**Final Report on the Optimization of the Ordered Open-End Bin-Packing Problem (OOEBPP)**

**Authors:**

* **Alessandro Morselli**, Optimization Methods for Data-Driven Engineering Processes, Digital automation engineering**,** 272033@studenti.unimore.it.
* **Francesco Leone**,Optimization Methods for Data-Driven Engineering Processes, Digital automation engineering**,** 271153@studenti.unimore.it.

**Section 1: Problem Definition**

The Ordered Open-End Bin-Packing Problem (OOEBPP) aims to efficiently pack a sequence of items with given weights into the minimum number of bins. These items are packed in a predefined order, and each bin has a capacity that can only be exceeded by the last item placed in it – termed the 'overflow item'.

*Objective:*

Minimize the number of bins into which a given ordered sequence of items can be packed. The OOEBPP departs from traditional bin-packing problems in that it allows the last item placed in each bin to exceed the bin's defined capacity. This exception introduces a nuanced complexity to the decision-making process, as it affects how items are selected and placed.

*Constraints:*

1. **Order Preservation:** The items must be packed in a bin in the same order as they appear in the input sequence. This constraint eliminates the possibility of rearranging items for better fit, increasing the problem's complexity.
2. **Capacity Overstep:** Each bin has a defined capacity limit which can only be exceeded by the last item placed in it (the overflow item). This condition brings an additional layer of strategy to the packing process, as the selection of the overflow item affects the space utilization of each bin.
3. **Overflow Limitation:** Only one item – the last in the sequence within the bin – is allowed to exceed the bin's capacity. This rule necessitates careful consideration of the order and selection of items to maximize the utilization of bin space without violating the overflow criteria.
4. **Exclusivity of Items:** Every item is packed exactly once, ensuring that the item sequence is both collectively exhaustive and mutually exclusive across the bins.

This problem has practical applications in various fields such as logistics, resource allocation, and operations management, where the order of operations and capacity limits are critical factors. The study of the OOEBPP provides insights into how to balance between the order-induced constraints and the need for minimization of resources or space.

In solving the OOEBPP, we encounter an interplay between these constraints and the objective, as each decision about where to place an item immediately influences the options available for subsequent items. This interdependence requires a solution method that can intelligently navigate the combinatorial landscape to find not just a feasible solution, but the most efficient one given the problem parameters.

**Section 2: Model Implementation**

The model implemented to solve the OOEBPP is a 0-1 Integer Linear Programming (ILP) formulation that represents the decision-making process of bin packing with an ordered sequence and an overflow allowance. Below is a detailed explanation of each component of the model:

*Variables:*

* ​*yi* : Binary variable indicating if the ith item is the overflow item in its bin (1 if true, 0 otherwise).
* *xij*​: Binary variable denoting if the ith item (where i < j) is packed in the bin where the jth item is the overflow.

*Objective Function:* The objective function given by:

Immagine che contiene Carattere, testo, bianco, Elementi grafici

Descrizione generata automaticamente

aims to minimize the total number of bins used, which corresponds to the sum of *yi*​ variables that are set to 1, indicating that the ith item is serving as an overflow item in a bin.

**Assignment Constraint:**

*Immagine che contiene Carattere, testo, bianco, tipografia

Descrizione generata automaticamente*

Ensures that every item is assigned to exactly one bin, either as an overflow item or packed before an overflow item.

**Capacity Constraint:**

*Immagine che contiene Carattere, calligrafia, bianco, tipografia

Descrizione generata automaticamente*

Enforces that the total weight of items in a bin, excluding the overflow item, must not exceed the bin capacity minus one. The capacity is effectively reduced by one to accommodate the potential overflow item.

**Variable Binary Constraints:**

Immagine che contiene Carattere, testo, calligrafia, bianco

Descrizione generata automaticamente

These constraints dictate that the decision variables are binary, indicating the presence or absence of an item in a specific position.

The ILP formulation captures the essence of the OOEBPP by translating the problem's objectives and constraints into mathematical expressions. This model allows for the use of standard optimization solvers to determine the optimal or near-optimal bin packing configuration under the given conditions.

**Section 3: Constructive Greedy Heuristic**

The greedy heuristic is designed to provide a quick, though possibly suboptimal, solution to the OOEBPP. It builds a solution iteratively by making the locally optimal choice at each step, hoping to find a global optimum.

*Heuristic Idea:* The heuristic starts with an empty set of bins. It proceeds through the ordered list of items, attempting to place each one into an existing bin, following the constraints. If the current item does not fit into any bin without causing an overflow, a new bin is initiated, and the item becomes the overflow item of this new bin. The heuristic prioritizes filling bins with larger items to maximize the potential overflow benefit.

The greedy nature of the heuristic comes from its shortsighted decision-making process, where it focuses on the immediate placement of each item rather than considering future repercussions. This approach is generally faster but may not always yield the optimal solution.

