Artificial Intelligence Assessed Exercise

2461352F

Section 1: Defining the states and actions

States:

A state in my implementation is a partially tiled grid. As attributes it has its board size a (initialised at the start and inherited from parent states) and a set of non-overlapping integer-sized rectangles $r=(r_x,r_y,r_w,r_h)$ where (r_x,r_y) are co-ordinates of the bottom right corner of the rectangle, and r_w and r_h are the rectangle width and height respectively. It also has a signature s, which is the list of sorted areas of each rectangle on the board. This is used to compare to other states to ensure we don't explore the same state twice in the search algorithm. The initial state has one rectangle (0, 0, a, a)

Actions:

I have defined two different types of actions to reduce the largest area and increase the smallest area of the rectangles, respectively: **Box cutting** and **merging**

Box cutting:

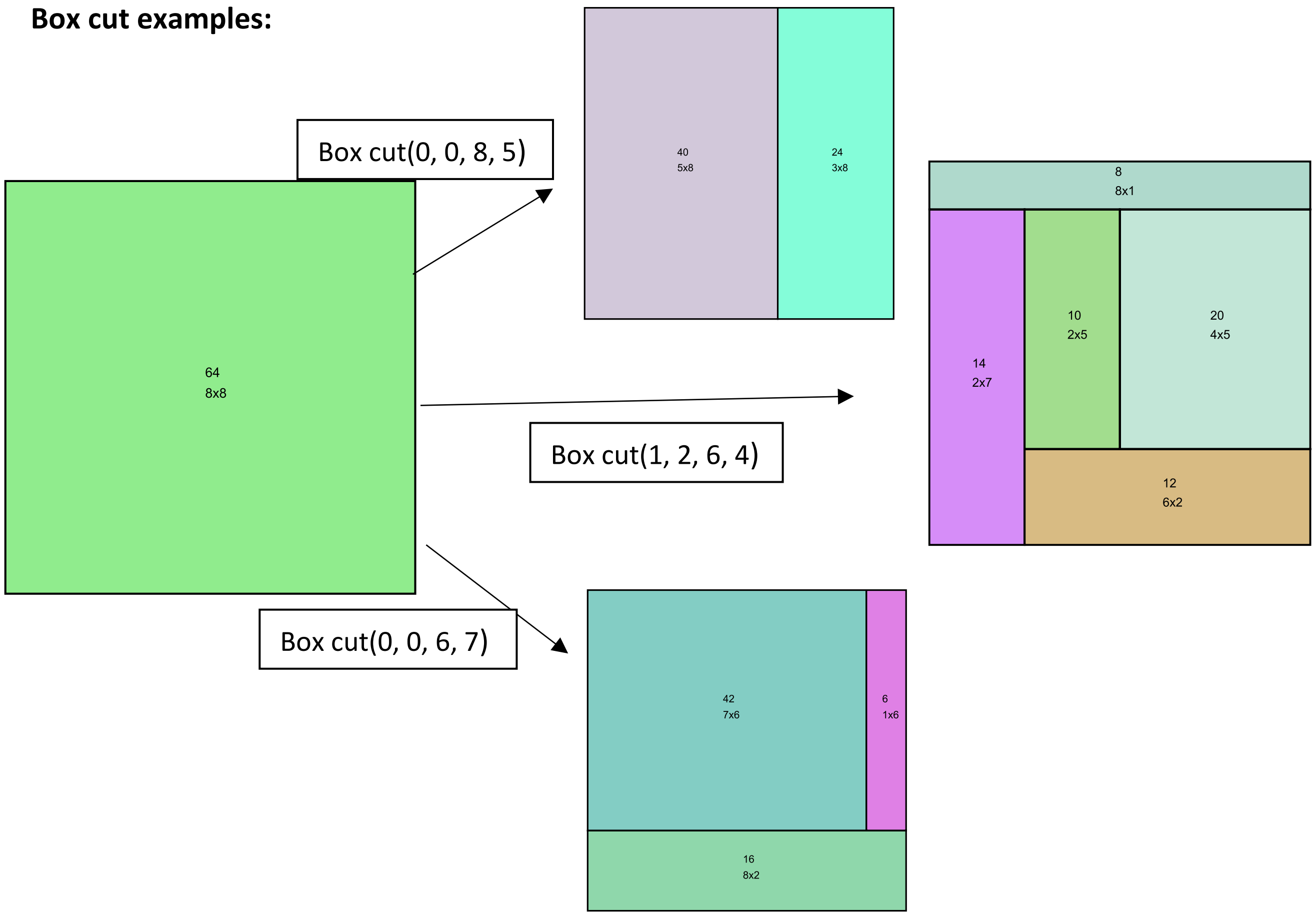
Choose the largest rectangle by area, rmax (if it is not unique, an rmax is chosen arbitrarily from the possible values) and perform a split (x1, y1, x1, y2) where x1,y1,x2,y2 are integers >=0, with constraints x1 < x2 <= $rmax_h$, y1 < y2 <= $rmax_w$. This split is performed by first splitting the rectangle horizontally at height x1, then the newly created bottom region vertically at width y1, then the new right region horizontally at height x2, then the new top region vertically at width y2. Any new rectangles with 0 area are discarded. Note if (x1, x2) = (0, $rmax_h$) or (y1,y2) =(0, $rmax_w$) then this is a purely horizontal or vertical split respectively.

