applied sciences

MDPI

Article

GRASP Optimization for the Strip Packing Problem with Flags, Waste Functions, and an Improved Restricted Candidate List

Edgar Oviedo-Salas, Jesús David Terán-Villanueva *, Salvador Ibarra-Martínez *, Alejandro Santiago-Pineda, Mirna Patricia Ponce-Flores, Julio Laria-Menchaca, José Antonio Castán-Rocha and Mayra Guadalupe Treviño-Berrones

Facultad de Ingeniería "Arturo Narro Siller", Universidad Autonoma de Tamaulipas (UAT), Centro Universitario Tampico Madero, Tampico 89109, Mexico; eaos9407@gmail.com (E.O.-S.); aurelio.santiago@uat.edu.mx (A.S.-P.); mirna_poncef@hotmail.com (M.P.P.-F.); jlaria@docentes.uat.edu.mx (J.L.-M.); jacastan@docentes.uat.edu.mx (J.A.C.-R.); mgtrevino@docentes.uat.edu.mx (M.G.T.-B.)

* Correspondence: jdTeran@docentes.uat.edu.mx (J.D.T.-V.); sibarram@uat.edu.mx (S.I.-M.)

Abstract: This research addresses the two-dimensional strip packing problem to minimize the total strip height used, avoiding overlapping and placing objects outside the strip limits. This is an NP-hard optimization problem. We propose a greedy randomized adaptive search procedure (GRASP), incorporating flags as a new approach for this problem. These flags indicate available space after accommodating an object; they hold the available width and height for the following objects. We also propose three waste functions as surrogate objective functions for the GRASP candidate list and use and enhanced selection for the restricted candidate list, limiting the object options to better elements. Finally, we use overlapping functions to ensure that the object fits in the flag because there are some cases where a flag's width can be wrong due to new object placement. The tests showed that our proposal outperforms the most recent state-of-the-art metaheuristic. Additionally, we make comparisons against two exact algorithms and another metaheuristic.

Keywords: strip packing problem; NP-hard optimization; GRASP; object placing flag

Citation: Oviedo-Salas, E.; Terán-Villanueva, J.D.; Ibarra-Martínez, S.; Santiago-Pineda, A.; Ponce-Flores, M.P.; Laria-Menchaca, J.; Castán-Rocha, J.A.; Treviño-Berrones, M.G. GRASP Optimization for the Strip Packing Problem with Flags, Waste Functions, and an Improved Restricted Candidate List. *Appl. Sci.* **2022**, *12*, 1965. https://doi.org/10.3390/app12041965

Academic Editors: Antonio J. Nebro and José Manuel García-Nieto

Received: 12 January 2022
Accepted: 8 February 2022
Published: 14 February 2022

1. Introduction

The strip packing problem (SPP) is a two-dimensional industrial minimization problem. Given a strip of infinite length and bounded width, the problem is to define a packing of rectangular objects into a strip that minimizes its final length. The SPP is present in the industrial sector in material fabrics as paper, wood, glass, plastics, and metal, among others; its purpose is to reduce the amount of waste of the strip. Additionally, this problem is similar to the two-dimensional knapsack problem [1], where considered are the width and height of the objects in the knapsack. Furthermore, the variable cost and size of the bin packing problem [2] have some similarities regarding the variable size of the bin, which relates to the size of the strip. Nevertheless, there are also extensions to these problems, such as the three-dimensional bin packing problem [3].

The SPP is known as an NP-hard optimization problem [4]. Therefore, in real-world scenarios, the use of heuristic and metaheuristic algorithms is advised [5,6].

The greedy randomized adaptive search procedure (GRASP) is a constructive multi-start optimization algorithm with proven accuracy in NP-hard combinatorial optimization problems. In [7,8], the authors use a simple GRASP to solve the job–shop problem, minimizing the makespan while maximizing the workflow, and the orienteering problem with hotel selection, which look for a path with time limit and maximum score in a graph.

Other papers, such as [9], use a GRASP with a reduced candidate list to tackle the DNA sequence motif discovery.

Additionally, hybrid GRASP algorithms are common; e.g., in [10], Bruni et al. solve the k-traveling repairman problem with profits. In [11,12], Santiago et al. and Saad et al.

tackle the precedence-constraint tasks scheduling using a GRASP hybridized with a cellular processing algorithm and a simulated annealing, respectively. Khamlichi et al. used another hybrid GRASP to solve the manufacturing planning [13] for demand fluctuations. Hence, we believe that GRASP is simple and has high hybridizable properties, making room for improvements.

1.1. Literature That Allows Object Rotation

The following is the most relevant literature regarding the strip parking problem. However, this subsection only contains papers that allow rotations, not considered in our research.

In 2004, Beltran et al. [14] implemented a hybrid algorithm using GRASP and VNS algorithm to solve the strip packing problem and consider guillotine cuts.

Later, in 2013, Silveira et al. [15] proposed a GRASP heuristic and a first-fit decreasing height technique with two approximation algorithms for the strip packing problem and the two-dimensional loading capacitated vehicle routing problem.

Gaticia et al. [16] presented, in 2016, an algorithm called strip packing problem game (SPPG) that uses real players, patterns, decision trees, data mining, and heuristics. Their proposed algorithm implemented a puzzle representing a strip packing instance to obtain solutions. The algorithm analyzes the player's movements through data mining techniques and pattern recognition. Their experimental results showed that SPPG identifies game patterns from previous plays, obtaining adjustments of 87.03% for known instances and 88.2% for unknown instances.

In 2019, Chen et al. [17] presented a heuristic algorithm to the two-dimensional rectangular packing problem using anticipation strategy and step method. They aim to maximize the speed of the fill rate of the strip and improve adjustments in the construction procedure. The computational test concluded that their proposal is competitive against recent algorithms.

Finally, in 2020, Martin et al. approached the constrained two-dimensional guillotine placement problem (C2GPP) in [18]; this problem is similar to the one presented by Beltran et al. in [14], which considers guillotine cuts. Their objective is to use orthogonal cuts with constrained patterns to select an optimal set of large objects. The authors proposed a nonlinear integer function to obtain linear programming for nonlinear models and decision trees. The study concludes that models based on ascending storage lead to optimal or semi-optimal solutions with a reasonable computational cost.

1.2. Literature That Does Not Allow Object Rotation

Here, we show the most relevant literature regarding no object rotating strip parking problem.

In 2003, Martello et al. [19] implemented a branch and bound algorithm for the strip packing problem using instances up to two hundred objects. The computational results showed that they were able to solve 75% of the instances.

In 2008, Alvarez-Valdes et al. [20] implemented a reactive GRASP algorithm for the strip packing problem, improving their strategies and critical search options.

Later, in 2009, Alvarez-Valdes et al. [21] proposed a new branch and bound for the strip packing problem. They aim to reduce the tree search and generate better lower bounds.

Two years later, Leung et al. [22] presented a new approach named "two-stage intelligent search algorithm" for the strip packing problem. Their proposal consists of two stages; the scoring rule and the combination of simulated annealing algorithm with local search.

In 2016, Zhang et al. proposed an improved hybrid metaheuristic algorithm with variable neighborhood search that generates sets based on block pattern construction [23], called hybrid algorithm (HA), which has three phases. The first phase uses the least waste strategy, which consists of scoring rules to select the items that fit the least waste on the strip. The second phase selects a better sequence to improve the initial solution. Finally, the third

phase constructs different neighborhoods based on block patterns. Their computational tests show that the HA algorithm surpasses other approaches from state of the art for hard instances of the strip packing problem and conclude that the HA algorithm is efficient in selecting neighborhoods dynamically.

