Mondrian Tiling Problem

2479054v

November 2022

1 Introduction

This report presents a solution and its performance for Mondrian Tiling Puzzle. To run the solution **Python 3.10** or above is required. **Numpy** package is also needed.

2 Defining States and Actions

2.1 States

`State space` for the given problem is a finite set $S = \{s_0, s_1, \ldots, s_n \mid n \in \mathbb{N}\}$ where s_i is some valid configuration of the space (except s_0). Note, s_0 denotes the initial state, i.e. singular $N \times N$ box. Any state s_i is fully specified by an array of rectangles where no two rectangles have same or congruent dimensions.

2.2 Actions

`Action` that can be taken are as follows:

- `split_vertical(rectangle, ratio)` - splits selected rectangle vertically by given ratio
- `split_horizontal(rectangle, ratio)` - splits selected rectangle horizontally by given ratio

Each of these actions returns a new pair of rectangles emerging from the split or `None` if the action constraints are violated (see section 3).
As for `merging` rectangles, it is not a necessary actions as by splitting rectangles by some ratio, we can always arrive at same solution as by merging. Moreover, merging presents an issue of cycles as merging transforms otherwise unidirectional decision tree of state transitions to bidirectional one. Additionally, it requires further constraints thus complicating the problem.

2.3 Transition Between States

Figure 1 shows the transition from state on the left to state on the right by splitting the red rectangle using `split_horizontal(red, (1, 3))`.

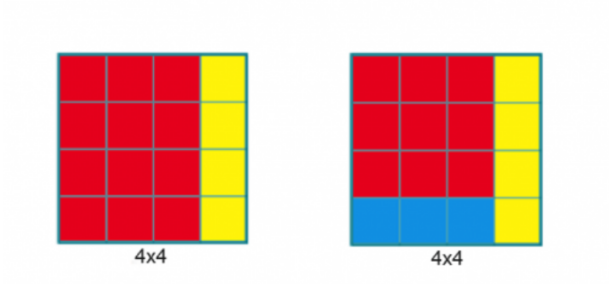

Figure 1: Image from www.bbvaopenmind.com

3 State Validity

Variety of actions can be taken, some of which lead to invalid states. Thus, the following constraints have been introduced:

1. Area of any rectangle must be ≥ 1
2. Height and width of any rectangle must $\in \mathbb{N} - \{0\}$
3. Height and width of any two rectangles can not be the same nor congruent

3.1 Checking State Validity

We can check if an action will yield a valid state as follows:

```
def is_valid(s:State) -> bool:
    seen = set()

    for r in s.rectangles:
        rw:int, rh:int = r.dimensions

        if rw <= 0 or rh <= 0:
            return False
        elif (rw, rh) in seen or (rh, rw) in seen:
            return False
        else:
            seen.add((rw, rh))
            seen,add((rh, rw))

    return True
```

Inputs:

- `s:State` - current state, `s.rectangles` returns list of all rectangles representing the current state configuration

Algorithm initiates by creating set to help us track what dimensions we have seen already. Then iterating over each rectangle `r` algorithm checks if each rectangle satisfies the already mentioned constraints.
By construction we ensure that dimensions of a rectangle are `int` and not `float`. This follows by checking that the dimensions of a rectangle follow the 2^{nd} rule. If this is true, then 1^{st} rule is also satisfied. Lastly, the elif/else branch make sure that no two rectangles have same or congruent dimensions as per 3^{rd} rule. If not rectangle violates the constraints, `True` is returned signaling the state is valid. Notice, checks if rectangles overlap are not required as per the nature of the defined actions. We are always splitting existing rectangles not placing new ones, thus overlap never happens.
Average time complexity is $O(n)$

3.2 Transition to Invalid States

Actions leading to invalid states will not be considered. Such actions would pose increasing complications in terms of implementation. Moreover, as it was already explained, merging is not considered as an action. This in turn limits possible recovery from invalid states, for example if there were 2, 1×1 rectangles, the only viable option for recovery from such a state would be merging. Thus, as of now, it is not deemed as worth while investment.
Nonetheless, it presents and interesting venue to explore in the future to see relative performance to the current solution.

4 Computing Mondrian Score

We can compute Mondrian Score as follows:

```
def score(s:State) -> int | float:
    if len(s.rectangles) == 1:
        return float("inf")

    if not is_valid(s):
        return float("inf")

    return max(s.rectangles, key=lambda r: r.area) \
    - min(s.rectangles, key=lambda r: r.area)
```

Inputs:

- `s:State` - current state, `s.rectangles` returns list of all rectangles representing the current state configuration

Algorithm begins by checking if the number of rectangles is $= 1$. If that is true, it means we are in the initial state s_0 and thus "invalid" score is returned in from of $+inf$. If we are not in the initial state, we check if current state is valid

(using algorithm described in section 3.1). If not, we return the "invalid" score. Else, return the area difference between rectangles with maximum and minimum area.

