Name: Rudraksh Mehta
E-mail: rudymehtaaa@gmail.com

Coding assignment problem:

Assume a bin which has a width of 80 and height 40 in which 9 rectangles of random width and height have to be placed without overlap and within the bin optimally. Here are the constraints:

1. Rectangle no 1 and 2 are to be placed on the top and the bottom side of the bin.
2. Rectangle no 3 is to be placed close to 4, 5, 9
3. Rectangle no 7 is to be placed close to 6 and 2.
4. Remaining rectangle can be placed anywhere such that the area is optimal.
5. All rectangles have to have one unit separation between them.

Rules:

1. Python code needs to be created along with the plot and the weights to show the results.
2. Use Graphs neural networks to define the constraints. Use the algorithm to steer the Reinforcement learning or other meta heuristic algorithm for area optimisation.
3. Do share your end results.

FIRST ITERATION

In the first iteration, the goal was to develop a straightforward and functional placement algorithm capable of solving the problem of fitting rectangles into a bin while adhering to the basic placement constraints. At this stage, the algorithm did not incorporate advanced techniques like Reinforcement Learning or Graph Neural Networks (GCNN). Instead, we employed a simple, rule-based approach, complemented by randomized search to achieve a valid placement of rectangles within the bin. The primary aim was to ensure basic constraint satisfaction without focusing on optimality, providing a foundation for further improvements in future iterations.

Features of the First Iteration:

1. Greedy, Rule-Based Placement:
    - The algorithm began by following a set of basic, hardcoded rules to handle critical constraints early on.
    - Fixed Placement for Rectangles 1 and 2: These two rectangles were placed at specific positions, with Rectangle 1 positioned at the top of the bin and Rectangle 2 at the bottom. This helped satisfy the key constraint regarding their fixed locations.
    - Proximity-Based Placement for Rectangles 3, 4, 5, and 9: These rectangles were placed based on proximity to other rectangles, following the provided constraints to ensure the layout conformed to initial placement rules.
    - The algorithm was primarily designed to prioritize the correct placement of these critical rectangles before focusing on others.
2. Randomized Search for Remaining Rectangles:
    - For the remaining rectangles, the algorithm utilized a randomized search technique. Each rectangle was placed at a random position within the bin, but overlap was avoided by checking potential placements before finalizing each rectangle's location.
    - Although this approach ensured that no rectangles overlapped, it did not guarantee that the final placement would be optimal in terms of maximizing bin space utilization. The randomness introduced an element of unpredictability, and the arrangement could result in unused gaps or inefficient packing.
3. Constraint Satisfaction:
    - The focus of this iteration was on ensuring that the defined constraints were met. The algorithm handled the placement of the rectangles with explicit checks for the following conditions:

        - Rectangles 1 and 2: Positioned at the top and bottom of the bin, respectively, fulfilling the fixed placement requirement.
        - Rectangle 3: Positioned close to Rectangles 4, 5, and 9, ensuring proximity-based constraints were respected.
        - Rectangle 7: Placed near Rectangles 6 and 2, adhering to its proximity constraints.
    - These constraints were enforced through manual checks, but the algorithm's design did not incorporate any advanced techniques to learn or optimize placement, such as machine learning models or optimization algorithms.

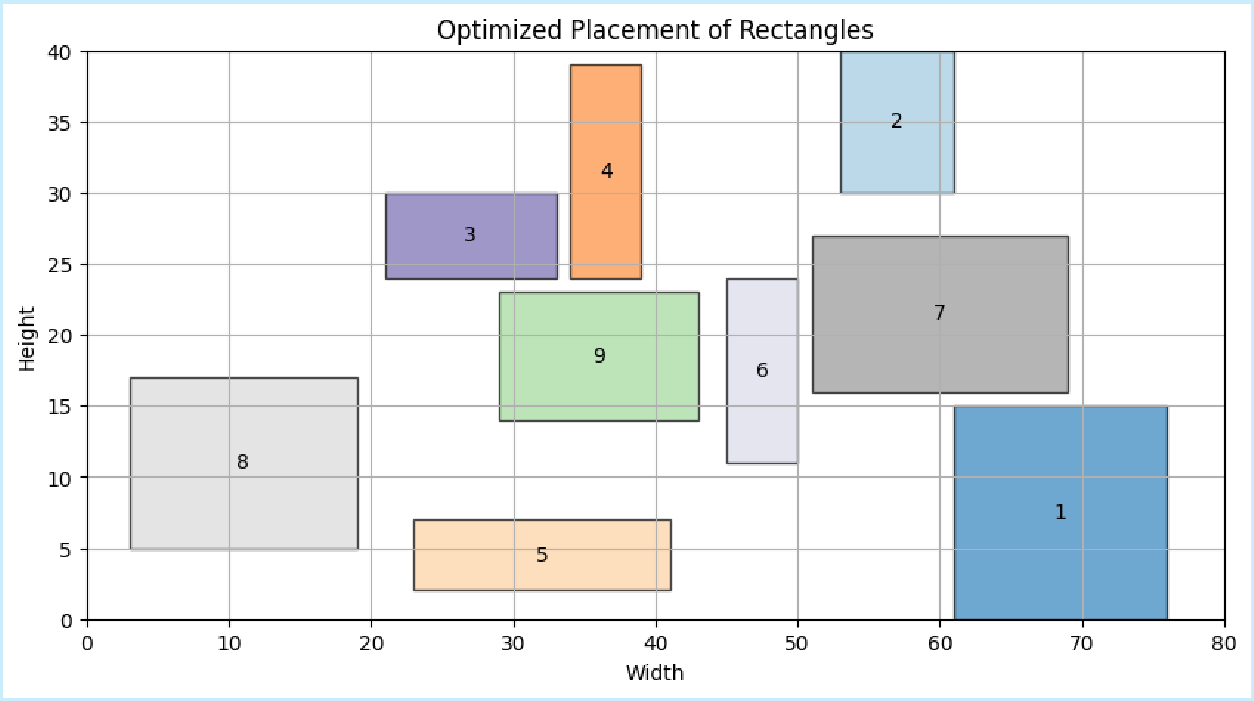
4. <u>No Advanced Optimization:</u>
    - The placement process was not optimized for minimizing unused space or maximizing bin packing efficiency. While overlap checks ensured that no two rectangles intersected, the arrangement of rectangles in the bin was not designed to optimize the use of space.
    - As a result, the bin layout might have significant unused areas, and the rectangle arrangement could be far from the most space-efficient. This was an intentional limitation for this first iteration, focusing more on basic functionality rather than optimization.

