IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 21, 2019, accepted November 7, 2019, date of publication November 14, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2953531

Hierarchical Search-Embedded Hybrid Heuristic Algorithm for Two-Dimensional Strip Packing Problem

MENGFAN CHEN[1], KAI LI[1], DEFU ZHANG[1], LING ZHENG[1], AND XIN FU[2]
[1]School of Informatics, Xiamen University, Xiamen 361005, China
[2]School of Management, Xiamen University, Xiamen 361005, China

Corresponding authors: Defu Zhang (dfzhang@xmu.edu.cn) and Ling Zheng (liz5@xmu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61672439.

ABSTRACT Two dimensional strip packing problem implies a sort of NP-hard combinational optimization, where a set of rectangles are defined all for filling a strip with finite width and infinite height being minimized. Optimization taking no account of rotation and guillotine constraints remains problematic in solving the problem. This paper proposes a new variant of a hybrid heuristic algorithm for optimizing solutions of the strip packing problem. This algorithm involves three key improvements: the building of δ scoring rules for selecting rectangles, the use of the red-black trees that stores rectangle indices for quickly locating the most suitable rectangles, and the embedding of a hierarchical method into a random local search to implement an optimization solution. Comparative studies show that the proposed algorithm variant outperforms state-of-the-art algorithms on almost all benchmark datasets.

INDEX TERMS Packing, heuristic, hierarchical search, red-black tree.

I. INTRODUCTION

The two-dimensional strip packing problem (2DSPP), similar to the bin packing problem, is a classical combinatorial optimization problem, which attempts to arrange a given set of rectangles on a strip of finite width and infinite height without overlapping while minimizing the region waste of the strip. It has many real-world applications, such as cutting teel plates or paper rolls and cutting large boards of wood into smaller rectangular panels in the furniture industry [1], [2]. These practical problems are defined in order to minimize the material waste or maximize the use of material. Due to constraint set differences, 2DSPP can be subdivided into four classes [3] based on the yes/no requirement of rectangle rotation and that of guillotine:

1) RF: rectangles may be rotated 90° (R) and no guillotine cutting is required (F).
2) RG: rectangles may be rotated 90° (R) and guillotine cutting is required (G).
3) OF: the orientation of rectangles is fixed (O) and no guillotine cutting is required (F).
4) OG: the orientation of rectangles is fixed (O) and guillotine cutting is required (G).

In the literature, 2DSPP has been well studied as an NP-hard problem. Most research has been focused on exact algorithms or heuristic methods. An exact algorithm can guarantee the optimal solution to be found for the problem. These variants were widely proposed by those authors [4]–[9], which are either based on the branch-and-bound strategies or mixed integer linear programming. However, the curse of dimensionality inevitably leads to the exponential growth of computation. Therefore, the exact algorithm is only suitable for small-scale datasets. The heuristic methods are therefore developed in bulk for the 2DSPP in order to find a trade-off between the time complexity and the best approximation of optimal solution. In attempts to initialize a reasonable filling sequence, Baker *et al.* [10] proposed bottom-left (BL) heuristic in 1980. Chazelle [11] proposed a BL variant, bottom-left-fill (BLF) which first determines all possible positions that the coming rectangle can fit, and then selects the lowest one of them. Hopper and Turton [12] combined BL with three kinds of meta-heuristic algorithms, including genetics algorithms (GA), simulated annealing (SA), and naive evolution (NE).

The associate editor coordinating the review of this manuscript and approving it for publication was Seyedali Mirjalili.

Burke *et al.* [13] proposed a best-fit heuristic which can dynamically search the list to get better candidate shapes for placement. Zhang *et al.* [14] presented a simple and intuitive algorithm, the heuristic recursive (HR) algorithm. Soon, Zhang *et al.* [15] improved HR with a genetic algorithm integrated. Alvarez-Valdés *et al.* [16] presented a greedy randomized adaptive search procedure (GRASP) for 2DSPP. Belov *et al.* [17] proposed two iterative heuristics: SVC and BS.

Considering the intrinsic local optimization trap of greedy heuristics, many meta-heuristics algorithms have been adopted for solving the 2DSPP. The authors of [18]–[21] proposed a diverse range of GA-based approaches for the problem. These GA variants can mainly be partitioned into two classes. One is the boxing algorithm-based GA, which is employed to generate the filling sequence, and the other is used to store the layout information through the tree structure. In [22]–[24], the authors applied a simulated annealing algorithm in combination with other meta-heuristics to solve the 2DSPP. Iori *et al.* [25] combined TS with GA in solving the 2DSPP. Leung *et al.* [26] proposed a two-stage intelligent search algorithm (ISA) with simple scoring rules, which has been used for cooperating with minimum waste limitation and a simple random strategy in proposing the SRA algorithm by [27]. Zhang *et al.* [28] combined a variable domain search with modified scoring rules in order to generate a better approximation of an optimal solution. In addition, machine learning techniques have also been applied in solving 2DSPP. Hu *et al.* [29] utilized intensive learning for a proposed three-dimensional packing problem. Wei *et al.* [30] proposed a skyline-based heuristic algorithm with a set of rules for measuring rectangular priority. Wei *et al.* [31] then extended the skyline heuristic. Zhang *et al.* [32] used a binary search algorithm in order to ensure the smoothness of unused space. Wang and Chen [33] used the strategy of maximizing the remaining space. Chen and Chen [34] proposed an algorithm framework CIBA. It created a novel concept of a corner increment that is used for the rectangle selection. In nature, meta heuristic approaches may not converge to a fixed solution. Therefore, the optimal solution can not be guaranteed.

Oliveira *et al.* [1] investigated the application of heuristic algorithm in 2DSPP. The results show that 40% of heuristic algorithms are fitness-based heuristics, which has become a research trend in this field, especially heuristic algorithms based on scoring rules. Leung *et al.* [26] used five scores in the Intelligent Search Algorithm (ISA), ranging from 0 to 4, as was the Hybrid Demon Algorithm (HDA) [35]. While in the Simple Randomized Algorithm (SRA) [27] considers 8 ratings and introduces a minimum waste strategy. The design of these different scoring rules are roughly intended to achieve a flatter skyline. A simpler corner increment strategy with only 4 values is proposed in CIBA. However, we can further improve the heuristic hybrid algorithm based on scoring rules to improve the quality of the solution. We noticed that the rules in CIBA are very simple, but this may ignore the relationship between the rectangle and the available space.

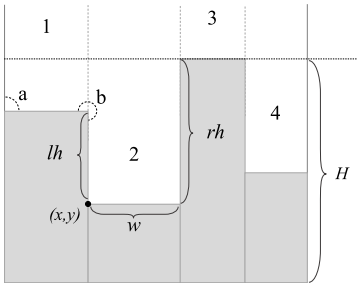


FIGURE 1. A layout.

This article improved the corner increment strategy and proposed new scoring rules. Given the complexity of the 2DSP problem, precise algorithms are often applied to small-scale datasets. The meta heuristic does not guarantee an optimal solution, but it can provide solutions in a fairly small amount of time. Combining the characteristics of the two algorithms, this paper considers combining precision and heuristic algorithms to develop a hybrid algorithm [36] to solve the 2DSPP with the OF constraint.

We propose a new algorithm with three core modifications: 1) a corner increment-based δ scoring rule is pre-defined for selecting rectangles in an effective way, 2) a red-black tree is used to memorize rectangle indices for the purpose of quickly determining the most-fit rectangles, and 3) a hierarchical strategy combined with a random-based local search is customized to expand the diversification of problem solutions. The remainder of this paper is organized as follows. Section II introduces some basic concepts of 2DSPP. Proposed approaches are elaborated in Section III. The experimental comparison and analysis are reported in Section IV. The conclusion and future perspectives of this paper are given in Section V.

II. BASIC CONCEPTS

Some concepts are systematically introduced in this section, which will be used hereafter:

1) R_j: j^{th} candidate rectangle ($1 \leq j \leq n$ where n is the number of rectangles).
2) $R_j.h$: the height of j^{th} candidate rectangle.
3) $R_j.w$: the width of j^{th} candidate rectangle.
4) $Seq = \{R_1, R_2, \ldots, R_n\}$: a sequence of rectangles.
5) *Layout*: a pattern of strip whose space is occupied by a set of rectangles. A layout example is shown in Figure 1.
6) *Empty Layout*: a pattern of strip containing none of a single rectangle.
7) H: a height storing the current highest y-coordinate of rectangles on the strip, which has been marked in Figure 1.
8) C_i($1 \leq i \leq m$ where m is the total number of caves, which will be assigned different values iteratively): indicates a cave that has the area excluding the occupied part of a segment, e.g., every white area numbered

TABLE 1. Value choice of corner increment and its corresponding conditions.

