Lebanese University
Faculty of Sciences
Section II - Fanar

CUTTING BOARD project

Final Report

Section: English
Group: B33
Supervisor: Dc. Joseph Merhej

59502 – Raphael El Mouallem

ABSTRTACT

This project explores the development and application of nesting algorithms, with a focus on their implementation using Python. Nesting algorithms, crucial in manufacturing and material optimization, arrange a set of shapes within a defined boundary efficiently to minimize waste. The project encloses the creation of Python modules designed to implement two nesting algorithms on three set of shapes, generating a dataset of nested shapes. This dataset serves as the foundation for training a machine learning model, aiming to explore improvements concerning precision and efficiency of nesting processes. By integrating algorithmic design, programming, and machine learning, this project demonstrates possibilities in optimizing material usage, contributing to both academic research and practical applications in industries such as manufacturing and logistics.

Catalog

1- Bibliography 5
2- Algorithms 6
3- Data sets 32
4- Machine Learning 40
5- Conclusion 42

Introduction

The efficient utilization of materials is a critical concern in various industries, particularly in manufacturing, logistics, and construction. One of the key challenges in this domain is the nesting problem, which involves arranging irregularly shaped objects within a bounded area in such a way as to minimize waste in order to reduce cost and material usage.

This project focuses on the study and implementation of nesting algorithms using Python. The primary objectives include developing Python modules to perform nesting, creating a dataset of nested shapes, and using this dataset to train a machine learning model to further improve nesting efficiency.

Nesting problems are computationally complex and often require heuristic or approximation methods to find near-optimal solutions (nearly never an exact solution). Traditional methods, while effective to some extent, can be limited in their adaptability and efficiency due to their heavy reliance on human input. Machine learning, with its ability to learn from data and improve over time, presents a promising avenue for enhancing nesting algorithms.

The first phase of this project required an in-depth review of existing nesting algorithms and their implementations. This review informs the design and development of custom Python modules that can generate and manipulate nested shapes.

The second phase required the generation of a dataset that would be, after treating, cleaning, feature engineering and normalising it, utilised to train a machine learning model. The performance of the trained model is evaluated against traditional methods to assess improvements in efficiency and material utilization.

PS:

The main focus of this project was to develop and train a machine learning (ML) model capable of efficiently nesting 2D irregular shapes. However, an initial challenge arose due to the unavailability of a suitable dataset tailored to the project's requirements. Consequently, the need arose to construct a dataset from scratch.

Unfortunately, existing libraries for dataset generation were incompatible with the ARM64 architecture of the Mac M1 machine, necessitating a reassessment of the project's approach. This led to a return to the foundational stage, where extensive research into algorithms relevant to 2D nesting was conducted.

Subsequently, efforts were directed towards successfully implementing these algorithms, constructing 2D nesting libraries, and devising a custom script for dataset generation of nested 2D irregular shapes.

The new objective of this project is to explore and apply algorithms related to 2D irregular shape nesting, in order to create a dataset that can be used to train an ML.

1- Bibliography

The field of 2D nesting algorithms has been explored extensively, with numerous studies focusing on heuristic and approximation methods. While machine learning applications in 2D nesting exist, they are predominantly private and not open source. The following works provide foundational insights and methodologies relevant to this project:

- Genetic Algorithms for the Nesting Problem in the Packing Industry by Roberto Selow and Heitor S. Lopez.
- Applying Simulated Annealing and the No Fit Polygon to the Nesting Problem by Edmund Burke and Graham Kendall.
- A simulated annealing approach to the nesting problem in the textilemanufacturing industry by Ralf Heckmann and Thomas Lengauer.
- A Fully General, Exact Algorithm for Nesting Irregular Shapes by Donald R. Jones.
- Rectangular exact nesting algorithm by Defu Zhanga, Yan Kangb and Ansheng Denga, A new heuristic recursive algorithm for the strip rectangular packing problem

These references form the basis of the theoretical and practical approaches used in this project, guiding the development of Python modules and the creation of a dataset for training a machine learning model to enhance nesting efficiency. Given the challenges associated with the complexity and implementation of these algorithms, the project emphasizes a practical approach to developing feasible and efficient solutions within the available time frame.

2- Algorithms

In this project, three distinct Python modules were developed to address the nesting problem for different types of shapes: two modules for regular shapes (circles and rectangles) and one module for irregular shapes. Each module was designed to efficiently arrange the shapes within a predefined boundary to minimize material waste. The following sections detail the approach and functionality of each module.

1. **Circle Nesting Algorithm**: Grid-Based Greedy Packing

The circle nesting module uses a Grid-Based Greedy Packing algorithm, which sorts circles from largest to smallest radius and places them systematically on a grid. Also support hallow circles in the packing process.

2. **Rectangle Nesting Algorithm**: Heuristic Approach

The rectangle nesting module employs a heuristic algorithm to place rectangles within a boundary efficiently. Also support the nesting of hallow rectangles with thicknesses varying on each of its sides.

3. **Irregular Shape Nesting Algorithm**: Grid-Based Greedy Packing

Similar to the circle nesting algorithm, the irregular shape nesting module uses a Grid-Based Greedy Packing algorithm. It detects shapes from an image using open-cv2 and uses ray-casting to detect collisions between the vertices of 2or more polygons.

2-1- Circles:

The circle nesting module uses a method called the Grid Laying Algorithm, a greedy algorithm designed to efficiently place circles within a defined boundary. This approach sorts the circles from largest to smallest radius and places them systematically on the x and y axes. Here's a detailed breakdown of the algorithm:

Algorithm overview:

- **Sorting**: The circles are initially sorted in descending order based on their radius. This ensures that the largest circles are placed first, which generally leads to more efficient use of space.
- **Initialization**: The algorithm starts by setting the coordinates of the first circle to (r, r), where r is the radius of the first circle being nested.
- **Step Parameter**: A user-defined step parameter determines the granularity of the grid search. The default value is 1. A smaller step size increases the accuracy of the nesting but also increases computation time.
- **Grid Iteration**: The algorithm iterates over the x-axis in increments of the step size, starting from x = y = r. If no suitable position is found along the x-axis for a given y, the y-coordinate is incremented by the step size, and the x-coordinate is reset to r.
- **Feasibility Check**: For each potential position, the algorithm checks if the circle can be placed without overlapping any previously placed circles or bring pushed outined the frame.

  This is done by verifying if x + radius and y + radius are less than or equal to the frame's width and height, respectively.

  Similarly, we check if x - radius and y - radius are greater than or equal to zero to ensure the circle stays within the frame's bounds.

  For each circle already placed within the frame, we calculate the distance between the centre points of the current circle and the placed circle.

  This distance is compared to the sum of their radii (r1 + r2). If the distance is less than the sum of the radii, it indicates a collision, and the current grid point is deemed unsuitable.

```
for c in nested_circles:
    if Circle.r + c.r < dist(Circle.x, Circle.y, c.x, c.y):
        return FALSE
```

- **Storage**: Successfully placed circles are stored in a list of nested circles.

Hollow Circles (Frame Circles): The algorithm also supports nesting hollow circles or "frame circles," which are circles with a defined thickness and empty space inside. To optimize the use of space within these frame circles:

- **Sorting**: Hollow circles are sorted from largest to smallest radius. Circles with the same radius are further sorted by the greatest empty area inside the frame.
- **Nested List**: Frame circles are stored in a nested list structure, where each frame circle can have a list of circles nested inside it.
- **Inner Circle Check**: Before checking if a circle can fit within the boundary, the algorithm checks if the circle can fit inside any already nested frame circles.

Complexity : O(nlogn + nG + n^2)

Sorting Circles: O(nlogn)
Grid Seaarch: O(nG) with G = number of positions = (1/step) * (frame.width * frame.height)
Collision Check: O(n^2) because every circle is checking collision with all circles

Module Overview:

The module consists of 4 classes, Circle to define a circle, Frame to define a frame, Algorithm that encloses the agorithm and a Help class as well a Credit function so that the user can refer to documentation:

```
  1  import matplotlib.pyplot as plt
  2  from matplotlib.patches import Circle as Circles
  3  import math
  4  import random
  5  from typing import List, Tuple
  6  import matplotlib.axes._axes as Axes
  7
  8  class Circle: ⋯
103
104  class Frame: ⋯
159
160  class Algorithm: ⋯
338
339  class Help: ⋯
412
413  def Credit() -> None: ⋯
419
```

1-**Circle class:** Contains 9 methods.

```python
class Circle:
    def __init__(self, r : float,  thickness : float = 0, x : float = - 1, y : float = -1) -> None:
        self.r: float = 0

        try:
            if isinstance(r, float) or isinstance(r, int):
                if r <= 0:
                    raise ValueError("Radius must be a positive float/int")
                self.r: float = r
        except ValueError as e:
            print(e)

        self.x : float = x
        self.y : float = y
        self.thickness : float = 0

        try:
            if isinstance(thickness, float) or isinstance(thickness, int):
                if thickness < 0 or thickness >= r:
                    raise ValueError("Thickness must be a non-negative float/int less than the radius")
                self.thickness : float = thickness
        except ValueError as e:
            print(e)

        self.__area : float = math.pi * (self.r ** 2)
        self.__empty_area : float = 0

        if(self.thickness > 0):
            self.__nested_circles : List[Circle] = []
            self.__empty_area = math.pi * ((self.r - self.thickness) ** 2)

