Journal of Information Systems Engineering and Management
2025, 10(9s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Efficient Component Packing with Algorithmic Optimizations

Nilesh P Sable[1], Kirti H. Wanjale[2], Swapnali Makdey[3], Ketan J. Raut[4], Ashok Kanthe[5], Varsha Jadhav[6]

[1]Department of Computer Science & Engineering (Artificial Intelligence), Vishwakarma Institute of Information Technology, Pune. drsablenilesh@gmail.com

[2]Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra, India kirti.wanjale@vit.edu

[3]Assistant Professor, Department of Artificial Intelligence and Data Science, Fr. Conceicao Rodrigues College of Engineering, Mumbai, Maharashtra, India. swapnalimakdey@gmail.com

[4]Department of Electronics and Telecommunication Engineering, Vishwakarma Institute of Technology (Kondhwa campus), Pune ketan.raut@viit.ac.in

[5]Associate Professor, Department of Computer Engineering, Fr. Conceicao Rodrigues College of Engineering, Mumbai, Maharashtra, India. ashokkanthe@gmail.com

[6]Associate Professor, Artificial Intelligence and Data Science , , Vishwakarma Institute of Technology, Pune, Maharashtra, India Varsha.jadhav@vit.edu

ARTICLE INFO

Received: 22 Oct 2024

Revised: 18 Dec 2024

Accepted: 30 Dec 2024

ABSTRACT

This research paper introduces a recursive algorithm that maximizes component placement by optimizing their orientations in each layer. A software utility was developed using this algorithm called Productive Package. This is used to calculate the maximum number of components that can be packed in the box, the utility enhances efficiency and reduces errors compared to manual calculations. This involves system analysis and design methods to optimize the packaging process. The evaluation of Productive Packager demonstrates its accuracy and effectiveness by comparing its output with manual packaging calculations for various box and component sizes. The utility offers time savings, improved accuracy, and customization options, making it a valuable tool for optimizing packaging processes and enhancing transportation efficiency.

Keywords: Optimisation algorithm, Box Packing Problem, Python, 3D Optimization, STEP, STL

1. INTRODUCTION

Packaging is an essential aspect of product distribution, and optimizing the use of space within a container or box is critical to ensure efficient and cost-effective transportation. The task of calculating the optimal number of components that can be accommodated within a box through manual means is not only labor-intensive but also susceptible to inaccuracies. The Three-Dimensional Bin Packing Problem (3D-BPP) presents a tangible and practical combinatorial optimization challenge that significantly influences economic outcomes, environmental considerations, and safety concerns (Ramos et al., 2018). Packing methodologies have found extensive application across various industries, including the automotive sector, as an effective strategy for addressing the storage dilemmas associated with components within containers (Joung et al., 2014).

This research paper presents the development and evaluation of a software utility called Productive Packager, which aims to calculate the maximum number of components that can be packed inside a box of fixed size. The utility employs a custom recursive algorithm that optimizes the orientation of the components to fit the maximum number of them in a layer. The algorithm iterates layer by layer, and the final sum is displayed to the user.

The Productive Packager utility offers several advantages over traditional manual packaging calculations. Firstly, it reduces the time required to determine the optimal orientation of the components, thus improving efficiency. Secondly, it improves the accuracy of packaging calculations by accounting for the dimensions of each component and ensuring that they are arranged in the most efficient way possible. Finally, it offers flexibility by allowing the user to specify the dimensions of the box and the components, enabling customization to specific packaging requirements.

The evaluation was carried out by comparing the output of the Productive Packager algorithm with manual packaging calculations for a range of box and component sizes. The algorithm also demonstrated a high level of accuracy, with the output matching the manual calculations.

In the development of the proposed algorithm, the principles of system analysis and design played a crucial role. System analysis involved a thorough examination of the existing packaging processes, identifying challenges, and defining the requirements for an optimized solution. System design encompassed the creation of a recursive algorithm that maximizes component placement through optimized orientations in each layer. By applying these principles, we aimed to align business objectives with the technical aspects of the system, creating a tool that provides tangible benefits in terms of time savings, improved accuracy, and customization options.

2. RELATED WORK

The problem of packing components into a box is a classic problem in operations research. There are many different algorithms that have been proposed for solving this problem, including heuristic algorithms, exact algorithms, and machine learning algorithms.

