Programming problems related to chip design

BTP PROJECT

by

AKANSHA MADAVI
(2103104)
CSE

FACULTY ADVISOR

DR. SHARAD SINHA

DATE: 27/11/2024

AGENDA

01 Introduction to Problem Statement

02 Work Done and Solutions Proposed

03 Results

04 Future Work

05 Conclusion

06 Q&A

INTRODUCTION TO PROBLEM STATEMENT

"UNDERSTANDING THE CHALLENGE"

---

1. Brief overview of chip design and its significance.

- Vital for compact, efficient modern electronics.
- Balances performance, size, cost, and energy use.
- Crucial in industries: consumer electronics, telecom, automotive.

2. Common Programming Problems

- Component Placement: No overlaps, optimized arrangement.
- Space Utilization: Maximize area usage, avoid gaps.
- Order of Placement: Find the best sequence for arrangements.
- Flexibility Constraints: Fixed dimensions/orientations.
- Inspired by bin-packing and knapsack computational problems.

INTRODUCTION TO PROBLEM STATEMENT

"UNDERSTANDING THE CHALLENGE"

3. Importance of Solving These Problems

- Cost Reduction: Less material waste, lower production costs.
- Performance Boost: Shorter interconnects, improved efficiency.
- Scalability: More functionality in the same area.
- Environmental Benefits: Sustainable resource use.

4. Specific Problem Addressed

Efficient Placement of Components:

- Arrange smaller rectangles in a large rectangle without overlaps.
- Fit within boundaries, maximize area usage.
- Power Constraints: Adjacent boxes not placed together if crossed a certain threshold
- Visualize arrangements, calculate utilization %.
- Advanced techniques: fill gaps, varied starting posit

WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

Summary of methods or algorithms used (e.g., optimization techniques, heuristic methods).

Optimization Techniques:

- Implemented an arrangement strategy inspired by the bin-packing problem to maximize area utilization.
- Explored multiple placement orders to find the best arrangement for efficiency.

Heuristic Methods:

- Used a greedy placement algorithm to fit blocks into the Big box.
- Checked each block sequentially to find the first available space that meets constraints.

Tools and Programming Languages:

Python Matplotlib File handling

WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

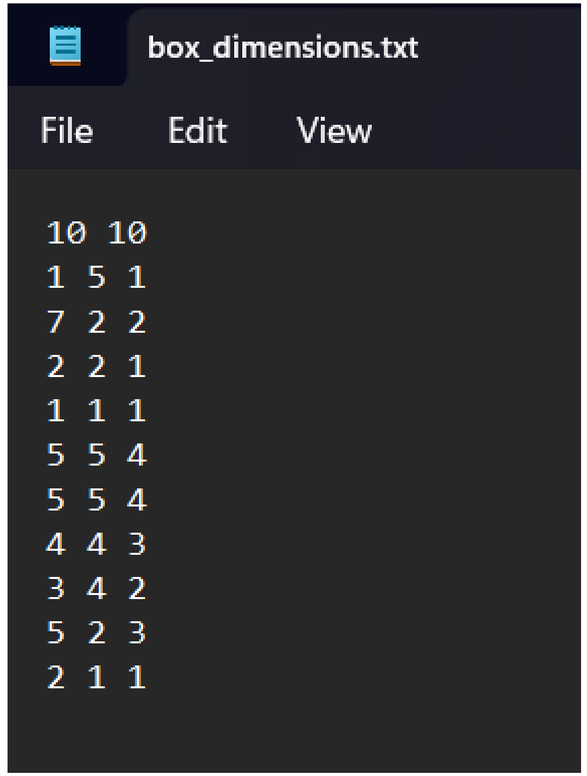
Step-by-Step Description of the Process

i. Input Stage:

A text file provided the following data:

- Dimensions of the large box (chip area).
- Dimensions of smaller boxes (chip components).
- Power consumption for each component.

Introduced a power consumption threshold for adjacent blocks.



WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

ii. Algorithmic Logic:

Placement Logic:

- Checked if a block could fit within the available space without overlapping.
- Applied constraints to ensure the power threshold was not exceeded for adjacent blocks.