5 Solution

This section presents a solution to `Mondrian Tiling` problem as well as statistical comparison of the performance for different dimensions and depth limits of the problem.

5.1 Algorithm

A modification of the `Best First Search` algorithm was used. Algorithm works by selecting a move that yields best score M moves ahead. This algorithm was chosen over normal `Best First Search` as it simply does not consider relative best move at the current level. As it can be seen in some solutions, we might pick move which is relatively worse to its peers, however yields better long-term result.
`Depth & Breath First Search` algorithms are not suited for this problem due to the large size of the `State Space` as well as lack of specific goal state. In this problem we assume we do not know best possible score for the tiling, thus, the goal state either becomes the first state with no further transitions or we have to search entire `State Space` which becomes exponentially more costly with larger dimensions.
A^* was also considered, however, no heuristic function was found which could be used for this algorithm. Note that if one was actually found, this algorithm would be preferred as it can find the best path by only comparing moves at the current level instead of having to "look into the future" which becomes more and more expensive with large tiling problems.

5.2 Adapting to $A \times B$ Tiling

This solution can be easily changed from $A \times A$ to $A \times B$ Tiling problem. Only change required would be changing generation of valid `ratios` which are supplied to `split` functions. At the moment, these are generated by double `for` loop with iterates over $[1 \dots A-1]$. This could be trivially changed to accommodate for different dimensions.

5.3 Results

This section presents results for different Mondrian Tiling problems. The limiting factor M is the maximum depth, i.e. the maximum number of future moves we consider when deciding on the current one. Note, this algorithm becomes simple `Best First Search` if $M = 0$. Additionally, when rectangle to be split is a square, we only consider splitting it vertically, as splitting it horizontally will yield same result and thus is redundant branch to explore.

5.3.1 Observations

As expected performance of the algorithm degrades proportionally to the size of the `Tiling Rectangle`. The only exception is the `8x8` Tiling.This can be explained by the fact that once the maximum depth reaches 8, the algorithm traverses the entire state space and find the branch leading to optimal solution. Thus, further increase in depth has no longer any effect of the performance as can be noted in Figure 2. The ideal depth for this Tiling Problem is 5 as it is the smallest depth where optimal solution can be reached.
As for other problems, only the `12x12` was able to achieve optimal solution within reasonable time limit. The optimal solution of 7 was achieved first for $M = 9$ with the processor time ≈ 55.3 seconds. The exponential increase in the execution time with respect to maximum depth is becoming rather noticeable (see Figure 2). The execution time increases by factor of ≈ 1.9.
As for `16x16` and `20x20` neither was able to achieve optimal solution. The `16x16` reached score 11 for $M = 7$ with execution time ≈ 115.3 seconds. The `20x20` achieved best score of 13 with $M = 6$ with execution time ≈ 111.3 seconds. The execution time increased by factors of ≈ 5 and ≈ 7.26 respectively.