Heuristic algorithms are typically the most efficient algorithms for packing components into a box. Some of the most well-known heuristic algorithms include the first-fit algorithm, the best-fit algorithm, and the next-fit algorithm. These algorithms are relatively simple to implement and can be used to achieve good packing solutions in a reasonable amount of time.

Exact algorithms can be used to find the optimal packing solution, but they are typically much slower than heuristic algorithms. The bin-packing problem represents a significantly challenging combinatorial optimization issue with strong NP-hard characteristics, as noted by Korte et al. (2012). Which means that there is no polynomial-time algorithm that can find the optimal solution in all cases.

Machine learning algorithms have recently been proposed for solving the packing problem. These algorithms can learn from historical data to improve their performance. Dynamic Programming algorithms are also introduced.

Typically these bin-packing problems are classified into 1, 2 or 3 dimensions. Secondly, the shape of the container, if it’s regular or irregular and also if the shapes are of one type or many.

There is a lot of research done on the 2D BPP where various algorithms were introduced to pack rectangles and squares effectively in a 2D space. Many of these algorithms are heuristic in nature and employ various optimisation strategies on top of the core logic. Chan et. al (2011) summarizes and analyzes many of these 2D BPP algorithms, including one phase heuristics, two phase heuristics and local search heuristics. Lodi et. al (2014) also summarizes 2D algorithms in addition to approximations to optimize the model.

Work on 3D has been slow but steady. Harrath (2021) proposed a new heuristic algorithm for solving the 3D bin packing problem under a balancing constraint. The balancing constraint requires that the bins be filled as evenly as possible. Joung et al. (2014) came up with a grouping algorithm, which is a heuristic algorithm for 3D packing of freeform objects. The algorithm clusters components with comparable shapes by evaluating the likeness of their enclosing boxes. The grouping algorithm is then used to determine the order in which the parts are loaded, the orientation of the parts, and the position of the parts. Erbayrak et al. (2021) combines these approaches to develop a new multi-objective optimization model for the 3D bin packing problem. The model considers two objectives: the number of bins used to pack the items and the load balance between the bins. The model also considers the family of each item, which is a set of items that have similar properties. Some algorithms like one proposed by Chen et. al (2010) employ genetic algorithms combining it with local search heuristics for better optimization.

However, despite the progress in prior research addressing specific aspects of packaging optimization and system analysis, there remains a need for a comprehensive software utility that integrates recursive algorithms and efficient orientation optimization to maximize component placement. This paper aims to address this gap by introducing the Productive Packager utility, which incorporates advanced system analysis and design principles to offer an efficient, accurate, and customizable solution for optimizing packaging processes and enhancing transportation efficiency. By building on the insights and approaches from the aforementioned works, this research contributes to the field by providing a practical tool that aligns business objectives with the technical aspects of packaging optimization.

2.2 MOTIVATION

The 3D Box Packing Problem (BPP) is a classic optimization problem that arises in various real-world applications, including logistics, warehousing, and transportation. The motivation for this paper comes from the practical difficulties faced by small and medium-sized industries in packing components inside boxes of fixed sizes. The cost of boxes and shipping charges can quickly add up, and thus, finding an optimal packing solution can help reduce packaging material and transportation costs significantly. The proposed algorithm in this paper aims to provide an efficient and effective solution to the BPP, ultimately helping industries save time, money, and resources.

Small to medium-sized industries often struggle with the problem of efficiently packing components inside boxes of fixed sizes, resulting in additional costs of purchasing larger boxes and higher shipping fees. This problem can have a significant impact on the overall cost of their operations, affecting their profitability and competitiveness in the market. To address this challenge, we propose a solution that leverages advanced algorithms and optimization techniques to efficiently pack components into boxes, reducing the need for larger boxes and lowering shipping costs . This solution has the potential to significantly improve the bottom line of these industries, allowing them to remain competitive in the market and invest in growth opportunities.

3. OBJECTIVES

The objectives of this project are as follows:

- To design and develop an algorithm that can accurately extract the dimensions of components from a 3D (STEP) file.
- To develop a custom algorithm that can calculate the maximum number of components that fit inside a fixed dimension box.
- To build a user-friendly PC application that can accept 3D (STEP) files as input and display the calculated maximum number of components as output.
- To test the application and algorithm and ensure that it can handle a range of input scenarios and produce accurate and reliable results.