Later, in 2019, Wei et al. [24] analyzed the SPP with unloading constraints, which consist of storing objects in a two-dimensional space for transport and unloading. Their objective was to minimize the total height and satisfy the discharge condition using segment trees, heuristics for open spaces, and random local search.

A year later, Rakotonirainy et al. [25] proposed two metaheuristics for the strip packing problem. The first consisted of a simulated annealing combined with a heuristic construction algorithm. The second implemented a simulated annealing in predefined packing layouts without encoding solutions. The authors used 1718 instances, and they concluded that their proposal improved its original version in terms of the quality of the solution.

Table 1 shows the differences among the most relevant approaches, most of which will be used as a comparison.

Table 1. Differences between approaches.

Differences between Approaches	
GRASPSPP (Our proposal)	Reactive GRASP [20]
Simple GRASP	Use of reactive GRASP
Use of flags	Use of best fit (BF)
Overlap functions	Evaluate and rearrange the solution
Height and width measurement functions	Use of floating accommodations
Avoid floating objects	-
New Branch and Bound [21]	Iterative Search [24]
Use of Branch and Bound	Use of first-fit
GRASP for improvements	Arrangement in open space
Evaluate and rearrange the solution	Change solutions
Floating objects	Use of random local search
Waste space validation	Combination of 2D-SPP and CVRP
Branch and Bound [19]	Two-stage ISA [22]
Use of Branch and Bound	Use of search algorithm
Available space by two points	GRASP makes improvements
Arrangements by constraints	Accommodations in six dynamic spaces
Height and width evaluation	Width measurement
Branches reduction by selection	-

2. Strip Packing Problem (SPP)

The strip packing problem is an NP-hard optimization problem commonly found in the industrial sector in materials such as paper, fabrics, wood, glass, plastics, and metal. Their purpose is to reduce the amount of waste in a strip, avoiding overlapping and placing objects outside the strip limits. This problem is defined using the following sets:

$$O = \{o_1, o_2, \ldots, o_n\} \quad (1)$$

$$OW = \{ow_1, ow_2, \ldots, ow_n\} \quad (2)$$

$$OH = \{oh_1, oh_2, \ldots, oh_n\} \quad (3)$$

where O is the set of rectangular objects; OW is the set of object widths, and OH is the set of heights.

For every o_i their correspondent width and height are ow_i and oh_i respectively. The objective of the strip packing problem is to accommodate O into a delimited strip with a fixed width and infinite height, avoiding overlapping and aiming to reduce the total strip length used.

We implement a GRASP meta-heuristic algorithm to the strip packing problem, prioritizing the objects that generate less waste of area in the strip to minimize the height used.

One interesting element of our proposal is the use of flags. These flags are in the upper left and bottom right corners of an accommodated object. Additionally, they store the existing information in the strip, such as the widths and heights available for the next candidate.

On the other hand, we analyze the total height, waste area, the algorithm performance, and computational time. The impact of waste functions is also analyzed, and finally, functions that validate overlaps for objects are incorporated to determine if the current flag is disabled or updated.

3. GRASP Algorithm for SPP

The greedy randomized adaptive search procedure algorithm (GRASP) is a meta-heuristic proposed by Feo et al. [26,27]. The GRASP algorithm is a multi-start process that has two phases. The first phase corresponds to a heuristic construction, which provides a high-quality solution, and the second phase improves the solution through a local search. However, given the nature of the solution, a slight change in the placement of the rectangular objects for an already-completed solution would produce a huge problem for repairing highly-possible overlaps or would increase wastes. Therefore, we avoid the use of local search, limiting the algorithm to the heuristic construction phase.

Algorithm 1 shows the general procedure of our proposed optimization method; it generates the solutions with a defined number of iterations. The process starts with the *LoadInstance* function that loads the instance. The *InitializeParameters* function initializes the main parameters to use in GRASP construction. The *ReorderObjects* function consists of ordering the objects from the smallest area to the largest area. The *SPPGRASP* function constructs the GRASP solutions; we validate the solutions to obtain the best solution. This code is available at https://github.com/csalas07/GraspSpp (accessed on 4 February 2022).

Algorithm 1 GRASP General Procedure

1: $Sol = \varnothing, best = \emptyset$
2: $LoadInstance()$
3: $InitializeParameters()$
4: $ReorderObjects()$
5: **for** $i = 1 \rightarrow MaxIter$ **do**
6: $\quad Sol = SPPGRASP()$
7: $\quad$ **if** $f(Sol) < best$ **then**
8: $\quad\quad best = f(Sol)$
9: $\quad$ **end if**
10: **end for**

Output: best

3.1. GRASP Construction

One of the main contributions of this paper is the use of flags, and at the beginning of the algorithm, we set a single flag at the lowest left corner of the strip to mark the first place for setting an object.

Algorithm 2 describes the initial construction that consists of three parts. The first is to select an available flag index with the *FindFlagIndex* function (see line 4), which iterates a list of available flags and selects the index of the flag that has the lowest height, and if there were several flags with the same height, it chooses the one furthest to the left. The second part consist of the use of waste functions (see Equations (5)–(7)) to analyze the

objects' waste. Finally, the last part is constructing the candidate list (CL) and the restricted candidate list (RCL).

Algorithm 2 GRASP Construction Part 1

Input: k

1: $miss = |O|$
2: **while** $miss > 0$ **do**
3: $CL = \varnothing$
4: $j = FindFlagIndex(Flags)$
5: **for** $i = 0 \rightarrow |O|$ **do**
6: **if** $s_i = 0$ && $ow_i \leq tfw_j$ **then**
7: $CL = CL \cup \{o_i\}$
8: $wk_{ij} = CalculateWaste(k)$
9: $W = W \cup \{w_{ij}\}$
10: **end if**
11: **end for**
12: **if** $CL \neq \emptyset$ **then**
13: $kElems = max(|CL| \times 0.10, 5)$
14: $RCL = \varnothing$
15: $RW = \varnothing$
16: **for** $i = 1 \rightarrow kElem$ **do**
17: $imin = \arg\min_i wk_{ij} \forall wk_{ij} \in W$
18: $RCL = RCL \cup \{o_{imin}\}$
19: $RW = RW \cup \{wk_{imin}\}$
20: $W = W \setminus \{wk_{imin}\}$
21: **end for**
22: $Limit = max(W) + (\beta \times (max(W) - min(W)))$
23: **for** $i = 1 \rightarrow |RCL|$ **do**
24: **if** $wk_{ij} \in RW > Limit$ **then**
25: $RW = RW \setminus \{wk_{ij}\}$
26: $RCL = RCL \setminus \{o_i\}$
27: **end if**
28: **end for**
29: $O_r = Roulette(RCL, RW)$

The algorithm checks that the set value s_i is equal to zero, where $S = \{s_1, s_2, \ldots, s_n\}$ is a set that represents the availability of the objects (see Equation (4)).

$$s_i = \begin{cases} 1 & \text{The object is set in the strip} \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

The $CalculateWaste(k)$ function calculates the waste area of an object wk_{ij} (see line 8). We use three waste functions (see Equations (5)–(7)) to analyze the objects' waste. Each function validates different components among the rectangular objects and a specific flag.

$$w_{ij}^1 = min(\left|dfw_j - ow_i\right|, \left|tfw_j - ow_i\right|) + min(\left|lfh_j - oh_i\right|, \left|rfh_j - oh_i\right|) \tag{5}$$

$$w_{ij}^2 = min(\left|lfh_j - oh_i\right|, \left|rfh_j - oh_i\right|) \tag{6}$$

$$w_{ij}^3 = (min(\left|dfw_j - ow_i\right|, \left|tfw_j - ow_i\right|) \times max(oh, min(lfh_j, rfh_j)) + (min(\left|lfh_j - oh_i\right|, \left|rfh_j - oh_i\right|) \times ow_i) \tag{7}$$

where $O = \{o_1, o_2, \ldots, o_n\}$ is the set of objects and wk_{ij} is the waste of the object i on the flag j. Hence, in the algorithm, we calculate the waste for all the available objects o_i for the selected flag f_j. Figure 1 shows the characteristics of the flags and the objects, i.e., dfw_j and tfw_j are the flag's desirable and total width, respectively; lfh_j and rfh_j are the height to the left and to the right of the flag. Finally, SP is the starting point and ow_i and oh_i are the current object width and height, respectively.