```

```python
    def add_nested_circle(self, nested_circle : 'Circle') -> bool:
        if(self.thickness > 0 and self.can_add(nested_circle)):
            self.__nested_circles.append(nested_circle)
            return True
        else:
            print("Cannot nest circles in a filled circle or in a smaller empty circle")
            return False

    def get_circle_data(self) -> str:
        string1 = "radius: " + str(self.r) + " thisckness: " + str(self.thickness) + " x: " + str(self.x) + " y: " + str
        (self.y) + " area: " + str(self.__area)
        string2 = ""
        string3 = ""
        if(self.thickness > 0):
            string2 = "\n\tnumber of nested circles: " + str(len(self.__nested_circles))
            for circle in self.__nested_circles:
                string3 += "\n\t\t" + circle.get_circle_data()

        return string1 + string2 + string3

    def get_nested_circles(self) -> List['Circle'] : return self.__nested_circles
```

```
60      def plot_circle(self, ax : Axes) -> None:
61          radius = self.r
62          thickness = self.thickness
63          x = self.x
64          y = self.y
65          center = (x, y)
66
67          color = (random.random(), random.random(), random.random())
68
69          if self.thickness:
70              # Draw the colored circle representing the main body
71              colored_circle_patch = Circles((x, y), radius, edgecolor='black', facecolor=color)
72              ax.add_patch(colored_circle_patch)
73
74              # Draw the white circle on top to simulate thickness
75              white_circle_patch = Circles((x, y), radius - self.thickness, edgecolor='black', facecolor='white')
76              ax.add_patch(white_circle_patch)
77          else:
78              # Draw the circle without thickness
79              circle_patch = Circles((x, y), radius, edgecolor='black', facecolor=color)
80              ax.add_patch(circle_patch)
81
82          # Add a text annotation to mark the center, radius, thickness, and position of the circle
83          ax.text(x, y, f"Center: ({x}, {y})\nRadius: {radius}\nThickness: {thickness}\nPosition: ({x}, {y})", ha='center',
            va='center', fontsize = 5)
84
85          if thickness > 0:
86              ax.text(x, y - 2*radius, f"Thickness: {thickness}", ha='center', va='center', fontsize=5)
```

```
88      def compare_to(self ,circle : 'Circle') -> bool:
89          return self.__area > circle.__area
90
91      def can_add(self, circle : 'Circle') -> bool:
92          if(self.thickness > 0):
93              if isinstance(circle, Circle) and (self.r - self.thickness > circle.r):
94                  return True
95
96          return False
97
98      def get_area(self) -> float:
99          return self.__area
100
101     def get_empty_area(self) -> float:
102         return self.__empty_area
```

2-Frame class: Contains 7 methods

```python
class Frame:
    def __init__(self, width : float, height : float) -> None:
        self.width : float = 0
        self.height : float = 0

        try:
            if isinstance(width, float) or isinstance(width, int) or isinstance(height, float) or isinstance(height, int):
                if height <= 0 or width <= 0:
                    raise ValueError("Dimensions must be a positive float/int")
                self.width : float = width
                self.height : float = height
        except ValueError as e:
            print(e)

        self.__area : float = width * height
        self.__occupied_area : float = 0
        self.__nested_circles : float= []

    def add_circle(self, circle : Circle) -> bool:
        if isinstance(circle, Circle) and (self.__occupied_area + circle.get_area() - circle.get_empty_area() <= self.__area):
            self.__nested_circles.append(circle)
            self.__occupied_area += circle.get_area() - circle.get_empty_area()
            return True
        return False

    def get_area(self) -> float: return self.__area

    def get_occupied_area(self) -> float: return self.__occupied_area

    def get_nested_circles(self) -> List[Circle]: return self.__nested_circles


    def get_frame_data(self) -> str:
        string1 = "width: " + str(self.width) + " height: " + str(self.height) + " area: " + str(self.__area) + " occupied area: " + str(self.__occupied_area)
        string2 = "\n\tnumber of nested circles: " + str(len(self.get_nested_circles()))
        string3 = ""
        for circle in self.get_nested_circles():
            string3 += "\n\t\t" + circle.get_circle_data().replace("\t", "\t\t\t")

        return string1 + string2 + string3

    def plot(self) -> None:
        fig, ax = plt.subplots()
        ax.set_aspect('equal')
        ax.set_xlim(0, self.width)
        ax.set_ylim(0, self.height)

        for circle in self.get_nested_circles():
            circle.plot_circle(ax)

        ax.set_title('Nested Circles in Frame')
        ax.set_xlabel('Width')
        ax.set_ylabel('Height')
        ax.grid(True)

        plt.show()
```

3-Algorithm class: Contains 6 methods

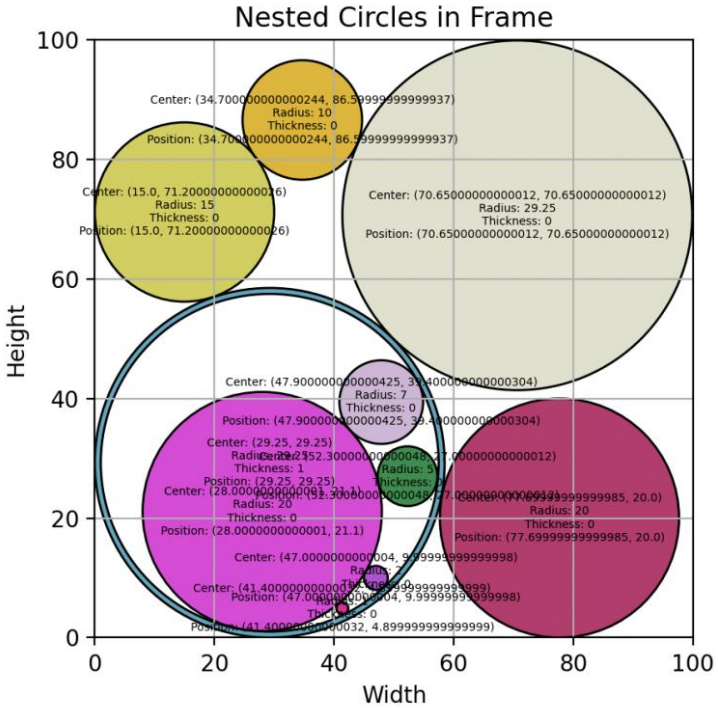
```python
class Algorithm:
    @staticmethod
    def sort_circles(circles : List[Circle]) -> List[Circle]:
        if all(isinstance(circle, Circle) for circle in circles):
            return sorted(circles, key=lambda circle: (-circle.r, -circle.get_empty_area()))
        else:
            raise ValueError("All elements in the circles list must be instances of the Circle class.")

    @staticmethod
    def sort_circle_by_radius(circles : List[Circle], reverse : bool = False) -> List[Circle]:
        if all(isinstance(circle, Circle) for circle in circles):
            return sorted(circles, key=lambda circle: circle.r, reverse=reverse)
        else:
            raise ValueError("All elements in the circles list must be instances of the Circle class.")

    @staticmethod
    def sort_circle_by_empty_area(circles : List[Circle], reverse : bool = False) -> List[Circle]:
        if all(isinstance(circle, Circle) for circle in circles):
            return sorted(circles, key=lambda circle: circle.get_empty_area(), reverse=reverse)
        else:
            raise ValueError("All elements in the circles list must be instances of the Circle class.")

    @staticmethod
    def distance(x1 : float, y1 : float, x2 : float, y2 : float) -> float:
        return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)


    @staticmethod
    def calculate_average_radius(circles : List[Circle]) -> float:
        total_radius = 0
        num_circles = len(circles)

        # Sum up the radii of all circles
        for circle in circles:
            total_radius += circle.r

        # Calculate the average radius
        if num_circles > 0:
            average_radius = total_radius / num_circles
            return average_radius
        else:
            return 0


    @staticmethod
    def nest_circles(circles : List[Circle], frame : Frame, step : float = 1, automatic : bool = False) -> Tuple[List
    [Circle], List[Circle]]:
        sorted_circles = Algorithm.sort_circles(circles)
        nested = []
        not_nested = []
        inside = []

        for circle in sorted_circles:
            nestable = True

            if((circle.r*2)**2 > frame.get_area()):
                not_nested.append(circle)
                continue

            if(circle.get_area() > frame.get_area() - frame.get_occupied_area()):
                not_nested.append(circle)
                continue

            avg = Algorithm.calculate_average_radius(circles)

            if(c (variable) new_step: float 1 and automatic):
                new_step = step * (circle.r / avg)
                print(new_step)
            else:
                new_step = step
```

```
228             if(inside):
229                 nestable = True
230                 for insider in inside:
231                     nestable = True
232                     if insider.r - insider.thickness >= circle.r - circle.thickness:
233                         minX = insider.x - insider.r + insider.thickness + circle.r
234                         maxX = insider.x + insider.r - insider.thickness - circle.r
235                         minY = insider.y - insider.r + insider.thickness + circle.r
236                         maxY = insider.y + insider.r - insider.thickness - circle.r
237 
238                         y = minY - new_step
239                         while(y < maxY):
240                             y += new_step
241 
242                             x = minX - new_step
243                             while(x < maxX):
244                                 x += new_step
245                                 if(Algorithm.distance(x, y, insider.x, insider.y) > (insider.r - insider.thickness -
                                    circle.r)):
246                                     nestable = False
247                                     continue
248                                 else:
249                                     nestable = True
250 
251                                 if(insider.get_nested_circles()):
252                                     for c in insider.get_nested_circles():
253                                         x1 = c.x
254                                         y1 = c.y
255                                         r1 = c.r
256                                         dist = Algorithm.distance(x, y, x1, y1)
257 
258                                         if(dist < r1 + circle.r):
259                                             nestable = False;
260                                             break
261                                 else:
262                                     circle.x = x
263                                     circle.y = y
264                                     nested.append(circle)
265                                     frame.add_circle(circle)
266                                     insider.add_nested_circle(circle)
267                                     if(circle.thickness > 0):
268                                         inside.append(circle)
269                                     nestable = True
270                                     break
271 
272                                 if nestable:
273                                     circle.x = x
274                                     circle.y = y
275                                     nested.append(circle)
276                                     frame.add_circle(circle)
277                                     insider.add_nested_circle(circle)
278                                     if(circle.thickness > 0):
279                                         inside.append(circle)
280                                     nestable = True
281                                     break
```