**First-Fit Decreasing Heuristic**

The FFD heuristic was implemented in Python and applied to each instance. The heuristic's performance, in terms of the number of bins used, was recorded. As expected, the heuristic's computation time was negligible across all instances, completing in under a second even for the largest instance of 200 items. This reaffirms the FFD's suitability for quick approximations in large-scale problems.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

The idea behind using a greedy approach is to enhance computational efficiency. While it does not guarantee an optimal solution, it significantly reduces the processing time, making it a practical approach for large instances where exact solutions may not be computationally feasible.

By focusing on the current item and its immediate placement, the heuristic simplifies the decision-making process.

**Section 4: Computational Results and Performance Analysis**

This section provides an analysis of the First-Fit Decreasing (FFD) heuristic and the exact solution approach for the bin packing problem across five instances with varying number of items (N) and bin capacities (b). The instances are as follows: small-scale instances with 10, 20, and 50 items, a mid-scale instance with 100 items, and a larger-scale instance with 150 and 200 items. The weights of the items (a\_j) for these instances are detailed previously.

**4.1 Exact Solution Approach Using PuLP**

The exact solution approach was modeled for each instance using **PuLP**. The exact solutions provided benchmarks for the minimum number of bins required. As anticipated, the computational time for the exact approach escalated with instance size, ranging from seconds for the smallest instance to several hours for the 150 and the 200-item instance.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

| Instance | N Items | Bin Capacity (b) | FFD Bins Used | Exact Bins Used | FFD Time (s) | Exact Time (s) | Solution Quality Difference (%) |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Small-1 | 10 | 30 | 5 | 5 | <0.001 | 0.01 | 0 |
| Small-2 | 20 | 50 | 7 | 7 | <0.001 | 0.06 | 0 |
| Medium-1 | 50 | 200 | 16 | 15 | <0.001 | 1.41 | 6.26 |
| Medium-2 | 100 | 500 | 36 | 34 | <0.001 | 223.4 | 5.56 |
| Large-1 | 150 | 750 | 53 | 52 | <0.001 | 300 (L) | 1.89 |
| Large-2 | 200 | 1000 | 73 | 73 | <0.001 | 300 (L) | 0 |

\*L = Time limited

**Performance Analysis:**

*Computing Time:* The FFD heuristic demonstrated remarkable efficiency, with computation times being negligible across all instances. Even in the largest instance of 200 items, the FFD heuristic completed in less than a second. On the other hand, the exact solution approach had a noticeable increase in computation time, particularly for larger instances. The exact times for each instance are indicated in the table above. The image provided is a table displaying computational results and performance analysis for different instances of the Ordered Open-End Bin-Packing Problem (OOEBPP). It compares the First-Fit Decreasing (FFD) heuristic's performance with exact solutions across various instance sizes in terms of the number of bins used and the computation time, along with the solution quality difference expressed as a percentage. Here is a summary of the data presented in the table:

* **Small-1 Instance**: Both FFD heuristic and exact solution used 5 bins for 10 items with a bin capacity of 30. The computation times were less than 0.001 seconds for FFD and 0.01 seconds for the exact solution, with no quality difference.
* **Small-2 Instance**: 20 items with a bin capacity of 50 resulted in 7 bins used by both methods. The computation times were negligible for FFD and 0.06 seconds for the exact solution, also with no quality difference.
* **Medium-1 Instance**: For 50 items and a bin capacity of 200, FFD used 16 bins while the exact solution used 15 bins. The FFD heuristic was very fast, but the exact solution took 1.41 seconds, resulting in a solution quality difference of 6.26%.
* **Medium-2 Instance**: With 100 items and a bin capacity of 500, FFD used 36 bins compared to the exact solution's 34 bins. FFD was still under 0.001 seconds, while the exact solution time increased to 223.4 seconds, with a 5.56% solution quality difference.
* **Large-1 Instance**: For 150 items and a bin capacity of 750, the FFD heuristic used 53 bins, and the exact solution used 52 bins. The FFD remained extremely fast, whereas the exact solution hit a time limit of 300 seconds (L), resulting in a 1.89% solution quality difference.
* **Large-2 Instance**: In the largest tested instance of 200 items with a bin capacity of 1000, both methods used 73 bins, with the computation time for FFD remaining negligible and the exact solution also reaching the time limit of 300 seconds (L). There was no solution quality difference in this case.

Here are some **key points** that stand out from the analysis:

**Time Efficiency of the FFD**: The FFD heuristic proves to be extremely efficient with negligible computation times for all instances, demonstrating its practicality for problems of both small and large sizes.

**Increase in Time with Exact Solutions**: As expected, the computation time significantly increases with the exact solution as the size of the instance grows, becoming particularly notable for medium and large-sized instances.

**Solution Quality Difference**: For small to medium instances, the FFD shows a slight discrepancy in solution quality compared to the exact method, indicated by the percentage difference. However, for large instances, this difference becomes smaller, indicating that the FFD performs quite well in terms of solution quality, especially when computation time becomes a limiting factor for the exact method.

**Time Limits**: For the Large-1 and Large-2 instances, the exact method reaches the time limit of 300 seconds, indicated with (L), suggesting that the computation time for an exact solution could become prohibitive for even larger instances.