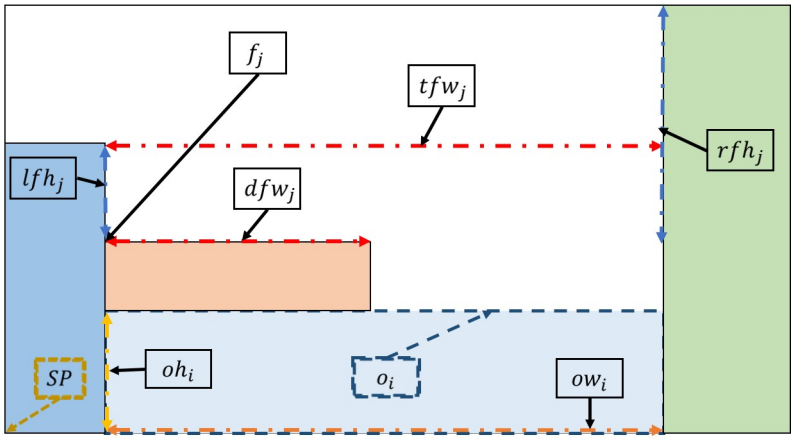

Figure 1. Characteristics of the flags and objects.

The selection of the first object is the one with the largest area. We select the rest of the elements according to the waste area function which is similar in some sense to the feature selection used in [28]; the WA value stores all waste area values for each object for a specific flag. Additionally, we validate that $CL \neq \emptyset$ in line 12. Therefore, if no object fits in the space of the flag, then we try to raise the flag (see line 35 of Algorithm 3).

Algorithm 3 GRASP Construction Part 2

```
30:         if NoObjectOverlapping() then
31:             FixFlagOverlapping()
32:             Set(O_r, j)
33:         end if
34:     else
35:         RiseFlag()
36:     end if
37: end while
```

Later, Equation (8) calculates a limit value, which will be compared with the waste areas of the candidate objects to create the restricted candidate list (RCL).

$$Limit = max(WA) + (\beta \times (max(WA) - min(WA))) \tag{8}$$

where β is a real number between 0 and 1; we made several tests to identify the best value, but we could not find it; hence, we chose to use random values between 0.4 and 0.5, which were the values with the best performance.. Additionally, the objects have a wide variety of sizes and, therefore, wastes, meaning that there might be lower or higher differences among the wastes. Hence, we could end up without candidates or with a lot of them, nullifying the purpose of the restricted candidate list.

Therefore, the candidate list selects $kElems$, which are either five elements or 10% of the objects in CL until exhaust CL (see line 13). As a result, the algorithm initializes the restricted candidate list with the elements in CL with the lowest waste.

Line 17 obtains the index of the object for which the waste wk_{ij} is minimal. Then, we update the restricted candidate list (RCL) and the restricted waste list (RW), respectively (see lines 18, and 19). Finally, the procedure removes wk_{imin} from W (see line 20).

The algorithm calculates the $Limit$ value to further restrict the RCL to those wastes that have a lower or equal value than $Limit$ (see line 22). Therefore, the algorithm updates RW and RCL in lines 25 and 26, respectively.

The procedure selects an object O_r in RCL randomly with a roulette technique [29] (see line 29); for further explanation see Section 3.2.

3.2. Roulette Procedure

Algorithm 4 describes the selection process using the roulette. The roulette assigns larger probabilities to the elements in RCL with lower waste.

Algorithm 4 General Roulette Procedure

Input: RCL, RW

1: $RW' = \{wk'_{ij} | wk'_{ij} = max(RW) + min(RW) - wk_{ij}, \forall wk_{ij} \in RW\}$
2: $t = \sum_{wk'_{ij} \in WK'} (wk'_{ij})$
3: $p = Random(1, t)$
4: **for** $\forall wk'_{ij} \in RW'$ **do**
5: $\quad p = p - wk'_{ij}$
6: $\quad$ **if** $p \leq 0$ **then**
7: $\quad\quad$ **return** $(o_i \in RCL)$
8: $\quad$ **end if**
9: **end for**

The procedure starts obtaining the maximum and minimum waste values from RW transforming high values of wk_{ij} to lower values of wk'_{ij} and vice versa (see line 1 of Algorithm 4). Additionally, t stores the total sum of RW' and p obtains a random value between 1 and t in lines 2 and 3, respectively.

Finally, the procedure iterates the converted values in RW' and subtracts wk'_{ij} from p; see lines 4 and 5. The process ends when the p value is less than or equal to zero and returns the selected object o_i in lines 6 and 7.

3.3. Object Arrangement

Once we have the selected object o_r, we try to accommodate it in f_j. However, first, we need to check for overlapping of the current object o_r regarding other objects in the strip. If there is any overlapping, then we update the data of the current flag and restart the selection of the object (see line 30). There is another possible overlapping, which occurs between the object o_r and another flag f_k; if this happens, we merge the information of f_j and f_k together and proceed to accommodate the object o_r in f_j (see lines 31 and 32, respectively).

Once o_r is placed in the strip, we create two new flags at the upper left and the lower right corners of the object (see Figure 2).

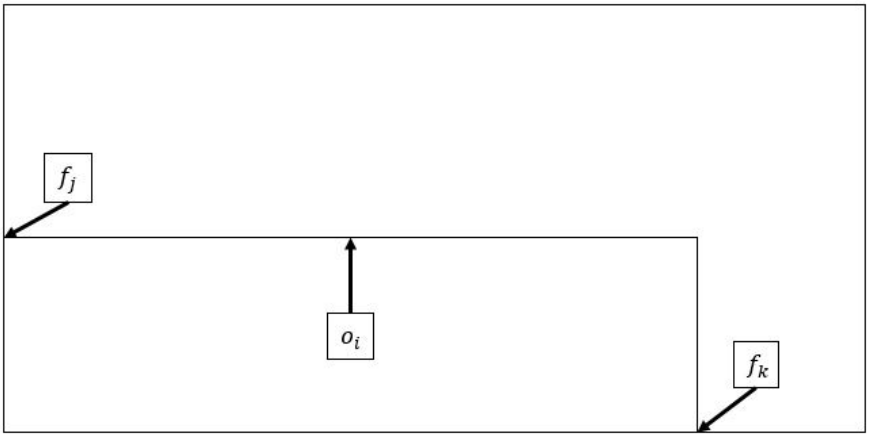

Figure 2. Flag placing; f_j and f_k are the new flags.

3.4. Overlapping among Objects

Figure 3 shows the idea of the overlaps detection among rectangles. The function validates that the current rectangular object does not cover an active rectangular object in the strip (see line 30 in Algorithm 3).