Merging:

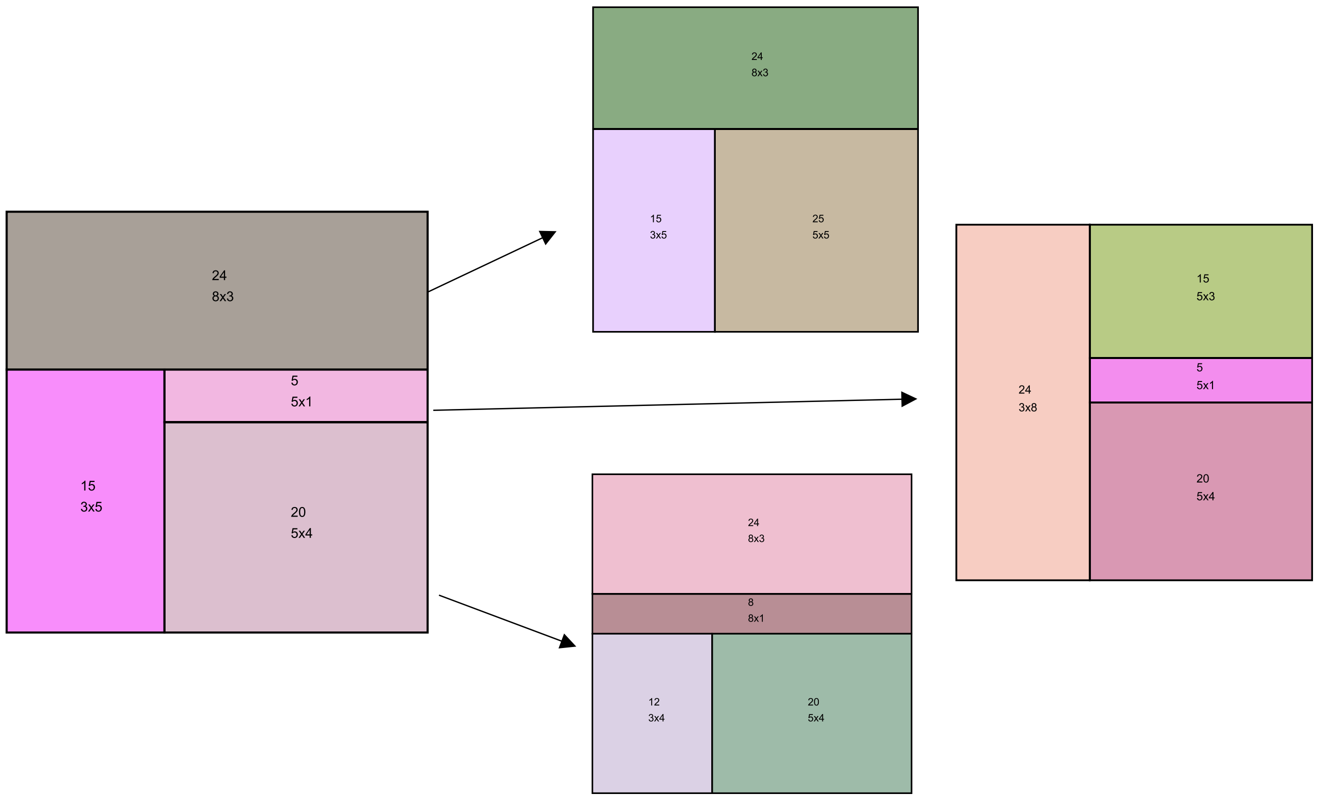
Choose any two rectangles which share a non-opposite corner (e.g. they share at least part of an edge at a corner). For the larger of the two rectangles, split off the area at this shared edge section (cut off a rectangle whose dimensions are the length of the shared edge on its axis, and the length of the larger rectangle on the other) and add it to the smaller rectangle. If these rectangles share a whole edge, this will merge the two rectangles into one (the area 0 rectangle is discarded). If they do not, the merge is only generated if the smaller rectangle is shorter than the larger one in the axis of the shared edge.

