57º SBPO
SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
Gramado • RS
DE 5 A 9 DE OUTUBRO DE 2025

# Greedy Column Generation: A Novel Constructive Heuristic for 2D Knapsack and Strip Packing

**Felipe Silvestre Cardoso Roberto**
Universidade Federal de São Paulo
São José dos Campos/SP
felipe.silvestre@unifesp.br

**Victor Ulisses Pugliese**
Universidade Federal de São Paulo
São José dos Campos/SP
pugliese@unifesp.br

**Sanderson Lincohn Gonzaga de Oliveira**
Universidade Federal de São Paulo
São José dos Campos/SP
sanderson.oliveira@unifesp.br

**Oseias Faria Ferreira**
EMBRAER S.A.
São José dos Campos/SP
oseias.ferreira@embraer.com.br

**Fabio Augusto Faria**
Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
fabio.faria@tecnico.ulisboa.pt

## ABSTRACT

This paper addresses two-dimensional Cutting and Packing Problems, with a focus on the Knapsack Problem (KP) and Strip Packing Problem (SPP), both of which are essential for optimizing material utilization. We propose a constructive heuristic called Greedy Column Generation (GCG), designed to place irregular polygons efficiently. GCG constructs compact, adjacent columns using the Bottom-Left (BL) placement rule, guided by a convex hull (CH) metric for polygon selection and orientation. To assess its effectiveness, we compared GCG with five established heuristics on fifteen ESICUP benchmark instances, evaluating both packing efficiency and execution time. In the KP, GCG consistently outperformed the alternatives, particularly in more complex instances. For the SPP, it also delivered competitive results, although with a smaller performance margin. We further applied the Simulated Annealing (SA) metaheuristic to all methods, and GCG-SA maintained strong performance across both problems, suggesting that GCG provides stable and reliable results.

**KEYWORDS. 2D Knapsack. 2D Strip Packing. Constructive Heuristics. Metaheuristic. No-Fit Polygon.**

**Paper topics: Cutting and Packing, Constructive Heuristics, Industrial Applications.**

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

## 1. Introduction

Cutting and packing problems are fundamental to numerous industries (e.g., aerospace, automotive, metallurgy, textiles, logistics), as they involve optimizing the arrangement of polygons to minimize material waste or space usage. Each industry has specific constraints, leading to the development of several problem variants [Dyckhoff, 1990; Guo et al., 2022; Iori et al., 2022], which differ in dimensionality, objective functions, application contexts, and the regularity of polygon geometries. These distinctions are essential to address the unique requirements of each sector correctly.

We have two variants of these optimization problems: The 2D Knapsack problem, which involves a single finite bin, is typically used when it is not necessary to pack all polygons, and the goal is to maximize space utilization, and the "profit" can be directly associated with the area occupied by the polygons; and the Strip Packing Problem, which assumes bins with width much larger than height aiming to minimize the width used. These problems are classified as NP-hard [Garey and Johnson, 1979], making exact solution approaches feasible only for small instances.

Given the complexity of these problems, constructive heuristics provide a fast way to generate initial solutions by defining promising insertion orders and placement rules that determine how each polygon is positioned within the bin. These initial layouts can later be refined by metaheuristics or used as standalone strategies [Guo et al., 2022], potentially leading to more satisfactory solutions and speeding up the overall search process.

For instance, Valle et al. [2012] employed an extended search with a rectangular hull to initialize the GRASP metaheuristic, followed by a local search. Similarly, Elkeran [2013] applied three constructive heuristics based on the Bottom-Left (BL) placement rule, each prioritizing polygons by a different criterion: largest area, largest perimeter, or highest rectangularity, to initialize the Guided Cuckoo Search. While metaheuristics play a central role in producing competitive final solutions, the quality of their initial solutions—shaped by these constructive methods—is equally critical. This highlights the need for further research into more sophisticated constructive heuristics to generate high-quality starting points beyond basic strategies.

This paper introduces a novel heuristic, called Greedy Column Generation (GCG), which constructs solutions for the 2D Irregular Knapsack and Irregular Strip Packing Problems by forming compact, adjacent columns. The design is motivated by the observation that combining the Bottom-Left (BL) placement heuristic with convex hull metrics often leads to tightly packed columns. Based on this insight, the method decomposes the solution into a sequence of such columns, each built to maximize spatial efficiency.

## 2. Background

This section presents the main theoretical concepts necessary to contextualize the cutting and packing problems, as well as the algorithmic approaches commonly used to address them.

### 2.1. Cutting and packing problems

Cutting and packing problems are a class of combinatorial optimization problems that seek to minimize material waste or maximize space utilization. A typical formulation involves a cutting area $A$ and a set of polygons $I$, where the objective is to select and position a subset of $I$ to optimize the usage of $A$. Among the various problem variants, the one involving a single bin and polygons with arbitrary shapes is commonly referred to as the Knapsack Problem, in which the profit of each polygon corresponds to its area. This variant has given rise to others, such as the Strip Packing Problem (SPP). In contrast to the Knapsack Problem, the SPP entails packing all polygons into a bin with fixed width $W$ and theoretically infinite length, to minimize the total length occupied [Guo et al., 2022]. This problem arises across various industries, including textiles, metal fabrication, and composite manufacturing, and becomes especially complex when irregular

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

shapes are involved, requiring advanced geometric techniques such as no-fit polygons for efficient packing [Dodge et al., 2021; Burke et al., 2006] as shown in Figure 1 which compares two packing solutions for a set of patterns. Solution (a) is feasible but has wasted space. Solution (b) is optimal, demonstrating efficient packing.