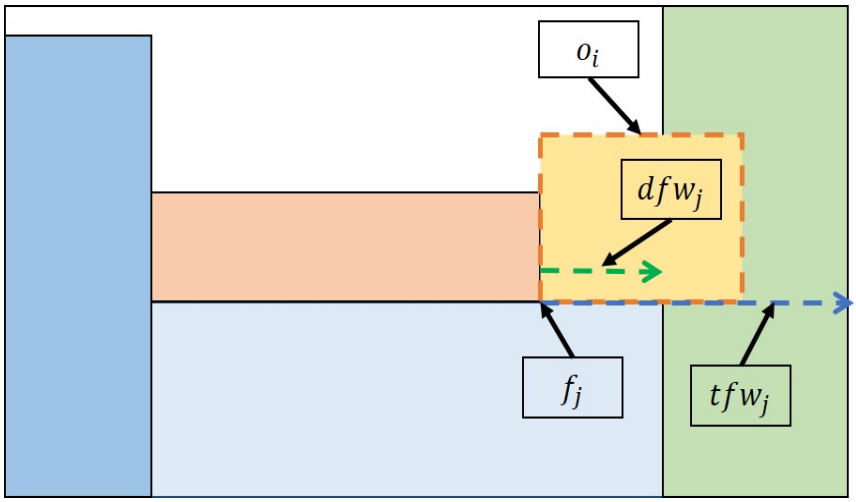

Figure 3. Overlaps between objects.

Here, when the algorithm created f_j, it also set tfw_j before the object to the right was there. Therefore, it contains wrong information. Hence, we check for collisions of o_i with the objects in the strip.

3.5. Overlaps between Rectangles and Flags

Figure 4 shows the overlaps between objects and flags. The function validates that the current rectangular object o_i does not cover the nearest flag space f_k. In case of overlapping, the process updates the current flag f_j, merging the information with f_k and finally deleting it.

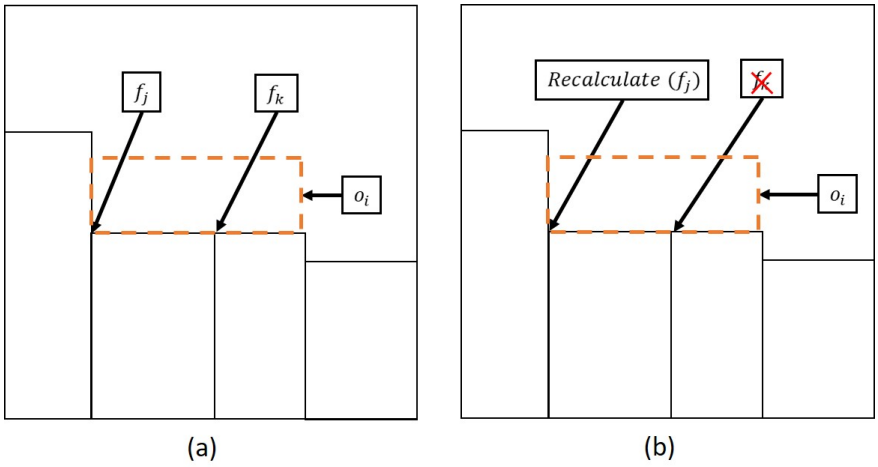

Figure 4. (**a**) Shows the overlap between an object and a flag. (**b**) The overlap fix.

3.6. Rise Flag

Figures 5 and 6 present the rise flag concept. The procedure expands the total width to the right and changes the height of the flag, allowing the algorithm to improve the management of the space, avoiding larger empty spaces (see line 35 in Algorithm 3).

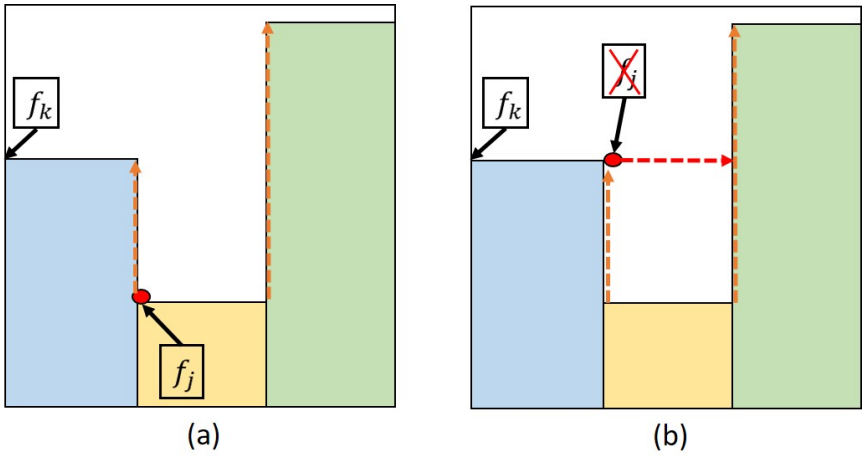

Figure 5. (**a**) Rise flag first case. (**b**) The flag fix.

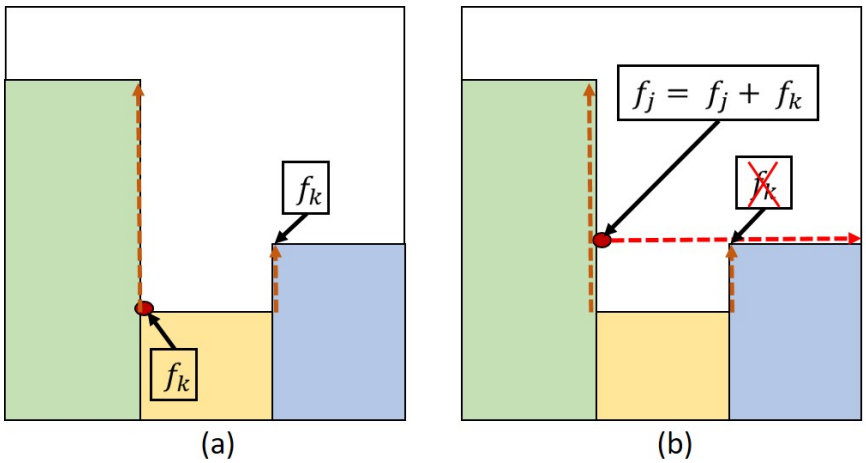

Figure 6. (**a**) Rise flag second case. (**b**) The flag fix.

The rise flag function changes the current flag information when no available object fits in the current flag. There are two possible cases for this function. The first case happens when the left height is lower than the right height; if this happens, then the flag is deleted. However, for the second case, if the right height is lower than the left height, we merge both flags $f_j = f_j + f_k$.

4. Experimental Results

This section presents the experimental results from the proposed optimization method called *GRASPSPP*, which includes a comparison among the waste functions and four state-of-the-art papers, using nonparametric tests.

4.1. Configuration and Instances

The algorithm runs computational tests on a computer with 2.50 GHz with 8 GB of RAM, Windows 7, and the Microsoft Visual Studio platform with C++ as the programming language.

Table 2 shows the data of the instances used, where the first column shows the instance name, the second column shows the author's name, and the third and fourth columns show the total of the instances and their sizes. The instances are available at https://mega.nz/folder/9slymA7b#kUSGqJGcfGszPDxF7a5sJg (accessed on 4 February 2022).

Table 2. Used instances.

Name	Autor	Total	n
2lcvrp	Gendreau [30]	180	15–255
Chr/cgcut	Christofides & Whitlock [31]	3	10–70
Brk/N1-13	Burke [32]	13	10–500
Ben/beng	Bengtsson [33]	10	20–200
Htu/ht	Hopper and Turton [34]	114	16–28
Hop	Hopper and Turton [35]	350	17–199
Bea/gcut-ngcut	Beasley [36,37]	25	10–22
Class 1–4	Martello and Vigo [19]	200	20–100
Class 5–10	Berkey and Wang [38]	300	20–100
50cx–15,000cx	Pinto and Oliveira [39]	6	50–15,000
P1	Ramesh Babu [40]	1	1000

4.2. Waste Comparison

In this section, we present the comparison among the three waste functions regarding quality and time. The waste functions detect the amount of wasted space of any object. For the computational test, the algorithm uses three waste functions. We evaluate the performance and efficiency to determine the best waste function using the Friedman and the Wilcoxon tests. The waste functions are w_{ij}^1, w_{ij}^2, and w_{ij}^3 (see Equations (5)–(7)).