3.1 Constraints.

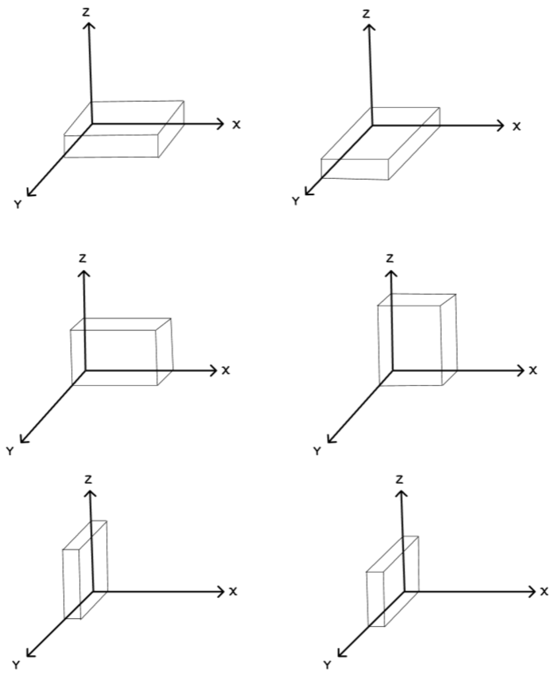
- Every individual component must be fully contained within the confines of the box.
- Overlap among components inside the box is strictly prohibited.
- Every component is oriented in parallel alignment with the surfaces of the box.

4. WORKING

- The user first inputs the dimensions of the outer box (length, breadth and height). This is achieved using a Graphical user interface (GUI) which is built in python using the Tkinter library.
- Users opt for a 3D Standard for the Exchange of Product Data (STEP) file. These STEP files are designed to facilitate the generation, modification, and distribution of 3D model designs among diverse computer-aided design (CAD) software. This file format adheres to standardized specifications.
- Algorithm converts the STEP file to a STL file which is easier to work with. Stereolithography (STL) is a file format commonly used for 3D printing and computer-aided design (CAD). It lacks color and texture. Here, extra details of the component are smoothed out and converted to a simpler model so that the computation of dimensions is faster. This uses the Trimesh and GMSH modules which are available in Python and some other languages.
- Now using the STL model as the base, the dimensions of the components are calculated, i.e. length, breadth and height. This is basically calculating the dimensions of the bounding box of the component. NumPy-STL is the python module that helps us calculate the extents and thereby the bounding box of the stl model.
- The custom algorithm calculates the optimal orientation of the component for a layer. The logic is based on a simple mathematical algorithm.

The challenge of determining how parts should be positioned is known as the part-orientation problem. In two-dimensional situations, the calculation of orientation is constrained within a span of 0 to 360 degrees. However, in 3D scenarios, orientation calculation becomes more complex as it needs to consider each axis individually. This increased complexity poses computational challenges. Santosh et al. addressed this issue by defining the orientation of parts based on a 90-degree criterion [12]. Particularly, when handling a right cuboid component characterized by dimensions (l × b × h), it is possible to derive six distinct orientations, as illustrated in Figure 1.

Figure 1. Six Orientations of a single component



4.1 ALGORITHM

Data : Dimensions of box(x,y,z) and item(a,b,c)

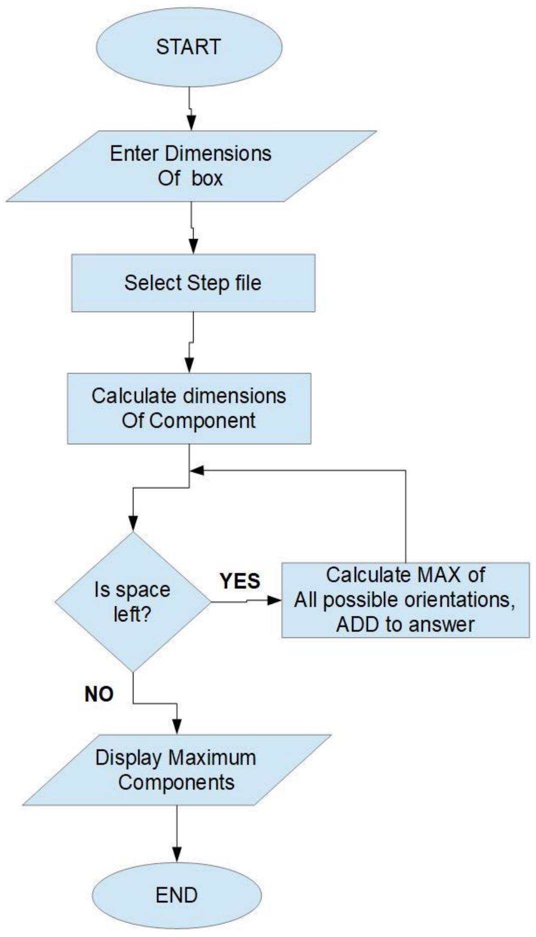
Result : No. of items that can fit inside the box

```
set answer to 0
function boxes(dimensions, spaceleft)

    val ← call (bestorientation)
    add val to answer

    when val = 0
    return

    spaceleft ← call(freespace)

    call (boxes)
call (boxes)
print (answer)
```