Corner Increment	Condition
-4	$(C_i.w = R_j.w)$ *and* $(C_i.lh = C_i.rh)$ *and* $(C_i.lh = R_j.h)$
-2	$(C_i.w = R_j.w)$ *and* $(C_i.lh \neq C_i.rh)$ *and* $(C_i.lh \neq R_j.h)$
0_w	$(C_i.w = R_j.w)$ *and* $(C_i.lh \neq C_i.rh)$ *and* $(C_i.lh \neq R_j.h)$
0_h	$(C_i.w \neq R_j.w)$ *and* $((C_i.lh = R_j.h)$ *or* $(C_i.rh = R_j.h))$
2	$(C_i.w \neq R_j.w)$ *and* $(C_i.lh \neq R_j.h)$ *and* $(C_i.rh \neq R_j.h)$

by 1-4 in Figure 1 means a cave having the following properties,

a) $C_i.w$: the width of C_i.
b) $C_i.cood$: the coordinate $(x,\ y)$ determines the left bottom corner of C_i.
c) $C_i.lh$: the height of the left border of C_i, which can be positively infinite. $C_i.lh = C_{i-1}.y - C_i.y$, where $C_{i-1}.y > C_i.y$ and C_{i-1} represents the previous adjacent cave of C_i.
d) $C_i.rh$: the height of the right border of C_i, which can be positively infinite. $C_i.rh = C_{i-1}.y - C_i.y$, where $C_{i+1}.y > C_i.y$ and C_{i+1} represents the next adjacent cave of C_i.
e) $C_i.maxH$: the higher height in the left and right borders of the C_i.
f) $C_i.minH$: the lower height in the left and right borders of the C_i.

9) *Corner*: each orthogonal white space in *Layout* is called a corner. The degree of a corner can be either 90° or 270°. *a* and *b* tagged in Figure 1 are two example corners, whose corresponding angle degrees are 90° and 270°.
10) *Corner increment*: the increment of the corner after a rectangle is placed into the strip.
11) *Cave smooth*: the merger operation of two adjacent caves. There are two cases where *cave smooth* is revoked. The first is where y-coordinates of the two adjacent caves are equal; in that case, the two caves will be merged. The other is where no single rectangle can be placed on either of those two adjoining caves, in which case they will be merged together.

III. HYBRID HEURISTIC-BASED ALGORITHM FOR 2DSPP

A new algorithm based on a hybrid heuristic for solving the 2DSPP will be described in detail. Section III-A gives the introduction of scoring rule building. Section III-B describes the process of rectangle selection using δ scoring rules and the red-black tree. Section III-C introduces a constructive heuristic based on δ scoring rules. Section III-D elaborates the hierarchical search. Section III-E gives an overview of the hybrid heuristic-based algorithm.

A. SCORING RULES

The construction of scoring rules is described in this section. Firstly, the definition of corner increment is explained in Section III-A1, from which δ scoring rules are derived. Secondly, the building process of δ scoring rules is detailed in Section III-A2.

1) CORNER INCREMENT

The corner increment was first proposed by [34]. Corner increment is the number of corners increased after a single packing move while corner indicates an orthogonal unoccupied area. The degree of corners is either 90° or 270°. The value of the corner increment simply has four choices: -4, -2, 0, or 2. The -4 value indicates that the four corners are flattened after a best-fit rectangle is inserted, and the remaining numbers respectively imply the flattening of two corners by the rectangle insertion move, no influence after a single packing move, and the larger vertical gap between every pair of rectangle top lines when an improper rectangle was arranged into a cave. Figure 2 enumerates five cases being considered for building δ scoring rules. The case in III-A1 is the most desirable in minimizing the waste of strip space. Table 1 is built for a clearer explanation of every value choice of corner increment taken under specified conditions. Note that there are two sets of conditions going into the same value of zero. For the sake of discriminating these two sets of conditions, the number '0' will be tagged with different subscripts. They are 0_w and 0_h as detailed in Table 1. The value 0_w indicates that candidate cave and a given rectangle are the same in width and different in height. The number 0_h signifies that the height of a given rectangle is equal to that of the border of a candidate cave and they are easily distinguished in width.

2) δ SCORING RULES

The proposed scoring strategy is a branch rule system of the corner increment, which is named as δ scoring rules and will be used as such throughout the rest of this paper. The original scoring rule of the corner increment only takes five cases into account. In particular, many other matching cases between caves and rectangles are omitted. However, when the corner increment scores the value of -4 or -2, the matching situation is reserved and no other cases can be achieved. Each δ scoring rule is therefore established when the values of the corner increment take either 0 or 2, and twelve new δ scoring rules are generated as listed in Table 2. Moreover, this is easily observed in Table 1 and Figure 2. The new rule system includes an additional imported parameter of δ imported, which is the reason for the name "δ scoring rules".

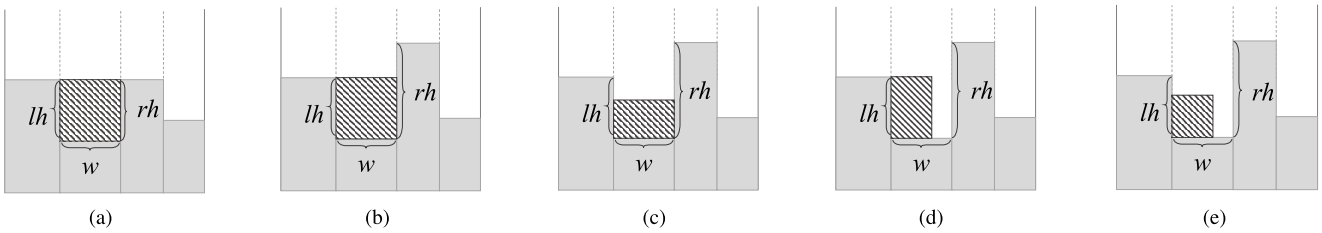


FIGURE 2. Corner increment cases.

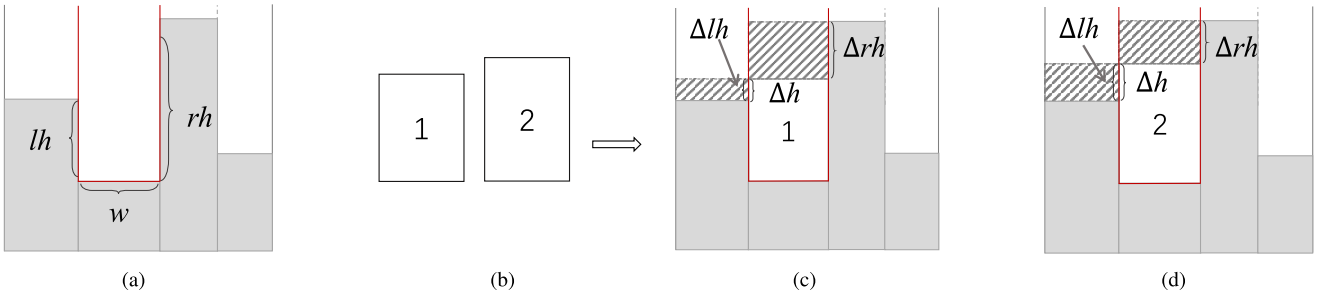


FIGURE 3. An example of δ scoring rules indicating (1)-(4) Table 2 in for rectangle selection.