Some examples of each action are shown below:

Box cut examples:



Merge examples:



Section 2: Actions leading to invalid states

1.) The way I have defined the states and actions above can lead to invalid states. However, as both defined actions produce non-overlapping sets of integer-sized rectangles which partition the axa square, it is sufficient to ensure that there are no congruent rectangles in a valid state. Shown below is pseudocode for this purpose:

```
Function IsStateValid:

Input: A state S (including a set of arbitrarily ordered rectangles r=(r_x,r_y,r_w,r_h) as defined above)

Output: True (if the state is valid) or False (if the state is invalid)

Process:

For each rectangle r in state S:
    For each rectangle r' after r in state S:
        If $r_w = r'_w$ and $r_h = r'_h$ then return False
        If $r_h = r'_w$ and $r_w = r'_h$ then return False
    End for
End for
Return True
```

2.) I chose not to include transitions to invalid states as I found that they did not help in my algorithm to find many solutions which could not be uncovered through a path of valid states and including them would often reduce the search space of realistic valid solutions (invalid solutions can be better than valid solutions as they have less restrictions, so any BFS based search method will choose the best invalid states over the best valid states).

I tried limiting this issue by only allowing invalid states for one step (do not allow invalid states generated from other invalid states to be explored), and although this sometimes improved the solution it was more often detrimental to the final optimal Mondrian score.

Section 3: Computing the Mondrian score of a state

Shown below is pseudocode to compute the Mondrian score of a given state:

```
Function FindMondrianScore:

Input: A state S (including a set of arbitrarily ordered rectangles $r=(r_x,r_y,r_w,r_h)$ and board size a as defined above)

Output: score, the Mondrian score of state S ($\max_{r\in S}(r_w r_h)-\min_{r\in S}(r_w r_h)$)

Process:

If number of rectangles in S = 1 then return a**2

Set max_area = 0

Set min_area = $a^2$

For each rectangle r in state S:

    Set area = $r_w * r_h$

    If area < min_area then set min_area = area

    If area > max_area then set max_area = area

End for

Return max_area - min_area
```

Section 4: A discrete state-space search method

The state-space search method I chose to use is **beam search** (modified best first search). The limiting parameter **M** I will use is the **beam size** of the beam search (how many solutions do we search at every step). Another possible M could be depth (how many layers deep do we go to find the best solutions) while fixing the beam size. I found that increasing depth past a certain point made little difference to the solutions found, so I instead chose to limit by beam size and fix depth=10.

Results:

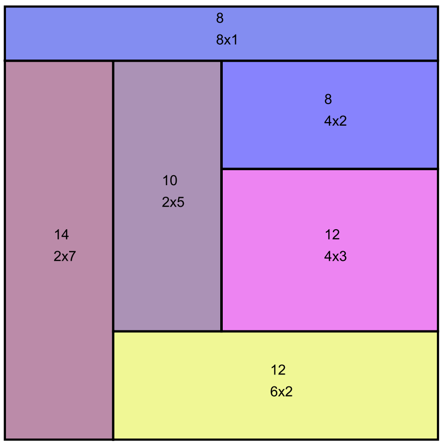
Shown below is a table of best Mondrian score found for different values of a (board size) and M (tree depth):