Exploring Arrangements:

- Tested multiple placement sequences to evaluate area utilization.
- Calculated the percentage of utilized space for each configuration.

Optimization Enhancements:

- Filled gaps left in previous rows to improve area usage.
- Tried placements starting from different positions (corners, center).

WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

iii. Output Visualization and Metrics Tracked:

Metrics:

- Total used area versus available area.
- Percentage of area utilization.
- Components that fit and those that didn’t.

Visualization:

- Generated visual plots showing the arrangement of components.
- Highlighted labels for blocks, managing power constraints.

WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

---

iv. Visual Aid: Flowchart or Pseudocode

Flowchart Steps:

1. **Start:** Read input dimensions and constraints.
2. **Initialize:** Create an empty layout and set power thresholds.
3. **Placement Attempt:**
    - **For each block, check:**
        - Does it fit in the available space?
        - Does it meet power consumption constraints?
4. **Record Placement:** Update layout and metrics.
5. **Evaluate:**
    - If all blocks are placed, calculate area utilization.
    - If not, explore alternative arrangements.
6. **End:** Visualize the optimal arrangement.

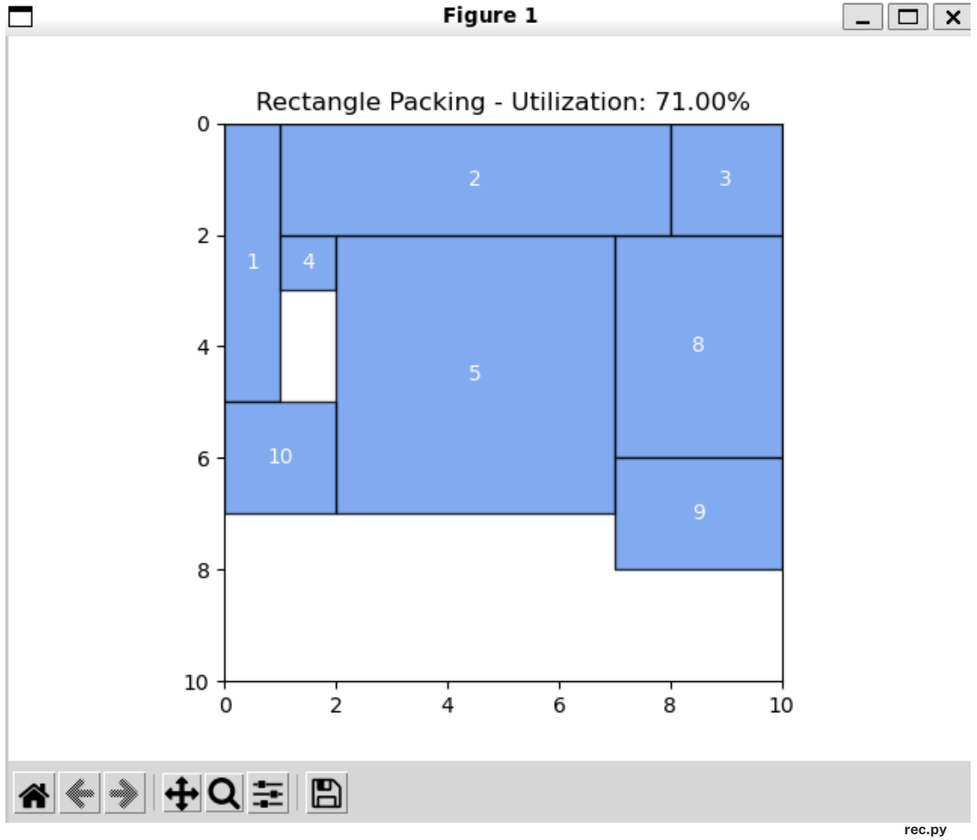
WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

"An algorithm to optimize space utilization in packing small boxes into a big box) without overlaps, maximizing space usage"

How It Works:

- **Input**: Dimensions of the big box and small boxes.
- **Goal**: Optimize packing of smaller rectangles into a larger rectangle to maximize space usage.
- **Process**: Greedy placement and Places boxes without overlaps; calculates utilization.
- **Output**: Visualization of packing with 71% utilization.
- **Challenges**: Some boxes remain unplaced; gaps reduce efficiency.
- **Applications**: Logistics, manufacturing, and layout design.



WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

“Objective: To arrange small boxes inside a large box to maximize space utilization while adhering to a power consumption constraint between adjacent boxes.”

How It Works:

Input:

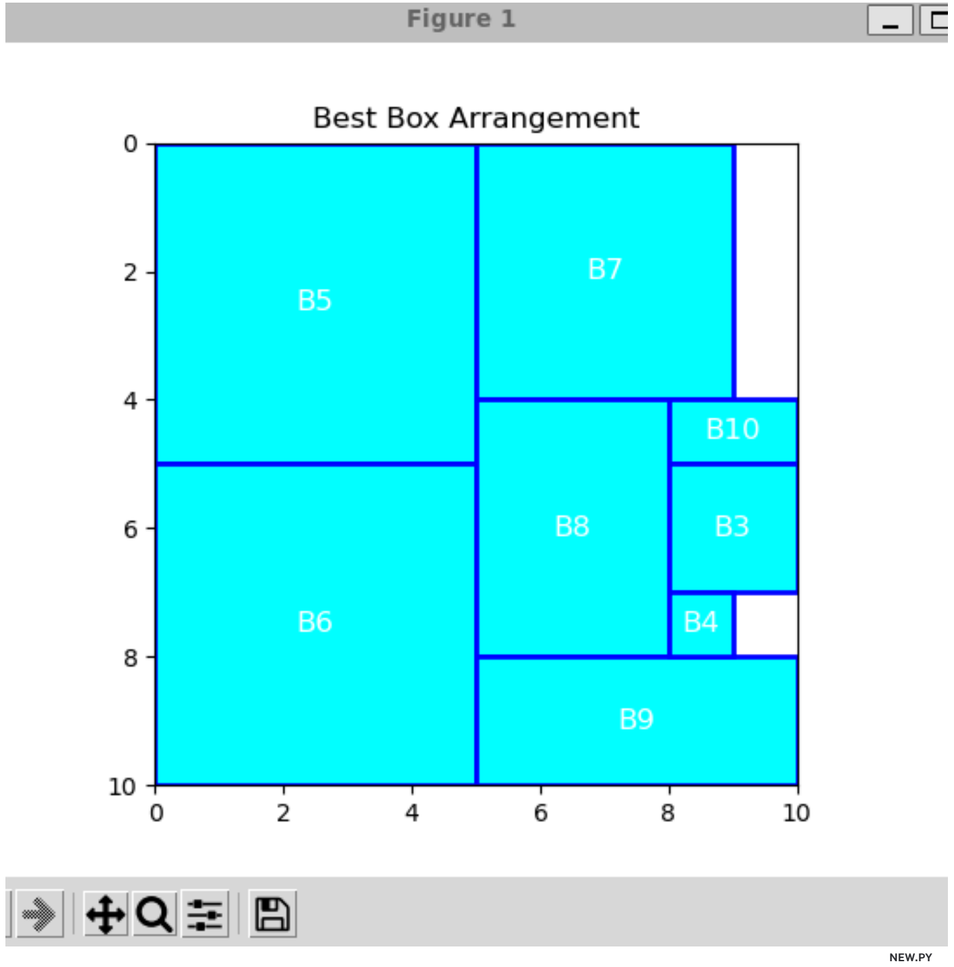
- Reads dimensions of the big box and small boxes
- Power consumption affects placement if adjacent boxes exceed a specified threshold.

Different Arrangements:

- Tries various orderings of box placement (highest utilization).

Efficient Space Usage: Optimizing layouts in storage or manufacturing.

Applications: Useful in electronics or packaging, where power consumption impacts placement.



WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

“Objective: This heuristic-based code aims to efficiently pack small boxes into a larger box by prioritizing larger boxes first and attempting different orientations.”