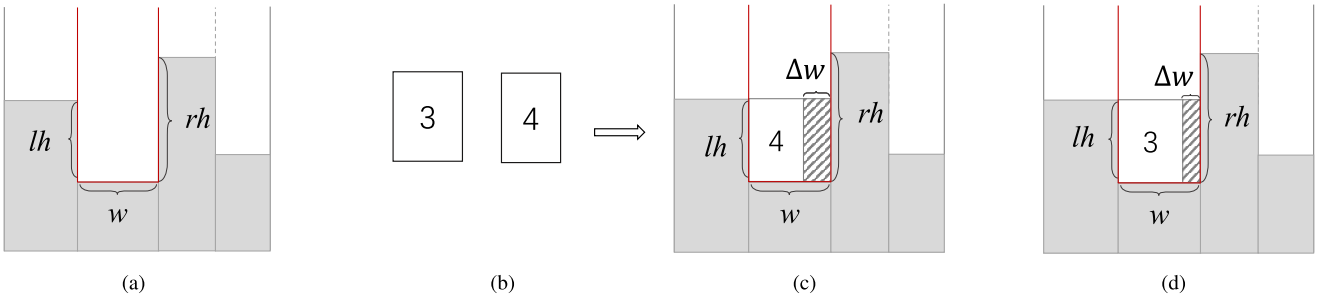


FIGURE 4. An example of δ scoring rules indicating (1)-(4) Table 2 in for rectangle selection.

The specific taken value of δ will be illustrated in Section IV, where the description of experimentation is given.

Figure 3 and Figure 4 give two examples in order to demonstrate the way to apply δ scoring rules in matching caves with rectangles. In Figure 3 (a) and Figure 4 (a), the region separated by red lines indicates a candidate cave which will be loaded with rectangles in Figure 3 (2) and Figure 4 (2), respectively. The purpose of rules (1)-(4) in Table 2 is to minimize the sum of differences between the currently-inserted rectangle and its adjacent existing rectangles. To find a candidate rectangle that decrease this difference, given a cave C_i and rectangle R_j, the equation in Equation 1 is then defined as follows:

$$\Delta h = \min(\Delta lh, \Delta rh) \tag{1}$$

where $\Delta lh = |C_i.lh - R_j.h|$ and $\Delta rh = |C_i.rh - R_j.h|$. The notation of $C_i.rh$, $C_i.lh$ and $R_j.h$ was given in Section II. Δh is a difference taking the minimum value between Δlh and Δrh. In the example shown in Figure 3, the Δh of Figure 3 (c) is observed to be less than that of Figure 3 (d). This means rectangle top lines become more even after the insertion of rectangle 1 rather than rectangle 2. The rules (5)-(7) built in Table 2 do not concern themselves with the height difference at all. These rules are focused on minimizing the width difference between a cave C_i and a rectangle R_i. The calculation of width difference is given in Equation 2.

$$\Delta w = |C_i.w - R_j.w| \tag{2}$$

where $C_i.w$ and $R_j.w$ were defined in Section II. The reason for taking into account the width difference Δw is that the width of R_j may exceed that of C_i. For instance, Δw of Figure 4 (c) is larger than that of Figure 4 (d). The rectangle 3 is then voted to be placed in the cave.

B. RECTANGLE SELECTION USING δ SCORING RULES AND RED-BLACK TREE

In [34], an AVL tree was used for storing a sequence of unpacked rectangles in the rectangle selection process, where

TABLE 2. δ scoring rules.

Corner Increment	Rules	δ Condition	Score
0_w	(1)	$C_i.lh(1-\delta) \le R_j.h \le C_i.lh(1+\delta) \quad or \quad C_i.rh(1-\delta) \le R_j.h \le C_i.rh(1+\delta)$	4
	(2)	$R_j.h > C_i.maxH(1+\delta)$	3
	(3)	$C_i.maxH(1-\delta) < R_j.h < C_i.minH(1+\delta)$	2
	(4)	$others$	1
0_h	(5)	$C_i.w(1-\delta) \le R_j.w \le C_i.w$	3
	(6)	$R_j.h = C_i.maxH$	2
	(7)	$R_j.h = C_i.minH$	1
2	(8)	$C_i.lh(1-\delta) \le R_j.h \le C_i.lh(1+\delta) \quad or \quad C_i.rh(1-\delta) \le R_j.h \le C_i.rh(1+\delta)$	4
	(9)	$(C_i.w(1-\delta) \le R_j.w \le C_i.w)$	4
	(10)	$R_j.h > C_i.maxH(1+\delta)$	3
	(11)	$C_i.maxH(1-\delta) < R_j.h < C_i.minH(1+\delta)$	2
	(12)	$others$	1

a single rectangle will be selected at each iteration from this AVL tree and then the node storing the rectangle is deleted. Such high frequency of the deleting operation leads to massive rotation moves for rebalancing AVL tree structures. Therefore, the red-black tree is introduced in order to avoid time cost in the tree structure rebalancing, although the average time complexity of the AVL tree and red-black tree are the same, with both taking $O(log(n))$. More importantly, our approaches used two red-black trees for the rectangle storage. They are built in different retrieval orders. A red-black tree T_{hwi} is constructed by recursively inserting rectangles into nodes in obedience to the criteria listing below:

1) $R_i.h > R_j.h$,
2) $R_i.h = R_j.h$ *and* $R_i.w > R_j.w$,
3) $R_i.h = R_j.h$ *and* $R_i.w = R_j.w$ *and* $i > j$.

Supposing R_j is any rectangle being in parent node, any given rectangle R_i of higher height than the rectangles being stored in parent nodes will be inserted into the left nodes and otherwise to the right nodes. When the height of R_i and R_j came to be equal, R_i of larger width than R_j is then stored in the left nodes and to the right nodes if smaller. When both the height and width of R_i and R_j are equal, R_i initially indexed by a number bigger than R_j will be put into the left nodes and to the right nodes, otherwise. The other red-black tree T_{whi} is created in the same manner of building T_{hwi}, following the rules below:

1) $R_i.w > R_j.w$,
2) $R_i.w = R_j.w$ *and* $R_i.h > R_j.h$,
3) $R_i.w = R_j.w$ *and* $R_i.h = R_j.h$ *and* $i > j$.

The only difference in creating these two red-black trees is that T_{whi} compares the width of R_i being inserted with that of R_j already saved in the parent nodes at first order, then the height and finally the index number, while T_{hwi} firstly compares the height between R_j and R_i, then width and the index number at last.

The rectangle selection strategies, mainly based on the combination of δ scoring rules and red-black trees, include five schemes. They are not independent and are used as ordered in the list below. The selection process stops when the first scheme is satisfied. Given a set of input rectangles $\mathbb{R} = \{R_1, R_2, \cdots, R_n\}$, a cave C_i, and a single fictional assistant rectangle R that is to be created according to the properties of cave C_i or directly assigned with a candidate rectangle, the five schemes are described as follows in the recalled order:

1) For the first scheme, the assistant rectangle R is created by assigning $R.w$ with $C_i.w$ and equating $R.h$ with $C_i.lh$. When the cave C_i is placed with the resulting R, the corner increment of the target strip will be decreased by four. Ideally, a rectangle with the same height and width as R is desired to be in the red-black tree T_{hwi} which is used for searching R-like rectangles. If no such rectangle is found before the process of tree traversal is completed, the next scheme will be invoked. Otherwise, the rectangle selection process stops and returns R.
2) This scheme is useful when the first one is disabled. In the scheme, R is built in two possible ways. One is that $R.w$ is given by $C_i.w$ and $R.h$ is equal to $C_i.maxH$. The other is that $R.w$ is given by $C_i.w$ and $R.h$ is equal to $C_i.minH$. Although the obtained R is imperfect when compared to the rectangle construction of the scheme (1), two corner increments will be eliminated after the insertion of this R into the candidate cave C_i. If a rectangle was matched with R by traversing the T_{hwi}, R returns; otherwise, move to the next scheme.
3) When the previous two schemes (1) and (2) are invalid, this scheme will be activated where the number of corner increments is not decreased or increased at all after the auxiliary rectangle R being inserted into the C_i and δ scoring rules is used. The current scheme will create a auxiliary set of rectangles $\mathbb{R}' = \{R_j \mid j \in \mathbb{N}^+\} \subseteq \mathbb{R}$. The $\mathbb{R}'$ is established obeying the conditions $R_j.w \equiv C_i.w$ and $0 \le R_j.h \le \infty$ and then $\mathbb{R}' = \{R_j \mid R_j.w \equiv C_i.w,\ 0 \le R_j.h \le \infty,\ j \in \mathbb{N}^+\}$. The intersection set of rectangles $\mathbb{R}_\cap = \mathbb{R}' \cap T_{whi}$ is the sequence of candidate rectangles, which will be scored by the rules (1)-(4) in Table 2. The complete T_{whi} will be traversed in order to search for such an $\mathbb{R}_\cap$. Traversing the T_{whi} until an R_j with $R_j.w > C_i.w$ is found or the tree