bestorientation : $\lfloor \frac{x}{a} \rfloor \times \lfloor \frac{y}{b} \rfloor \times \lfloor \frac{z}{c} \rfloor$ will give you number of boxes in one orientation. We have to find the maximum number of boxes for the current remaining space. For that we must check for each orientation of the item. Simply permute a,b,c. Compute each orientation and take the largest.

freespace : The permutation that gives the most number of boxes helps in finding the free space for the next iteration. It is observed that the box dimension that leaves the most space(remainder) when divided with its respective item dimension i.e.[max(dim_box % dim_item)] is the only box dimension that changes for the next iteration. Other two remain the same. Flowchart for the algorithm is depicted in figure 2.

Figure 2. Flowchart for the algorithm


START
Enter Dimensions Of box
Select Step file
Calculate dimensions Of Component
Is space left?
YES
Calculate MAX of All possible orientations, ADD to answer
NO
Display Maximum Components
END

4.2 TECHNOLOGIES USED

We have used different modules of Python to achieve various functionalities.

1. **Tkinter -** Tkinter serves as an excellent technology stack for developing graphical user interfaces (GUIs) in Python. As a native integration with the Tk GUI toolkit, Tkinter seamlessly connects Python code with the underlying Tk library, leveraging its powerful features. With its cross-platform compatibility, Tkinter empowers developers to create GUI applications that can run smoothly on various operating systems, including Windows, macOS, and Linux. One of the key advantages of Tkinter is its ease of use, as it provides a straightforward and intuitive interface for GUI development. Its extensive widget toolkit enables the creation of interactive and visually appealing user interfaces, including windows, buttons, labels, menus, and more. Tkinter also offers support for event-driven programming, allowing developers to respond to user interactions effectively. Moreover, Tkinter integrates smoothly with other Python libraries and tools, enhancing its flexibility and expanding its capabilities. Its inclusion in the Python standard library ensures its availability and stability for developers worldwide. Overall, Tkinter empowers developers to create feature-rich GUI applications in Python with relative ease, making it a popular choice in the Python ecosystem.

2. **Trimesh -** Trimesh provides a comprehensive set of functionalities for working with 3D mesh data. One of its key capabilities is the ability to load and process STEP files, which are commonly used for representing 3D CAD models. By using Trimesh, you can import a STEP file, gaining access to the geometric information and topology of the CAD model. Trimesh allows you to extract vertices, faces, and other relevant information from the loaded model, enabling further processing and analysis.

3. **Gmsh -** Gmsh is a specialized library for mesh generation and manipulation. In the context of converting STEP files to STL files, Gmsh plays a crucial role. Once you have loaded the CAD model using Trimesh, you can utilize Gmsh to convert the extracted geometry into a format compatible with Gmsh. Gmsh supports various mesh formats, including its native MSH format and the widely used STL format. You can utilize Gmsh to generate an unstructured mesh based on the extracted geometry, ensuring that it satisfies your specific requirements. Gmsh provides extensive control over meshing parameters, allowing you to achieve the desired mesh quality and refinement. Finally, Gmsh enables you to export the generated mesh in the STL format, which is widely accepted and supported by various applications and 3D printers.By leveraging the capabilities of Trimesh and Gmsh together, you can seamlessly convert STEP files to STL files. Trimesh allows you to access the geometric information and topology of the CAD model, while Gmsh provides the tools and algorithms for generating a high-quality mesh representation compatible with the STL format.

   This combined workflow ensures efficient and accurate conversion from STEP to STL, enabling you to further utilize the resulting STL file for 3D printing, simulation, or other applications that require a mesh representation of the original CAD model.