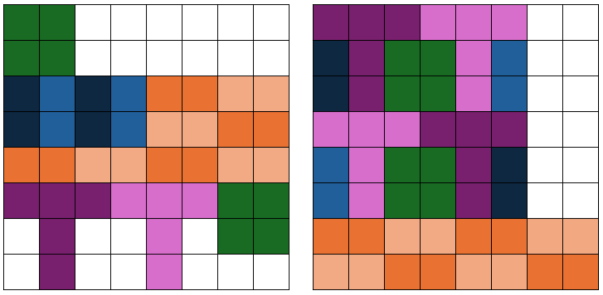



Figure 1: An optimal Cutting and Packing

### 2.2. Constructive Heuristic

The constructive heuristic defines promising insertion orders and placement rules that determine how each polygon is positioned within the bin.

- Bottom-Left (BL) aims to position geometric patterns at the lowest and leftmost available spot on the sheet without overlaps [Burke et al., 2006] . Elkeran [2013] applies three constructive heuristics based on this placement rule, each prioritizing polygons by a different criterion: largest area (LBL), largest perimeter (PBL), or highest rectangularity (RBL).

- Extended Search (ES) evaluates all feasible positions defined by the No-Fit Polygon and chooses the one that maximizes a spatial efficiency metric [Adamowicz and Albano, 1976].

### 2.3. Metaheuristic

For complex optimization problems that are computationally difficult to solve precisely, making exact methods impractical, metaheuristics offer powerful strategies to find solutions that are close to optimal. These methods differ from traditional heuristics, which are tailored to specific problems, by being problem-independent. Their design is frequently inspired by natural processes such as evolution, the behavior of swarms, or physical annealing [Bandaru and Deb, 2016].

#### 2.3.1. Simulated Annealing

Simulated Annealing (SA) is a metaheuristic inspired by the physical annealing process in metallurgy. It begins with an initial solution and iteratively explores the search space by applying small perturbations. At each step, new solutions are accepted based on a probabilistic criterion that occasionally permits transitions to solutions with worse objective values. This temporary allowance for worsening is essential for escaping local optima and exploring other regions of the search space. The probability of accepting such moves is governed by a temperature parameter, which gradually decreases over time, following a predefined cooling schedule. Higher temperatures facilitate broader exploration by permitting more diverse moves, while lower temperatures promote convergence towards regions of lower cost [Rutenbar, 1989].

### 2.4. Spatial Metrics

To measure how well a packing solution uses space, two common metrics are the rectangular hull (RH) and the convex hull (CH). The rectangular hull (or bounding box) is the smallest axis-aligned rectangle that covers all the polygons, and its compactness is calculated as the ratio between the total polygon area and the rectangle's area. The convex hull is the smallest convex shape that encloses all the polygons. It provides a tighter fit, better reflecting the layout's shape. Its compactness is also measured by the area ratio [Valle et al., 2012].

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

### 2.5. No-Fit Polygon (NFP)

The No-Fit Polygon is a geometric tool used to determine whether polygon $A$ overlaps with polygon $B$ in a specific position, ensuring compliance with the non-overlapping constraint in packing problems. The NFP is a polygon that represents all relative positions where polygon $A$ would touch or overlap polygon $B$. By analyzing the position of a reference point (e.g., a vertex) of $A$ relative to the NFP, it can be determined whether the polygons overlap (if the point is inside the NFP), touch (if the point is on the boundary), or remain separate (if the point is outside the NFP) [Burke et al., 2006]. This is typically computed via the Minkowski sum.

The direct computation of NFP$(A, B)$ is accurate only when both polygons are convex. For concave shapes, it may produce invalid boundaries and misclassify feasible placements as infeasible [Dean, 2002]. To address this, Burke et al. [2007] proposed several alternatives. One of them is to decompose $A$ and/or $B$ into convex parts and compute the NFP for each part, which is the approach adopted in this work.

## 3. Greedy Column Generation (GCG)

We propose a novel algorithm, named *Greedy Column Generation (GCG)*, to address the 2D Knapsack and strip packing problems with irregular instances. Despite its name, the GCG algorithm is unrelated to traditional Column Generation methods. Instead, it iteratively builds compact, high-occupancy columns by integrating the *BL* strategy with the convex hull metric, aiming to maximize area utilization. Figure 2 illustrates the algorithm's workflow, guiding the primary decisions in the solution construction process.