Algorithm 1 Rectangle Selection

Input:
C_i is the cave to be placed
T_{whi} and T_{hwi} store the rectangle sequence
Output:
R is the selected rectangle

$R_{temp} =$ null
// case −4
$R.w = C_i.w,\ R.h = C_i.lh$, traversing T_{hwi} finds a rectangle R_{temp} equals R
if $R_{temp} \neq null$ **then**
    $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
    return R
end
// case −2
$R.w = C_i.w,\ R.h = C_i.maxH$, traversing T_{hwi} finds a rectangle R_{temp} equals R
if $R_{temp} \neq null$ **then**
    $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
    return R
else if $R_{temp} == null$ **then**
    $R.w = C_i.w,\ R.h = C_i.minH$, traversing T_{hwi} finds a rectangle R_{temp} equals R
    if $R_{temp} \neq null$ **then**
        $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
        return R
    end
// case 0_w
traversing T_{whi} finds all rectangles that satisfy the condition $R_i.w = C_i.w$ *and* $0 < R_i.h < \infty$
using (1) - (4) rules score all rectangles, the highest score and the smallest index number is stored in R_{temp}
if $R_{temp} \neq null$ **then**
    $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
    return R
end
// case 0_h
traversing T_{hwi} finds all rectangles that satisfy the condition $R_i.h = C_i.lh$ *and* $0 < R_i.w < C_i.w$
using (5) -(7) rules score all rectangles, the highest score and the smallest index number is stored in R_{temp}
if $C_i.lh \neq C_i.rh$ **then**
    traversing T_{hwi} finds all rectangles that satisfy the condition $R_i.h = C_i.rh$ *and* $0 < R_i.w < C_i.w$
    using (5) - (7) rules score all rectangles, the highest score and the smallest index number is stored in R_{temp}
end
if $R_{temp} \neq null$ **then**
    $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
    return R
end
// case 2
traversing T_{whi} finds all rectangles that satisfy the condition $0 < R_i.w < C_i.w$
using (8) - (12) rules score all rectangles, the highest score and the smallest index number is stored in R_{temp}
if $R_{temp} \neq null$ **then**
    $R = R_{temp}$, delete R_{temp} from T_{whi} and T_{hwi}
    return R
end

traversal terminates. If $\mathbb{R}_\cap$ is empty, it will proceed to next scheme, otherwise R will be assigned with a rectangle that had the highest score and the smallest index in the $\mathbb{R}_\cap$ and returned.

4) As in scheme (3), the insertion of the assistant R into the C_i did not change the total number of corner increment in the strip for this scheme. An auxiliary set $\mathbb{R}' = \{R_j \mid j \in \mathbb{N}^+\} \subseteq \mathbb{R}$ is built by using the

following conditions: $0 < R_j.w \leq C_i.w$ and $R_j.h \in \{C_i.lh, C_i.rh\}$. The resulting $\mathbb{R}'$ is $\{R_j \mid 0 < R_j.w \leq C_i.w,\ R_j.h \in \{C_i.lh, C_i.rh\},\ j \in \mathbb{N}^+\}$. A candidate set of rectangles $\mathbb{R}_\cap$ is constructed by intersecting $\mathbb{R}'$ and T_{hwi}. Each of rectangles in $\mathbb{R}_\cap$ will be given a score using rules (5)-(7). R takes the rectangle that achieved the best score being indexed by the minimum number in $\mathbb{R}_\cap$, and is then returned. When $\mathbb{R}_\cap$ is always none until T_{hwi} is completely traversed or an R_j is found that $R_j.h > \max(C_i.lh, C_i.rh)$ in T_{hwi}, the coming scheme will be recalled.

5) In this scheme, the placement of R in the cave C_i will cause an increase of corner increment by 2 in the strip. As a result of this placement, the C_i may be separated into two sub-caves. The cardinality of $\mathbb{R}' = \{R_j \mid j \in \mathbb{N}^+\} \subseteq \mathbb{R}$ is simply ranged following a single condition: $0 < R_j.w \leq C_i.w$. Then, the $\mathbb{R}' = \{R_j \mid 0 < R_j.w \leq C_i.w,\ j \in \mathbb{N}^+\}$ is created as the candidate set of rectangles. The intersection set $\mathbb{R}_\cap = \mathbb{R}' \cap T_{whi}$ is never none. This means there is always a best candidate rectangle that can be returned from the T_{whi} until the tree is empty due to the iteratively deleting of a single node. These candidate rectangles will be scored by rules (8)-(12). When the scores of candidate rectangles are the same, the candidate rectangle being tagged with the minimum index is returned as the best one.

Note that δ scoring rules are not employed for the schemes (1) and (2) due to the simplicity of rectangle construction, in which only one assistant rectangle is generated. In order to make the rectangle selection process more understandable, Algorithm 1 is given to stepwisely describe the process of selecting rectangles.

C. CONSTRUCTIVE HEURISTIC USING PROPOSED RECTANGLE SELECTION METHOD

The method of rectangle selection is a key component in a constructive heuristic. There have been many rectangle selection strategies proposed in literature such as [26]–[28], [30], [34], and [35]. The positioned-based constructive heuristics, such as the bottom-left, skyline, corner increment, which historically it was the most widely used strategy for solving 2DSPP, and it is still an important basis for creating new constructive heuristics. Rectilinear skyline approaches attempt to minimize the space waste, while corner increment methods are focused on locally minimizing the number of corner increments for a strip pattern. As described in Section III-B, our rectangle selection method is proposed upon red-black trees and δ scoring rules that are an extension of corner increment. The process of the constructive heuristic based on δ scoring rules and red-black trees is detailed in Algorithm 2. Cautiously, the *Cave smooth* operation is very interesting, in that it focuses on merging adjoining caves that have less difference in the y-coordinate of bottom cave lines together for the rectangle insertion.

The time complexity of Algorithm 2 is analyzed as follows. In line 4 of the algorithm, it takes $O(1)$ time to get a candidate cave due to the use of priority queue for the storage of caves. In line 5-9, $O(log(n))$ time is taken for the deletion and adjustment of the priority queue of a *Cave smooth* operation. Line 10, using red-black tree to find the candidate rectangle, takes $O(log(n))$ time. The time complexity of line 11 is $O(1)$. Line 13 takes $O(log(n))$ time. The operation on lines 13 simply requires $O(1)$ time. Therefore, the time complexity of the whole Algorithm 2 is $O(nlog(n))$.

Algorithm 2 Constructive Heuristic

Input:
Layout.caveList represents all caves in the layout
Seq indicates unpacked sequence of rectangles
Output:
H is a packing solution, which indicates the current highest y-coordinate of packed rectangles

```
1  W is the current smallest width of the unpacked rectangles
2  create two red-black trees based on Seq, T_whi and T_hwi
3  while Seq.length > 0 do
4      get the lowest and most left cave C from Layout.caveList
5      if C.w is smaller than W
6      then
7          Cave smooth
8          continue
9      end
10     R = RectangleSelection(C, T_whi, T_hwi)
11     pack rectangle R into the cave C
12     Cave smooth if necessary
13     update H when satifying the triger criteria
14 end
15 returnH
```

D. HIERARCHICAL SEARCH

Traditional greedy search methods are very easy to be trapped into the local optima. Therefore, a hierarchical search method is proposed, which was inspired by the multi-layer heuristic search algorithm proposed in [37]. The purpose of this hierarchical search method is to diversify the search space. The larger diversification of the search space may result in a higher possibility of escaping the local optima. The root parent will be initialized with an empty strip. A new child node is generated by placing a single unpacked rectangle into the strip pattern of its parent. There are two parameters for this hierarchical search method: *childNum* and k. The parameter *childNum* indicates the number of child nodes that can be generated from the same parent, and k is the number of best patterns in a generation. An example of the hierarchical search structure is given in Figure 5, where *childNum* is set to 2 and k is set to 3. Algorithm 3 details the process of hierarchical search. Due to the impossibility of enumerating all possible patterns of the strip, we only select the top k nodes for making the problem solution diverse in each generation.