<u>What We Wanted to Achieve in the First Iteration:</u>

The purpose of the first iteration was to establish a working prototype that adhered to the basic constraints. The goals were:

- Create a functional placement system that correctly followed the basic placement constraints.
- Generate a valid bin layout where rectangles were placed without overlap, fulfilling the given positional and proximity conditions.
- Visualize the initial arrangement of rectangles within the bin, providing a tangible representation of the placement for future comparison.

This iteration was primarily a proof of concept that laid the groundwork for future developments. We did not aim for an optimized solution at this stage; instead, the focus was on ensuring the basic functionality worked as expected.



Relation to Later Iterations:

The first iteration acted as the foundation for more sophisticated algorithms in later stages. While basic placement was achieved, the subsequent iterations introduced enhancements aimed at optimizing the bin packing process and refining constraint satisfaction.

1. Optimization of Space Utilization:
   - As future iterations progressed, the primary focus shifted from merely ensuring constraints were met to optimizing the layout for minimal empty space and efficient bin usage. This required careful consideration of the placement of rectangles to ensure that the final arrangement was both valid and optimal in terms of space efficiency.
   - Techniques like dynamic adjustment and feedback loops were introduced, allowing the algorithm to refine its placements based on previous attempts and outcomes.
2. Constraint Handling with Greater Precision:
   - In later iterations, constraint handling was enhanced. Constraints such as proximity and fixed placement were not just hardcoded but were instead dynamically adjusted based on the evolving state of the bin layout. This resulted in more flexible and precise constraint satisfaction.

SECOND ITERATION

In the second iteration, the focus shifts to a more sophisticated approach by introducing Graph Convolutional Networks (GCNN) combined with Reinforcement Learning (RL) to optimize the rectangle placement process. This iteration marks a significant improvement over the first, moving away from random placements and leveraging a learning-based model to enhance the quality of the layout.

Use of Graph Convolutional Networks (GCNN):

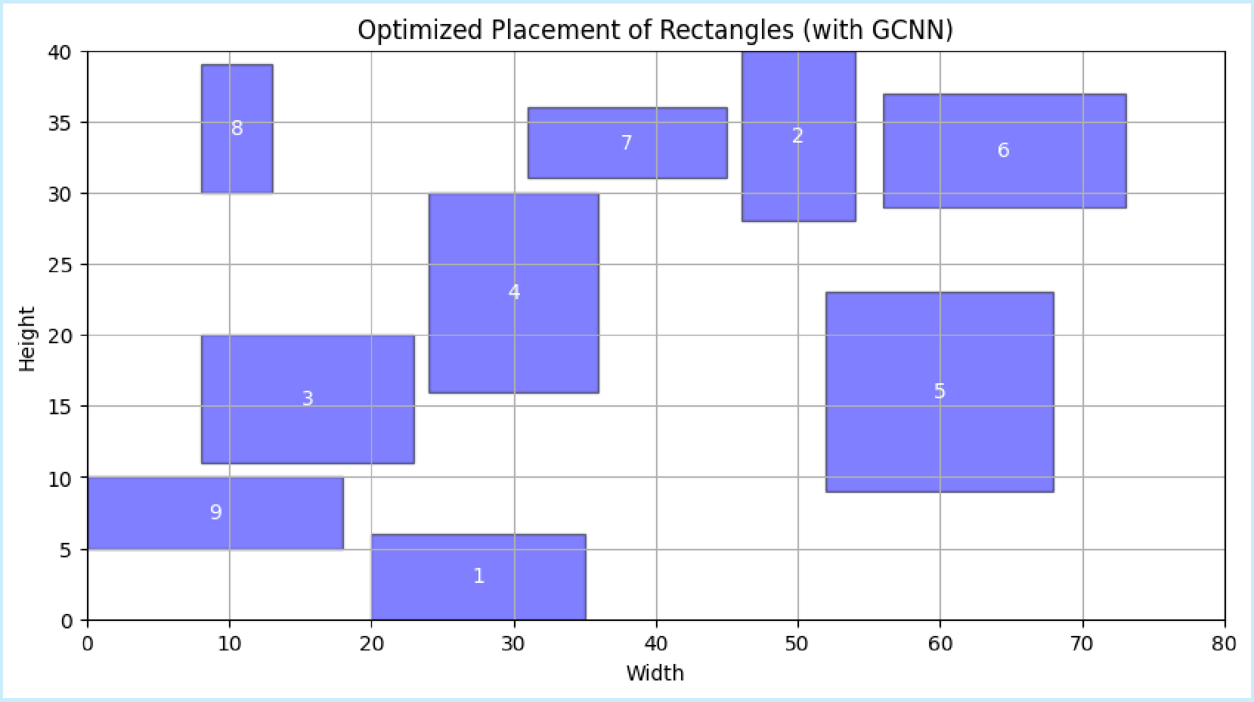
- Rather than relying on purely random placements, GCNN is introduced to process the placement problem in the context of graph-structured data.
- The GCNN treats the placement problem as a graph where:
  - Each node represents a rectangle.
  - Edges represent relationships between rectangles (such as proximity or other placement constraints).
- The GCNN uses two layers of graph convolutions, allowing it to learn complex relationships between rectangles. This enables the model to intelligently optimize the placement based on the graph's structure and the constraints between rectangles.