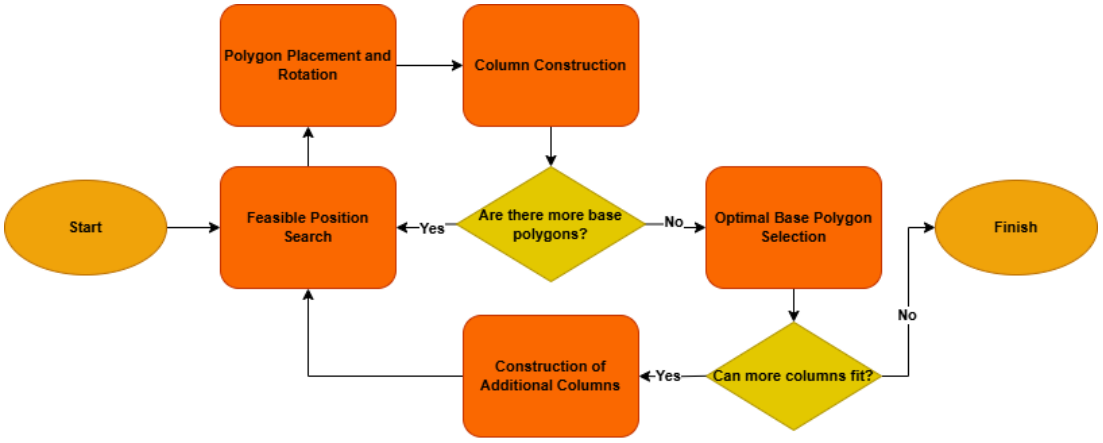



Figure 2: A Flowchart of the Greedy Column Generation.

1. **Feasible Position Search:** The *BL* algorithm identifies feasible positions for placing polygons within a specific range of the bin, defined by initial and final values. It searches for a valid position that does not intersect the *NFP* and returns the leftmost and lowest possible position.

2. **Polygon Placement and Rotation:** To place a single polygon within the column, the algorithm evaluates a list of possible rotations. For each angle, it applies the *BL* search to find a feasible position. Once the method identifies a feasible position, the algorithm calculates the convex hull to assess how efficiently the polygon utilizes the column's space and minimizes waste. After evaluating all rotations, the algorithm selects the orientation that maximizes occupancy and returns the corresponding convex hull efficiency. This function is referred to as `StockP` in Algorithm 1.

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

**Algorithm 1** StockP

```
1:  function StockP(S, W, H, polygon, rotations, column_start, column_end)
2:      best_polygon ← null
3:      best_position ← null
4:      best_convex_hull ← 0
5:      for each angle in rotations do
6:          polygon_rot ← Rot(polygon, angle)
7:          NFP ← NoFitPolygon(polygon_rot, S)
8:          (x, y) ← BL(polygon_rot, column_start, column_end, H, NFP)
9:          polygon_convex_hull ← CalculateCH(S, polygon_rot, x, y)
10:         if polygon_convex_hull > best_convex_hull then
11:             best_polygon ← polygon_rot
12:             best_position ← (x, y)
13:             best_convex_hull ← polygon_convex_hull
14:         end if
15:     end for
16:     if best_polygon is not null then
17:         return (best_polygon, best_position, best_convex_hull)
18:     else
19:         return (null, null)
20:     end if
21: end function
```

3. **Column Construction:** To construct a single column, the algorithm begins by selecting a base polygon onto which the remaining polygons will be stacked. Once the base polygon is chosen, the algorithm iterates through the remaining polygons, calculating the `StockP` value for each. The polygon that most efficiently occupies the column, based on the `StockP` metric, is selected and placed. This process repeats until no additional polygons can be stacked within the column. This procedure is encapsulated in the algorithm referred to as `BuildColumn` in Algorithm 2.

4. **Optimal Base Polygon Selection:** To determine the best base polygon—and consequently the most efficient column—the algorithm evaluates all polygons as potential bases. For each candidate base, it is passed as input to the `BuildColumn` function, which generates a column configuration. Each configuration is assessed using a metric that combines the occupied area and the convex hull area. The algorithm selects the column configuration with the highest metric value. This function, named `BestColumn` in Algorithm 3, outputs the most efficient column.

5. **Construction of Additional Columns:** After selecting the optimal column, the algorithm proceeds to analyze the remaining polygons that were not included in previous columns. Adjacent columns are constructed iteratively, and individually, using the same logic. The Algorithm 4 continues as long as there are unused polygons, and it is feasible to build new columns. When no more columns can be feasibly constructed, the algorithm attempts to pack the remaining polygons between the existing columns.