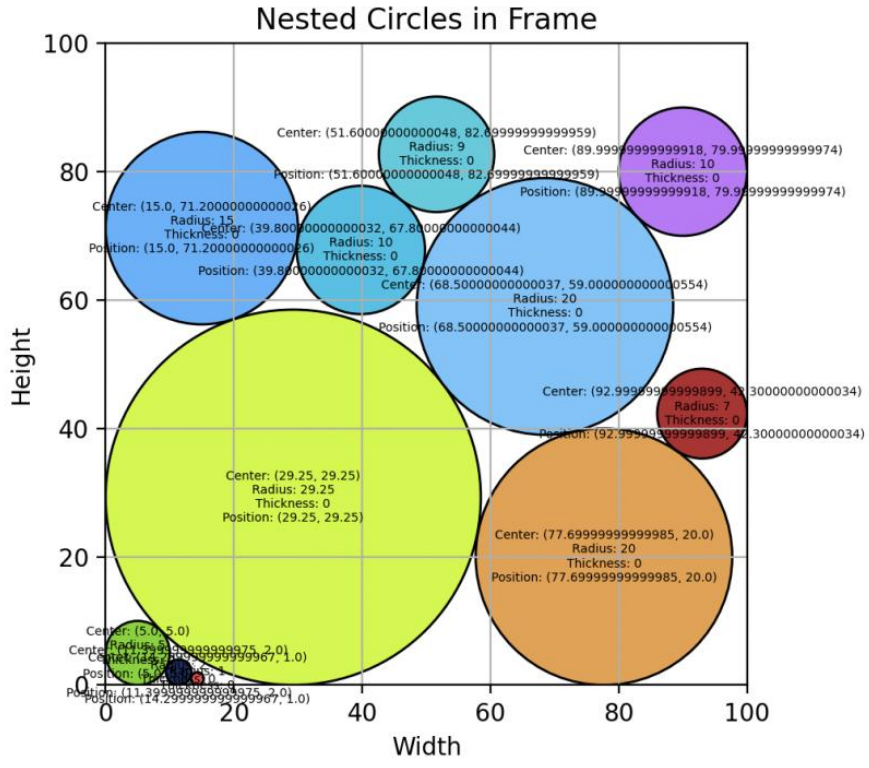
```
                        if nestable:
                            break

                    if nestable:
                        break

                else:
                    nestable = False

        if (not nestable or not inside):
            nestable = True

            y = circle.r - new_step
            while(y < frame.height - circle.r):
                y += new_step

                x = circle.r - new_step
                while(x < frame.width - circle.r):
                    x += new_step
                    if(not nested):
                        if(x + circle.r <= frame.width and y + circle.r <= frame.height):
                            circle.x = circle.r
                            circle.y = circle.r
                            nested.append(circle)
                            frame.add_circle(circle)
                            if(circle.thickness > 0):
                                inside.append(circle)
                            break

                    else:
                        nestable = True
                        for nest in nested:
                            x1 = nest.x
                            y1 = nest.y
                            r1 = nest.r
                            dist = Algorithm.distance(x, y, x1, y1)
                            if(dist < circle.r + r1  or circle.r + x > frame.width or circle.r + y > frame.height):
                                nestable = False
                                break

                        if nestable:
                            circle.x = x
                            circle.y = y
                            nested.append(circle)
                            frame.add_circle(circle)
                            if(circle.thickness > 0):
                                inside.append(circle)
                            break

                    if(nestable):
                        break

        if(not nestable):
            not_nested.append(circle)

    return nested, not_nested

```

Demo:

Nesting 8 normal circles and 1 hallow circle:



Nesting normal circles:



2-2- Rectangles:

The rectangular nesting module employs a Heuristic Recursive (HR) algorithm, inspired by the work of Defu Zhang, Yan Kang, and Ansheng Deng as detailed in their 2005 article. This approach is based on heuristic strategies and a recursive structure, focusing on efficient placement of rectangles within a bounded area.

Contribution: Special thanks to Eli Atamech for his 2019 final year report, which provided valuable insights and foundational concepts for this implementation.

Algorithm overview:

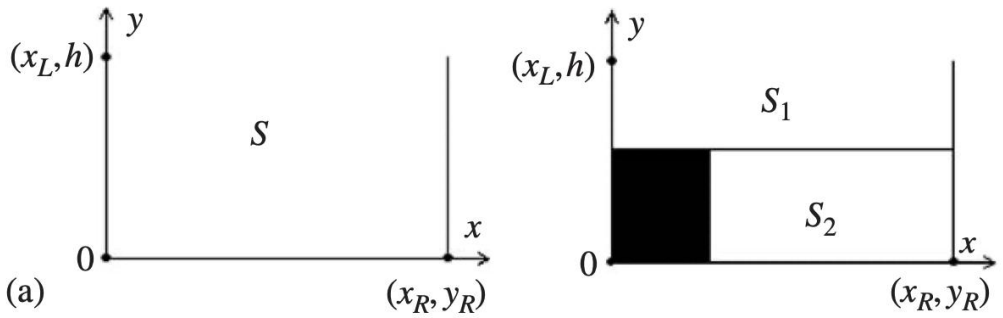
Heuristic algorithms aim to find feasible solutions without guaranteeing the optimal one. They are particularly useful for complex problems where exact solutions are computationally impractical. Even though a heuristic algorithm may occasionally find the best solution, it remains classified as heuristic until this solution can be definitively proven as optimal.

The HR algorithm follows a divide-and-conquer strategy, which involves:

1. **Divide**: Break down the original problem into smaller, non-overlapping subproblems.
2. **Solve**: Address each subproblem recursively.
3. **Combine**: Integrate the solutions of the subproblems to solve the original problem.

It works in the following manner:

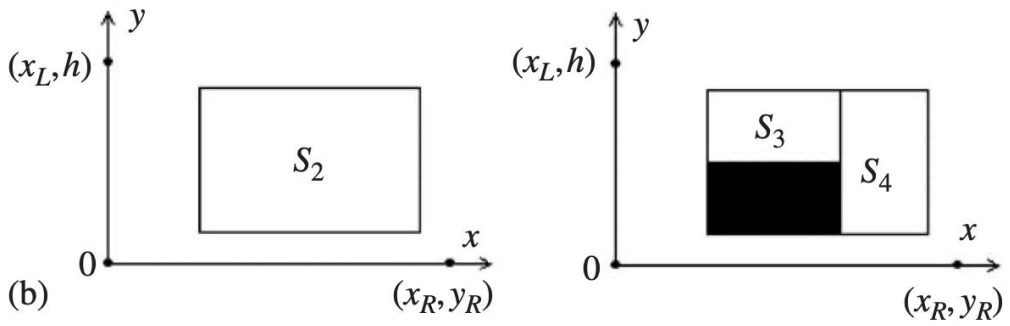
1. **Initial Placement**: Place a rectangle in a space where it fits, then divide this space into smaller subspaces.
2. **Recursive Packing**: Pack each subspace recursively using the same approach.
3. **Combining Solutions**: Combine the solutions of the subspaces to form the solution for the overall problem (e.g., the entire rectangular sheet).



- S is the original space (original problem) what we want to pack.
- S(i) are the subspaces (subproblems) we got from dividing S, and S(i-1)

There are two types of spaces in this algorithm:

- **Bounded Space**: A space with a fixed height, such as S2.
- **Unbounded Space**: A space without a fixed height, extending up to the top of the original space S, like S1.



Algorithm steps:

1. Pack the initial rectangle in a space where it fits, then divide this space into subspaces.
2. Apply the algorithm recursively to each subspace.
3. Combine the solutions of the subspaces to address the larger problem.

Optimization for Hollow Rectangles (Frames):

In addition to handling solid rectangles, the HR algorithm was optimized to accommodate hollow rectangles (frames) with varying thicknesses on each side. This allows the algorithm to nest smaller rectangles within the hollow areas of larger frame rectangles efficiently. The optimization steps include:

- **Sorting by Thickness**: Frame rectangles are sorted by their thickness to prioritize larger frames with more nesting potential.
- **Inner Nesting Check**: Before placing a frame rectangle in the main grid, the algorithm checks if smaller rectangles can be nested inside the frame.
- **Dynamic Subspace Management**: Adjust subspaces dynamically to account for the inner areas of frame rectangles, ensuring that no potential nesting space is wasted.

By incorporating these optimizations, the algorithm can handle complex nesting scenarios involving both solid and hollow rectangles, enhancing material utilization and overall efficiency.

Complexity: The writers of the paper concerning the HR algorithm stated that this algorithm has an average O(n^3).

Module Overview:

The module consists of 4 classes, Rectangle to define a rectangle, Frame to define a frame, Algorithm that encloses the agorithm and a Help class as well a Credit function so that the user can refer to documentation:

```
  1  import matplotlib.pyplot as plt
  2  import matplotlib.patches as patches
  3  from typing import List, Tuple
  4  import matplotlib.axes._axes as Axes
  5  import random
  6
  7  class Rectangle:…
100
101  class Frame:…
157
158  class Algorithm:…
330
331  class Help:…
404
405  def Credit() -> None:…
412
```