4. **Numpy-STL -** The numpy-stl library offers a convenient way to calculate dimensions of objects extracted from STL files. By loading the STL file using the library's mesh.Mesh.from_file() function, the mesh data is accessible through attributes of the resulting mesh.Mesh object. The mesh.vectors attribute provides access to the vertices of the mesh, while the mesh.points attribute holds the corresponding coordinates. With this data available, various dimensions can be calculated. For example, the bounding box dimensions can be determined by finding the minimum and maximum values along each axis (x, y, z) of the vertices. These calculations leverage the capabilities of numpy-stl to provide accurate and efficient measurements of the object's dimensions, enabling further analysis, simulation, and design optimization.

5. RESULTS

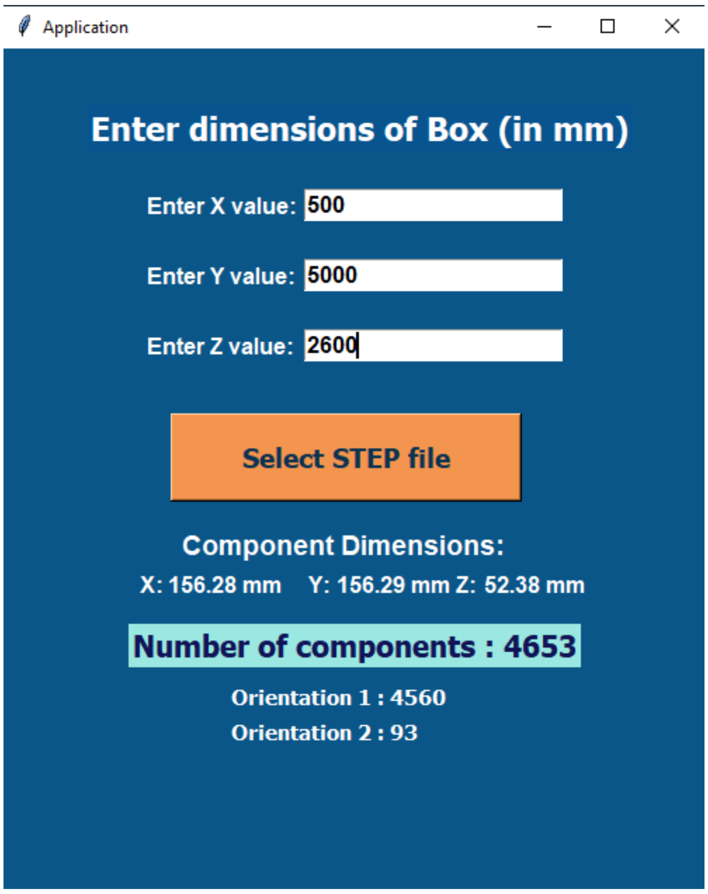
The GUI for the application is showcased in Figure 3. It has input fields for the dimensions of the box, i.e. length, breadth and height.

Next input is the selection of the step file. Clicking the button opens up the file open dialog box upon which the user can select the STEP file.

The dimensions of the components are calculated and displayed.

The total number of components that would fit inside the box are displayed and highlighted. Also the number of components that fit in an individual orientation are also displayed.

Figure 3. User Interface of the software utility



Enter dimensions of Box (in mm)

Enter X value: 500

Enter Y value: 5000

Enter Z value: 2600

Component Dimensions:

X: 156.28 mm Y: 156.29 mm Z: 52.38 mm

Number of components : 4653

Orientation 1 : 4560

Orientation 2 : 93

For example, if the component dimensions are (4,2,2) and the box dimensions are (6,6,6). Figure 4 demonstrates how they might be laid out by the manual method (components only in one orientation) while Figure 5 demonstrates how they are arranged according to our algorithm. The manual method only manages to pack 9 components with a packing efficiency of 66.67 % while our algorithm manages to pack 13 components with a packing efficiency of 96.67%. There is an improvement of 30%.

Figure 4. Components arranged using manual method (single orientation)

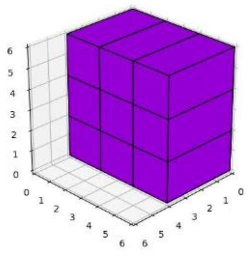
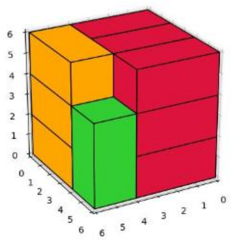


Figure 5. Components arranged using our algorithm (multiple orientations)



The following table summarizes the results for the tests conducted for verifying the effectiveness of our algorithm. The component size and box size are specified in cm. 'Manual' and 'Our Algorithm' specify the packing efficiency using the manual method and then using our algorithm respectively. The last column highlights the betterment in the packing efficiency due to our algorithm.