Figure 7 describes the quality result obtained from the different waste area functions. This comparison indicates that waste function w_{ij}^1 obtains lower heights than w_{ij}^2, and w_{ij}^3. However, w_{ij}^3 obtains lower heights than w_{ij}^2.

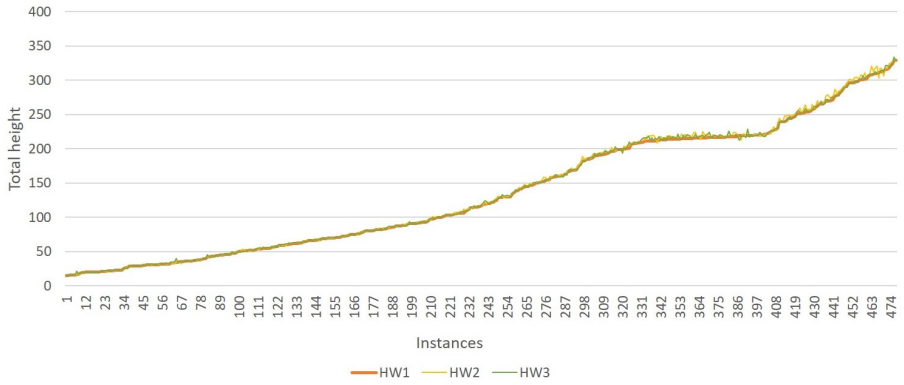

Figure 7. Waste performance comparison.

A Friedman test showed that w_{ij}^1 is the best-ranked waste function, followed by w_{ij}^3, and w_{ij}^2 in last place. Additionally, it shows that there is a statistically significant difference among them. Finally, the Wilcoxon test indicates that waste function w_{ij}^1 has statistically better quality than w_{ij}^3 and w_{ij}^2.

Regarding efficiency, most of the tests showed a computing time close to one second, which is not suitable for graphics. Here, the waste function w_{ij}^3 showed the lowest times. A Friedman test confirmed that the best ranked waste function is w_{ij}^3, followed by w_{ij}^1, and w_{ij}^2 in the last place. Finally, the Wilcoxon test indicated that there is no statistical difference between w_{ij}^1 and w_{ij}^3. However, there is a significant difference between w_{ij}^1 and w_{ij}^2. Therefore, w_{ij}^1 produces solutions with low computational times and the highest quality. On the other hand, w_{ij}^2 produces solutions with high computational times and low quality. Therefore, our proposed algorithm uses the waste function w_{ij}^1 for the rest of the computational test.

4.3. Comparison between GRASPSPP and Iterative Search Algorithm

In this section, we compare the results between the GRASPSPP algorithm and the iterative search (IS) algorithm presented in 2019 [24] by Wei et al. The authors performed 20 independent runs at 1000 iterations; thus, we configured our computational experimentation with the same runs and iterations.

Figure 8 shows the comparison of the average errors (*errAvg*) for the 20 runs for each instance set, for which we also calculated the average and standard deviation among all the sets of instances for the GRASPSPP (1.13 average, 0.06 standard deviation) and IS (1.12 average, 0.02 standard deviation). These *errAvg* are the average of the minimum error and the maximum error:

$$errAvg = \frac{minError + maxError}{2} \tag{9}$$

and the error is calculated as follows:

$$error = \frac{obtainedHeight}{bestHeight} \tag{10}$$

Additionally, this figure shows that GRASPSPP outperforms IS in six instances, and IS outperforms GRASPSPP in one instance. The Wilcoxon test result showed that GRASPSPP and IS produce statistically equivalent solutions with a significance value of 0.233.

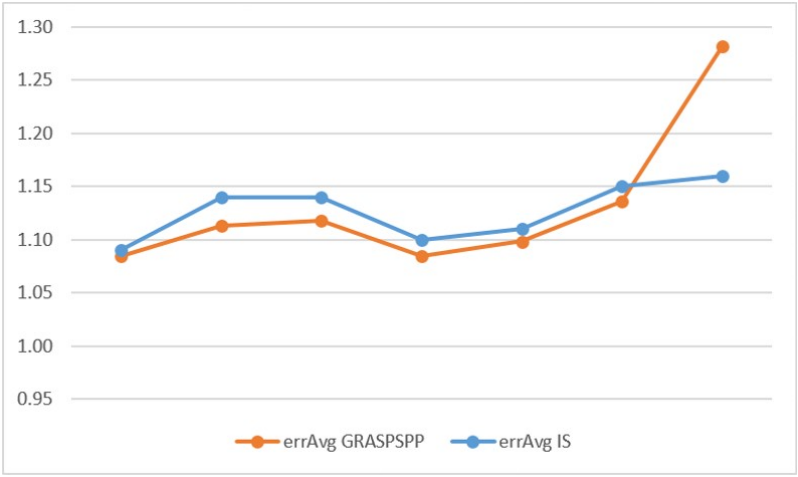

Figure 8. Average errors (errAvg) between GRASPSPP and IS.

Figure 9 shows the computational times between the GRASPSPP (2.70 average, 2.07 standard deviation) and IS (17.77 average, 9.51 standard deviation) algorithms. Here, the computational times are the average of the computational time of the 20 runs per set of instances. In all cases, the IS algorithm obtains higher computational times than GRASPSPP. Additionally, the Wilcoxon test result showed that GRASPSPP statistically outperforms IS with a 98.2% certainty.

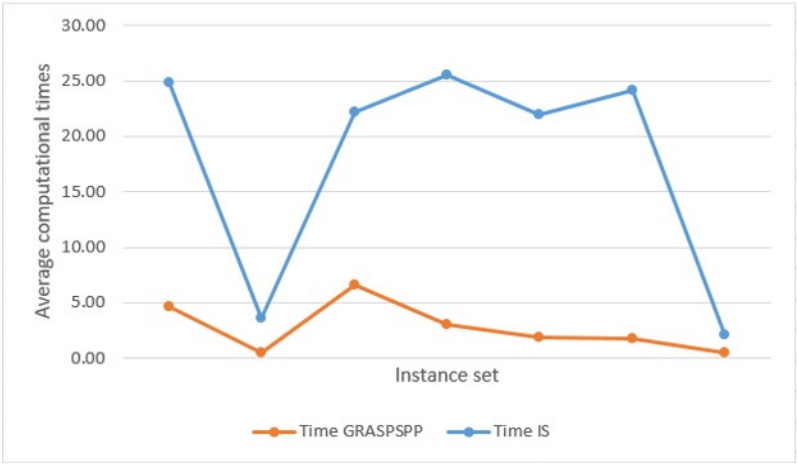

Figure 9. Time comparison between GRASPSPP and IS algorithm.

4.4. Comparison between GRASPSPP and Branch and Bound Algorithm

This section shows the comparison between GRASPSPP and a Branch and Bound algorithm (B&B) presented in 2003 [19] by Martello et al, where the authors executed their approach with a time limit of one hour and reported the time in which they found their best result. For this comparison, we executed the GRASPSPP algorithm with a time limit of 5 min in an attempt to produce an equivalent computational effort because our processor is an Intel Core i5 at 2.50 Ghz, which outperforms the authors' Pentium III at 800 Mhz processor.