M Value \ a value	8	12	16	20
1	10	8	18	13
10	6	8	15	13
25	6	8	12	13
50	6	8	11	13

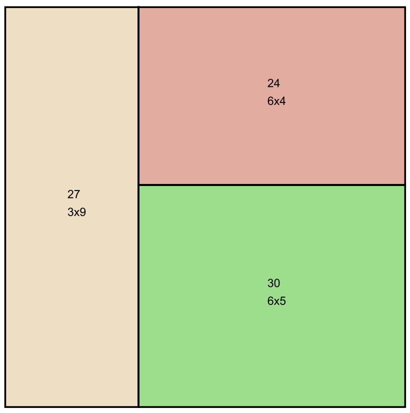
Optimal solutions found:

Here is a display of all optimal solutions found at M=25

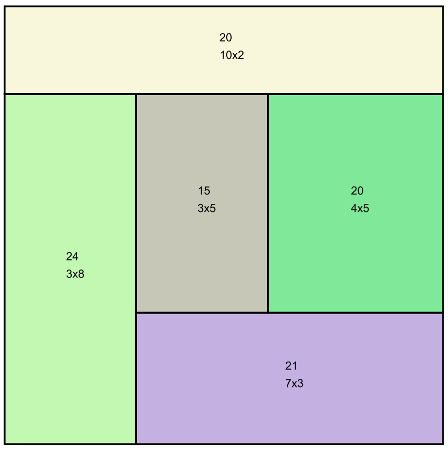
Board Size: 8x8 M=25 Mondrian score: 6



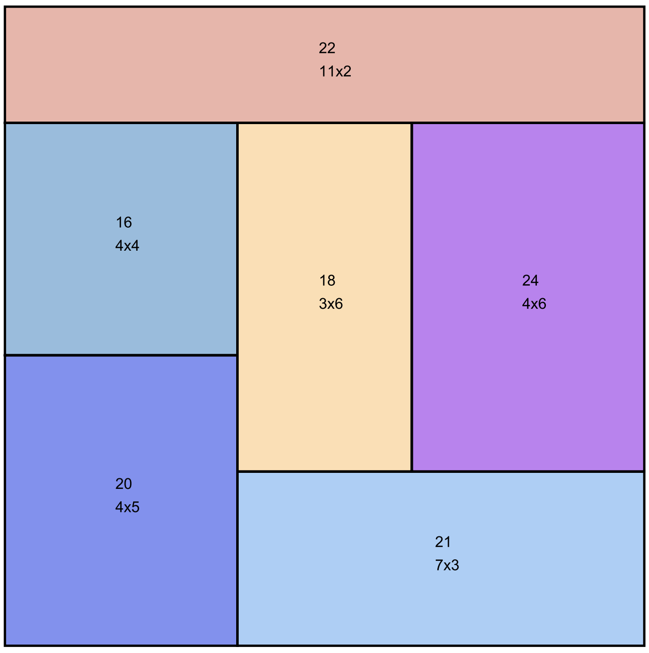
Board Size: 9x9 M=25 Mondrian score: 6



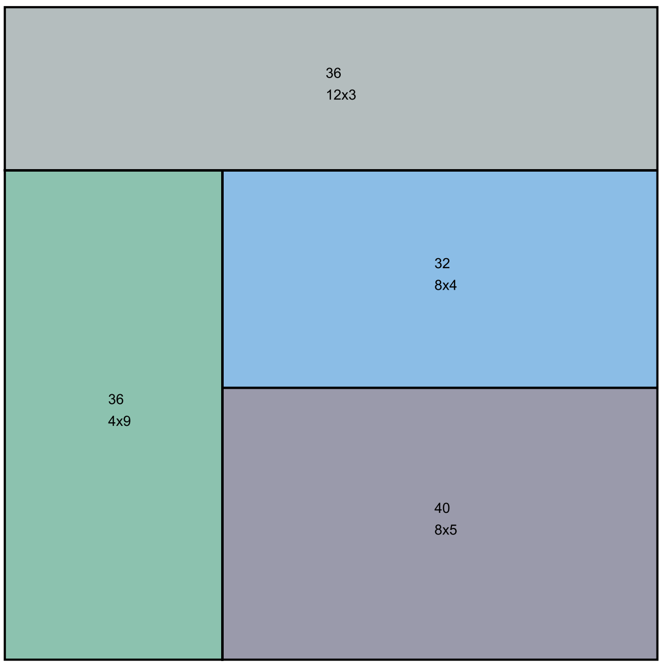
Board Size: 10x10 M=25 Mondrian score: 9