Algorithm 3 Hierachical search

Input:
Layout.caveList represents all caves in the layout
Seq indicates unpacked sequence of rectangles
Output:
H is a packing solution

```
1  nodeQueue = null
2  for i = 0 → k − 1 do
3      new a node, node.unpackedSeq is equal to Seq
       minus R_i, node.layout is equal to the layout after R_i
       is put in
4      nodeQueue.add(node)
5  end
6  while the stop criterion is not satisfied do
7      currentNodeQueue represents the top k nodes in the
       next layer
8      while nodeQueue.size > 0 do
9          tempNode = nodeQueue.poll()
10         get the lowest and most left cave C_i form
           tempNode.layout.caveList
11         canPutSeq is all the rectangles form
           node.unpackedSeq that fit into cave C_i
12         while j = 0 → canPutSeq.size and j <
           childNum do
13             new a childNode, childNode.unpackedSeq
               is equal to node.unpackedSeq minus R_j,
               childNode.layout is equal to the layout after
               R_i is put in
14             childNode.fitness =
               ConstructiveHeuristic(childNode.layout,
               childNode.unpackedSeq)
15             if childNode is in the top k in terms of
               fitness then
16                 currentNodeQueue.add(childNode)
17             end
18         end
19     end
20     nodeQueue = currentNodeQueue
21 end
22 minNode is the best fitness node from nodeQueue
23 H = minNode.fitness
24 returnH
```

E. OVERALL HYBRID HEURISTIC ALGORITHM

Although the hierarchical search could escape from the local optima, it may take massive computational effort. To ease this issue, we therefore integrated local random search into hierarchical search. Local random search accepts a current optimal solution, adds disturbances, and then searches to increase the chances of finding a better solution. The initialization of input sequences is very significant, which may lead to the best initial search point that can possibly achieve the optimal solution. Algorithm 4 describes the overall algorithm framework. At the beginning of hybrid heuristic algorithm

Algorithm 4 Overall hybrid heuristic algorithm

Input:
W is the width of the strip
Seq indicates unpacked sequence of rectangles
Output:
H is a packing solution

```
1  EmptyLayout.caveList is an initial sequence of caves,
   an Empty Layout with only one cave, which consists of
   the whole strip, with a width equal to W
2  bSeq = null, H = infinite
3  for each sortig rule do
4      sortSeq = using the sortig rule sort Seq
5      tempH =
       ConstructiveHeuristic(EmptyLayout.caveList,
       sortSeq)
6      if tempH < H then
7          bSeq = sortSeq
8          H = tempH
9      end
10 end
11 while the stop criterion is not satisfied do
12     tempH = hierachicalSearch(EmptyLayout.caveList,
       bSeq)
13     if tempH < H then
14         bSeq = SwapRandom(bSeq)
15         H = tempH
16     end
17 end
18 returnH
```

(HHA), several initial sequences are generated in a certain order which will be passed to the constructive heuristic in turn for the purpose of obtaining several initial solutions. This move is described in lines 3-10. The sequence of the optimal solution is selected as the input sequence of the hierarchical search. The packing sequence corresponding to the solution returned by the hierarchical search is perturbed at line 14. We will randomly swap the positions of the two rectangles and pass the rearrangement sequence to the hierarchical search algorithm for the next search. Lines 11-17 are the process of random local search, and the search returns the optimal solution until the stop condition is met.

We use the following six rules [34] to generate the initial sequence: (1)

1) Sort the rectangles in descending order of area and sort in descending order of width if the areas are equal;
2) Sort the rectangles in descending order of area and sort in descending order of width if the areas are equal;
3) Sort the rectangles in descending order of circumference and sort in descending order of height if the areas are equal;
4) Sort the rectangles in descending order of circumference and sort in descending order of height if the areas are equal.

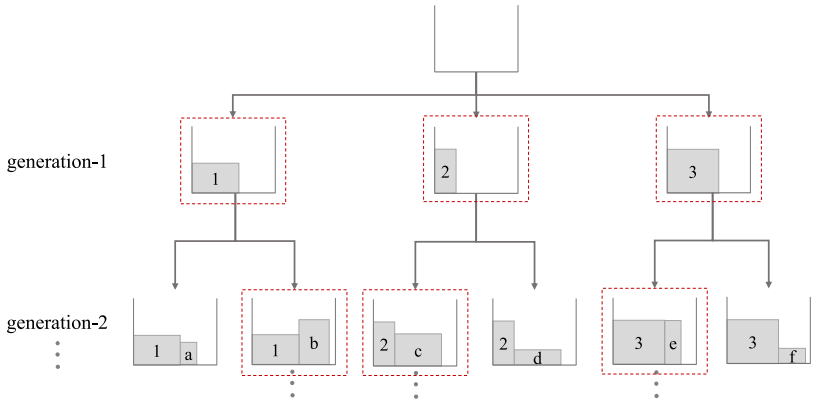


FIGURE 5. An example of hierarchical search.

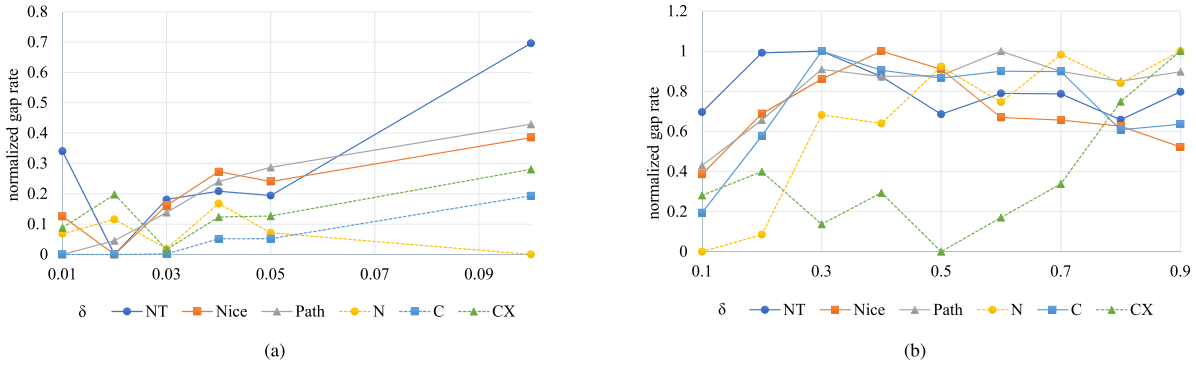


FIGURE 6. δ sensitivity analysis.

TABLE 3. Benchmark datasets.

Dataset	*Instances*	*n*	*zero-waste*	*Data Source*
C	21	16-197	Yes	[12]
N	13	10-3152	Yes	[13]
NT	70	17-199	Yes	[38]
CX	7	50-15000	Yes	[39]
2SP	38	7-200	No	[40–43]
BWMV	500	20-100	No	[44, 45]
Nice&Path	72	25-5000	No	[46]
ZDF	16	580-75032	No	[47]

5) Sort the rectangles in descending order of height and sort in descending order of width if the heights are equal.
6) Sort the rectangles in descending order of width and sort in descending order of height if the widths are equal.

The time complexity analysis of the entire algorithm is conducted as follows. In Algorithm 4, in line 4, sorting the rectangular sequences takes $O(nlog(n))$ time. In line 5, The time complexity of constructive heuristic is $O(nlog(n))$, line 6-9, takes O(1) time, so lines 3-10 takes $O(nlog(n))$ time. In line 12, we set $k = 100$ and $childNum = 20$, so the hierarchical search takes $O(nlog(n))$ time. The time cost of lines 13-16 is $O(1)$. The total time complexity of Algorithm 4 is therefore $O(nlog(n))$.

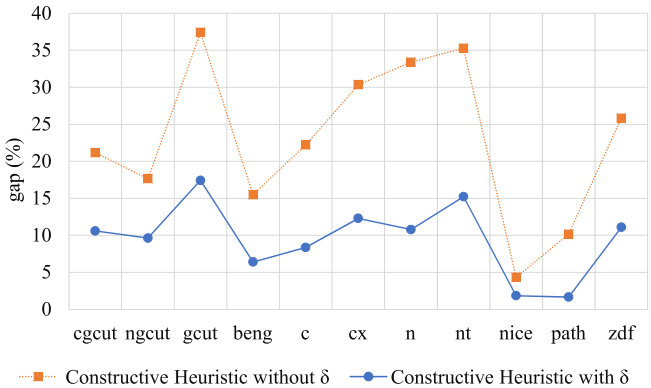


FIGURE 7. Test constructive heuristics on datasets.