Figure 10 shows the percentage error (see Equation (11)) comparison between GRASP-SPP (2.62 average, 7.17 standard deviation) and B&B (0.97 average, 1.79 standard deviation). We obtain the *%error* as follows:

$$\%error = (100 \times (obtainedHeight - bestHeight))/bestHeight \quad (11)$$

Here, GRASPSPP outperformed B&B in eight instances; while B&B outperformed GRASPSPP in 11 instances. Among those 11 instances, three of them presented an extremely

high *%error* in GRASPSPP; therefore, we require further analysis of the properties of those instances to identify the cause and implement corrections; those instances were ht4, ht7, and ht9. Additionally, the Wilcoxon test showed that the GRASPSPP and B&B are statistically equivalent.

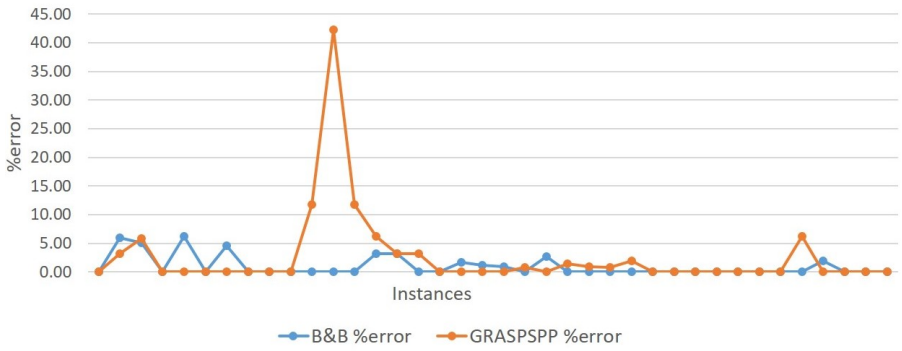

Figure 10. Percentage error (%error) results between GRASPSPP and B&B.

Figure 11 shows the computational time comparison between GRASPSPP (0.000 average, 0.001 standard deviation) and B&B (1116.30 average, 1632.09 standard deviation). These values are the computational times of each instance set. Here, the B&B algorithm presents some instances with the highest possible values, because they reached the time limit of 3600 s. On the other hand, the proposal processes the instances with times closer to one second.

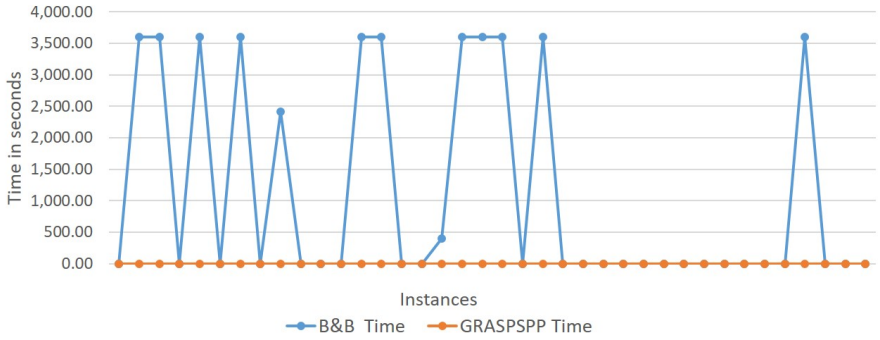

Figure 11. Time comparison between GRASPSPP and B&B algorithm.

A visual comparison shows that the GRASPSPP algorithm is faster than B&B. Additionally, the Wilcoxon test corroborates that the GRASPSPP outperforms B&B statistically with a *p*-value of 0.001.

4.5. Comparison beteween GRASPSPP and a New Branch & Bound Algorithm

This section compares the experimental results between GRASPSPP and a new branch and bound algorithm (NB&B) proposed by Alvarez et al. in 2009 [21], where the authors executed their approach with a time limit of 1200 s. For this comparison, we executed our GRASPSPP algorithm with a time limit of 150 s to produce equivalent results because our processor is an Intel Core i5 at 2.50 Ghz, which outperforms the authors' Pentium 4 at 2 Ghz processor.

Figure 12 shows the percentage error comparison (see Equation (11)) values between GRASPSPP (6.43 average, 7.13 standard deviation) and NB&B (1.37 average, 1.72 standard deviation) algorithm. Here, we can see that the NB&B algorithm outperforms GRASPSPP in most cases. However, GRASPSPP achieved a tie in 18 instances and won over NB&B in five instances. Finally, the Wilcoxon test showed that the NB&B statistically outperforms GRASPSPP with a *p*-value of 0.000.

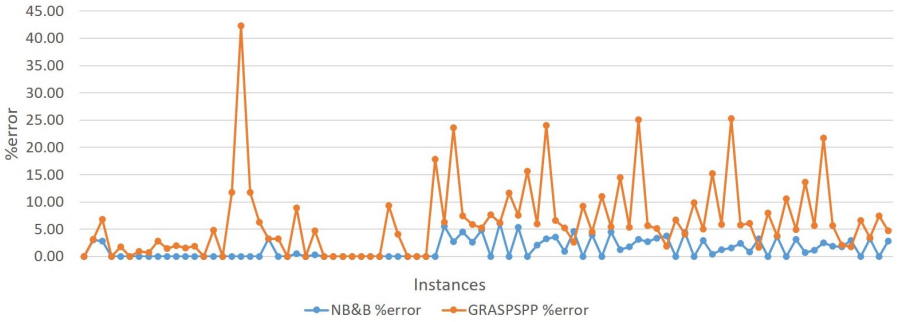

Figure 12. GAP comparison between GRASPSPP and NB&B algorithm.

Figure 13 shows the computational time comparison between GRASPSPP (0.42 average, 1.67 standard deviation) and NB&B (19.33 average, 51.20 standard deviation) algorithm. These values are the computational times of each instance set. Here, the NB&B algorithm reached the time limit of 1200 s, or even exceeded it, for five instances. On the other hand, GRASPSPP produced times near 1 s for most cases. Finally, the Wilcoxon test shows that the GRASPSPP algorithm outperforms NB&B with a *p*-value of 0.000.

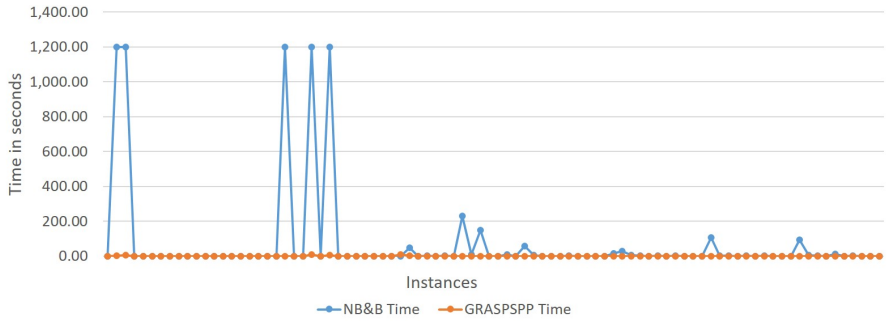

Figure 13. Time comparison between GRASPSPP and NB&B.

4.6. Comparison between GRASPSPP and Reactive GRASP Algorithm

This section shows the comparison results between the GRASPSPP and reactive GRASP algorithm proposed by Alvarez et al. in 2008 [20]. The comparisons only have the best height and average values because the author prioritizes the results of the instances and not the computational times. Therefore, we cannot guarantee a fair equivalence regarding quality and efficiency; hence, we included these comparisons merely as a complement using a time limit of 60 s.