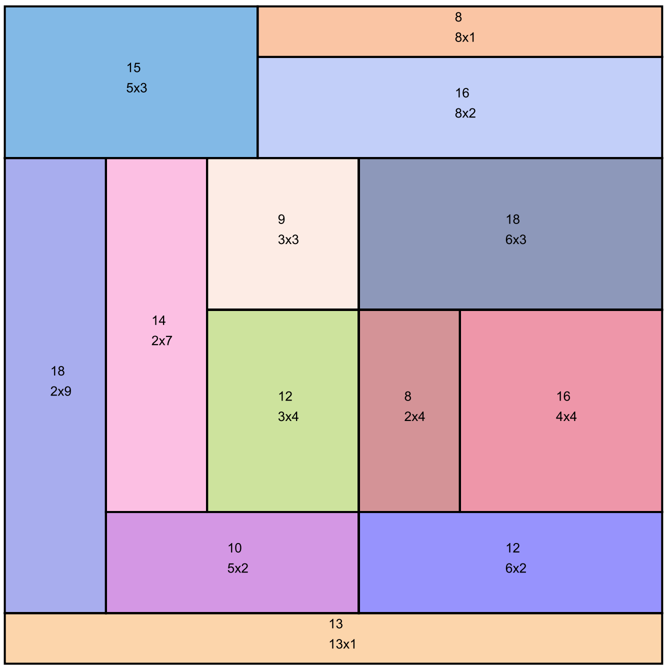
Board Size: 11x11 M=25 Mondrian score: 8



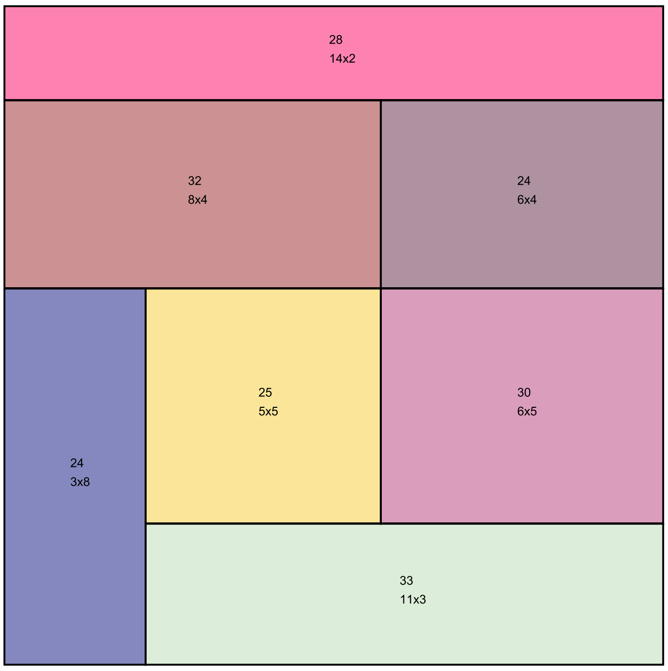
Board Size: 12x12 M=25 Mondrian score: 8



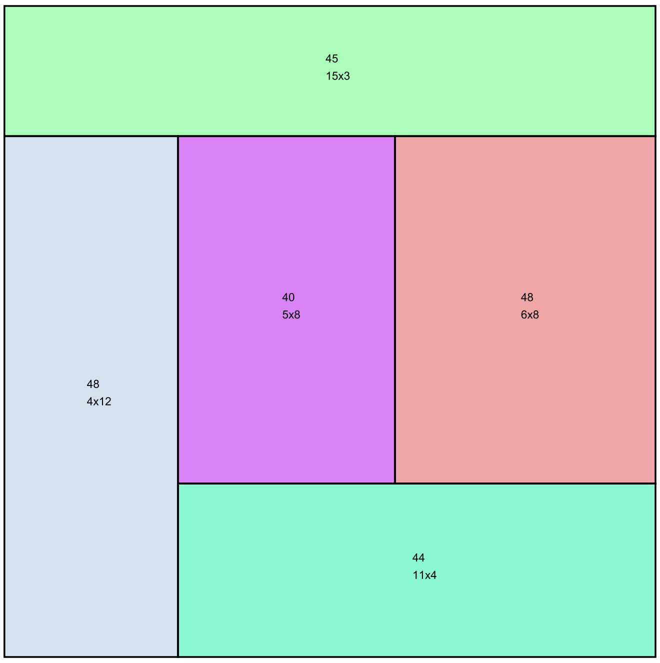
Board Size: 13x13 M=25 Mondrian score: 10