IV. EXPERIMENTAL RESULTS

In this section, a series of experiments are conducted for analyzing the proposed methods. Section IV-B illustrates some settings concerning the configuration of the machine that is used to run the experiments. Section IV-C gives information on relevant benchmark datasets. Section IV-D discusses the experimental results. Section IV-A analyzes the key parameters as they are set to different values.

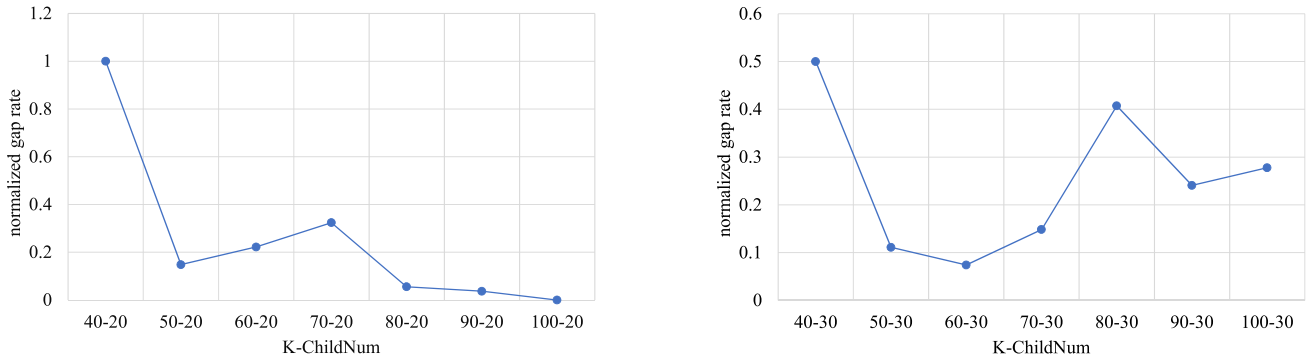


FIGURE 8. Analysis parameters *k* and *childNum*.

TABLE 4. Experimental results on instance C [12].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
C1	16	20	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C2	17	20	20	**0.0**	5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
C3	16	20	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C4	25	40	15	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C5	25	40	15	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C6	25	40	15	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C7	28	60	30	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C8	29	60	30	3.3	3.3	3.3	3.3	3.3	3.3	**0.0**
C9	28	60	30	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C10	49	60	60	1.7	1.7	1.7	1.7	1.7	1.7	1.7
C11	49	60	60	1.7	1.7	1.7	1.7	1.7	1.7	1.7
C12	49	60	60	1.7	1.7	1.5	1.7	1.7	**0.0**	**0.0**
C13	73	60	90	1.1	1.1	1.1	1.1	0.9	0.4	**0.0**
C14	73	60	90	1.1	1.1	0.9	1.1	0.4	0.3	**0.0**
C15	73	60	90	1.1	1.1	1.1	1.1	1.1	1.1	1.1
C16	97	80	120	1.7	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**
C17	97	80	120	0.8	0.8	0.8	0.8	0.8	0.8	0.8
C18	97	80	120	1.7	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**	**0.8**
C19	196	120	240	1.7	0.8	0.8	**0.4**	**0.4**	**0.4**	**0.4**
C20	197	120	240	1.3	0.8	**0.4**	**0.4**	**0.4**	**0.4**	**0.4**
C21	196	120	240	1.3	0.8	0.8	**0.4**	**0.4**	**0.4**	**0.4**
Average				0.95	1.03	0.76	0.71	0.69	0.58	**0.39**

TABLE 5. Experimental results on instance N [13].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
n1	10	40	40	0.0	0.0	0.0	0.0	0.0	0.0	0.0
n2	20	30	50	0.0	0.0	0.0	0.0	0.0	0.0	0.0
n3	30	30	50	2.0	**0.0**	0.2	**0.0**	**0.0**	**0.0**	**0.0**
n4	40	80	80	1.3	1.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
n5	50	100	100	2.0	1.0	1.0	**0.0**	**0.0**	**0.0**	**0.0**
n6	60	50	100	1.0	1.0	0.9	0.7	0.2	**0.0**	**0.0**
n7	70	80	100	1.0	1.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
n8	80	100	80	1.3	1.3	1.3	1.3	1.3	1.3	1.3
n9	100	50	150	0.7	0.7	0.6	0.7	0.6	**0.0**	**0.0**
n10	200	70	150	0.7	0.7	0.5	0.7	0.4	**0.0**	**0.0**
n11	300	70	150	0.7	0.7	0.5	0.5	0.1	**0.0**	**0.0**
n12	500	100	300	1.3	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**	**0.3**
n13	3152	640	950	0.5	0.3	**0.0**	**0.0**	0.1	0.1	**0.0**
Average				0.96	0.64	0.41	0.32	0.23	0.13	**0.12**

A. PARAMETER ANALYSIS

The value of delta affects the results of constructive heuristics, so we analyzed the sensitivity of it on 6 datasets, running constructive heuristics at different delta values and normalizing the results. As shown in Figure 6 (the abscissa is the dataset and the ordinate is the gap rate), Figure 6 (a) is the

TABLE 6. Experimental results on instance NT [38].