Reinforcement Learning for Optimization:

- The rectangle placement process is now driven by Reinforcement Learning (RL), enabling the model to improve over time.
- The algorithm learns from its placements by receiving rewards for favorable outcomes and penalties for undesirable ones. Specifically, the reward function evaluates:
  - Proximity adherence: Rectangles that should be placed close to one another (based on constraints) receive higher rewards for being placed near each other.
  - Space efficiency: Placements that utilize the bin space effectively, maximizing the occupied area, earn higher rewards.
  - Penalty for overlaps: Negative rewards are applied if there are any overlaps between rectangles, discouraging invalid placements.
- This reward-based system allows the model to iteratively improve its placement strategy.

Algorithm Breakdown:

1. Graph Construction:
    - The problem's constraints (such as proximity between rectangles) are converted into a graph structure using a library like networkx. Each node represents a rectangle, and edges capture the relationships between them (e.g., proximity constraints).
2. GCNN Layers:
    - The GCNN processes the graph using multiple convolutional layers, learning patterns in how rectangles should be placed based on the structure of the graph and their relationships. As the model trains, it updates its weights to refine its placement strategy.
3. Reinforcement Learning:
    - The agent (GCNN) interacts with the environment (the bin and rectangles) and is rewarded or penalized based on the quality of its placements.
        - Positive rewards: Given for placements that satisfy proximity constraints and use space efficiently.
        - Negative rewards: Applied for invalid placements, such as overlaps or inefficient use of space.
4. Reward Function:
    - The reward function is designed to evaluate placements based on:
        - Constraint satisfaction: Higher rewards for meeting proximity and other placement constraints.
        - Efficiency: Higher rewards for minimizing empty space in the bin.
        - Penalty for overlap: A negative reward for any overlap between rectangles.
5. Training:
    - The GCNN is trained using the Adam optimizer to improve its placement strategy by minimizing the penalties and maximizing the rewards.
    - This iterative training process leads to improved placements over time.


Optimized Placement of Rectangles (with GCNN)
Height
Width
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
60
70
80
1
2
3
4
5
6
7
8
9

<u>**THIRD ITERATION**</u>

In the third iteration, the focus was on refining the rectangle placement algorithm by leveraging Graph Convolutional Networks (GCN). The goal was to optimize placements while ensuring that all defined constraints were satisfied, creating a more intelligent approach to solving the bin packing problem.

The Algorithm:

1. Problem Representation:
   - Rectangles are represented as nodes in a graph, with edges signifying the "close_to" relationships between rectangles as dictated by the problem's constraints. This graph-based approach allows the algorithm to effectively handle spatial relationships between rectangles and adhere to specific placement rules (such as keeping certain rectangles near one another).
2. Graph Data Preparation:
   - Adjacency Matrix: This matrix captures the relationships between rectangles, where an edge between two nodes signifies that the corresponding rectangles should be placed close to each other, based on the given constraints.
   - Feature Matrix: The feature matrix holds the attributes of each rectangle, such as its x and y positions and dimensions (width and height). This information is crucial for the GCN to process and optimize the placement effectively.
3. GCN Architecture:
   - The GCN architecture consists of two convolutional layers followed by fully connected layers. The first convolutional layer aggregates features from neighboring nodes (rectangles), and the second layer refines these features further, enabling the network to capture more complex relationships between the rectangles.
   - After the convolutional layers, the model outputs the new x, y positions for each rectangle, which are used to adjust their placements within the bin.
4. Optimization Loop:
   - The GCN is trained over multiple epochs, where the rectangle placements are optimized at each step. The new positions of the rectangles are computed and validated for overlap. If no overlap occurs, the new positions are accepted; otherwise, the model adjusts the placements to resolve conflicts.

The Loss Function:

The loss function is designed to optimize two key objectives: avoiding overlap and satisfying constraints. It provides feedback to the model by penalizing suboptimal placements and rewarding more efficient ones.

1. Overlap Avoidance:
    - A critical aspect of the loss function is preventing overlaps between rectangles. During the placement check, any position that causes two rectangles to overlap is rejected. This prevents invalid placements and ensures that all rectangles fit within the bin without conflicting with each other.
2. Constraint Satisfaction:
    - A significant portion of the reward (or the minimization of loss) is derived from ensuring that the "close_to" constraints are satisfied. For example, if two rectangles need to be placed near each other, the model receives a positive reward for positioning them closer together. This encourages the model to prioritize the satisfaction of placement constraints.
3. Loss Function Design:
    - The loss function employs a mean squared error (MSE) between the predicted new positions and the current positions of the rectangles. This penalizes deviations from the current placements, guiding the model to incrementally improve the arrangement while maintaining the integrity of existing placements. The MSE helps refine placements gradually, ensuring smooth adjustments.
4. Training Process:
    - The model uses the Adam optimizer, which is highly efficient for problems with many parameters and large search spaces. The optimizer adjusts the model's weights to minimize the loss function, leading to better placements with each training epoch.
5. Penalty for Suboptimal Placements:
    - The model also integrates penalties for suboptimal placements. Specifically, if rectangles are placed too far from their desired positions or violate constraints, they are penalized. This ensures that the algorithm focuses not only on fitting rectangles into the available space but also on respecting the specific placement rules.