1-Rectangle class: Contains 6 methods.

```
7  class Rectangle:
8      def __init__(self, width : float, height : float, thicknessXleft : float = 0, thicknessYtop : float = 0,
       thicknessXright : float = 0, thicknessYbottom : float = 0 , bounded : bool =  False, x : float = -1, y : float = -1)
       -> None:
9          self.width : float = 0
10         self.height : float = 0
11
12         self.x : float = x
13         self.y : float = y
14
15         self.thickness : bool = False
16         self.thicknessXleft : float = 0
17         self.thicknessXright : float = 0
18         self.thicknessYtop : float = 0
19         self.thicknessYbottom : float = 0
20
21         self.rotate : bool = False
22
23         try:
24             if isinstance(width, float) or isinstance(width, int) or isinstance(height, float) or isinstance(height, int):
25                 if height <= 0 or width <= 0:
26                     raise ValueError("Dimensions must be a positive float/int")
27                 self.width : float = width
28                 self.height : float = height
29                 if (height > width):
30                     self.rotate90()
31         except ValueError as e:
32             print(e)
33
34         self.area : float = self.width * self.height
35
36         try:
37             if thicknessXleft != 0 and thicknessXright != 0 and thicknessYtop != 0 and thicknessYbottom != 0:
38                 if thicknessXleft > 0 and thicknessXright > 0 and thicknessYtop > 0 and thicknessYbottom > 0:
39                     if (thicknessXleft + thicknessXright) <= self.width and (thicknessYtop + thicknessYbottom) <= self.
                       height:
40                         self.thicknessXleft = thicknessXleft
41                         self.thicknessXright = thicknessXright
42                         self.thicknessYtop = thicknessYtop
43                         self.thicknessYbottom = thicknessYbottom
44                         self.thickness = True
45                     else:
46                         raise ValueError("Sum of thicknesses exceeds width/height")
47                 else:
48                     raise ValueError("Must input all the positive values of the thickness of every side")
49         except ValueError as e:
50             print(e)
51
52         if(self.thickness):
53             self.rectangles : List[Rectangle] = []
54             self.bounded : bool = bounded
55
56     def rotate90(self) -> None:
57         temp = self.width
58         self.width = self.height
59         self.height = temp
60         self.rotate = not self.rotate
61
62     def set_position(self, x : float, y : float):
63         self.x = x
64         self.y = y
65
66     def compare_to(self, rectangle : 'Rectangle') -> bool: return self.area > rectangle.area
67
68     def get_rectangle_data(self) -> str:
69         string1 = "\nwidth: " + str(self.width) + " , height: " + str(self.height)
70         string2 = "\nx: " + str(self.x) + " , y: " + str(self.y) + " , rotated: " + str(self.rotate) + " , area: " + str
           (self.area)
71         string3 = ""
72         if(self.thickness):
73             string3 = "\nleft thickness: " + str(self.thicknessXleft) + " , right thickness: " + str(self.
               thicknessXright)  + " , top thickness: " + str(self.thicknessYtop) + " , bottom thickness: " + str(self.
               thicknessYbottom)
74             for r in self.rectangles:
75                 string3 += "\n" + str(len(self.rectangles)) + " rectangles are nested inside this rectangle:\n\t" + r.
                   get_rectangle_data().replace("\n", "\n\t")
76
77         return string1 + string2 + string3
78
```

```
78
79      def plot_rectangle(self, ax : Axes) -> None:
80          color = (random.random(), random.random(), random.random())
81          x = self.x
82          y = self.y
83          width = self.width
84          height = self.height
85
86          if self.thickness:
87              # Draw the colored rectangle representing the main body
88              colored_rectangle_patch = patches.Rectangle((x, y), width, height, edgecolor='black', facecolor=color)
89              ax.add_patch(colored_rectangle_patch)
90
91              # Draw the white rectangle on top to simulate thickness
92              white_rectangle_patch = patches.Rectangle((x + self.thicknessXleft, y + self.thicknessYbottom), width - self.
                thicknessXleft - self.thicknessXright, height - self.thicknessYtop - self.thicknessYbottom,
                edgecolor='white', facecolor='white')
93              ax.add_patch(white_rectangle_patch)
94          else:
95              rectangle_patch = patches.Rectangle((x, y), width, height, edgecolor='black', facecolor=color)
96              ax.add_patch(rectangle_patch)
97
98          # Add text annotations
99          ax.text(self.x + self.width / 2, self.y + self.height / 2, f"Width: {self.width}\nHeight: {self.height}
            \nRotation: {self.rotate}",ha='center', va='center', fontsize=5)
```

2-**Frame Class**: Contains 5 methods

```python
class Frame:
    def __init__(self, width : float, height : float, x : float, y : float, bounded : bool) -> None:
        self.width : float = 0
        self.height : float = 0

        self.x : float = x
        self.y : float = y

        self.area : float = self.width * self.height
        self.bounded : bool = bounded
        self.rectangles : List[Rectangle] = []

        try:
            if isinstance(width, float) or isinstance(width, int) or isinstance(height, float) or isinstance(height, int):
                if height < 0 or width < 0:
                    raise ValueError("Dimensions must be a positive float/int")
                self.width : float = width
                self.height : float = height
        except ValueError as e:
            print(e)

    def can_add(self, rectangle : 'Rectangle') -> bool:
        for i in range(2):
            if(self.width >= rectangle.width and self.height >= rectangle.height):
                return True
            rectangle.rotate90()

        return False

    def add(self, s : 'Rectangle') -> None:
        s.x = self.x
        s.y = self.y
        self.bounded = True


    def plot(self):
        fig, ax = plt.subplots()
        ax.set_aspect('equal')
        ax.set_xlim(0, self.width)
        ax.set_ylim(0, self.height)

        for rectangle in self.rectangles:
            rectangle.plot_rectangle(ax)

        ax.set_title('Nested Rectangles in Frame')
        ax.set_xlabel('Width')
        ax.set_ylabel('Height')
        ax.grid(True)

        plt.show()

    def get_frame_data(self) -> str:
        string1 = ""
        for r in self.rectangles:
            string1 += r.get_rectangle_data()

        return string1

```

3-**Algorithm Module**: Contains 11 method

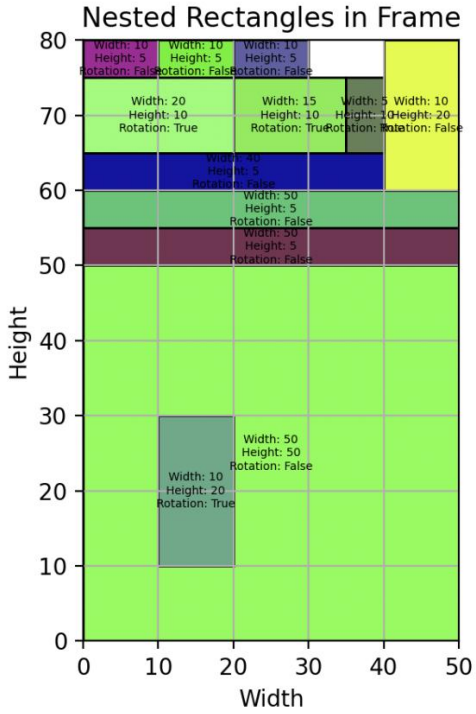
```python
class Algorithm:
    def __init__(self, width : float, height : float, x : float, y: float) -> None:
        self.initialFrame : Frame

        self.topLeftX : float = x
        self.topLeftY : float = y

        self.spaceTaken : float = 0
        self.spaceLeft : float

        self.totalNbRectangles : int = 0
        self.addedNbRectangles : int = 0

        self.overFlow : int = 0

        self.notes : str = ""

        try:
            if isinstance(width, float) or isinstance(width, int) or isinstance(height, float) or isinstance(height, int):
                if height <= 0 or width <= 0:
                    raise ValueError("Dimensions must be a positive float/int")
                self.initialFrame : Frame = Frame(width, height, 0, 0, False)
                self.spaceLeft : float = width * height
                self.notes : str = "no notes"
        except ValueError as e:
            print(e)

        self.shelves : List[Frame] = []

        self.rectangles : List[Rectangle] = []
        self.newRectangles : List[Rectangle] = []

    @staticmethod
    def compare(r1 : Rectangle, r2 : Rectangle) -> bool:
        return r1.area > r2.area

    def set_frame(self, wid : float, hei: float, x1 : float, y1 : float, bounded : bool) -> None:
        self.initialFrame = Frame(wid, hei, x1, y1, bounded)

    def setRectangles(self, s : List[Rectangle]) -> List[Rectangle]:
        self.rectangles : List[Rectangle] = s
        self.rectangles = sorted(self.rectangles, key=lambda rectangle : (-rectangle.area, -max(rectangle.width,
        rectangle.height)))
        self.totalNbRectangles = len(self.rectangles)
        return self.rectangles

    def bounded_packing(self, frame : Frame):
        pickedRectangle : Rectangle = None
        indexPacking : int = 0

        for i, rect in enumerate(self.rectangles):
            currentRect = rect

            if(frame.can_add(currentRect)):
                #print("R CA")
                pickedRectangle = currentRect
                indexPacking = i
                break

        if(pickedRectangle == None):
            return
        else:
            frame.add(pickedRectangle)
            self.rectangles.pop(indexPacking)
            self.newRectangles.append(pickedRectangle)
            self.addedNbRectangles += 1
            self.spaceTaken += pickedRectangle.area
```

```
224
225             if(pickedRectangle.thickness):
226                 wi = (pickedRectangle.width - pickedRectangle.thicknessXleft - pickedRectangle.thicknessXright)
227                 he = pickedRectangle.height - pickedRectangle.thicknessYbottom - pickedRectangle.thicknessYtop
228                 sort = Algorithm(pickedRectangle.width - pickedRectangle.thicknessXleft - pickedRectangle.
                    thicknessXright, pickedRectangle.height - pickedRectangle.thicknessYbottom - pickedRectangle.
                    thicknessYtop, pickedRectangle.thicknessXleft + pickedRectangle.x, pickedRectangle.thicknessYbottom +
                    pickedRectangle.y)
229                 sort.set_frame(wi, he, pickedRectangle.thicknessXleft + pickedRectangle.x, pickedRectangle.
                    thicknessYbottom + pickedRectangle.y, False)
230                 sort.setRectangles(self.rectangles)
231                 sort.run()
232                 self.newRectangles.extend(sort.get_new_rectangle())
233                 self.rectangles = sort.rectangles
234                 pickedRectangle.rectangles.extend(sort.get_new_rectangle())
235
236             s3X = frame.x
237             s3Y = frame.y + pickedRectangle.height
238             s3width = pickedRectangle.width
239             s3height = frame.height - pickedRectangle.height
240
241             s4X = frame.x + pickedRectangle.width
242             s4Y = frame.y
243             s4width = frame.width - pickedRectangle.width
244             s4height = frame.height
245
246             s3 : Frame = Frame(s3width, s3height, s3X, s3Y, True)
247             s4 : Frame = Frame(s4width, s4height, s4X, s4Y, True)
248
248
249             if(s3.area > s4.area):
250                 self.bounded_packing(s3)
251                 self.bounded_packing(s4)
252             else:
253                 self.bounded_packing(s4)
254                 self.bounded_packing(s3)
255
256     def unbounded_packing(self, frame : Frame):
257         if(len(self.rectangles) == 0): return
258
259         pickedRectangle : Rectangle = None
260         indexPacking = 0
261
262         for i, rect in enumerate(self.rectangles):
263             currentRect = rect
264
265             if(currentRect.width < currentRect.height):
266                 currentRect.rotate90()
267             if(frame.can_add(currentRect)):
268                 #print("P CA ", currentRect.width)
269                 pickedRectangle = currentRect
270                 indexPacking = i
271                 break
272
273         if(pickedRectangle == None):
274             return
```