5.3.2 Figures

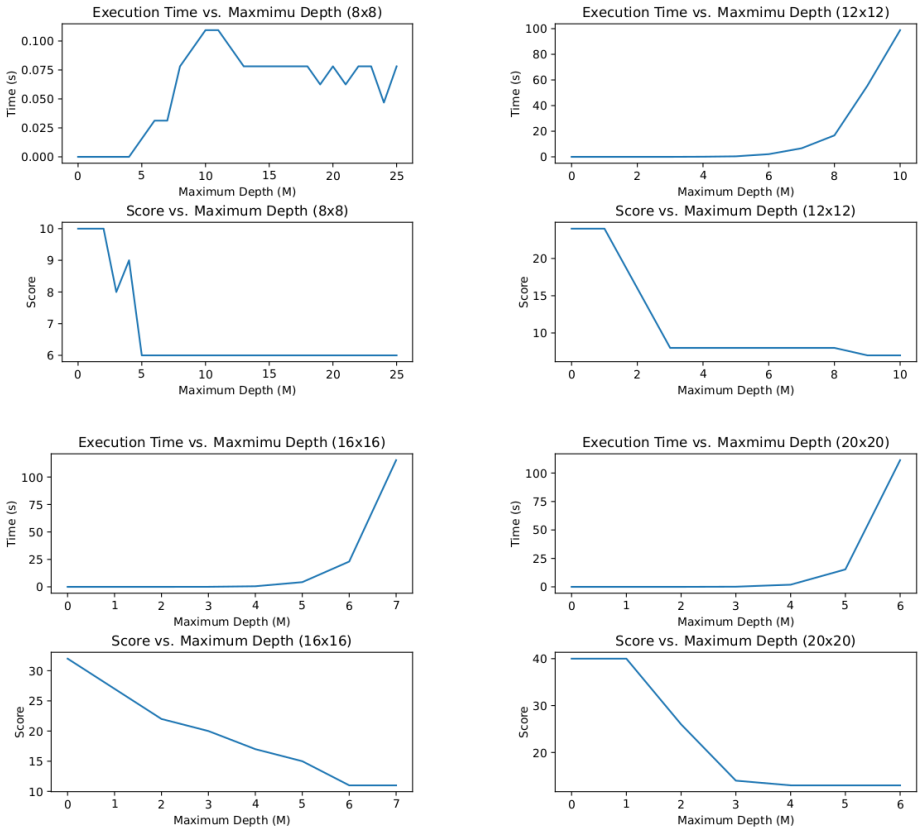

Figure 2: 8x8, 12x12, 16x16, 20x20 performance measurements

5.3.3 Visualization

This section presents each solution in a visual format similar to `https://oeis.org/A276523`.

```
[['A' 'A' 'A' 'A' 'A' 'A' 'A' 'A']
 ['d' 'd' 'b' 'b' 'b' 'b' 'b' 'b']
 ['d' 'd' 'b' 'b' 'b' 'b' 'b' 'b']
 ['d' 'd' 'C' 'C' 'E' 'E' 'E' 'E']
 ['d' 'd' 'C' 'C' 'E' 'E' 'E' 'E']
 ['d' 'd' 'C' 'C' 'f' 'f' 'f' 'f']
 ['d' 'd' 'C' 'C' 'f' 'f' 'f' 'f']
 ['d' 'd' 'C' 'C' 'f' 'f' 'f' 'f']]
```

(a) 8x8 Tiling, $M = 5$, $Score = 6$

```
[['A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A']
 ['f' 'f' 'f' 'b' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C']
 ['f' 'f' 'f' 'b' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C']
 ['f' 'f' 'f' 'b' 'h' 'h' 'd' 'd' 'd' 'd' 'd' 'd']
 ['f' 'f' 'f' 'b' 'h' 'h' 'd' 'd' 'd' 'd' 'd' 'd']
 ['f' 'f' 'f' 'b' 'h' 'h' 'E' 'E' 'I' 'I' 'I' 'I']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'I' 'I' 'I' 'I']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'I' 'I' 'I' 'I']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'j' 'j' 'j' 'j']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'j' 'j' 'j' 'j']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'j' 'j' 'j' 'j']
 ['G' 'G' 'G' 'b' 'h' 'h' 'E' 'E' 'j' 'j' 'j' 'j']]
```

(b) 12x12 Tiling, $M = 9$, $Score = 7$

```
[['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G']
 ['G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G' 'G']
 ['b' 'b' 'b' 'b' 'd' 'd' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f']
 ['b' 'b' 'b' 'b' 'd' 'd' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f']
 ['b' 'b' 'b' 'b' 'd' 'd' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f' 'f']
 ['b' 'b' 'b' 'b' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'C' 'C' 'C' 'C' 'C']
 ['b' 'b' 'b' 'b' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'C' 'C' 'C' 'C' 'C']
 ['b' 'b' 'b' 'b' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'C' 'C' 'C' 'C' 'C']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'C' 'C' 'C' 'C' 'C']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'C' 'C' 'C' 'C' 'C']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']
 ['h' 'h' 'h' 'h' 'd' 'd' 'I' 'I' 'I' 'A' 'A' 'E' 'E' 'E' 'E' 'E']]
```

(c) 16x16 Tiling, $M = 7$, $Score = 11$

```
[['C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'  'C' 'C']
 ['C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'  'C' 'C']
 ['C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'  'C' 'C']
 ['C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C' 'C'  'C' 'C']
 ['d' 'd' 'd' 'd' 'd' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'  'A' 'A']
 ['d' 'd' 'd' 'd' 'd' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'  'A' 'A']
 ['d' 'd' 'd' 'd' 'd' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'  'A' 'A']
 ['d' 'd' 'd' 'd' 'd' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'  'A' 'A']
 ['d' 'd' 'd' 'd' 'd' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A' 'A'  'A' 'A']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']
 ['d' 'd' 'd' 'd' 'd' 'b' 'b' 'b' 'b' 'b' 'b' 'b' 'E' 'E' 'E' 'E' 'E' 'E'  'E' 'E']]
```

(d) 20x20 Tiling, $M = 6$, $Score = 13$

Figure 3: 8x8, 12x12, 16x16, 20x20 solution visualization