

3.2 Launch

To launch the algorithm, do the following steps in SWI-Prolog :

1. Set the working directory to the folder where the source code and the data files are located by typing "working_directory(_, Code_directory).".
2. Do File->Consult on 'program.pl'.
3. Type "start.".

3.2.1 Example

We can see at Figure 6 an example of a launch of the program.

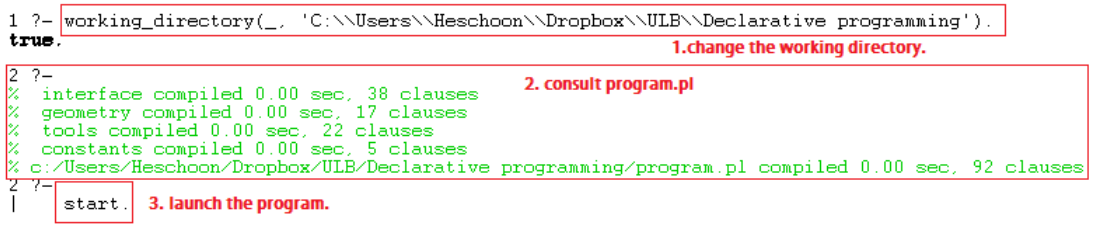

FIGURE 6 – Launch of the program.

4 Results

These are the some optimal solutions found for different metrics and extensions.

4.1 Filling the boxes

When trying to fill the boxes, the algorithm only tries to minimize the empty spaces.

```
Box:
---
|      11      11      11      11      11      11      18                      |
|      15      15      15      15      15      15      15      3               1      |
|      15      15      15      15      15      15      15      3               1      |
|      15      15      15      15      15      15      15      5       5       1      |
|      15      15      15      15      15      15      15      5       5       1      |
|      2       2       2       2       2       2       2       5       5       1      |
|      2       2       2       2       2       2       2       8       8       8      |
|      2       2       2       2       2       2       2       8       8       8      |
|      2       2       2       2       2       2       2       8       8       8      |
|      2       2       2       2       2       2       2       8       8       8      |
\      -       -       -       -       -       -       -       -       -       -      /
Box:
---
|      9       9       9       9       9       9       9       4               6      |
|      9       9       9       9       9       9       9       10      10      6      |
|      9       9       9       9       9       9       9       10      10      6      |
|      9       9       9       9       9       9       9       10      10      6      |
|      14      14      14      14      14      14      14      10      10      6      |
|      14      14      14      14      14      14      14      16      16      16     |
|      14      14      14      14      14      14      14      16      16      16     |
|      14      14      14      14      14      14      14      16      16      16     |
|      14      14      14      14      14      14      14      16      16      16     |
|      14      14      14      14      14      14      14      16      16      16     |
\      -       -       -       -       -       -       -       -       -       -      /
```

FIGURE 7 – The optimal solution of data1, with the emptyness minimizing heuristic. There is six empty spaces

```
Box:
---
|      15      15      15      15      15      15                              |
|      8       8       8       8       8       8       8       8       8       8      |
|      14      14      14      14      14      14      14      14      14      14     |
|      14      14      14      14      14      14      14      14      14      14     |
|      14      14      14      14      14      14      14      14      14      14     |
|      10      10      10      10      10      10                              |
|      10      10      10      10      10      10      1       1       1       1      |
|      10      10      10      10      10      10      1       1       1       1      |
|      10      10      10      10      10      10      1       1       1       1      |
|      10      10      10      10      10      10      1       1       1       1      |
\      -       -       -       -       -       -       -       -       -       -      /
Box:
---
|      11      11      11      11      11      11      11      11      11      12     |
|      11      11      11      11      11      11      11      11      11      12     |
|      11      11      11      11      11      11      11      11      11      4      |
|                                      13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
|      5       5       5       5       5       13      13      13      13      4      |
\      -       -       -       -       -       -       -       -       -       -      /
```

FIGURE 8 – The optimal solution of data2, with the emptyness minimizing heuristic. There is 13 empty spaces.