Loss Function Intuition:

The loss function is structured around two key objectives:

1. Spatial Optimization:
   - The model strives to minimize the distance between rectangles that should be placed close to each other, while simultaneously avoiding overlaps. This helps optimize the use of space within the bin.
2. Constraint Satisfaction:
   - The model is guided to respect the "close-to" relationships and other constraints, ensuring that the final placement configuration complies with the problem's requirements.

By focusing on these two core objectives, the model is able to refine the placements of rectangles progressively. The iterative training loop, supported by a well-designed loss function, leads to continuous improvement in the rectangle arrangement.

<u>Visualizing the Optimization:</u>

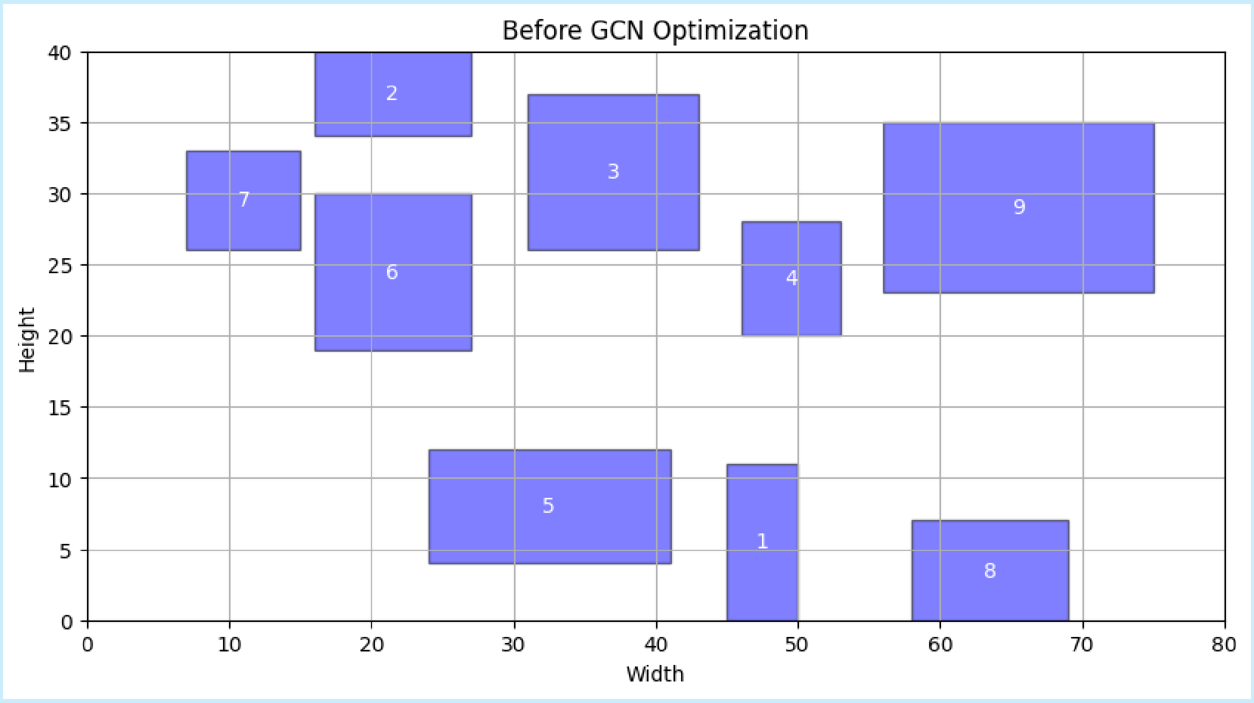
At the end of the training process, the model's performance is visualized by showing the "Before" and "After" of the optimization. The visual representation highlights the effectiveness of the GCN approach in improving the placement over time. The "Before" visualization illustrates the initial placements, while the "After" shows the optimized arrangement, where the constraints are satisfied, overlaps are minimized, and the space is more efficiently utilized.