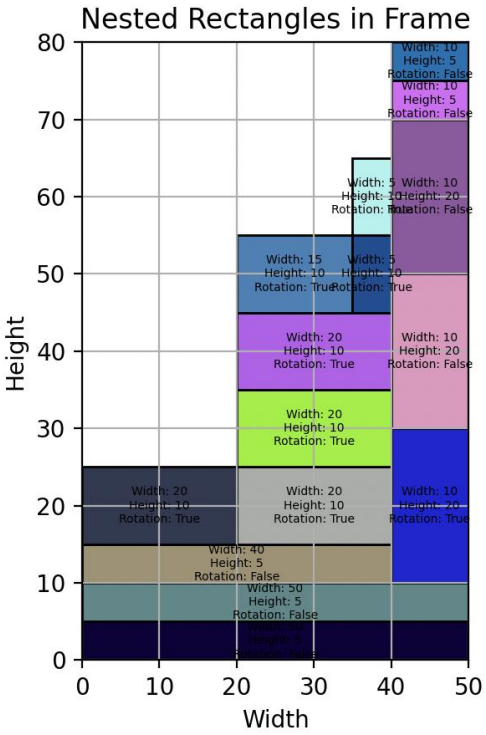
```
275             else:
276                 frame.add(pickedRectangle)
277                 self.rectangles.pop(indexPacking)
278                 self.newRectangles.append(pickedRectangle)
279                 self.addedNbRectangles += 1
280                 self.spaceTaken += pickedRectangle.area
281 
282                 if(pickedRectangle.thickness):
283                     wi = (pickedRectangle.width - pickedRectangle.thicknessXleft - pickedRectangle.thicknessXright)
284                     he = pickedRectangle.height - pickedRectangle.thicknessYbottom - pickedRectangle.thicknessYtop
285                     sort = Algorithm(pickedRectangle.width - pickedRectangle.thicknessXleft - pickedRectangle.
                        thicknessXright, pickedRectangle.height - pickedRectangle.thicknessYbottom - pickedRectangle.
                        thicknessYtop, pickedRectangle.thicknessXleft + pickedRectangle.x, pickedRectangle.thicknessYbottom +
                        pickedRectangle.y)
286                     sort.set_frame(wi, he, pickedRectangle.thicknessXleft + pickedRectangle.x, pickedRectangle.
                        thicknessYbottom + pickedRectangle.y, False)
287                     sort.setRectangles(self.rectangles)
288                     sort.run()
289                     self.newRectangles.extend(sort.get_new_rectangle())
290                     self.rectangles = sort.rectangles
291                     pickedRectangle.rectangles.extend(sort.get_new_rectangle())
292 
293                 s1X = frame.x
294                 s1Y = frame.y + pickedRectangle.height
295                 s1width = pickedRectangle.width
296                 s1height = frame.height - pickedRectangle.height
297 
298                 s2X = frame.x + pickedRectangle.width
299                 s2Y = frame.y
300                 s2width = frame.width - pickedRectangle.width
301                 s2height = frame.height
302 
303                 s1 : Frame = Frame(s1width, s1height, s1X, s1Y, False)
304                 s2 : Frame = Frame(s2width, s2height, s2X, s2Y, True)
305 
306                 s : Frame = s1
307 
308                 self.bounded_packing(s2)
309                 self.unbounded_packing(s)
310 
311     def get_new_rectangle(self) -> List[Rectangle]: return self.newRectangles
312 
313     def get_unnested_rectangle(self) -> List[Rectangle]: return self.rectangles
314 
315     def run(self) -> None:
316         self.unbounded_packing(self.initialFrame)
317         self.overFlow = self.totalNbRectangles - self.addedNbRectangles
318 
319         if(self.overFlow > 0):
320             self.notes = str(self.overFlow) + " rectangles haven't been nested"
321         else:
322             self.notes = "All rectangles got nested"
323 
324         self.initialFrame.rectangles = self.newRectangles
325 
326     def plot(self) -> None:
327         self.initialFrame.plot()
328 
329     def get_data(self) -> str: return self.initialFrame.get_frame_data()
```

Demo:

Nesting 11 normal rectangles and 1 rectangle with various thicknesses



Nesting 15 normal rectangles:



2-3- Irregular Shapes:

The irregular shape nesting algorithm in this project is derived from the grid laying techniques used in the circular nesting module. The approach involves reading polygon shapes from images using OpenCV (cv2), where contours are detected and approximated using a chain approximation method. The images undergo a cleaning process to ensure the polygons are correctly interpreted for nesting purposes. This algorithm aims to efficiently arrange irregular shapes within a predefined boundary by leveraging grid-based techniques, rotations, and mirroring.

Image Input:

1. **Reading the Image**: The process begins by loading the image using OpenCV (cv2). The image is then converted to grayscale to simplify contour detection.
2. **Contour Detection**: Contours are detected using the cv2.findContours method. This method retrieves the outlines of shapes within the image.
3. **Chain Approximation**: The detected contours are approximated to reduce the number of points while retaining the shape's geometry. This is achieved using the cv2.approxPolyDP method.
4. **Closing the Loop**: After approximation, it's essential to ensure the polygon is a closed shape. This involves verifying that the start and end points of the polygon are connected.
5. **Polygon Methods**: Once the polygons are obtained, various methods such as area calculation, point-in-polygon testing, and bounding box computation can be applied to facilitate the nesting algorithm.

Algorithm Overview:

The irregular shape nesting algorithm employs a grid-based greedy packing approach, similar to the circular nesting algorithm, with additional complexity to handle the irregular shapes.

1. **Initial Sorting**: Polygons are sorted based on their area to prioritize the placement of larger shapes first.
2. **Grid Laying Technique**: The algorithm iterates over a grid, attempting to place each polygon at various positions. The positions are checked incrementally based on a user-defined step size for both x and y axes.
3. **Rotations and Mirroring**: Each polygon is rotated in increments (320 times for a full 360-degree rotation) and mirrored at each rotation to explore all possible orientations for optimal placement.
4. **Raycasting for Collision Detection**: Raycasting is used to detect collisions between the polygons and other polygons or the frame. This method involves casting rays from the vertices of the polygon and checking for intersections.

By integrating grid-based techniques, rotations, mirroring, and advanced collision detection, the algorithm efficiently nests irregular shapes within a bounded area, minimizing material waste and optimizing space utilization.

Complexity:

Image Processing: O(p) with p = number of pixels

Grid Search: O(r.m.G.V.C) with r = 360 rotation, m = mirroring of factor of 2, G = potential grid laying points of (1/step)*(frame.width*frame.height), V = number of vertices in a polygon and C is the complexity of the collition detection.

Collision Detection: O((n^2).(V.VP).VP)

Module Overview:

The module consists of 3 classes, Polygon to define a polygon, ComputerVision to that includes the means to detect imags and convert them to polygons, Algorithm that encloses the agorithm:

```
1   import matplotlib.pyplot as plt
2   import matplotlib.path as mpath
3   import numpy as np
4   import cv2
5   from typing import List, Tuple, Optional, Union
6   import pymunk
7   import math
8   import random
9
10  > class Polygon: …
346
347 > class ComputerVision: …
650
651 > class Algorithm: …
906
```

PS: Due the code won't fit, it will be published on my gitHub repository.

1-**Polygon Class**: Contains 23 methods

```python
class Polygon:
    def __init__(self, vertices : List[np.array]):…

    def plot(self) ->  None:…

    def simplify(self, epsilon : float = 1.2) -> List[np.array]:…

    def area(self) -> float:…

    def rectangular_area(self) -> float:…

    def centroid(self) -> Tuple[float, float] :…

    def translate_to_point(self, target_point : Tuple[float, float] = [0, 0]) -> List[np.array]:…

    def resize_ratio(self, ratio : float) -> List[np.array]:…

    def resize(self, new_width : float, new_height : float) -> List[np.array]:…

    def point_in_polygon(self, point : Tuple[float, float]) -> Union[bool, None]:…

    def point_in_polygon_ray_cast(self, point) -> Union[bool, None]:…

    def rotate(self, angle : float) -> Union[List[np.array], None]:…

    def mirror(self):…

    def draw_polygon(self, ax: plt.Axes, frame : bool = False) -> None:…

    @staticmethod
    def calculate_edge_vector(vertex1 : List[np.array], vertex2 : List[np.array]) -> List[int]:…

    @staticmethod
    def normalize_vector(vector : List[np.array]) -> List[int]:…
    @staticmethod
    def calculate_angle(vector1, vector2):…

    @staticmethod
    def identify_tip(self, previous_vector, current_vertex, next_vertex, threshold_angle):…

    def find_tips(self, threshold_angle : float = 120) -> List[np.ndarray]:…

    def get_x_values_for_y(self, y_value : float) -> List[float]:…

    def equal_polygons(self, poly : 'Polygon') -> bool:…

    def exact_polygons(self, poly : 'Polygon') -> bool:…

    def shift_polygon_to_origin(self) -> Union[List[np.array], None]:…
```

2-ComputerVision Class: Contains 11 methods

```
347  class ComputerVision:
348      def __init__(self, image_path : str, threshold : int = 127 ,eps : float = 0.00000000000000001) -> None:⋯
360
361      def get_polygon_list(self, polygons : List[np.array] = None) -> List[Polygon]:⋯
368
369      @staticmethod
370      def read_image(image_path : str) -> Union[np.ndarray, None]:⋯
383
384      @staticmethod
385      def convert_to_binary(image : Union[np.ndarray, None], threshold : int =127) -> Union[np.ndarray, None]:⋯
399
400      @staticmethod
401      def detect_shapes(binary_image : Union[np.ndarray, None]) ->  Tuple[List[np.ndarray], np.ndarray]:⋯
417
418      @staticmethod
419      def approximate_polygons(contours : List[np.ndarray], eps : float = 0.00000000000000001) -> List[np.ndarray]:⋯
435
436      @staticmethod
437      def remove_polygon_by_index(polygons : List[np.ndarray], hierarchy : np.ndarray, index : int) -> Tuple[List[np.
         ndarray], np.ndarray]:⋯
463
464      def clean_polygons(self, polygons : List[np.ndarray], hierarchy : np.ndarray, image : Union[np.ndarray, None]) ->
         Tuple[List[np.ndarray], np.ndarray]:⋯
497
498      def plot_result(self, approximated_polygons : List[np.ndarray] = None, binary_image : Union[np.ndarray, None] = None)
         -> None:⋯
526
527      @staticmethod
528      def merge_polygons(polygon1 : List[np.array], polygon2 : List[np.array]) -> List[np.array]:⋯
533
534      #check why polygons are going missing
535      def merge_polygons_by_hierarchy(self) -> Union[List[np.array], None]:⋯
589
```