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

**Algorithm 2** Build Column

```
1:  function BUILDCOLUMN(W, H, base_polygon, polygons, rotations, column_start)
2:      column ← [base_polygon]
3:      column_end ← column_start + WIDTH(base_polygon)
4:      best_position ← not null
5:      while best_position is not null do
6:          best_position ← null
7:          best_polygon ← null
8:          best_convex_hull ← 0
9:          for each polygon in polygons do
10:             (rot_polygon, position, convex_hull) ←
11:                 STOCKP(column, W, H, polygon, rotations, column_start, column_end)
12:             if convex_hull > best_convex_hull then
13:                 best_polygon ← rot_polygon
14:                 best_position ← position
15:                 best_convex_hull ← convex_hull
16:             end if
17:         end for
18:         if best_polygon is not null then
19:             APPEND(column, best_polygon)
20:         end if
21:     end while
22:     return column
23: end function
```

**Algorithm 3** Best Column

```
1:  function BESTCOLUMN(S, N, W, H, rotations, column_start)
2:      best_column ← null
3:      best_metric ← 0
4:      for each polygon in N do
5:          column ← BUILDCOLUMN(W, H, polygon, N, rotations, column_start)
6:          metric ← CALCULATE_METRIC(column)
7:          if metric > best_metric then
8:              best_column ← column
9:              best_metric ← metric
10:         end if
11:     end for
12:     return best_column
13: end function
```

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

**Algorithm 4** Greedy Column Generation

```
1:  function GCG(N, W, H, rotations)
2:      S ← []
3:      column_start ← 0
4:      best_column ← not null
5:      while best_column is not null do
6:          best_column ← BESTCOLUMN(S, N, W, H, rotations, column_start)
7:          if best_column is not null then
8:              column_start ← column_start + WIDTH(best_column)
9:              APPEND(S, best_column)
10:         else
11:             break
12:         end if
13:     end while
14:     for polygon in RemainingPolygons(S, N) do
15:         STOCKP(S, W, H, polygon, rotations, 0, W)
16:     end for
17:     return S
18: end function
```

## 4. Experimental Methodology

This section describes the experimental protocol adopted in this work.

### 4.1. Baseline Heuristics

We employed five baseline algorithms to compare with our proposed GCG algorithm. The first baseline, $\mathcal{H}_1$, combines the BL placement rule with the convex hull metric. It serves as a simplified precursor to GCG, distinguished by not structuring the layout into columns. The second baseline, $\mathcal{H}_2$, is a deterministic adaptation of the semi-greedy method proposed by Valle et al. [2012]. Originally designed as the initial constructor for their GRASP algorithm, that method included a randomness factor $P$ for polygon selection during an extended search guided by rectangular hull metrics. In our implementation, we fix $P = 0$ to ensure entire determinism. The remaining baselines, $\mathcal{H}_3$, $\mathcal{H}_4$, and $\mathcal{H}_5$, are based on the initial solution strategies described by Elkeran [2013]. All three employ the BL placement rule, but differ in the heuristic used to prioritize the next polygon: $\mathcal{H}_3$ selects the polygon with the highest area; $\mathcal{H}_4$ prioritizes the polygon with the largest perimeter; and $\mathcal{H}_5$ chooses the polygon with the best bounding-box occupancy (i.e., the most rectangular).

### 4.2. Dataset

To evaluate the performance of GCG and the baseline heuristics, we selected 15 standard 2D irregular packing instances from the ESICUP (European Working Group on Cutting and Packing) repository. These instances are widely used in the literature as benchmarks and exhibit a variety of characteristics, including shape irregularity, number of polygon types ($m$), total number of polygons ($n$), bin dimensions (Height $\times$ Width), and allowed rotation angles.

For the KP, the bin area is defined by the fixed dimensions listed in the table. For the SPP, since the objective is to minimize the $Width$, the bin's $Height$ is fixed (as shown in the table), while the $Width$ is assumed to be theoretically infinite, allowing the heuristics to place all polygons freely in pursuit of minimizing the total width used.

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

Table 1: Characteristics of benchmark data instances.

| Instances | m | n | Height | Width | Rotations (°) |
|---|---|---|---|---|---|
| FU | 11 | 12 | 38 | 34 | {0, 90, 180, 270} |
| JACKOBS1 | 25 | 25 | 40 | 13 | {0, 90, 180, 270} |
| JACKOBS2 | 25 | 25 | 70 | 28.2 | {0, 90, 180, 270} |
| SHAPES0 | 4 | 43 | 40 | 63 | {0} |
| SHAPES1 | 4 | 43 | 40 | 59 | {0, 180} |
| SHAPES2 | 7 | 28 | 15 | 27.3 | {0, 180} |
| DIGHE1 | 16 | 16 | 100 | 138.14 | {0} |
| DIGHE2 | 10 | 10 | 100 | 134.05 | {0} |
| ALBANO | 8 | 24 | 4900 | 10122.63 | {0, 180} |
| DAGLI | 10 | 30 | 60 | 65.6 | {0, 180} |
| MAO | 9 | 20 | 2550 | 2058.6 | {0, 90, 180, 270} |
| MARQUES | 8 | 24 | 104 | 83.6 | {0, 90, 180, 270} |
| SHIRTS | 8 | 99 | 40 | 63.13 | {0, 180} |
| SWIM | 10 | 48 | 5752 | 6568 | {0, 180} |
| TROUSERS | 17 | 64 | 79 | 245.75 | {0, 180} |

Additionally, we employed the constructive heuristic to generate an initial solution for SA. We set the following hyperparameters. We set the initial temperature of 1.000, the cooling rate of 0.99, 30 iterations per Markov chain, and a runtime of 600 seconds. Then, we evaluated the performance of both approaches.