Figure 14 shows the percentage error (see Equation (11)) comparison between GRASPSPP (0.03 average, 0.01 standard deviation) and reactive GRASP (0.84 average, 0.57 standard deviation) algorithm from instance C1–C7 from the set Htu/ht (see Table 2. The reactive GRASP produces, in general, high average errors. The Wilcoxon test shows that the GRASPSPP algorithm outperforms statistically the reactive GRASP with a *p*-value of 0.046.

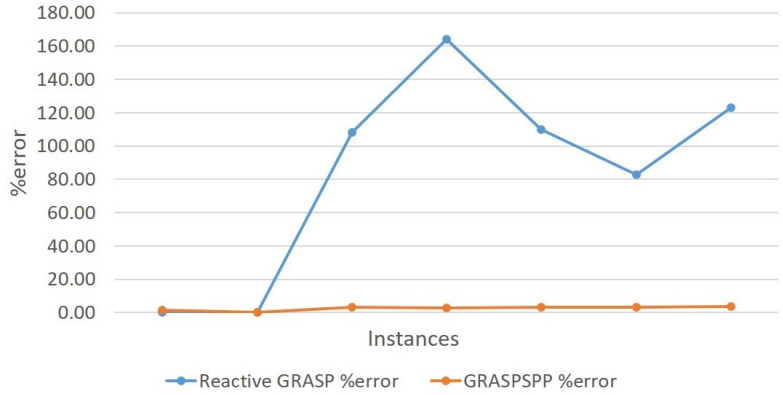

Figure 14. Average error between GRASPSPP and reactive GRASP.

Figures 15 and 16 shows the comparison of best values between the GRASPSPP (958.94 average, 1847.64 standard deviation) and reactive GRASP (948.59 average, 1786.24 standard deviation) algorithms. It is important to highlight that the author does not provide the rest of the average errors for these instances. Instead, they choose to use the best value, and, according to Figure 14, we believe that their algorithm produces solutions with a high variance.

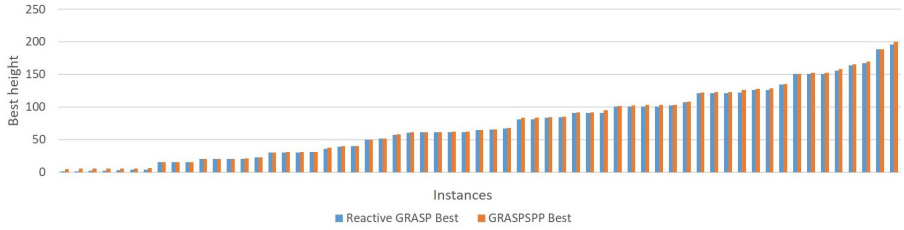

Figure 15. Best values comparison between GRASPSPP and reactive GRASP, part 1.

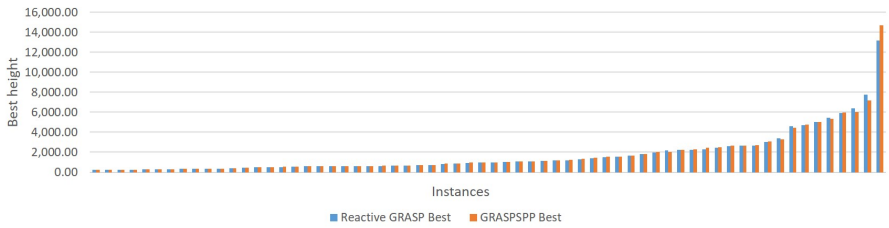

Figure 16. Best values comparison between GRASPSPP and reactive GRASP, part 2.

As we can see, the differences between both algorithms are minimal except for a handful of instances.

The Wilcoxon test shows that the reactive GRASP has statistically better performance than GRASPSPP with a p-value of 0.000.

5. Conclusions

In this paper, we tackle the strip packing problem, which is NP-hard [4]. For this problem, we proposed a GRASP algorithm with flags, three waste functions, and an improved selection of the restricted candidate list (RCL). Additionally, we carried out an extensive experimentation with state-of-the-art algorithms and datasets.

The GRASP algorithm processes the instance with a new approach through flags, which indicate ideal available spaces for new objects. However, there are some cases where old flags keep outdated information; therefore, we check for collisions for every object.

Additionally, we proposed three waste functions that help to identify the possible wasted space for each object in the candidate list. These waste functions consider different

measures. Among them, the best waste function is w_{ij}^1, which considers the minimum distance of the object width regarding the flag desired and total width, plus the minimum distance of the object height regarding the left and right flag height. The other two waste functions consider the vertical waste and the total area waste.

Regarding the improved selection of the restricted candidate list, we noticed that using the limit in the restricted candidate list was not enough to prevent the algorithm from using bad objects for any specific flag. Therefore, we selected a subset of the candidate list with the best waste to enhance the selection process of the restricted candidate list. This improvement was carried out and tested before the rest of the tests, and it significantly enhanced the performance of GRASPSPP.

The computational tests showed that GRASPSPP outperformed the most recent state of the art metaheuristic (IS) regarding quality and efficiency. However, GRASPSPP outperformed the two exact B&B algorithms regarding efficiency, while achieving equivalent performance regarding quality with the first exact algorithm (B&B). Finally, the last metaheuristic (reactive GRASP) outperformed GRASPSPP regarding quality; however, the authors did not report computational time. Nevertheless, GRASPSPP outperformed this metaheuristic regarding average errors.

As future work, we will consider analyzing the characteristics of some instances that produced a high percentage of error for further improvement. Additionally, we consider that an object permutation can code this problem, placing the objects according to the flag at the bottom and further to the left.

Author Contributions: Conceptualization, J.D.T.-V., M.P.P.-F. and S.I.-M.; methodology, J.D.T.-V. and S.I.-M.; software, E.O.-S.; validation, S.I.-M., A.S.-P., M.P.P.-F. and J.L.-M.; formal analysis, A.S.-P. and E.O.-S.; investigation, E.O.-S. and J.D.T.-V.; writing—original draft preparation, E.O.-S., M.G.T.-B. and J.A.C.-R.; writing—review and editing, J.D.T.-V. and E.O.-S.; visualization, J.A.C.-R.; supervision, S.I.-M., M.P.P.-F. and J.L.-M.; project administration, S.I.-M.; funding acquisition, S.I.-M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Universidad Autonoma de Tamaulipas.

Institutional Review Board Statement: No applicable.

Informed Consent Statement: No applicable.

Data Availability Statement: No applicable.

Acknowledgments: We thank Consejo Nacional de Ciencia y Tecnología (Conacyt) for supporting the project with number: 782021.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Egeblad, J.; Pisinger, D. Heuristic approaches for the two- and three-dimensional knapsack packing problem. *Comput. Oper. Res.* **2009**, *36*, 1026–1049. [CrossRef]
2. Fadda, E.; Fedorov, S.; Perboli, G.; Barbosa, I.D.C. Mixing machine learning and optimization for the tactical capacity planning in last-mile delivery. In Proceedings of the 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, Madrid, Spain, 12–16 July 2021; Institute of Electrical and Electronics Engineers Inc.: Piscataway, NJ, USA, 2021; pp. 1291–1296. [CrossRef]
3. Martello, S.; Pisinger, D.; Vigo, D. The Three-Dimensional Bin Packing Problem. *Oper. Res.* **2000**, *48*, 256–267. [CrossRef]
4. Baker, B.S.; Coffman, E.G., Jr.; Rivest, R.L. Orthogonal Packings in Two Dimensions. *SIAM J. Comput.* **1980**, *9*, 846–855. [CrossRef]
5. Routledge. *Optimization in Industry: Volume 2, Industrial Applications*, 2nd ed.; Routledge: Philadelphia, PA, USA, 2017; p. 270.
6. Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549. [CrossRef]
7. Rajkumar, M.; Asokan, P.; Anilkumar, N.; Page, T. A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. *Int. J. Prod. Res.* **2011**, *49*, 2409–2423. [CrossRef]
8. Sohrabi, S.; Ziarati, K.; Keshtkaran, M. A Greedy Randomized Adaptive Search Procedure for the Orienteering Problem with Hotel Selection. *Eur. J. Oper. Res.* **2020**, *283*, 426–440. [CrossRef]