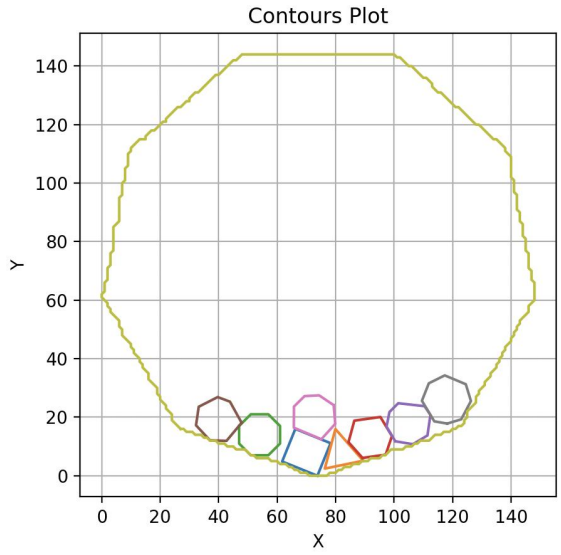
3-Algorithm Class: Contains 8 methods

```
651  class Algorithm:
652      def __init__(self, polygons : List[Polygon], frame : Polygon) -> None:…
659
660      def get_nesting_results(self) -> List[Polygon]:…
664
665      def plot_results(self) -> None:…
682
683      def shift_polygons_to_origin(self) -> List[Polygon]:…
691
692      def sort_polygons_by_area(self) -> List[Polygon]:…
695
696      def polygon_in_frame(self, polygon : Polygon) -> bool:…
709
710      @staticmethod
711      def collision_detection(polygon1 : Polygon, polygon2 : Polygon) -> Union[bool, None]:…
752
753      def nest_polygons(self, rotating : bool = True, mirroring : bool = True, step : float = 1):…
906
```

Demo:

Nesting 8 polygons (~40 vertices each) in 1 greater polygon with 360 rotation and mirroring per polygon on each coordinate.

Elapsed time: 2566.6485130786896 seconds ~= 43 minutes



3- Data sets

To train the machine learning model effectively, two mini data sets were created: one for circles and one for rectangles. Each data set comprises 100,000 test cases with negative tokens to ensure robust training and validation of the model.

3-1- Irregular Shape Data Set:

The creation of a data set for irregular shapes was considered. However, due to the complexity of the nesting operation for irregular polygons, generating such a data set was deemed impractical within the project's timeframe and computational resources.

1. Nesting Complexity: The nesting operation for just 10 polygons takes approximately 25 minutes on an M1 8GB 2020 MacBook Pro.
2. Time Constraints: Given that creating 100,000 test cases would require 100,000 * 25 minutes, the total time needed would be excessively high (~175 days), making it infeasible to generate such a data set within a reasonable period.

3-2- Circle Data Set:

The circle data set was generated using the grid-based greedy packing algorithm detailed earlier. The data set includes a diverse range of circle configurations to cover various scenarios encountered in real-world applications.

1. **Number of Test Cases**: 100 000
2. **Generation Method**: Each test case involves randomly generating a set of 2 to 20 solid circles with varying radii from 2 to 20 units and packing them within a predefined boundary (200, 200) using the grid-based greedy algorithm.
3. **Attributes Captured**: For each test case, attributes such as the number of circles, their radii, and their final positions after nesting are recorded.
4. **Negative Tokens**: For each test case involving less then 20 circles, it was filled with circles of radius = -1 to act as negative tokens. Circles with a radius of -1 are to have the nesting coordinates (-1, -1).

To generate circles:

```python
import random
import csv
from typing import List
from sklearn.preprocessing import MinMaxScaler
from circleNesting import Circle, Frame, Algorithm
import numpy as np

min_radius = 2
max_radius = 20
num_circles_range = (2, 20)
frame_width = 200
frame_height = 200
num_data_points = 100000

with open("/Users/raphael/Desktop/circle_test/nesting_algorithms/circle/data/mini_data/untreated_data.csv", "w",
newline="") as csvfile:
    writer = csv.writer(csvfile)

    writer.writerow(["test_case", "radius", "x", "y", "frame_width", "frame_height"])

    for id in range(num_data_points):
        print(id)
        num_circles = random.randint(*num_circles_range)

        circle_list : List[Circle] = []

        for i in range(num_circles):
            radius = random.randint(min_radius, max_radius)
            circle_list.append(Circle(radius))

        frame = Frame(frame_width, frame_height)
        nest, not_nested = Algorithm.nest_circles(circle_list, frame)

        unnested = 20 - len(nest)

        #writer.writerow([f"Input_{id}", None, None, None, frame_width, frame_height, None])
        for n in range(len(nest)):
            writer.writerow([f"Input_{id}", nest[n].r, None, None, frame_width, frame_height])

        for j in range(unnested):
            writer.writerow([f"Input_{id}", -1, None, None, None, None])

        for n in range(len(nest)):
            x1 = nest[n].x
            y1 = nest[n].y
            x2 = nest[n-1].x
            y2 = nest[n-1].y
            writer.writerow([f"Output_{id}", nest[n].r, x1, y2, frame_width, frame_height])

        for j in range(unnested):
            writer.writerow([f"Output_{id}", -1, -1, -1, None, None])


print("Circle nesting data generated and saved to circle_nesting_data.csv")
```

Remove inputs for outputs contain the same and more data:

```python
#This file removes the Inputs for there Outputs contains the same data
import csv

def filter_csv(input_file, output_file):
  with open(input_file, 'r') as infile, open(output_file, 'w', newline='') as outfile:
    reader = csv.reader(infile)
    writer = csv.writer(outfile)

    # Write the header row (assuming there's a header)
    writer.writerow(next(reader))

    for row in reader:
      if not row[0].startswith("Input_"):
        writer.writerow(row)

# Example usage
input_file = "/Users/raphael/Desktop/circle_test/nesting_algorithms/circle/data/mini_data/updated_treated_data.csv"
output_file = "filtered_output.csv"
filter_csv(input_file, output_file)

print(f"CSV filtered, results written to: {output_file}")

```

Normalise radii:

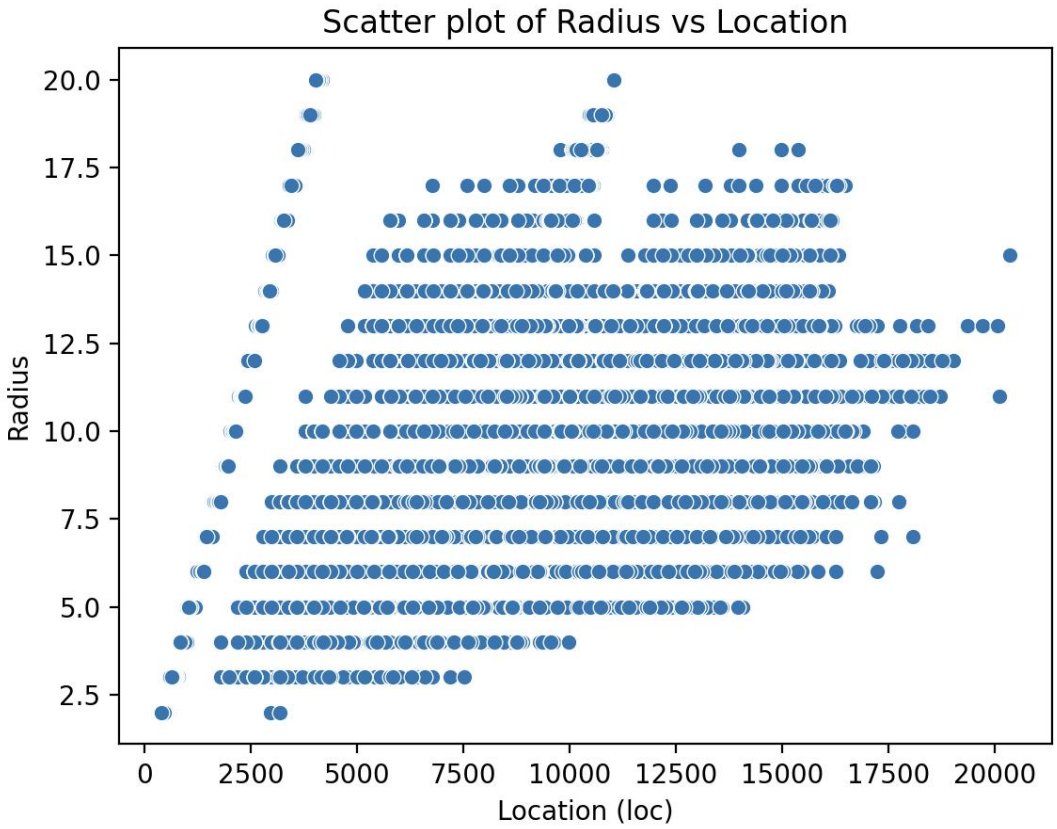
```python
import csv
import pandas as pd

def normalize_radii(input_file, output_file):
    data = pd.read_csv(input_file)
    data['r_norm'] = data['radius'].apply(lambda r: r / 20 if r > 0 else r)
    data.to_csv(output_file, index=False)

input_file = "/Users/raphael/Desktop/circle_test/filtered_output.csv"
output_file = "/Users/raphael/Desktop/circle_test/filtered_output.csv"
normalize_radii(input_file, output_file)

print(f"CSV normalized and updated, results written to: {output_file}")
```