The experiments were implemented in Python 3.11, using the Shapely library for accurate geometric calculations and the Turtle library to visualize the packing process. Finally, this study performed the experiments on a system equipped with an Intel(R) Xeon(R) CPU E5-2630 and 64 GB of RAM.

### 4.3. Metrics

We measured performance using packing efficiency ($P$), defined as the ratio between the area of the packed polygons and the total bin area. In the case of the Knapsack problem, the bin area corresponds to the fixed dimensions of the bin. For strip packing, the bin area is computed as the product of the bin's fixed height and the total width used to accommodate all polygons:

$$t = \text{execution time (s)}, \quad P = \frac{\text{area of packed polygons}}{\text{bin area}} \times 100\%.$$

## 5. Experiments

This section performs a comparative analysis of the GCG and five different baseline heuristics to assess their effectiveness in solving 2D KP and SPP.

### 5.1. Knapsack Problem

Table 2 shows that GCG achieves the best $P$ in most experiments (10 out of 15, including both wins and ties), followed by the performance of $\mathcal{H}_3$ (6 out of 15). GCG performs well on complex instances such as MARQUES, ALBANO, and FU. This performance is mainly attributed to GCG's integration of BL placement with the CH metric, which promotes the formation of compact and adjacent vertical columns. This strategy is especially effective in the early stages of packing, although its efficiency tends to decrease in more unstructured layouts.

Heuristic $\mathcal{H}_1$, which also employs BL and CH, suffers from a similar limitation: because it does not explicitly organize polygons into columns, the CH metric quickly loses its guiding effectiveness after the initial placements. GCG, on the other hand, uses CH only while it remains beneficial, then transitions to constructing compact adjacent columns for continued efficiency.

DOI: 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

Although in some cases their performance is inferior to GCG, heuristics $\mathcal{H}_3$–$\mathcal{H}_5$, which rely on more elementary polygon selection rules without CH guidance, produce reasonable results.

GCG struggles in scenarios with significant size variation—such as in SHIRTS, where small polygons are packed too early and would be better used to fill gaps under larger polygons—or in instances with highly fragmented shapes, like DIGHE1 and DIGHE2, where the structure resists column-based organization. These cases resemble a ”broken glass” layout and demand a more flexible strategy. Additionally, runtime tends to increase with the number of distinct polygon types, as seen in JACKOBS1.

Table 2: The results of each heuristic in terms of area utilization $P$ (%) on the Knapsack Problem.

| Instance | GCG | $\mathcal{H}_1$ | $\mathcal{H}_2$ | $\mathcal{H}_3$ | $\mathcal{H}_4$ | $\mathcal{H}_5$ |
|---|---|---|---|---|---|---|
| FU | **83.82** | 77.32 | 77.32 | 75.15 | 80.03 | 65.94 |
| JACKOBS1 | **75.38** | **75.38** | 67.69 | **75.38** | **75.38** | 72.69 |
| JACKOBS2 | **68.44** | **68.44** | 66.28 | 67.55 | 66.89 | 63.62 |
| SHAPES0 | **57.30** | 50.63 | 50.63 | 54.92 | 56.98 | 51.27 |
| SHAPES1 | 52.37 | 57.46 | 50.68 | **58.47** | 55.93 | 54.75 |
| SHAPES2 | **71.85** | 69.14 | 64.44 | **71.85** | 71.11 | 65.56 |
| DIGHE1 | 53.82 | 60.20 | 65.67 | **71.07** | 64.31 | 58.44 |
| DIGHE2 | **64.71** | 57.14 | 49.81 | 59.40 | 59.83 | 62.88 |
| ALBANO | **82.25** | 73.89 | 68.21 | 74.71 | 73.79 | 67.65 |
| DAGLI | 68.72 | 70.67 | 70.67 | **73.90** | 69.49 | 70.91 |
| MAO | 65.00 | 65.63 | 65.00 | 65.12 | **68.37** | 58.39 |
| MARQUES | **82.74** | 72.15 | 72.15 | 71.13 | 74.31 | 72.15 |
| SHIRTS | 78.89 | 75.48 | 71.21 | **82.30** | 81.15 | 61.35 |
| SWIM | **62.87** | 59.50 | 59.25 | 60.86 | 57.38 | 59.78 |
| TROUSERS | **81.13** | 66.32 | 58.92 | 78.78 | 78.30 | 66.32 |
| **Wins/Losses/Ties** | **7/5/3** | 0/13/2 | 0/15/0 | 4/9/2 | 1/13/1 | 0/15/0 |

### 5.2. Strip Packing Problem

Table 3 shows that GCG achieved the best metric $P$ in 33.33% of the experiments (5 out of 15, including both wins and ties), followed by $\mathcal{H}_1$ and $\mathcal{H}_3$, but its advantage is less pronounced than in the Knapsack scenario. Because the strip allows unlimited width, GCG may occasionally generate poorly filled columns toward the end of the process, inadvertently increasing the total width. This effect is noticeable in instances such as ALBANO, although GCG still delivered the best overall result in that case.