Heuristic-Based

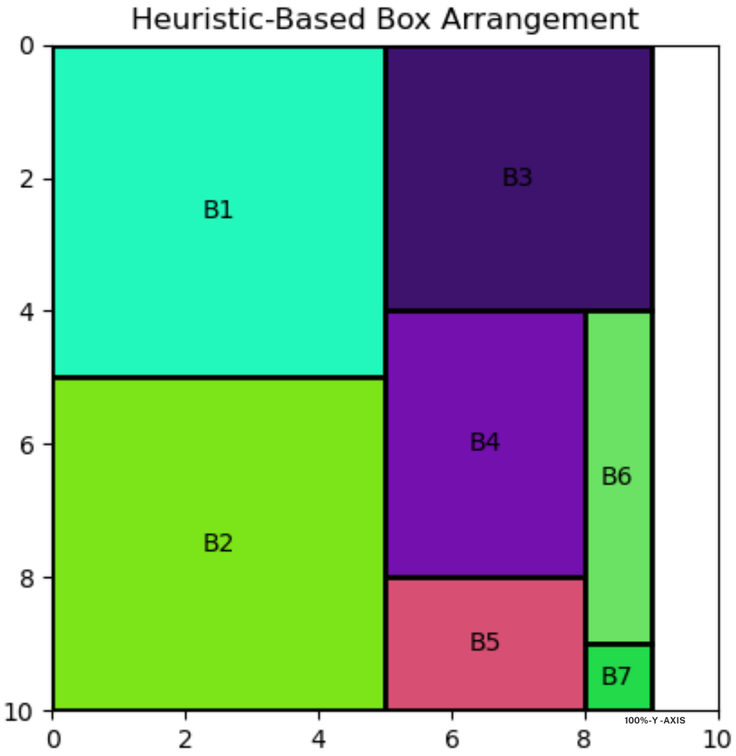
How It Works:

1.Input

2. Process:

- **Heuristic-Based Placement:**
  - **Sorting by Area**
  - **Placement Logic:**
    - Iterates through potential positions in the big box (x, y).
    - Tries both orientations (original and rotated) for each small box.
    - Checks for fit and overlap with already placed boxes.
- **Greedy Approach**

3.Area Utilization Calculation



WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

“Objective: This grid-based optimized code aims to efficiently place small boxes in a large container by using grid occupancy checks and rotation.”

How It Works

1. Inputs

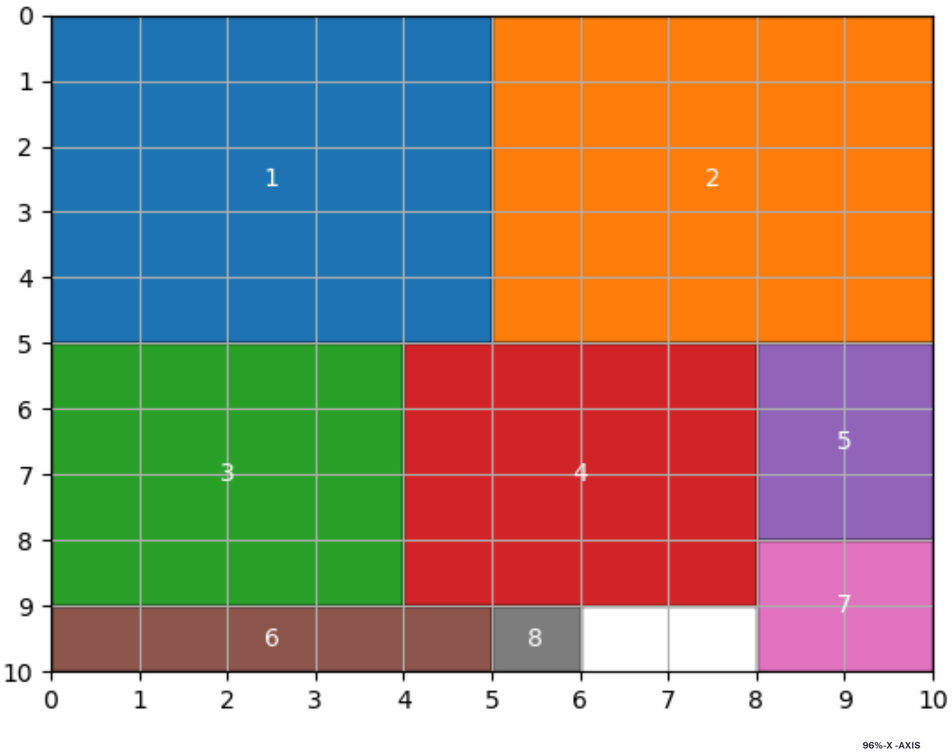
2. Process:

- **Grid-Based Approach**
- **Placement Strategy:**
  - Sorting by Area
  - Rotation Handling
- **Visualization**

3. Outputs:

- **Area Utilization**
- **Visualization**

Grid-Based Optimized Code



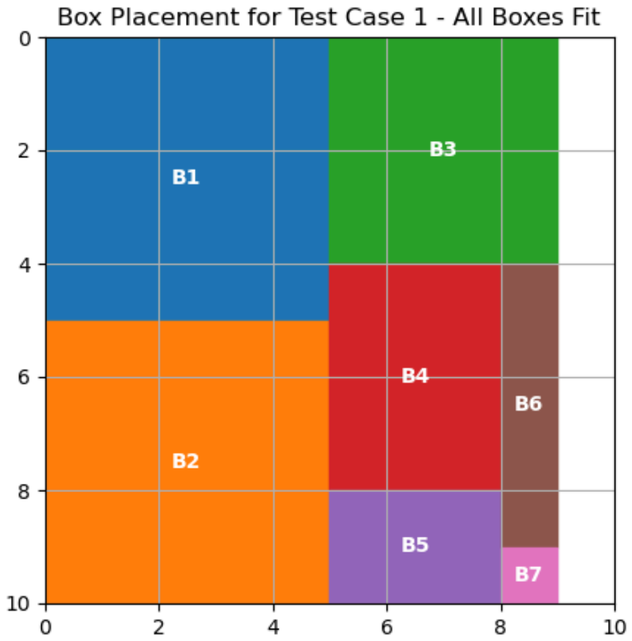
Why grid based approach is better?

WORK DONE

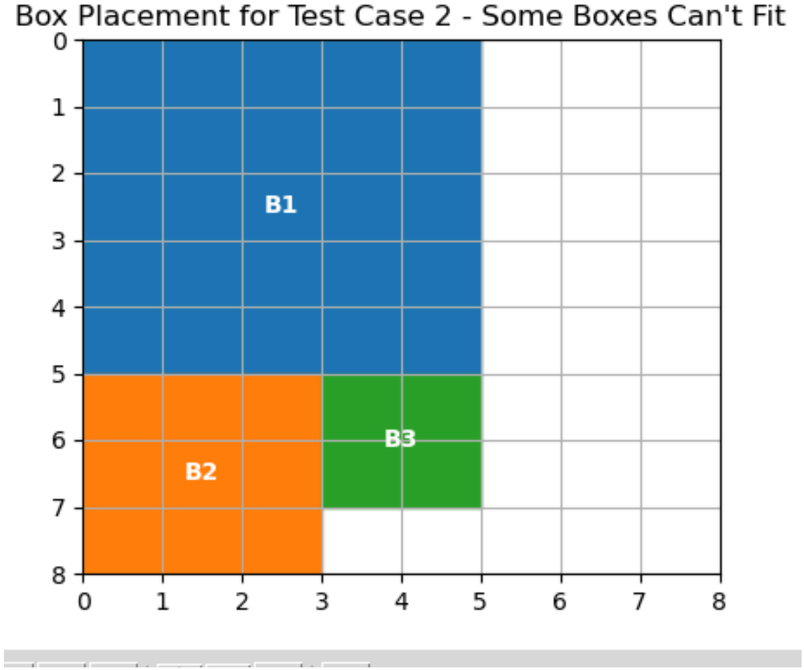
"APPROACH AND SOLUTIONS PROPOSED"

“Objective: Evaluates the box placement algorithm across multiple test cases, using grid-based placement to track space utilization and compare the results for different scenarios”