Instance				*average_gap_rate (%)*							
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	HA	CIBA	HHA
n1a	17	200	200	**0.0**	1.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
n1b	17	200	200	4.5	**0.0**	5.6	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
n1c	17	200	200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
n1d	17	200	200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
n1e	17	200	200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
n2a	25	200	200	3.0	2.5	2.0	0.7	**0.0**	**0.0**	**0.0**	**0.0**
n2b	25	200	200	3.0	4.5	4.7	5.0	**0.0**	**0.0**	**0.0**	**0.0**
n2c	25	200	200	4.0	4.5	4.3	3.2	1.5	**0.0**	1.5	**0.0**
n2d	25	200	200	4.5	3.5	3.9	3.0	2.9	2.2	1.8	**0.0**
n2e	25	200	200	3.0	2.5	3.3	3.1	2.6	2.3	1.5	**0.0**
n3a	29	200	200	4.5	4.0	3.1	3.2	**0.0**	**0.0**	3.1	**0.0**
n3b	29	200	200	4.0	3.5	4.5	4.5	4.3	4.0	3.5	**1.0**
n3c	29	200	200	2.5	3.5	3.1	2.5	**0.0**	**0.0**	2.9	2.3
n3d	29	200	200	3.5	4.0	2.2	2.3	2.0	2.0	**0.6**	1.4
n3e	29	200	200	3.5	3.5	4.0	4.1	3.6	3.8	2.0	**0.0**
n4a	49	200	200	3.0	2.5	3.0	3.0	2.6	2.4	2.2	**2.0**
n4b	49	200	200	3.5	2.5	2.5	2.3	2.2	**1.7**	2.6	2.4
n4c	49	200	200	2.5	2.5	3.0	2.7	2.6	2.3	2.2	**1.7**
n4d	49	200	200	3.0	2.5	2.4	2.5	2.6	2.1	**2.0**	2.1
n4e	49	200	200	2.5	2.5	3.0	3.1	3.3	2.7	**1.8**	2.0
n5a	73	200	200	2.5	2.0	2.6	2.8	2.4	2.0	1.8	**1.5**
n5b	73	200	200	2.0	2.0	1.8	**1.6**	1.9	**1.6**	**1.6**	**1.6**
n5c	73	200	200	3.0	2.0	2.2	2.3	2.1	1.8	1.9	**1.7**
n5d	73	200	200	2.0	2.5	2.5	2.3	2.3	**1.8**	2.0	2.2
n5e	73	200	200	3.0	2.5	2.3	2.5	2.4	1.8	1.8	**1.7**
n6a	97	200	200	2.0	1.5	1.4	1.3	1.5	**1.0**	1.2	1.5
n6b	97	200	200	2.0	2.0	1.5	1.5	1.5	**1.2**	**1.2**	1.3
n6c	97	200	200	2.0	2.0	1.8	1.6	1.7	1.4	**1.2**	1.3
n6d	97	200	200	2.1	**1.0**	1.9	1.5	1.5	1.3	1.5	1.5
n6e	97	200	200	**1.0**	1.5	1.8	1.5	1.4	1.2	1.3	1.2
n7a	199	200	200	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
n7b	199	200	200	1.5	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
n7c	199	200	200	1.5	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
n7d	199	200	200	1.5	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
n7e	199	200	200	1.5	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
t1a	17	200	200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
t1b	17	200	200	**0.0**	5.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
t1c	17	200	200	**0.0**	5.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
t1d	17	200	200	**0.0**	**0.0**	5.9	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
t1e	17	200	200	**0.0**	4.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
t2a	25	200	200	2.0	3.5	3.5	3.2	3.4	3.0	1.8	**0.0**
t2b	25	200	200	4.0	2.5	3.5	2.9	2.1	2.8	1.1	**0.0**
t2c	25	200	200	4.0	3.0	3.0	3.7	3.1	3.1	1.4	**0.6**
t2d	25	200	200	3.0	3.5	4.7	2.2	1.6	**0.0**	1.3	**0.0**
t2e	25	200	200	3.0	3.5	3.7	3.0	3.0	2.3	1.0	**0.0**
t3a	29	200	200	3.5	4.0	4.5	4.5	1.7	2.4	1.3	**1.2**
t3b	29	200	200	4.5	3.5	4.1	4.0	3.5	3.8	3.6	**2.8**
t3c	29	200	200	3.0	3.5	3.3	3.2	3.1	2.8	0.8	**0.0**
t3d	29	200	200	3.5	4.0	3.2	3.2	**0.0**	**0.0**	0.9	1.6
t3e	29	200	200	4.0	3.0	2.5	2.5	2.5	2.5	3.6	**1.0**
t4a	49	200	200	2.5	2.5	2.5	2.3	2.6	2.1	2.3	**2.0**
t4b	49	200	200	2.5	2.5	3.1	3.0	3.2	3.0	2.2	**1.6**
t4c	49	200	200	3.0	2.5	2.5	2.3	2.3	**1.7**	1.8	2.1
t4d	49	200	200	3.0	2.5	2.8	2.7	2.6	2.2	2.2	**2.0**
t4e	49	200	200	2.5	2.5	2.6	2.8	**2.2**	2.5	2.6	2.3
t5a	73	200	200	3.0	2.0	2.2	2.3	2.2	1.8	1.8	**1.7**
t5b	73	200	200	2.0	2.0	2.0	2.2	2.1	1.6	1.5	**1.2**
t5c	73	200	200	2.5	2.0	2.5	2.8	2.3	2.0	1.8	**1.5**
t5d	73	200	200	2.0	2.5	2.5	2.3	2.4	**1.8**	2.0	2.1
t5e	73	200	200	2.0	2.0	2.0	2.3	2.3	2.0	**1.6**	2.3
t6a	97	200	200	2.0	2.0	1.6	1.8	2.0	1.5	1.3	**1.2**
t6b	97	200	200	2.0	**1.0**	1.7	1.6	1.5	1.3	1.1	1.3
t6c	97	200	200	2.0	2.0	1.5	1.3	1.5	1.3	1.5	**1.2**
t6d	97	200	200	2.0	2.0	1.8	1.8	1.8	1.5	**1.2**	1.3
t6e	97	200	200	2.5	2.0	1.8	1.8	1.7	1.5	1.4	**1.3**
t7a	199	200	200	1.5	**0.5**	0.6	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
t7b	199	200	200	1.5	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
t7c	199	200	200	2.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
t7d	199	200	200	1.0	1.0	1.0	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
t7e	199	200	200	1.5	1.0	0.8	**0.5**	**0.5**	**0.5**	**0.5**	**0.5**
Average				2.32	2.27	2.24	1.93	1.60	1.37	1.30	**0.97**

test result with a delta value of $0.01 - 0.1$ (step size is 0.01), and Figure 6 (b) is the test result with a delta value of $0.1 - 0.9$ (step size is 0.1). From the results we can conclude that most of the datasets have poor solutions when the delta value is greater than 0.1. When the delta value is set to 0.02 or 0.03, the algorithm performs satisfactorily on most datasets.

In order to illustrate the effectiveness of the δ scoring rules, the δ scoring rules based constructive heuristics and the architectural heuristics without scoring rules are compared on 8 datasets. As shown in Figure 7 (the abscissa is the dataset and the ordinate is the gap rate), the δ scoring rules based constructive heuristics are significantly better than the original

TABLE 7. Experimental results on instance CX [39].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
50	50	400	600	2.2	0.5	3.4	1.2	0.3	0.3	**0.0**
100	100	400	600	2.8	2.7	2.6	2.9	3.3	2.6	**1.3**
500	500	400	600	0.8	0.7	0.2	0.2	**0.0**	**0.0**	**0.0**
1000	1000	400	600	0.3	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
5000	5000	400	600	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10000	10000	400	600	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15000	15000	400	600	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Average				0.88	0.57	0.88	0.61	0.52	0.41	**0.18**

TABLE 8. Experimental results on instance 2SP [40]–[43].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
cgcut1	16	10	23	**0.0**	**0.0**	**0.0**	4.3	4.4	**0.0**	**0.0**
cgcut2	23	70	63	3.2	3.2	3.2	3.2	3.2	3.2	3.2
cgcut3	62	70	636	3.9	3.9	**3.8**	4.1	4.0	**3.8**	**3.8**
gcut1	10	250	1016	0.0	0.0	0.0	0.0	0.0	0.0	0.0
gcut2	20	250	1133	5.1	**4.8**	**4.8**	**4.8**	**4.8**	**4.8**	**4.8**
gcut3	30	250	1803	0.0	0.0	0.0	0.0	0.0	0.0	0.0
gcut4	50	250	2934	**2.3**	2.8	2.6	**2.3**	2.4	2.6	2.8
gcut5	10	500	1172	8.6	8.6	8.6	8.6	8.6	8.6	8.6
gcut6	20	500	2514	4.5	4.7	4.7	4.6	4.5	4.5	**4.3**
gcut7	30	500	4641	1.1	1.1	1.1	1.1	1.1	1.1	1.1
gcut8	50	500	5703	3.7	**3.0**	3.3	3.2	3.2	3.1	3.2
gcut9	10	1000	2022	14.6	14.6	14.6	14.6	14.6	14.6	14.6
gcut10	20	1000	5356	11.4	11.4	11.4	11.4	11.4	11.4	11.4
gcut11	30	1000	6537	5.5	5.4	5.3	5.3	**5.0**	5.2	5.2
gcut12	50	1000	12522	17.3	17.3	17.3	17.3	17.3	17.3	17.3
gcut13	32	3000	4772	4.7	4.3	4.1	4.0	4.2	4.2	**3.3**
ngcut01	10	10	23	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut02	17	10	30	**0.0**	**0.0**	3.3	3.3	3.3	**0.0**	**0.0**
ngcut03	21	10	28	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut04	7	10	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut05	14	10	36	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut06	15	10	29	6.9	6.9	6.9	6.9	6.9	6.9	6.9
ngcut07	8	20	20	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut08	13	20	32	**3.1**	6.3	6.3	6.3	6.3	6.3	6.3
ngcut09	18	20	49	**2.0**	4.1	6.1	4.1	4.1	4.1	**2.0**
ngcut10	13	30	80	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ngcut11	15	30	50	4.0	4.0	4.0	4.0	4.0	4.0	4.0
ngcut12	22	30	87	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng01	20	25	30	**0.0**	**0.0**	3.3	2.3	3.3	3.3	**0.0**
beng02	40	25	57	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng03	60	25	84	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng04	80	25	107	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng05	100	25	134	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng06	40	40	36	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng07	80	40	67	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng08	120	40	101	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng09	160	40	126	0.0	0.0	0.0	0.0	0.0	0.0	0.0
beng10	200	40	156	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Average				**2.68**	2.80	3.02	3.04	3.07	2.87	2.70

constructive heuristics in all the tested datasets, which indicates that the δ scoring rules can significantly improve the quality of the solution.