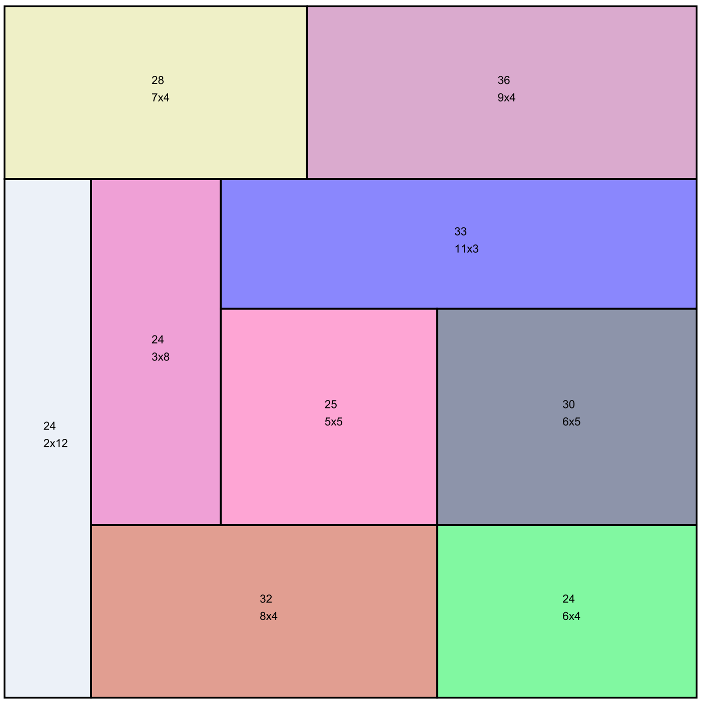
Board Size: 14x14 M=25 Mondrian score: 9



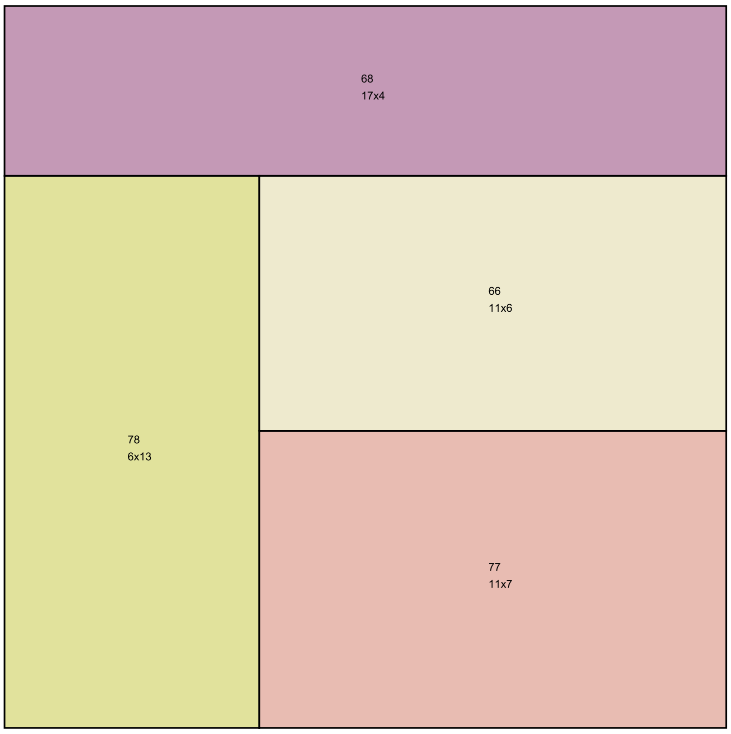
Board Size: 15x15 M=25 Mondrian score: 8