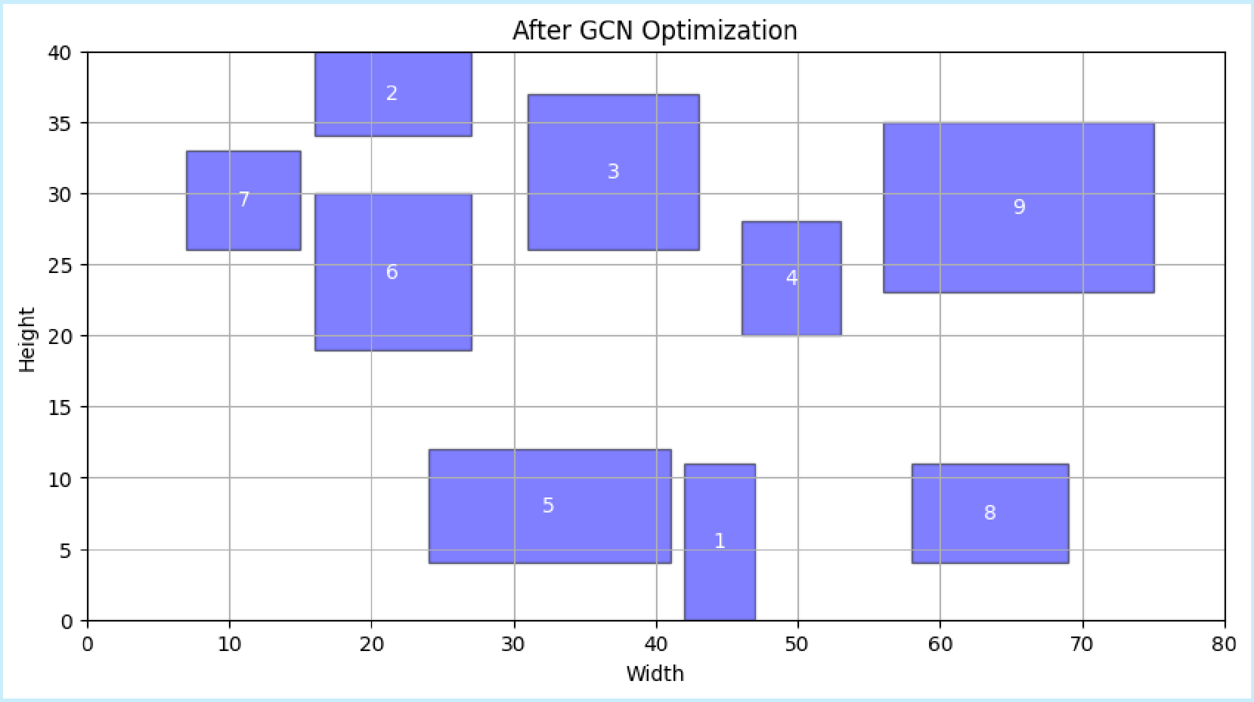
This visualization serves as a powerful tool for assessing the success of the optimization algorithm, clearly demonstrating how the GCN refined the arrangement of rectangles and achieved a more efficient, constraint-compliant layout.

Improvements:

- Graph-based Problem Representation: The use of a graph structure for representing relationships between rectangles allows for more sophisticated placement strategies that respect proximity constraints.
- GCN Architecture: The GCN’s two convolutional layers and fully connected layers enable it to learn complex relationships between rectangles, progressively improving the placement strategy.
- Reinforced by the Loss Function: The loss function ensures that the model optimizes spatial efficiency while respecting constraints, avoiding overlap, and gradually refining the placements over training epochs.

- Penalty System: Penalties for suboptimal placements drive the model toward better solutions, ensuring that it not only fits rectangles but also adheres to the problem's specific rules.





FOURTH ITERATION

In the fourth iteration of the project, we introduced a strategic modification to how rectangles 1 and 2 were handled during the optimization process, building upon the foundation laid in the third iteration. While the third iteration focused on adjusting all rectangles, including rectangles 1 and 2, during optimization, the fourth iteration sought to refine this approach for more controlled and efficient placement.

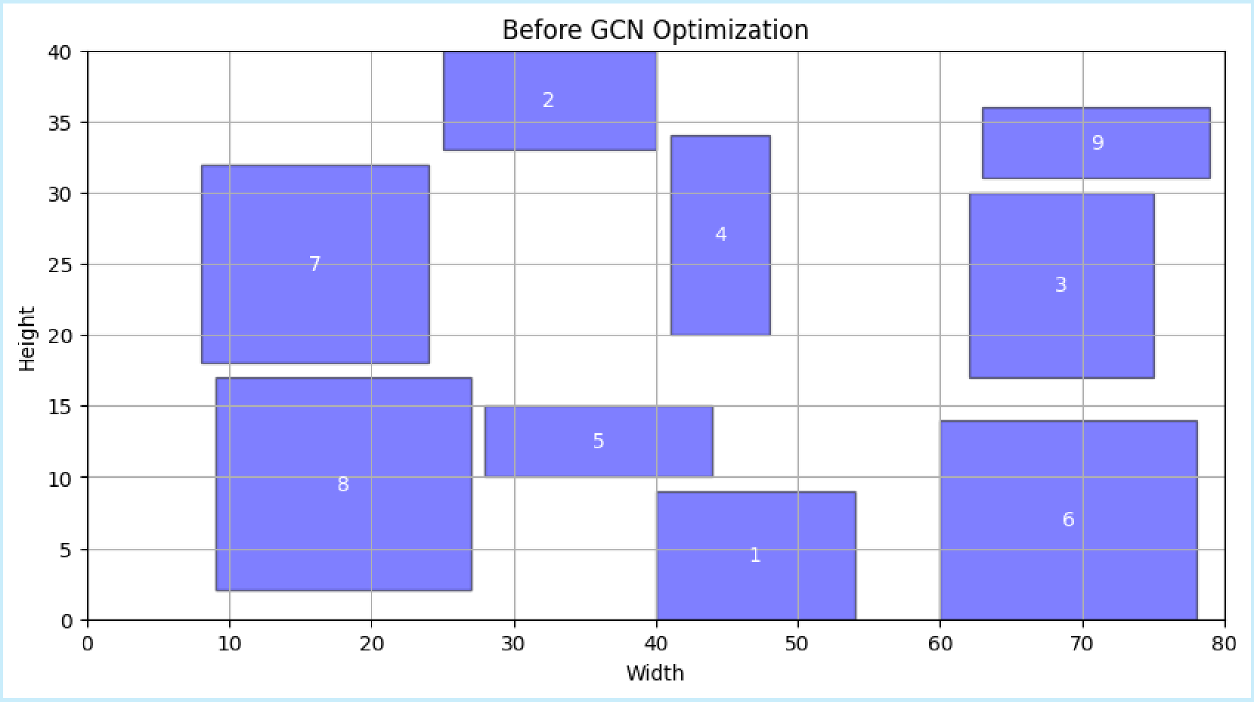
Changes in the Optimization Process:

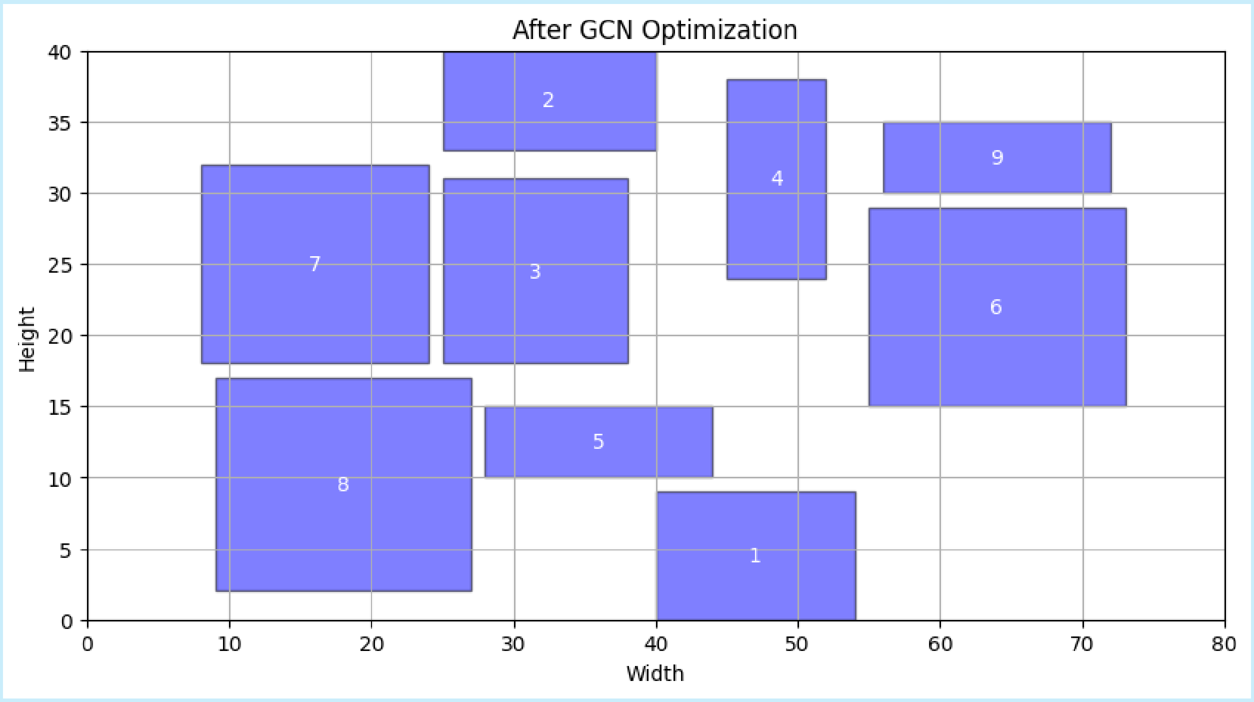
1. Rectangle Placement Adjustments:
    - In Iteration 3, rectangles 1 and 2 were manually placed at the top and bottom of the bin, respectively, but their positions were still dynamically adjusted during the optimization phase using the GCN model. The optimization loop included all rectangles, regardless of their predefined positions.
    - In Iteration 4, we maintained the initial fixed positions for rectangles 1 and 2 but made a critical change: these rectangles were excluded from the optimization loop. While they were still placed at the top and bottom of the bin manually at the start, the GCN model did not alter their positions during the optimization process. This was achieved by explicitly skipping these rectangles in the optimization loop using a continue statement for rect_id == 1 or 2.

Fixed Positions for Rectangles 1 and 2:

- The decision to keep rectangles 1 and 2 fixed was based on the recognition that their positions were already constrained by the problem's rules. Rectangle 1 was always placed at the top of the bin, and rectangle 2 was always placed at the bottom. Further optimization on their placements could have led to unnecessary adjustments, potentially disrupting their intended positions.
- By fixing rectangles 1 and 2, we focused optimization efforts solely on the remaining rectangles. This strategy allowed for a more controlled and efficient optimization process, ensuring that the positions of the constrained rectangles were not altered by the GCN model.

- The GCN model continued to adjust the remaining rectangles based on the predefined graph constraints, ensuring that their placement was optimized for space utilization while respecting the required proximity relationships.


Before GCN Optimization
Height
Width
1
2
3
4
5
6
7
8
9


After GCN Optimization
Height
Width
1
2
3
4
5
6
7
8
9

FIFTH ITERATION

In the fifth and final iteration, we took a more advanced approach by further refining the bin-packing solution with a Graph Convolutional Network (GCN). This iteration focused on optimizing the placement of rectangles while adhering to the constraints and introducing more sophisticated handling of the x and y coordinates.