As in the previous setting, GCG performs well when compact column formation is feasible. However its effectiveness declines in instances characterized by high size disparities in polygon sizes (e.g., SHIRTS) or fragmentation (e.g., DIGHE1, DIGHE2), which hinder stacking structure. The runtime also increases with the number of distinct polygon types (e.g., JACKOBS1, JACKOBS2), although it remains within practical limits.

In contrast, other heuristics—particularly $\mathcal{H}_1$, $\mathcal{H}_3$, and $\mathcal{H}_4$—while less structured, offer greater flexibility in the final stages of packing, often enabling tighter placement of the remaining pieces. In instances like FU and SHAPES2, where polygons are of similar dimensions, GCG's column-based strategy proved highly effective, resulting in superior space utilization.

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

Table 3: The results of each heuristic in terms of area utilization $P$ (%) on the Strip Packing Problem.

| Instance | GCG | $\mathcal{H}_1$ | $\mathcal{H}_2$ | $\mathcal{H}_3$ | $\mathcal{H}_4$ | $\mathcal{H}_5$ |
|---|---|---|---|---|---|---|
| FU | **83.82** | 75.00 | 72.52 | 67.86 | 75.00 | 75.00 |
| JACKOBS1 | 77.72 | **79.77** | 38.17 | 75.38 | 78.32 | 73.55 |
| JACKOBS2 | 67.01 | **69.62** | 57.85 | 68.44 | 68.44 | 64.85 |
| SHAPES0 | 51.82 | 50.51 | 43.85 | 55.03 | **57.83** | 51.82 |
| SHAPES1 | 57.00 | **58.68** | 45.29 | 55.03 | 57.83 | 45.29 |
| SHAPES2 | **74.48** | 66.06 | 62.79 | 71.76 | 72.97 | 65.85 |
| DIGHE1 | 46.30 | 64.94 | 54.35 | **68.81** | 60.48 | 55.25 |
| DIGHE2 | 56.54 | 57.26 | 49.75 | 63.78 | **68.47** | 61.17 |
| ALBANO | **74.70** | 73.64 | 71.69 | 73.99 | 68.60 | 69.37 |
| DAGLI | 61.85 | 70.34 | 62.61 | **76.26** | 70.05 | 69.66 |
| MAO | **68.62** | 66.04 | 38.09 | 64.14 | 61.88 | 60.96 |
| MARQUES | **77.72** | 72.06 | 41.17 | 73.59 | 69.17 | 67.82 |
| SHIRTS | 71.15 | 75.63 | 70.50 | 78.83 | **81.57** | 65.14 |
| SWIM | 56.83 | **62.77** | 33.78 | 60.17 | 58.15 | 56.95 |
| TROUSERS | 73.33 | 71.65 | 69.14 | **76.96** | **76.96** | 69.36 |
| **Wins/Losses/Ties** | **5/10/0** | 4/11/0 | 0/15/0 | 3/12/0 | 4/11/0 | 0/15/0 |

### 5.3. Metaheuristics

Table 4 shows that GCG-SA achieved the best results for both $P$ and $t$ in most knapsack experiments (8 out of 15), with execution time used as a tiebreaker. This result reinforces the strong performance previously observed in Table 2, where GCG also led in instances FU, SHAPES2, ALBANO, MARQUES, SWIN, and TROUSERS. In contrast, $\mathcal{H}3$ showed a decline in performance. $\mathcal{H}3$ outperformed it. $\mathcal{H}5$ achieved the best results in 3 out of 15 instances.

Table 4: Value utilization $P$ (%) and execution time $t$ (s) for each heuristic on the Knapsack Problem.