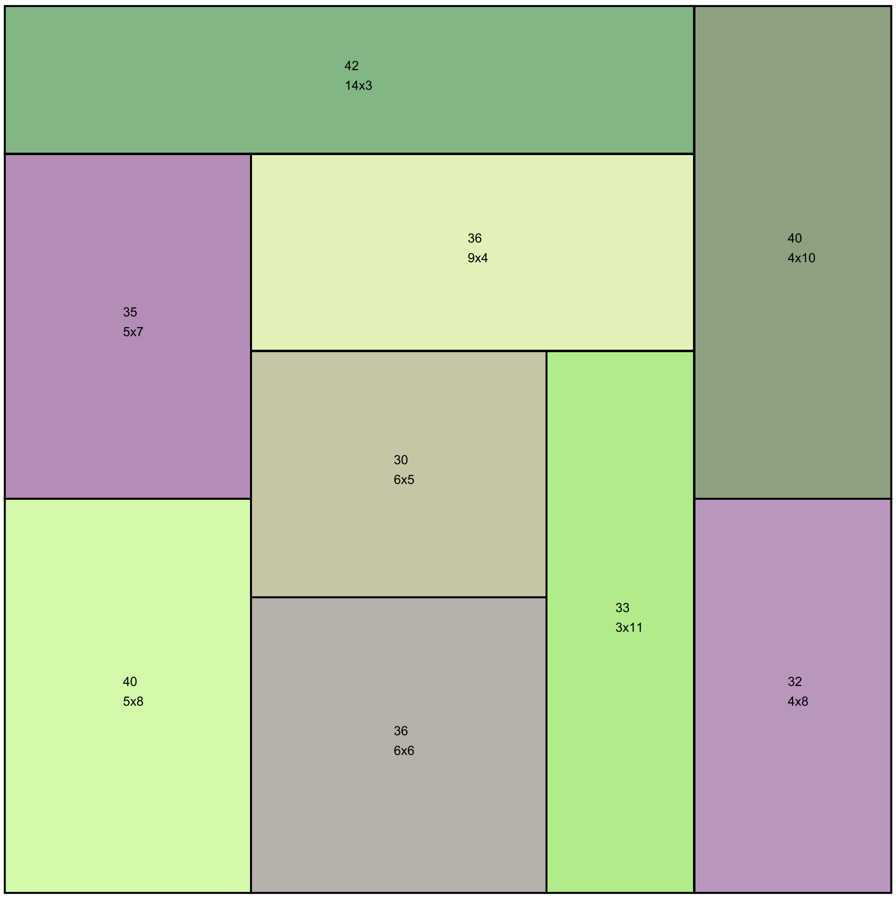
Board Size: 16x16 M=25 Mondrian score: 12



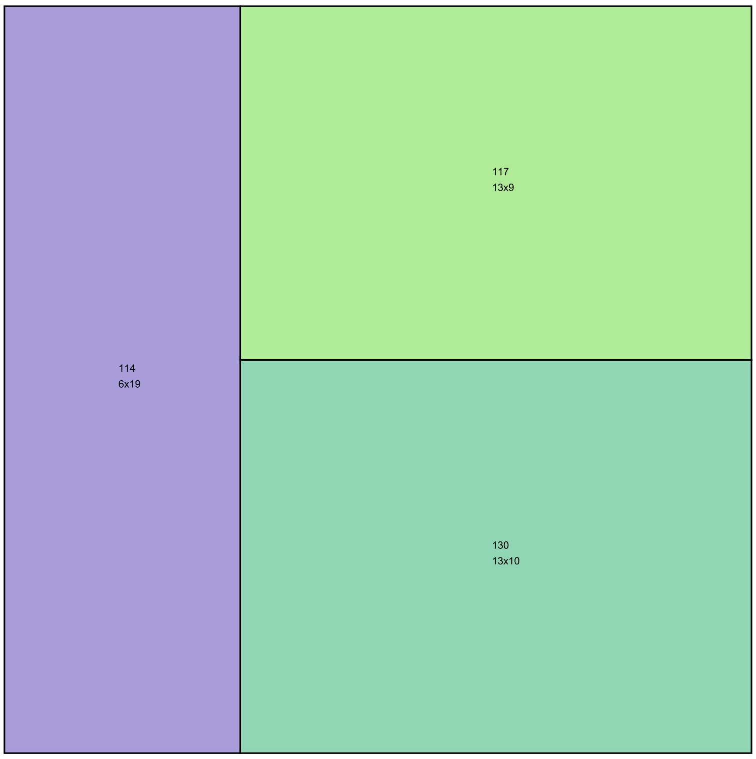
Board Size: 17x17 M=25 Mondrian score: 12