Steps in the Fifth Iteration

1. Graph Construction and Constraints Handling:
    - We began by constructing a graph where each rectangle was represented as a node. The graph edges were drawn to indicate the proximity constraints, i.e., rectangles that need to be placed near each other.
    - For example:
        - Rectangle 1 had to be fixed at the top of the bin.
        - Rectangle 2 was fixed at the bottom.
        - Other rectangles had to respect their relative positions according to the defined constraints (e.g., rectangle 3 had to be placed near rectangles 4, 5, and 9).
    - This graph was then used to ensure that the constraints were represented properly during optimization.
2. Initial Rectangle Placement:
    - Rectangles 1 and 2 were manually placed at their fixed positions in the bin, ensuring that their y-coordinates were respected (rectangle 1 at the top and rectangle 2 at the bottom).
    - The x-coordinates for these rectangles were left open for optimization, meaning only their horizontal positions were adjusted during the optimization process.
    - The remaining rectangles were initially placed using a random placement strategy while respecting proximity constraints. If an overlap was detected, the algorithm would reattempt placement until a valid configuration was found.

3. Graph Neural Network (GCN) for Optimization:
    - A GCN was used to process the graph and output optimized positions for the rectangles.
    - The GCN took in a feature matrix representing rectangle properties such as width, height, and the current positions (x, y coordinates) and used this data to iteratively improve the rectangle placements over 5000 epochs.
    - The key modification in this iteration was that:
        - Rectangles 1 and 2 had their y-coordinates fixed, but their x-coordinates were optimized by the GCN.
        - The remaining rectangles had both their x and y coordinates optimized based on learned relationships from the graph structure.
4. Overlap Check and Update:
    - After obtaining new positions from the GCN, we checked for overlap between the rectangles. If any overlap was detected, the GCN model would adjust the placements and reattempt until a valid configuration was achieved.
    - This optimization process ensured that the rectangles were placed efficiently while minimizing wasted space in the bin.

Loss Function and Training

In this iteration, the loss function underwent a critical adjustment to reflect the optimization focus on the x-axis for rectangles 1 and 2, while keeping their y-positions fixed. Here's how the loss function was structured:

1. Loss Based on X-Axis (for Rectangles 1 and 2):
    - Since the y-positions of rectangles 1 and 2 were fixed, the loss function for these rectangles focused exclusively on optimizing their x-coordinates.
    - The model was penalized for any deviation from the desired x-positions. This ensured that rectangles 1 and 2 were placed at the top and bottom of the bin but with their optimal horizontal placement.

2. Overlap Avoidance and Constraint Satisfaction:
    - The primary loss function still focused on minimizing overlap and ensuring that the constraints (i.e., proximity requirements) were satisfied. The GCN was trained to respect the relative positions of the rectangles while adjusting the coordinates.
    - The loss function included penalties for overlap and deviations from desired proximity, encouraging the model to find the most efficient placements.
3. Training Process:
    - The Adam optimizer was used to adjust the GCN model's weights during backpropagation. By minimizing the loss function over time, the model gradually improved the placement of the rectangles.
    - The optimization process was designed to focus on efficiency by reducing the wasted space in the bin, while also ensuring that the proximity constraints were respected.