| Instance | GCG-SA | | H1-SA | | H2-SA | | H3-SA | | H4-SA | | H5-SA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ |
| fu | **83.82** | **3.98** | 83.82 | 5.50 | 83.82 | 589.86 | 83.82 | 29.32 | 83.82 | 202.27 | 83.82 | 17.18 |
| jackobs1 | 75.38 | 3.04 | 75.38 | 5.63 | 75.38 | 49.69 | 75.38 | 0.60 | 75.38 | 0.59 | **75.38** | **0.51** |
| jackobs2 | 68.44 | 2.61 | 68.44 | 2.70 | 68.44 | 50.07 | **68.44** | **0.51** | 68.44 | 0.52 | 68.44 | 0.55 |
| shapes0 | 57.30 | 15.55 | 59.05 | 516.20 | 59.05 | 511.32 | 59.05 | 581.42 | 59.05 | 349.27 | **59.05** | **300.32** |
| shapes1 | 57.80 | 508.77 | 61.86 | 52.20 | 60.85 | 135.00 | 60.85 | 152.57 | **62.20** | **543.14** | 61.86 | 194.92 |
| shapes2 | **77.28** | **409.50** | 75.80 | 442.93 | 73.46 | 580.32 | 74.57 | 543.32 | 74.57 | 456.12 | 75.80 | 498.12 |
| dighe1 | 68.61 | 430.56 | 72.4 | 183.32 | 72.4 | 494.92 | 72.4 | 483.59 | 72.4 | 28.89 | **72.4** | **20.07** |
| dighe2 | 67.06 | 5.09 | 74.60 | 50.00 | 74.60 | 255.99 | 74.60 | 21.79 | **74.60** | **17.74** | 74.60 | 96.10 |
| albano | **85.15** | **59.05** | 81.76 | 529.05 | 79.92 | 455.51 | 80.39 | 452.95 | 82.87 | 468.22 | 82.11 | 30.60 |
| dagli | 74.78 | 386.66 | **76.84** | **7.80** | 74.78 | 543.52 | 76.84 | 36.00 | 76.84 | 41.00 | 76.84 | 32.00 |
| mao | **71.60** | **48.78** | 71.60 | 67.25 | 71.60 | 64.61 | 71.60 | 57.35 | 71.60 | 83.43 | 71.60 | 51.23 |
| marques | **82.74** | **101.00** | 82.74 | 262.12 | 82.74 | 318.96 | 82.74 | 157.12 | 82.74 | 241.51 | 82.74 | 132.54 |
| shirts | **81.43** | **162.23** | 81.15 | 266.82 | 75.56 | 397.42 | 81.03 | 128.78 | 81.03 | 306.99 | 81.15 | 277.28 |
| swim | **66.23** | **386.60** | 65.65 | 322.35 | 62.46 | 543.52 | 65.65 | 350.74 | 65.65 | 534.12 | 65.65 | 234.12 |
| trousers | **83.51** | **476.12** | 81.13 | 320.34 | 76.15 | 276.32 | 82.84 | 298.23 | 82.84 | 129.21 | 81.13 | 435.45 |
| **W/L/T** | 8/7/0 | | 1/14/0 | | 0/15/0 | | 1/14/0 | | 2/13/0 | | 3/12/0 | |

Table 5 indicates that GCG-SA achieved the best performance in the strip packing experiments, ranking first in 5 out of 15 instances when considering both $P$ and $t$, with execution time serving as a tiebreaker. This result reinforces the strong performance previously observed in Table 2, where GCG also excelled in instances FU, SHAPES2, ALBANO, and MARQUES. In con-

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

trast, the other heuristics showed reduced effectiveness, except for $\mathcal{H}_5$, which demonstrated some improvement by leading in 3 out of 15 instances.

Table 5: Area utilization $P$ (%) and execution time $t$ (s) for each heuristic on the Strip Packing Problem.

| Instance | GCG-SA | | H1-SA | | H2-SA | | H3-SA | | H4-SA | | H5-SA | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ | $P$ | $t$ |
| fu | **86.36** | **41.27** | 83.82 | 63.69 | 77.45 | 118.33 | 86.36 | 232.90 | 86.36 | 259.48 | 86.36 | 231.15 |
| jackobs1 | 79.56 | 399.08 | 83.07 | 314.16 | 77.32 | 371.51 | **83.76** | **347.28** | 81.94 | 571.02 | 83.07 | 390.57 |
| jackobs2 | 69.03 | 582.27 | **75.86** | **209.56** | 70.54 | 312.70 | 74.35 | 426.16 | 73.89 | 245.80 | 73.22 | 92.45 |
| shapes0 | 52.50 | 34.11 | 58.68 | 162.21 | 52.85 | 195.53 | 58.25 | 35.04 | 59.11 | 309.06 | **59.55** | **436.47** |
| shapes1 | 56.20 | 1.54 | 60.45 | 440.67 | 58.50 | 252.18 | 60.36 | 326.21 | 60.45 | 560.31 | **62.34** | **108.12** |
| shapes2 | **76.33** | **354.16** | 75.26 | 360.79 | 69.90 | 289.19 | 75.26 | 272.63 | 76.06 | 496.26 | 76.06 | 473.24 |
| dighe1 | 65.79 | 474.54 | 73.89 | 113.49 | 70.75 | 240.65 | 74.60 | 3.87 | **76.53** | **394.69** | 73.17 | 294.20 |
| dighe2 | 61.17 | 14.97 | 78.55 | 246.24 | **81.41** | **179.90** | 79.11 | 437.24 | 79.11 | 204.03 | 79.11 | 50.77 |
| albano | **84.09** | **149.05** | 83.97 | 404.81 | 79.27 | 14.82 | 82.74 | 0.18 | 82.20 | 20.69 | 83.86 | 369.33 |
| dagli | 74.58 | 45.69 | **80.25** | **248.61** | 74.91 | 557.86 | 79.37 | 298.80 | 78.75 | 198.09 | 78.02 | 126.06 |
| mao | 72.47 | 88.60 | **78.24** | **485.90** | 71.41 | 635.21 | 77.74 | 277.65 | 76.53 | 137.29 | 77.50 | 274.90 |
| marques | **85.84** | **221.37** | 83.34 | 20.21 | 79.51 | 499.55 | 84.36 | 299.39 | 84.36 | 74.42 | 83.34 | 52.38 |
| shirts | **83.72** | **115.46** | 81.94 | 477.56 | 72.97 | 599.03 | 82.19 | 1.20 | 82.07 | 1.33 | 79.53 | 116.46 |
| swim | 60.83 | 487.31 | 66.30 | 536.27 | 60.30 | 510.86 | 66.86 | 26.85 | 65.59 | 172.13 | **68.01** | **253.00** |
| trousers | 82.82 | 104.11 | 81.57 | 319.05 | 76.69 | 475.18 | 83.77 | 475.69 | **85.41** | **394.10** | 80.97 | 160.44 |
| **W/L/T** | 5/10/0 | | 3/12/0 | | 1/14/0 | | 1/14/0 | | 2/13/0 | | 3/12/0 | |