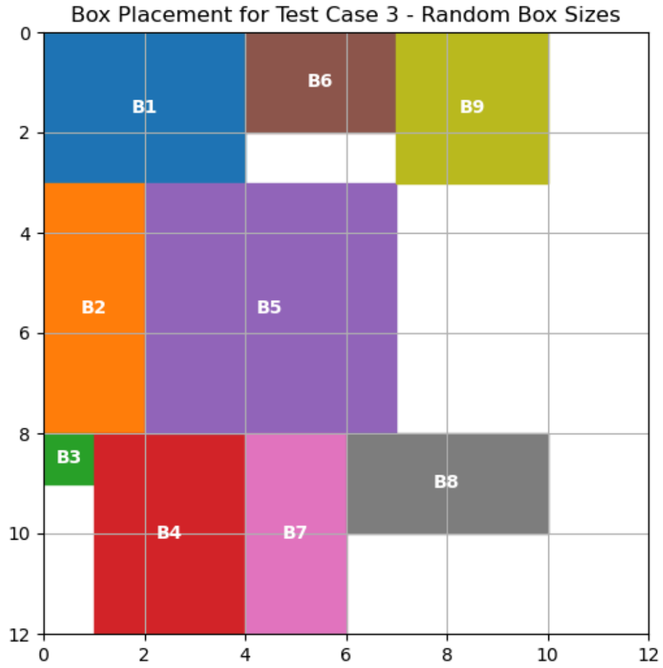
TEST CASE -1



TEST CASE -2



TEST CASE -3



Insights from the Test Cases

1.Optimal Packing: Test Case 1 shows the algorithm's peak performance when all boxes fit without gaps or unplaced boxes.

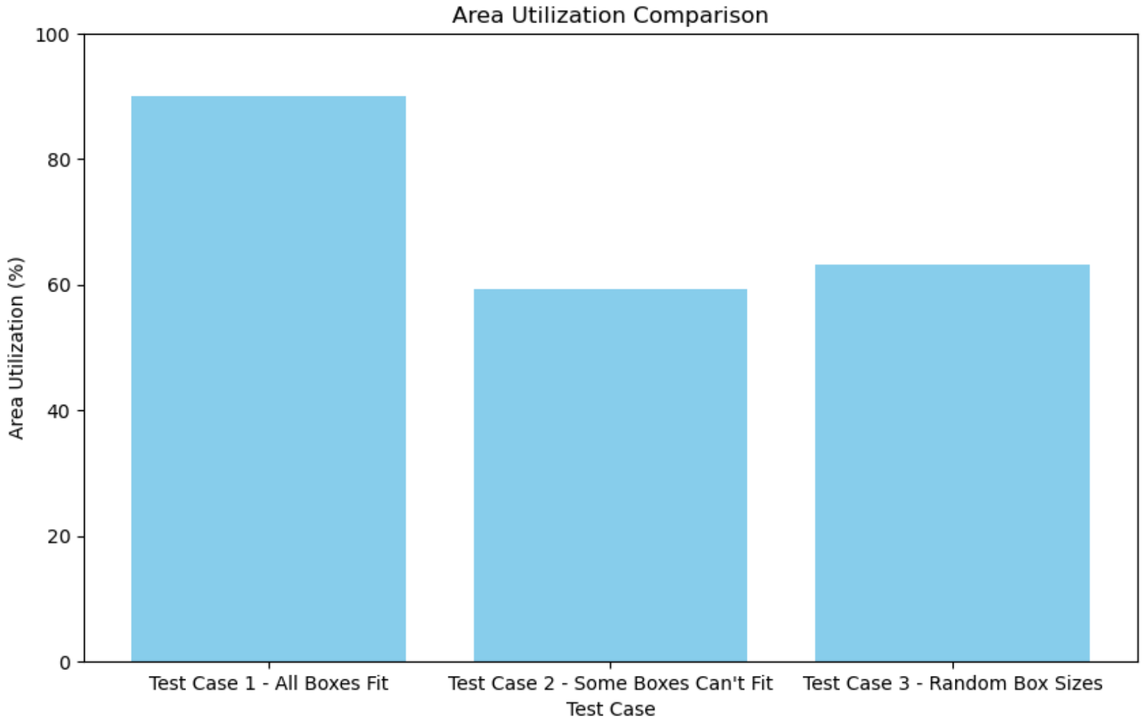
2.Handling Overpacking: Test Case 2 illustrates the algorithm's behavior when the grid is too small, emphasizing the need to prioritize box placement.