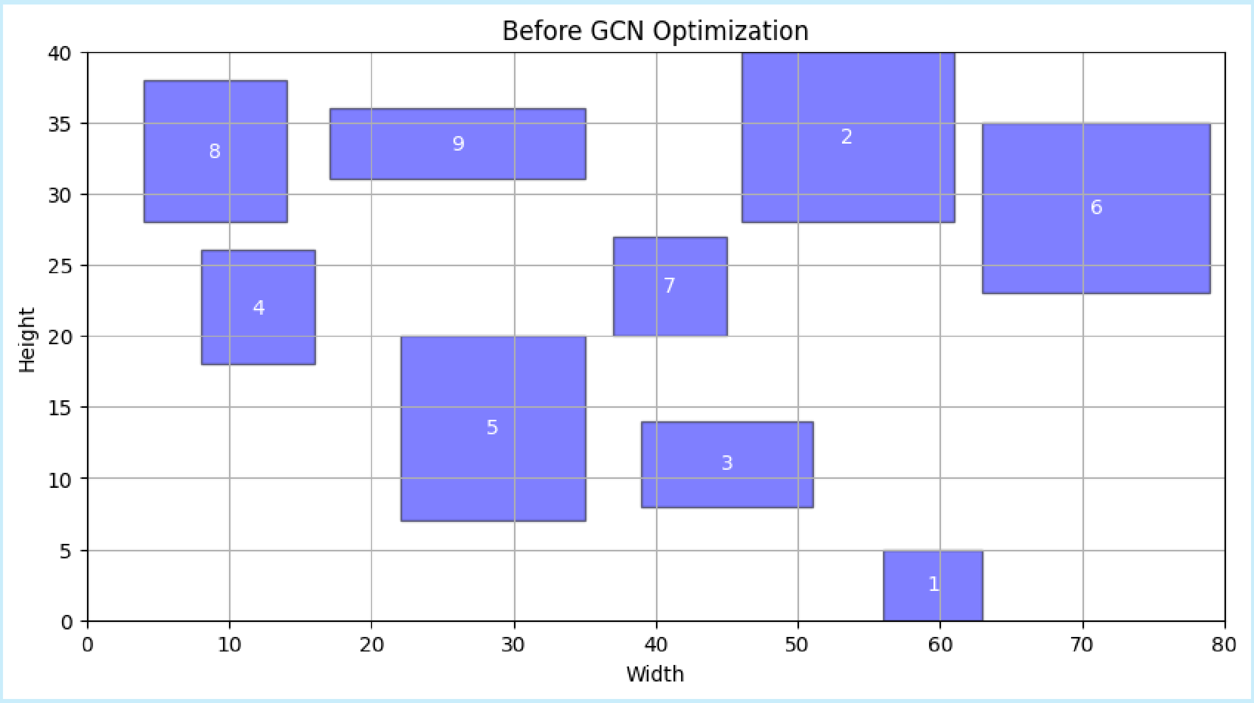
Results After Optimization

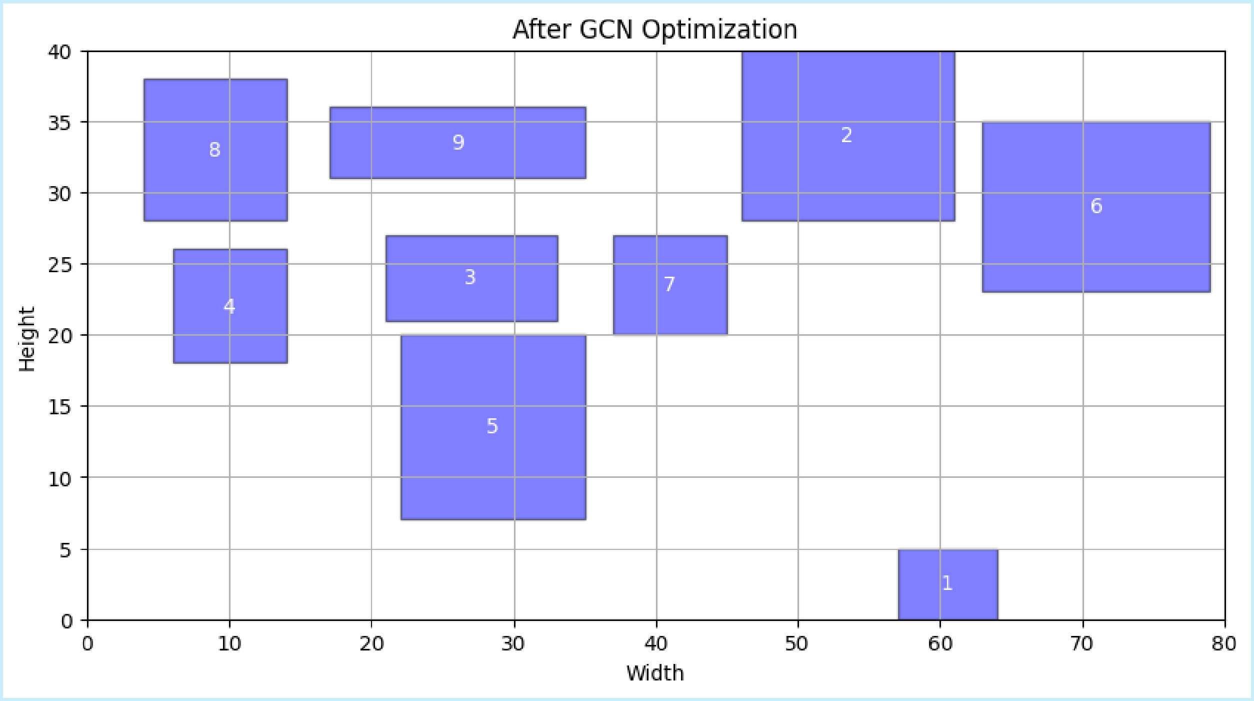
- Before GCN Optimization: The initial random placements were suboptimal. Rectangles were not fully utilizing available space, and overlaps occurred due to the random nature of the initial placements.
- After GCN Optimization:
    - The GCN refined the positions of rectangles, minimizing overlap and ensuring that all constraints were respected.
    - Rectangles 1 and 2, which had fixed y-positions, had their x-positions optimized to find the most efficient use of space while respecting their fixed vertical placements.
    - The remaining rectangles were placed optimally, and the overall layout showed better space utilization and fewer empty areas in the bin.

Visualizing the Results

The optimization results were visualized before and after the GCN adjustments, showcasing a clear improvement:

- The before visualization showed suboptimal placements with overlapping rectangles.
- The after visualization demonstrated a more compact arrangement, with no overlap and optimized use of available space.


Before GCN Optimization
Height
Width
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
60
70
80
8
9
2
6
4
7
5
3
1


After GCN Optimization
Height
Width
0
5
10
15
20
25
30
35
40
0
10
20
30
40
50
60
70
80
8
9
2
6
4
3
7
5
1

In conclusion, this project effectively addressed the rectangular bin-packing problem by using Graph Neural Networks (GCNs) to optimize rectangle placement within a fixed bin, adhering to predefined constraints. Through an iterative process, we refined the solution by addressing challenges, improving the optimization, and ensuring the final arrangement was efficient and compliant with constraints.

The first iteration involved random placement of rectangles as a baseline. In the second iteration, constraints were introduced to fix rectangles 1 and 2 at the top and bottom, but placement of other rectangles remained random with basic proximity considerations. The third iteration incorporated a GCN to optimize rectangle placements, considering proximity and overlap, with rectangles 1 and 2 fixed along the y-axis but optimized in the x-direction.

The fourth iteration further refined the process by focusing on the x-coordinates of rectangles 1 and 2 while optimizing the other rectangles, leading to a more compact bin layout. In the final iteration, the optimization was refined to efficiently handle the fixed positions of rectangles 1 and 2, with only their x-coordinates optimized, ensuring efficient space utilization and minimal overlap.

By the end, the GCN-based approach provided an optimized solution that adhered to all constraints and minimized wasted space, demonstrating the power of machine learning in solving complex bin-packing problems.