## 6. Conclusions

This paper addressed two classic Cutting and Packing problems: the 2D Irregular Knapsack Problem and the 2D Irregular Strip Packing Problem. Both problems focus on minimizing material waste and enhancing layout efficiency. We introduced Greedy Column Generation (GCG) as a novel constructive heuristic designed to tackle these problems by building compact, adjacent, and high-occupancy columns. GCG achieves this by integrating the BL placement rule with convex hull metrics to maintain spatial coherence and effectively guide polygon placement, particularly in the early stages.

The experimental results showed that GCG outperformed the baseline heuristic in the Knapsack Problem, especially on more complex instances. In the Strip Packing Problem, GCG also performed well, although with a less pronounced advantage. GCG-SA maintained the performance across both problems, indicating that GCG delivers consistently stable results. In contrast, the other heuristics experienced a decline in performance when combined with SA.

A key distinction of GCG is its explicit column-based layout organization. Unlike simpler BL approaches where geometric guidance can become less effective, GCG deliberately builds compact columns, keeping the convex hull metric relevant and reducing layout fragmentation. However, the heuristic faces limitations with instances exhibiting significant size variation or highly fragmented shapes, which hinder structured column formation. Additionally, the SPP unlimited width can sometimes lead to less tightly packed columns towards the end, wich increases total width.

For future research, two promising directions may enhance the effectiveness of GCG-based approaches. One possibility is to reduce the deterministic nature of GCG by incorporating it into a metaheuristic framework that perturbs the solution during column construction. The idea is to make the column construction less deterministic, building different columns during the search process, which could be especially effective for the Knapsack Problem. In the context of the Strip Packing Problem, another approach would be to relax the constraint of non-overlapping columns,

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br

allowing partial overlaps during initial construction. These overlaps could then be penalized and resolved by a post-processing algorithm aimed at removing intersections while progressively reducing the bin width. This strategy could lead to more compact and efficient final layouts.

**Acknowledgements**

The authors thank the National Council for Scientific and Technological Development (CNPq) for the PIBIC scholarship of student F. Silvestre and the Doctoral Academic Industrial scholarship of V. Pugliese in partnership with EMBRAER.

## References

Adamowicz, M. and Albano, A. (1976). Nesting two-dimensional shapes in rectangular modules. *Computer-Aided Design*, 8:27–33.

Bandaru, S. and Deb, K. (2016). Metaheuristic techniques. In *Decision sciences*, p. 709–766. CRC Press.

Burke, E. K., Hellier, R. S. R., Kendall, G., and Whitwell, G. (2007). Complete and robust no-fit polygon generation for the irregular stock cutting problem. *European Journal of Operational Research*, 179(1): 27–49. Received 22 April 2005; accepted 8 March 2006.

Burke, E., Hellier, R., Kendall, G., and Whitwell, G. (2006). A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem. *Operations research*, 54(3):587–601.

Dean, H. T. (2002). *Minimizing Waste in the 2-Dimensional Cutting Stock Problem*. PhD thesis, University of Canterbury, Christchurch, New Zealand. Doctoral thesis, Department of Mechanical Engineering.

Dodge, M., MirHassani, S., and Hooshmand, F. (2021). Solving two-dimensional cutting stock problem via a dna computing algorithm. *Natural Computing*, 20:145–159.

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159.

Elkeran, A. (2013). A new approach for sheet nesting problem using guided cuckoo search and pairwise clustering. *European Journal of Operational Research*, 231(3):757–769.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. Mathematical Sciences Series. W. H. Freeman. ISBN 9780716710448.

Guo, B., Zhang, Y., Hu, J., Li, J., Wu, F., Peng, Q., and Zhang, Q. (2022). Two-dimensional irregular packing problems: A review. *Frontiers in Mechanical Engineering*, 8:1–15. URL `https://www.frontiersin.org/articles/10.3389/fmech.2022.966691/full`. Published 10 August 2022.

Iori, M., Loti de Lima, V., Martello, S., and Monaci, M. (2022). 2dpacklib: a two-dimensional cutting and packing library. *Optimization Letters*, 16:471–480.

Rutenbar, R. A. (1989). Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine*, 5(1):19–26.

Valle, A. M. D., de Queiroz, T. A., Miyazawa, F. K., and Xavier, E. C. (2012). Heuristics for two-dimensional knapsack and cutting stock problems with items of irregular shape. *Expert Systems with Applications*, 39 (4):3676–3686.

**DOI:** 10.59254/sbpo-2025-212113
https://proceedings.science/p/212113?lang=pt-br