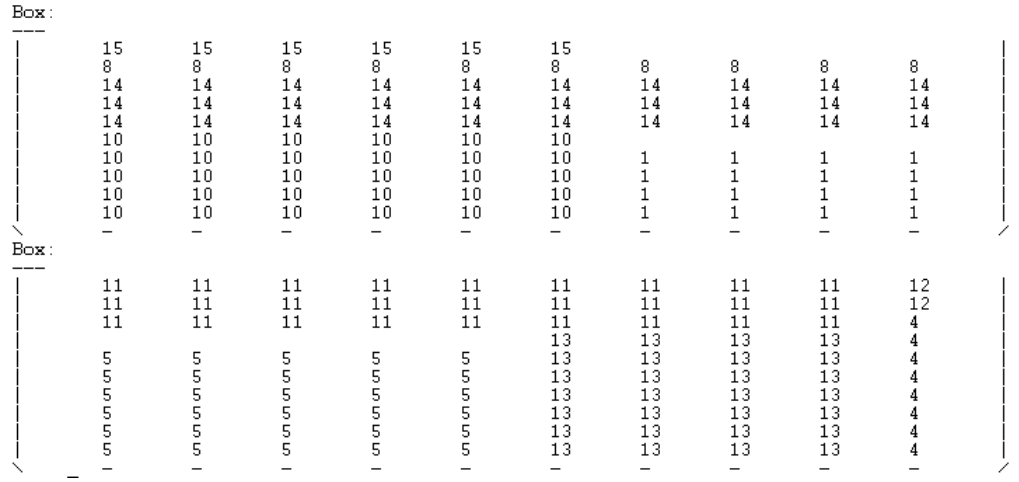

FIGURE 9 – The optimal solution of data3, with the emptyness minimizing heuristic. There is 99 empty spaces.

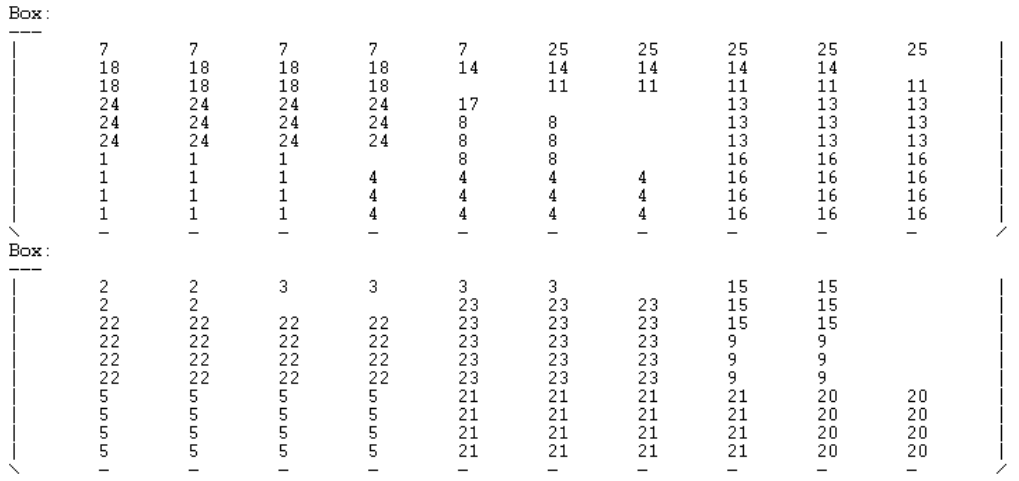

FIGURE 10 – The optimal solution of data4, with the emptyness minimizing heuristic. There is 17 empty spaces.

4.2 Balancing the boxes

When trying to balance the boxes, the algorithm will only accept solutions with the same volume of objects put in each box, while minimizing the empty spaces.

FIGURE 11 – The optimal solution of data5, with the emptyness minimizing heuristic. There is 100 empty spaces.