9. Gokalp, O. DNA Sequence Motif Discovery Using Greedy Construction Algorithm Based Techniques. In Proceedings of the 2020 5th International Conference on Computer Science and Engineering (UBMK), Diyarbakir, Turkey, 9–11 September 2020; pp. 176–180. [CrossRef]
10. Bruni, M.; Beraldi, P.; Khodaparasti, S. A hybrid reactive GRASP heuristic for the risk-averse k-traveling repairman problem with profits. *Comput. Oper. Res.* **2020**, *115*, 104854. [CrossRef]
11. Santiago, A.; Terán-Villanueva, J.D.; Martínez, S.I.; Rocha, J.A.C.; Menchaca, J.L.; Berrones, M.G.T.; Ponce-Flores, M. GRASP and Iterated Local Search-Based Cellular Processing algorithm for Precedence-Constraint Task List Scheduling on Heterogeneous Systems. *Appl. Sci.* **2020**, *10*, 7500. [CrossRef]
12. Saad, A.; Kafafy, A.; El Raouf, O.A.; El-Hefnawy, N. A GRASP-Simulated Annealing approach applied to solve Multi-Processor Task Scheduling problems. In Proceedings of the 2019 14th International Conference on Computer Engineering and Systems (ICCES), Cairo, Egypt, 17–18 December 2019; pp. 310–315. [CrossRef]
13. Khamlichi, H.; Oufaska, K.; Zouadi, T.; Dkiouak, R. A Hybrid GRASP Algorithm for an Integrated Production Planning and a Group Layout Design in a Dynamic Cellular Manufacturing System. *IEEE Access* **2020**, *8*, 162809–162818. [CrossRef]
14. Beltran, J.D.; Calderon, J.E.; Cabrera, R.J.; Perez, J.A.M.; Moreno-Vega, J.M. GRASP/VNS hybrid for the strip packing problem. In Proceedings of the First International Workshop on Hybrid Meta-Heuristics (HM04), Valencia, Spain, 22–23 August 2004.
15. Da Silveira, J.L.; Miyazawa, F.K.; Xavier, E.C. Heuristics for the strip packing problem with unloading constraints. *Comput. Oper. Res.* **2013**, *40*, 991–1003. [CrossRef]
16. Gaticia, G.; Reyes, P.; Contreras-Bolton, C.; Linfati, R.; Escobar, J.W. Un algoritmo para el Strip Packing Problem obtenido mediante la extracción de habilidades de expertos usando minería de datos. *Ing. Investig. Tecnol.* **2016**, *17*, 179–190. [CrossRef]
17. Chen, M.; Wu, C.; Tang, X.; Peng, X.; Zeng, Z.; Liu, S. An efficient deterministic heuristic algorithm for the rectangular packing problem. *Comput. Ind. Eng.* **2019**, *137*, 106097. [CrossRef]
18. Martin, M.; Morabito, R.; Munari, P. A bottom-up packing approach for modeling the constrained two-dimensional guillotine placement problem. *Comput. Oper. Res.* **2020**, *115*, 104851. [CrossRef]
19. Martello, S.; Monaci, M.; Vigo, D. An Exact Approach to the Strip-Packing Problem. *Informs J. Comput.* **2003**, *15*, 310–319. [CrossRef]
20. Alvarez-Valdes, R.; Parreño, F.; Tamarit, J.M. Reactive GRASP for the strip-packing problem. *Comput. Oper. Res.* **2008**, *35*, 1065–1083. [CrossRef]
21. Alvarez-Valdes, R.; Parreño, F.; Tamarit, J.M. A branch and bound algorithm for the strip packing problem. *OR Spectr.* **2009**, *31*, 431–459. [CrossRef]
22. Leung, S.C.; Zhang, D.; Sim, K.M. A two-stage intelligent search algorithm for the two-dimensional strip packing problem. *Eur. J. Oper. Res.* **2011**, *215*, 57–69. [CrossRef]
23. Zhang, D.; Che, Y.; Ye, F.; Si, Y.W.; Leung, S.C. A hybrid algorithm based on variable neighbourhood for the strip packing problem. *J. Comb. Optim.* **2016**, *32*, 513–530. [CrossRef]
24. Wei, L.; Wang, Y.; Cheng, H.; Huang, J. An open space based heuristic for the 2D strip packing problem with unloading constraints. *Appl. Math. Model.* **2019**, *70*, 67–81. [CrossRef]
25. Rakotonirainy, R.G.; van Vuuren, J.H. Improved metaheuristics for the two-dimensional strip packing problem. *Appl. Soft Comput. J.* **2020**, *92*, 106268. [CrossRef]
26. Feo, T.A.; Resende, M.G. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.* **1989**, *8*, 67–71. [CrossRef]
27. Feo, T.A.; Resende, M.G. Greedy Randomized Adaptive Search Procedures. *J. Glob. Optim.* **1995**, *6*, 109–133. [CrossRef]
28. An, N.T.; Dong, P.D.; Qin, X. Robust feature selection via nonconvex sparsity-based methods. *J. Nonlinear Var. Anal.* **2021**, *5*, 59–77. [CrossRef]
29. Gendreau, M.; Potvin, J.Y. *Integrated Methods for Optimization*, 2nd ed.; Springer: New York, NY, USA, 2012; Volume 146, p. 648. [CrossRef]
30. Gendreau, M.; Iori, M.; Laporte, G.; Martello, S. A Tabu Search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Networks* **2008**, *51*, 4–18. [CrossRef]
31. Christofides, N.; Whitlock, C. An Algorithm for Two-Dimensional Cutting Problems. *Oper. Res.* **1977**, *25*, 30–44. [CrossRef]
32. Burke, E.K.; Kendall, G.; Whitwell, G. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Oper. Res.* **2004**, *52*, 655–671. [CrossRef]
33. Bengtsson, B.E. Packing rectangular pieces—A heuristic approach. *Comput. J.* **1982**, *25*, 353–357. [CrossRef]
34. Hopper, E.; Turton, B.C. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artif. Intell. Rev.* **2001**, *16*, 257–300. [CrossRef]
35. Hopper, E.; Turton, B.C.H. Problem Generators for Rectangular packing problems. *Stud. Inform. Univ.* **2002**, *2*, 123–136.
36. Beasley, J.E. An Exact Two-Dimensional Non-Guillotine Cutting Tree Search Procedure. *Oper. Res.* **1985**, *33*, 49–64. [CrossRef]
37. Beasley, J.E. Algorithms for unconstrained two-dimensional guillotine cutting. *J. Oper. Res. Soc.* **1985**, *36*, 297–306. [CrossRef]
38. Berkey, J.O.; Wang, P.Y. Two-Dimensional Finite Bin-Packing Algorithms. *J. Oper. Res. Soc.* **1987**, *38*, 423–429. [CrossRef]

39. Ferreira, E.; Oliveira, J. Algorithm based on graphs for the non-guillotinable two-dimensional packing problem. In Proceedings of the Second ESICUP Meeting, Southampton, UK, 1 January 2005.
40. Babu, A.R.; Babu, N.R. Effective nesting of rectangular parts in multiple rectangular sheets using genetic and heuristic algorithms. *Int. J. Prod. Res.* **1999**, *37*, 1625–1643. [CrossRef]