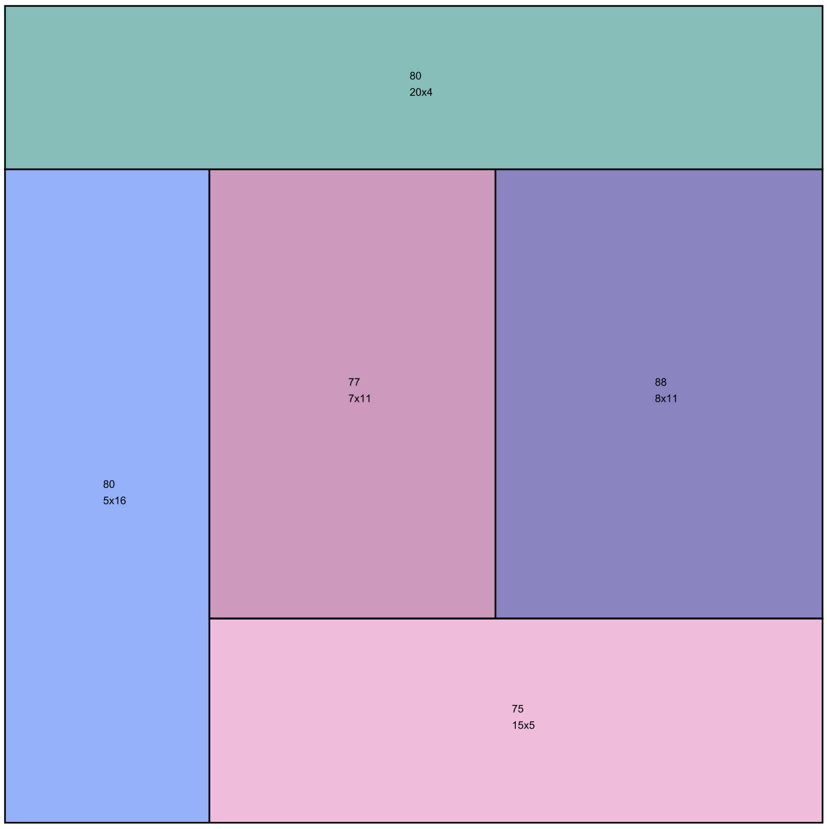
Board Size: 18x18 M=25 Mondrian score: 12



Board Size: 19x19 M=25 Mondrian score: 16



Board Size: 20x20 M=25 Mondrian score: 13



Evaluation:

We can see that as the beam size increases (e.g. the algorithm gets less greedy at every step) then it can sometimes find new better solutions (as it did at a=16). This is because considering a less initially optimal solution can allow for more space to improve upon the Mondrian score (the best solutions at a given depth often involve a few large rectangles of very large area, impossible to improve upon). Often the algorithm is still too greedy, and the start of the best solution is not to be found anywhere near the top 50 solutions at a given depth. This problem worsens at higher levels of a where the solution is deeper.

The algorithm becomes less efficient at higher M's, as the search will explore (9*M + 1) states, for each of which it will generate all possible actions. This grows very fast as M grows: if M doubles, it means considering approximately 18*A more actions, where A is the average number of actions per state (dependent on number of rectangles and largest rectangle of state).

Generalisation for arbitrary rectangles:

My solution is already designed to work for a rectangular board, as everything is defined for arbitrary rectangles, not just squares: states are already defined as non-overlapping rectangles which partition the space, and every action is performed on rectangles or pairs of rectangles and will not exceed the space of the rectangles involved in the operation. The only change needed would be to change the state parameter a to a vector **a** = ($board_w$, $board_h$) and instead initialise the board state with a single rectangle (0, 0, $board_w$, $board_h$).