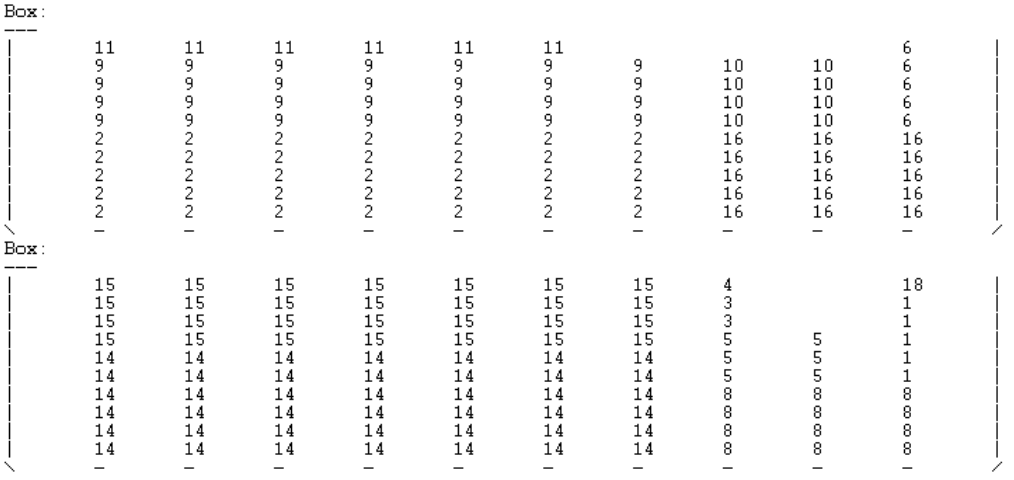

FIGURE 12 – The optimal solution of data1, with the balancing heuristic. Both boxes have equal weight

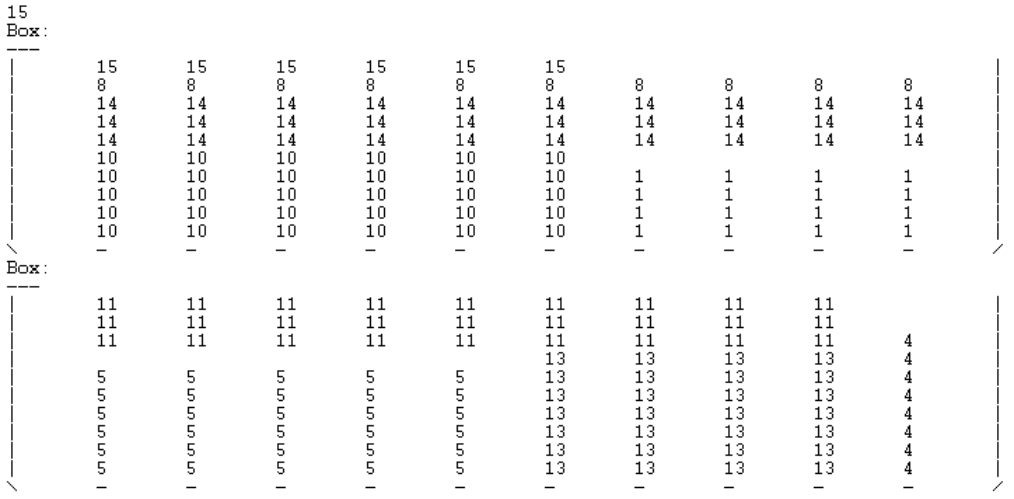

FIGURE 13 – The optimal solution of data2, with the balancing heuristic. There is a difference of one between the boxes.

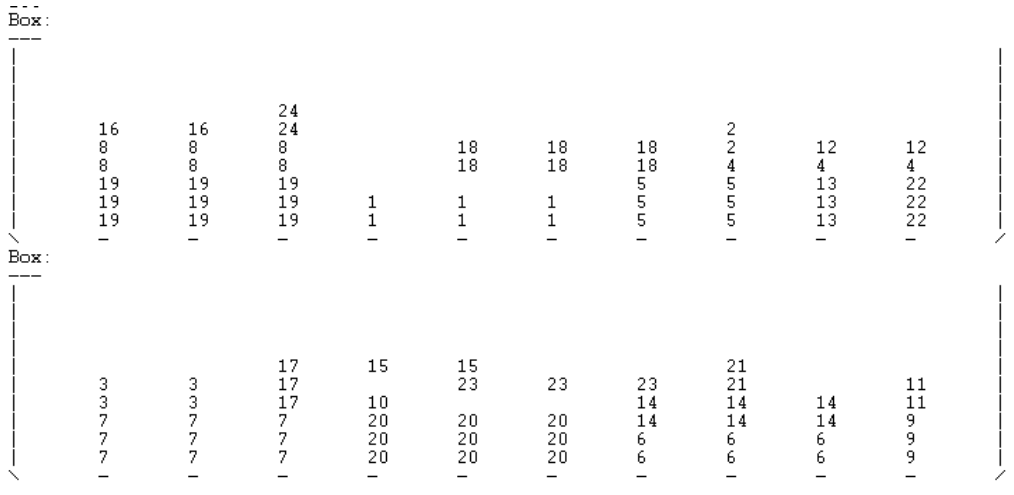

FIGURE 14 – The optimal solution of data3, with the balancing heuristic. Both boxes have equal weight. There is a volume of objects of 50 in each box.