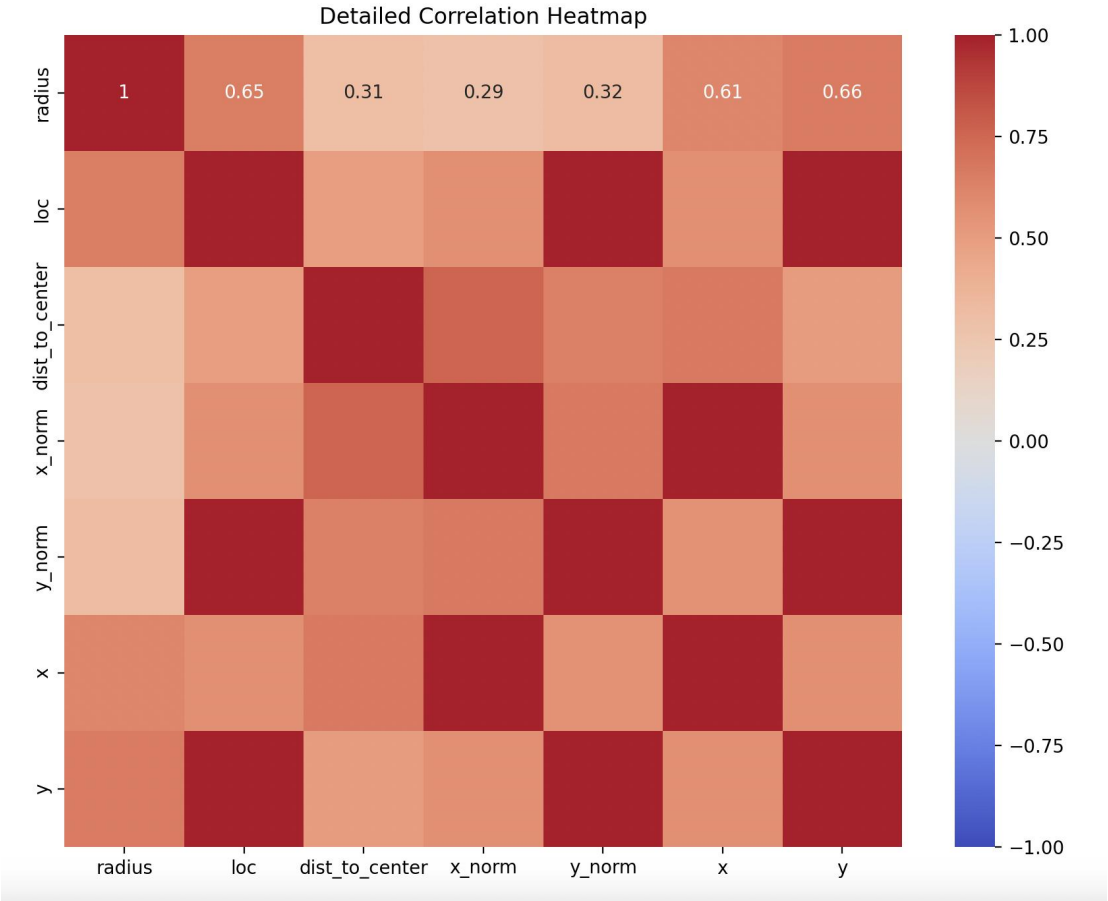
Feature Engineer:

```python
#this file adds the following columns
#x_norm         : normalized x with x_norm = x / 200
#y_norm         : normalized y with y_norm = y / 200
#loc            : loc = y * 200 + x
#dist_to_center : distance from center of nested circle to (0, 0)

import pandas as pd
import numpy as np;

data = pd.read_csv('/Users/raphael/Desktop/circle_test/nesting_algorithms/circle/data/mini_data/untreated_circle_data.csv')
frame_width = 200
frame_height = 200

# Calculate the 'loc' feature
def calculate_loc(row):
    if pd.notna(row['x']) and pd.notna(row['y']) and row['x'] != -1 and row['y'] != -1:
        return row['y'] * frame_width + row['x']
    else:
        return -1

# Apply the function to calculate 'loc' for Output rows
data['loc'] = data.apply(lambda row: calculate_loc(row) if 'Output' in row['test_case'] else pd.NA, axis=1)

filtered_data = data[(data['radius'] > 0)].copy()

filtered_data['dist_to_center'] = np.sqrt((filtered_data['x'] - frame_width / 2)**2 + (filtered_data['y'] - frame_height / 2)**2)

# Normalize x and y coordinates
filtered_data['x_norm'] = filtered_data['x'] / frame_width
filtered_data['y_norm'] = filtered_data['y'] / frame_height

data = data.merge(filtered_data[['test_case', 'radius', 'x', 'y', 'dist_to_center', 'x_norm', 'y_norm']], on=
```

```
    ['test_case', 'radius', 'x', 'y'], how='left')
33
34  data['dist_to_center'].fillna(0, inplace=True)
35  data['x_norm'].fillna(0, inplace=True)
36  data['y_norm'].fillna(0, inplace=True)
37
38  data.to_csv('/Users/raphael/Desktop/circle_test/nesting_algorithms/circle/data/mini_data/treated_data.csv', index=False)
```

Tesing the circles' data set:





Correlation Analysis: The correlation between the radius of the circles and their coordinates (x, y) were calculated to be 0.61 and 0.67 respectively. These values indicate an acceptable positive correlation, suggesting that the data is suitable for machine learning applications, particularly for models that use reinforcement learning with an agent and a reward/punishment system.

3-3- Rectangle Data Set:

Similarly, the rectangle data set was created using the Heuristic Recursive (HR) algorithm. This data set also includes 100,000 test cases, providing a comprehensive range of rectangle configurations.

1. **Number of Test Cases**: 100 000
2. **Generation Method**: Each test case involves randomly generating a set of 2 to 20 solid rectangles with varying width and height from 2 to 20 units and packing them within a predefined boundary (100, 100) using the HR algorithm.
3. **Attributes Captured**: For each test case, attributes such as the number of rectangles, their dimensions, and their final positions after nesting are recorded.
4. **Negative Tokens**: For each test case involving less then 20 rectangles, it was filled with rectangles of (width, height) = (-1, -1) to act as negative tokens. Rectangles with a dimentions of (-1, -1) are to have the nesting coordinates (-1, -1).

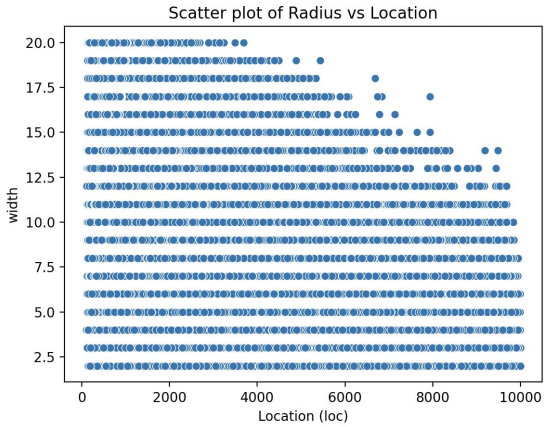
Generating the data set:

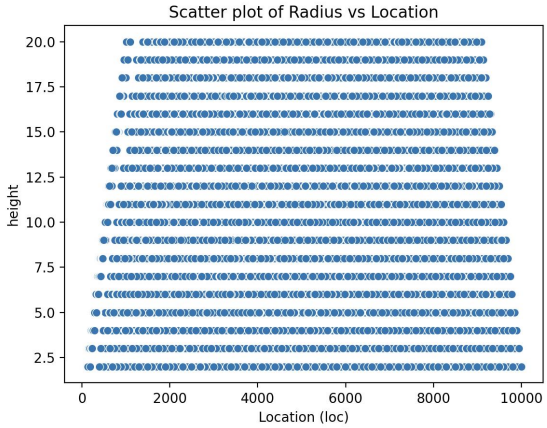
```python
import random
import csv
from typing import List
from sklearn.preprocessing import MinMaxScaler
from rectangleNesting import Rectangle, Frame, Algorithm
import numpy as np

min_dimension = 2
max_dimension = 20
num_rectangles_range = (2, 20)
frame_width = 100
frame_height = 100
num_data_points = 100000

with open("/Users/raphael/Desktop/circle_test/nesting_algorithms/rectangle/data/mini_data/rectangle_nesting_data.csv",
"w", newline="") as csvfile:
    writer = csv.writer(csvfile)

    writer.writerow(["test_case", "w", "h", "x", "y", "frame_width", "frame_height", "rotation"])

    for id in range(num_data_points):
        print(id)
        num_rectangles = random.randint(*num_rectangles_range)

        rectangles_list : List[Rectangle] = []

        for i in range(num_rectangles):
            width = random.randint(min_dimension, max_dimension)
            height = random.randint(min_dimension, max_dimension)
            rectangles_list.append(Rectangle(width, height))

        nest = Algorithm(frame_width, frame_height, 0, 0)
        nest.setRectangles(rectangles_list)
        nest.run()
        nested = nest.get_new_rectangle()

        unnested = 20 - len(nested)

        #writer.writerow([f"Input_{id}", None, None, None, frame_width, frame_height, None])
        for n in range(len(nested)):
            x1 = nested[n].x
            y1 = nested[n].y
            w = nested[n].width
            h = nested[n].height
            writer.writerow([f"Output_{id}", nested[n].width, nested[n].height, x1, y1, frame_width, frame_height, nested
            [n].rotate * 1])

        for j in range(unnested):
            writer.writerow([f"Output_{id}", -1, -1, -1, -1, None, None, -1])


print("Rectangle nesting data generated and saved to rectangle_nesting_data.csv")
```