Table 1. Comparison of efficiency of manual method and our algorithm.

Component Size (cm)	**Box Size (cm)**	**Manual (%)**	**Our Algorithm (%)**	**Betterment (%)**
4x2x2	6x6x6	66.67	96.67	30
5x15x25	20x50x70	85.71	96.43	10.72
5x6x6	10x20x25	64.8	75.6	10.8
5x5x15	20x20x100	90.00	99.37	9.37
4x7x7	16x20x20	61.25	73.5	12.25
5x5x13	18x24x44	61.55	70.10	8.55
1.5x2x3.5	4x9x12	66.66	77.77	11.11
9x12x144	20x144x666	77.8	89.1	11.3

6. CONCLUSION

The Productive Packager algorithm is an effective and efficient tool for optimizing the use of space when packing components into a box. The custom recursive algorithm used by the utility ensures that components are arranged in the most efficient way possible, while the user-friendly interface allows for customization to specific packaging requirements. The utility offers significant advantages over traditional manual packaging calculations, including improved efficiency, accuracy, and flexibility. The results of the evaluation demonstrate the effectiveness of the Productive Packager utility and suggest that it could be a valuable tool for various industries that require efficient packaging solutions.

ACKNOWLEDGMENT

This research did not receive dedicated funding support from any funding agency in the public, commercial, or nonprofit sectors.

REFERENCES

[1] Youn-Kyoung Joung, Sang Do Noh. “Intelligent 3D packing using a grouping algorithm for automotive container engineering”, in Journal of Computational Design and Engineering, Vol. 1, No. 2 (2014) 140~151

[2] Santosh T, Fadel G, Fenyes P. A fast and efficient compact packing algorithm for free-form objects. In: ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference; 2008 Aug 3-6; New York, NY; p. 543-552.

[3] Ramos, A. G., Silva, E., & Oliveira, J. F. (2018). A new load balance methodology for container loading problems in road transportation. European Journal of Operational Research, 266(3), 1140–1152.

[4] Lodi, A., Martello, S., Monaci, M., & Vigo, D. (2014). Two-Dimensional Bin Packing Problems in Paradigms of Combinatorial Optimization, 107–129.

[5] Abdolahad Noori Zehmakan (2015). Bin Packing Problem: Two Approximation Algorithms in International Journal in Foundations of Computer Science & Technology (IJFCST, July 2015, Volume 5, Number 4.

[6] Chan, T., Alvelos, F., Silva, E., & Valériode Carvalho, J. (2011). Heuristics for Two-Dimensional Bin-Packing Problems. Intelligent Systems, 1–18.

[7] Harrath, Y. (2021). A Three-Stage Layer-Based Heuristic to Solve the 3D Bin-Packing Problem under Balancing Constraint. Journal of King Saud University - Computer and Information Sciences.

[8] Erbayrak, S., Özkır, V., & Mahir Yıldırım, U. (2021). Multi-objective 3D bin packing problem with load balance and product family concerns. Computers & Industrial Engineering, 159, 107518.

[9] Moon, I., & Nguyen, T. V. L. (2013). Container packing problem with balance constraints. OR Spectrum, 36(4), 837–878.

[10] Wang, H., & Yanjie Chen. (2010). A hybrid genetic algorithm for 3D bin packing problems. 2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA).

[11] C. Paquay, S. Limbourg, and M. Schyns.(2018). A tailored two-phase constructive heuristic for the three-dimensional multiple bin size bin packing problem with transportation constraints. European Journal of Operational Research, vol. 267, no. 1, pp. 52–64.

[12] S. Walke, M. Zambare, and K. Wanjale, "A Comparative Study: Cloud Computing Service Providers", IJRESM, vol. 5, no. 5, pp. 208–211, May 2022.

[13] T. A. Toffolo, E. Esprit, T. Wauters, and G. Vanden Berghe. (2017). A two-dimensional heuristic decomposition approach to a three-dimensional multiple container loading problem. European Journal of Operational Research, vol. 257, no. 2, pp. 526–538.

[14] H.Gehring, A.Ortfeldt (1998). A genetic algorithm for solving the container loading problem. International Transactions in Operational Research.