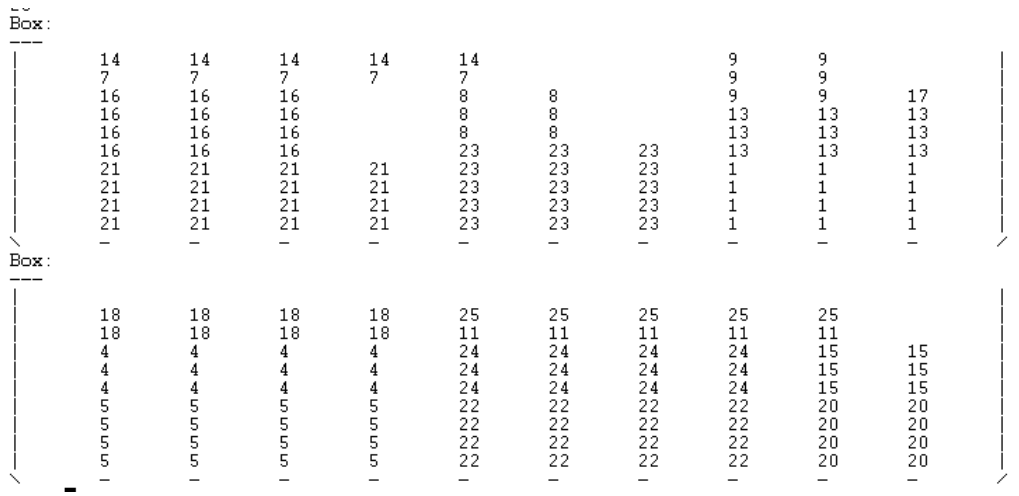

FIGURE 15 – The optimal solution of data4, with the balancing heuristic. There is one kilogram difference between the boxes. The total number of empty space is thus 25.

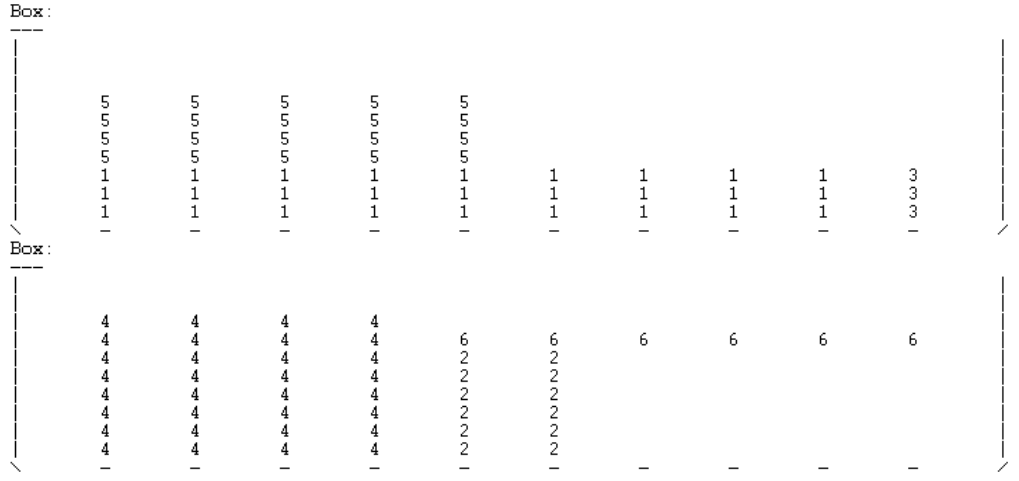

FIGURE 16 – The optimal solution of data5, with the balancing heuristic. Both boxes have equal weight.