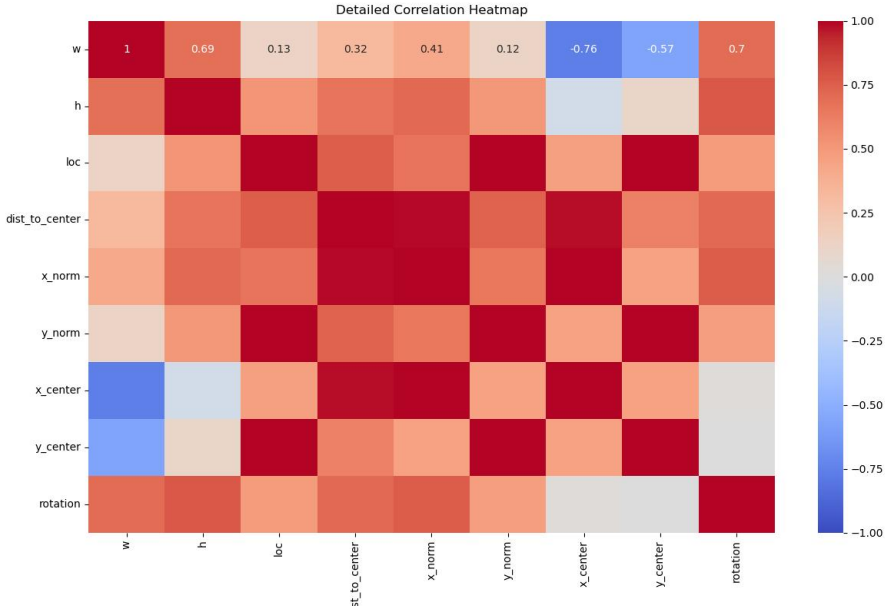
Treat and normalise data:

```python
import pandas as pd
import numpy as np;

data = pd.read_csv('nesting_algorithms/rectangle/data/mini_data/rectangle_nesting_data.csv')
frame_width = 100
frame_height = 100

# Calculate the 'loc' feature
def calculate_loc(row):
    if pd.notna(row['x']) and pd.notna(row['y']) and row['x'] != -1 and row['y'] != -1:
        return row['y'] * frame_width + row['x']
    else:
        return -1

# Apply the function to calculate 'loc' for Output rows
data['loc'] = data.apply(lambda row: calculate_loc(row) if 'Output' in row['test_case'] else pd.NA, axis=1)

filtered_data = data[(data['w'] > 0)].copy()

filtered_data['dist_to_center'] = np.sqrt((filtered_data['x'])**2 + (filtered_data['y'])**2)

# Normalize x and y coordinates
filtered_data['x_norm'] = filtered_data['x'] / frame_width
filtered_data['y_norm'] = filtered_data['y'] / frame_height
filtered_data['w_norm'] = filtered_data['w'] / 20
filtered_data['h_norm'] = filtered_data['h'] / 20

data = data.merge(filtered_data[['test_case', 'w', 'h', 'x', 'y', 'dist_to_center', 'x_norm', 'y_norm', 'w_norm',
'h_norm']], on=['test_case', 'w', 'h', 'x', 'y'], how='left')

data['dist_to_center'].fillna(0, inplace=True)
data['x_norm'].fillna(0, inplace=True)
data['y_norm'].fillna(0, inplace=True)
data['w_norm'].fillna(0, inplace=True)
data['h_norm'].fillna(0, inplace=True)

data.to_csv('/Users/raphael/Desktop/circle_test/nesting_algorithms/rectangle/data/mini_data/treated_data.csv',
index=False)
```

test data:







Correlation Analysis: The correlation between the width of the rectangles and their coordinates (x, y) were calculated to be -0.76 and -0.57 respectively. These values indicate an acceptable positive correlation, suggesting that the data is suitable for machine learning applications, particularly for models that use reinforcement learning with an agent and a reward/punishment system.
But the the correlation between the height of the rectangles and their coordinates (x, y) is very low, and not much can be done about it, this means the learning process will demand alot of data and will relied low accuracy results.

4- ***Machine Learning***

In this project, machine learning is employed to enhance the efficiency of the nesting algorithms through predictive modeling. Although more complex models like neural networks or reinforcement learning agents could be applied, a linear regression model was chosen based on the guidance of the project supervisor. Linear regression provides a straightforward and interpretable approach to understanding the relationships between input features and the nesting outcomes.

4-1- Circles

The goal of the linear regression model for circle nesting is to predict the coordinates (x, y) where each circle should be placed, given its radius. This predictive model helps enhance the nesting efficiency by minimizing overlap and optimizing space usage. The model is trained on a dataset of 100,000 test cases, with an 80-20 split for training and testing. Shuffling is employed to ensure the training data is well-mixed, which helps improve the generalization of the model.

Model:

```
42  def build_model(input_dim):
43      model = Sequential()
44      model.add(Dense(128, input_dim=input_dim, activation='relu'))
45      model.add(Dropout(0.2))
46      model.add(Dense(128, activation='relu'))
47      model.add(Dense(40))
48
49      model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
50      return model
```

- The build_model function constructs a neural network with an input dimension matching the number of features (radii).
- The model consists of Dense layers with ReLU activation and a Dropout layer to prevent overfitting.
- The output layer has 40 neurons to predict 20 (x, y) pairs.

Training:

```
53  def train_model(model, X_train, y_train, epochs=30, batch_size=10):
54      history = model.fit(X_train, y_train, epochs=epochs, validation_split=0.2, batch_size=batch_size, verbose=1)
55      return history
56
```

- The train_model function trains the model in this case with 100 epochs and a batch size of 10, using 80% of the data for training and 20% for validation.

Evaluate Model:

```
57  def evaluate_model(model, X_test, y_test):
58      loss, mae = model.evaluate(X_test, y_test, verbose=1)
59      print(f"Test Loss: {loss}, Test MAE: {mae}")
60      return loss, mae
```

- The evaluate_model function evaluates the model's performance on the test set, printing the loss and mean absolute error (MAE).

PS: Due to the time frame, multiple models have been created after this one, The code will be published on my gitHub repository.

4-2- Rectangles

The linear regression model for rectangular nesting is designed to predict the coordinates (x, y) and rotation for each rectangle. This model aids in arranging the rectangles within a defined boundary to maximize space efficiency and minimize material waste. Numerous models were created during the project, and the following explanation focuses on Model 6, which showed significant promise.

Model:

```
def create_model(input_shape):
    input_layer = Input(shape=input_shape)
    # Flatten the input to ensure compatibility with Dense layers
    flattened_input = Reshape((-1,))(input_layer)
    x = Dense(512, activation='relu')(flattened_input)
    x = BatchNormalization()(x)
    x = Dense(512, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(256, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(128, activation='relu')(x)
    x = BatchNormalization()(x)
    x = Dense(64, activation='relu')(x)
    output_layer = Dense(10 * 3, activation='linear')(x)
    output_layer = Reshape((10, 3))(output_layer)
    model = Model(inputs=input_layer, outputs=output_layer)
    return model
```

- Input Layer: Accepts the input with a shape defined by the number of features.
- Dense Layers: A series of dense layers with ReLU activation functions. Batch normalization is applied after each dense layer to stabilize and accelerate the training process.
- Output Layer: The final dense layer outputs predictions for 10 rectangles, each with 3 values (x, y, and rotation). The Reshape layer formats this output to a shape suitable for the problem.

Training & Evaluation:

- Compilation: The model is compiled using the Adam optimizer, with mean squared error as the loss function and mean absolute error as a metric.
- Training: The model is trained for 100 epochs with a batch size of 32. Validation split of 20% ensures that the model's performance is monitored on unseen data during training.
- Evaluation: After training, the model is evaluated on the test set to determine its loss and mean absolute error.

5- Conclusion

This project aimed to tackle the complex problem of nesting various shapes—circles, rectangles, and irregular polygons—within bounded areas to minimize material waste, using both traditional algorithmic approaches and modern machine learning techniques. Through the development and implementation of custom Python modules for each shape type, significant strides were made in understanding and enhancing nesting efficiency.

Summary of Achievments:

1-Algorithm Development:

- Circle Nesting: The Grid Laying Algorithm, a greedy approach, was successfully implemented to nest circles efficiently. This algorithm demonstrated the ability to handle both solid and hollow circles with varying radii.
- Rectangular Nesting: The Heuristic Recursive (HR) algorithm, inspired by existing literature and further optimized, provided an effective solution for nesting rectangles. Special adaptations allowed the nesting of hollow rectangles with variable thicknesses on each side.
- Irregular Shape Nesting: Leveraging techniques from circular nesting, the grid laying method was extended to handle irregular polygons. Image processing with OpenCV enabled accurate detection and contour approximation, critical for nesting complex shapes.

2-Data Set Creation:

- Two mini datasets, each containing 100,000 test cases for circles and rectangles, were generated. These datasets served as the foundation for training and evaluating the machine learning models.
- Despite initial intentions, creating a dataset for irregular shapes was deemed impractical due to the excessive computation time required for nesting operations.

3-Machine Learning Implementation:

- A linear regression model was trained on the circle dataset, showcasing acceptable positive correlations between radius and coordinates (0.61 for x and 0.67 for y). The model's performance indicates its potential for further optimization and application in real-world scenarios.
- For rectangles, multiple models were explored, with Model 6 demonstrating promising results. This model, with its deep neural network architecture, effectively predicted the placement and orientation of rectangles, highlighting the power of machine learning in solving complex nesting problems.

Challenges and Limitations: Throughout the project, several challenges were encountered:

Computational Complexity: The inherent complexity of nesting algorithms, especially for irregular shapes, posed significant computational demands. Optimizing these algorithms for efficiency was crucial but remains an area for further improvement.

Data Set Limitations: Generating a dataset for irregular shapes was impractical within the given time frame and computational resources, highlighting the need for more efficient algorithms or advanced computational infrastructure.

Development Time: The entire project, including algorithm development, dataset creation, and machine learning implementation, was completed within a month and a half. This tight timeline constrained the depth of exploration and optimization possible within each component.

Hardware and Software Constraints: The architecture of the development machine (a MacBook Pro with an M1 chip and 8GB RAM) and its operating system made running premade libraries for shape nesting near impossible, for they are designed for x64-86 chips rather than ARM chips.

Future Work: This project lays the groundwork for several future research and development avenues:

Algorithm Optimization: Further optimization of existing algorithms and exploration of novel approaches could significantly enhance nesting efficiency and reduce computation time.

Advanced Machine Learning Models: Experimenting with more sophisticated machine learning models, such as reinforcement learning with a reward/punishment system, could lead to better performance and adaptability in various nesting scenarios.

Real-World Applications: Extending the project's scope to real-world industrial applications, such as textile manufacturing or metal cutting, would validate the practical utility and robustness of the developed solutions.

Acknowledgements:

Special thanks to Eli Atamech for his 2019 final year report, which provided invaluable insights and foundational concepts, particularly for the rectangular nesting algorithm.

In conclusion, this project successfully demonstrated the potential of combining traditional algorithms with machine learning techniques to address the nesting problem. While significant progress was made, the journey toward optimal and efficient nesting solutions continues, promising exciting developments and applications in the future.

Girhub Repository: https://github.com/Raphael-m03/Cutting-Board.git