3.Adaptability to Variations: Test Case 3 highlights the algorithm's adaptability to random inputs, though gaps reduce utilization.

WORK DONE

"APPROACH AND SOLUTIONS PROPOSED"

TEST CASE -2



- The test case-based approach serves as an evaluation framework to assess the performance of the box packing algorithm under different conditions· conditions (e.g., all boxes fitting, overpacking, random inputs).
- Test Cases and Why They’re Important:
  - Tests multiple scenarios to ensure adaptability.
  - Quantifies performance for better comparison and analysis.
  - Highlights real-world challenges like overpacking and irregular inputs.
- Comprehensive analysis
- Real-world Applicabiity
- Provides insights into where the algorithm excels or fails, enabling targeted improvements.

RESULTS

"KEY FINDINGS AND PERFORMANCE"

Final Recommendations

1.For Simple and Efficient Packing: Use the First Code or Heuristic-Based Code for scenarios with minimal complexity.

2.For Constraint-Driven Applications: The Second Code is the best for solving problems with additional requirements, such as power or adjacency rules.

3.For Maximum Precision and Utilization: Use the Grid-Based Optimized Code for highly complex or irregular packing tasks.

4.For Comprehensive Algorithm Evaluation: The Test Case-Based Approach is essential for validating and comparing algorithms under diverse, real-world scenarios.

RESULTS

"KEY FINDINGS AND PERFORMANCE"

KEY DIFFERENCES BETWEEN ALL

Feature	First Code (Basic Packing)	Second Code(Power-Constarined Packing)	Heuristic-Based Code	Grid-Bases Optimized Code	Test Case-Baseed Approach
Primary Objective	Simple packing,maximize maximize utilization	Packing with constraints (Power Consumption)	Prioritize larger boxes for efficiency	Grid-level precision for optimal utilization	Evaluate Algorithm performance undr various scenarious
Flexibility	Limited to simple packing	Handles constraints but fixed placement	Limited flexibility,sorts by size	High flexibility,handles rotations	Converse diverse scenarios (overpacking irregular sizes)
Performance Analysis	No analysis,simple results	Evaluates packing under constraints	Modderate,focused on size-based logic	Precise placement,optimized utilization	Quantifies and compares performance across different scenario's

RESULTS

"KEY FINDINGS AND PERFORMANCE"

MAXIMUM UTILIZATION / MORE BOX PLACEMENTS - What Should Be Prioritized?

Aspect	Placing Maximum Boxes	Maximumizing Utilization
Why It's Important	- High functional density. - Supports future workloads. - Essential for high-performance chips (e.g., CPUs). Ex: Multi-core processors, GPUs, logic-heavy chips.	- Efficient area use. - Reduces silicon cost. - Simplifies interconnections, lowering power. Ex: Memory chips, IoT devices, compact SoCs.
When Prioritized	Fixed chip size, high logic density needed.	Fixed component count, focus on PPA optimization.
Drawbacks	-Potential gaps or wasted area. - Creates thermal hotspots.	- Risk of overlooking critical components. - Potential power density issues.
Key Considerations	Thermal management.	Cost reduction, manufacturability.
Conclusion	Focus on **functionality and performance.**	Focus on **efficiency and cost-effectiveness.**
Balanced Approach	Optimize for Performance, Power, and Area (PPA).	

FUTURE WORK

"WHAT'S NEXT?"

1. Enhanced Placement Algorithms
   - Optimization Techniques: Employ advanced techniques like genetic algorithms, simulated annealing, or machine learning to improve component placement.
   - Dynamic Constraints: Adapt to real-time constraints like thermal dissipation and power routing.

2. Interconnect and Routing Optimization
   - Focus on reducing wirelength, delay, and signal integrity issues in routing.

3.AI for Layout Optimization
   - Use machine learning models to predict the best placement strategies.

4. Parallel Processing for Optimization
   - Implement multi-threaded or distributed algorithms to speed up layout optimization.

5. User-Friendly Interface for Design
   - Create GUI-based tools for chip designers to easily input constraints and visualize solutions.

CONCLUSION

- Recap of Key Objectives
- Key Findings
- Practical Implications
- Future Prospects
- Final Thought

Thank you!