To select a reasonable value of k (the number of nodes per layer) and *childNum* (the number of child nodes), we performed parameter sensitivity analysis on 8 datasets. Since the running time of the algorithm is limited, the values of k and *childNum* cannot be set too large. In the test experiment, we set the value of k to a number in $\{40, 50, 60, 70, 80, 90, 100\}$ when setting *childNum* value to 20 and 30. First the different parameters are arranged and combined, then the HHA performs under the different

TABLE 9. Experimental results on instance BWMV [44], [45].

Instance				*average_gap_rate (%)*							
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	HA	CIBA	HHA
C01	20	10	60.3	1.8	1.8	1.7	1.7	1.7	1.7	1.7	**1.5**
	40	10	121.6	**0.2**	0.3	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**
	60	10	187.4	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	80	10	262.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	100	10	304.4	0.2	0.2	0.2	0.2	0.2	0.2	0.2	**0.1**
C02	20	30	19.7	**0.5**	**0.5**	1.0	**0.5**	0.9	1.0	0.7	**0.5**
	40	30	39.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	60	30	60.1	0.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	80	30	83.2	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	100	30	100.5	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C03	20	40	157.4	3.9	4.6	4.2	4.0	4.1	4.1	4.0	**3.4**
	40	40	328.8	1.6	1.6	1.5	1.5	1.4	**1.3**	**1.3**	**1.3**
	60	40	500	1.3	1.4	1.2	1.2	1.1	**1.0**	1.1	1.1
	80	40	701.7	1.1	1.2	1.1	1.1	1.1	1.1	1.1	**1.0**
	100	40	832.7	0.9	0.9	**0.6**	0.7	**0.6**	**0.6**	**0.6**	**0.6**
C04	20	100	61.4	**3.1**	3.9	4.1	3.3	3.9	3.6	3.4	**3.1**
	40	100	123.9	1.9	1.9	1.8	1.5	1.6	1.5	1.3	**1.1**
	60	100	193	1.9	1.3	1.3	1.3	1.1	1.0	0.9	**0.8**
	80	100	267.2	1.8	1.2	1.0	1.1	0.9	0.9	**0.8**	**0.8**
	100	100	322	1.6	1.0	0.8	0.8	0.7	0.6	0.5	**0.4**
C05	20	100	512.2	**4.2**	5.0	4.4	4.3	4.3	4.3	4.3	**4.2**
	40	100	1053.8	2.0	2.1	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**	**1.9**
	60	100	1614	2.0	2.1	**1.8**	1.9	**1.8**	**1.8**	**1.8**	2.0
	80	100	2268.4	1.0	**0.9**	**0.9**	**0.9**	1.0	**0.9**	1.0	**0.9**
	100	100	2617.4	1.3	1.4	**1.0**	1.1	1.3	**1.0**	1.1	1.2
C06	20	300	159.9	4.6	6.1	6.1	5.5	5.6	4.9	5.4	**4.2**
	40	300	323.5	3.1	2.8	3.2	3.0	3.0	2.8	2.8	**2.2**
	60	300	505.1	2.9	2.4	2.8	2.6	2.6	2.4	2.4	**1.8**
	80	300	699.7	2.7	2.1	2.3	2.2	2.2	2.0	2.0	**1.6**
	100	300	843.8	2.5	2.0	2.1	2.1	1.9	1.8	1.8	**1.4**
C07	20	100	490.4	2.3	2.3	2.3	2.3	2.3	2.3	2.3	2.3
	40	100	1049.7	**0.9**	1.0	**0.9**	**0.9**	**0.9**	1.0	**0.9**	**0.9**
	60	100	1515.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	80	100	2206.1	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	100	100	2627	**0.6**	**0.6**	0.7	0.7	0.7	0.7	0.7	0.7
C08	20	100	434.6	5.5	6.1	5.5	5.4	5.4	5.3	5.2	**4.9**
	40	100	922	3.5	3.7	3.2	3.2	3.2	3.1	**2.9**	**2.9**
	60	100	1360.9	3.2	3.1	2.8	3.1	2.8	2.7	**2.4**	2.8
	80	100	1909.3	3.3	2.9	2.4	2.7	2.7	2.5	**2.4**	2.7
	100	100	2362.8	3.1	2.6	2.0	2.4	2.3	**2.1**	**2.1**	2.3
C09	20	100	1106.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	40	100	2189.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	60	100	3410.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	80	100	4578.6	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	100	100	5430.5	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
C10	20	100	337.8	3.8	4.1	3.7	3.6	3.6	3.6	3.6	**3.5**
	40	100	642.8	3.4	3.8	3.3	3.3	3.1	**3.0**	**3.0**	**3.0**
	60	100	911.1	2.6	2.8	2.4	2.4	2.4	2.3	**2.2**	2.3
	80	100	1177.6	2.7	3.0	2.3	2.3	2.4	2.2	**2.1**	2.2
	100	100	1476.5	2.4	2.5	1.9	2.0	2.0	**1.8**	1.9	**1.8**
Average				1.77	1.80	1.67	1.63	1.63	1.56	1.54	**1.45**

parameters, and finally we normalize the running results of the algorithm. The experimental results are shown in Figure 8. When the value of parameters is set too small (such as $40-20$, $40-30$), the algorithm obtains a poor solution. When the value of parameters is set too large (such as $70-30$, $80-30$, $90-30$, etc.), the result is not satisfactory. When the parameters are set to $80-20$, $90-20$, $100-20$, $50-30$, and $60-30$, the results are better. When $k = 100$ and $childNum = 20$, the algorithm performs best.

B. EXPERIMENTATION SETTING

The machine configurations for experimentation are listed as follows, Ubuntu version 16.04 OS, Intel(R) Core(TM) i5 CPU with 8 cores (which individually have the clock time

TABLE 10. Experimental results on instance Nice [46].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
nice1	25	1000	1000	3.4	3.4	4.1	3.5	3.7	3.7	**3.3**
nice2	50	1000	1001	4.6	4.6	4.6	3.7	3.7	3.6	**2.5**
nice3	100	1000	1001	4.0	4.0	3.5	3.0	3.3	3.2	**2.4**
nice4	200	1000	1001	3.6	3.6	2.9	**2.2**	**2.2**	2.3	2.4
nice5	500	1000	1000	2.4	2.4	1.5	**0.8**	**0.8**	**0.8**	1.0
nice6	1000	1000	999	2.1	1.5	1.2	0.5	**0.3**	0.5	0.6
nice11t	1000	1000	1001	2.5	1.4	1.0	0.5	**0.4**	**0.4**	0.6
nice21t	1000	1000	1001	2.1	1.5	0.9	0.4	**0.3**	**0.3**	0.5
nice31t	1000	1000	1000	2.0	1.3	1.1	0.5	**0.3**	0.4	0.6
nice41t	1000	1000	1000	1.9	1.3	1.0	0.5	**0.4**	**0.4**	0.6
nice51t	1000	1000	1000	2.2	1.4	1.0	0.5	**0.4**	**0.4**	0.5
nice61t	1000	1000	1001	1.9	1.3	0.9	0.4	**0.1**	0.2	0.4
nice71t	1000	1000	1000	2.2	1.4	1.0	0.6	**0.3**	0.5	0.6
nice81t	1000	1000	1001	2.0	1.5	1.1	0.6	**0.3**	0.4	0.5
nice91t	1000	1000	1000	2.2	1.7	1.2	0.5	**0.3**	0.4	0.5
nice101t	1000	1000	1001	2.6	1.5	1.1	0.6	**0.3**	0.4	0.6
nice12t	2000	1000	1001	1.5	0.7	0.5	0.3	**0.2**	**0.2**	**0.2**
nice22t	2000	1000	1001	1.4	1.0	0.4	0.4	**0.1**	**0.1**	**0.1**
nice32t	2000	1000	1000	1.6	0.8	0.7	0.5	**0.2**	**0.2**	**0.2**
nice42t	2000	1000	1000	1.4	0.7	0.6	0.3	**0.2**	**0.2**	**0.2**
nice52t	2000	1000	1000	1.5	0.8	0.6	0.3	**0.1**	0.2	0.3
nice62t	2000	1000	1000	1.6	0.2	0.5	0.4	**0.1**	**0.1**	0.2
nice72t	2000	1000	1001	1.5	0.6	0.6	0.3	**0.1**	**0.1**	0.2
nice82t	2000	1000	1001	1.3	0.5	0.5	0.2	**0.1**	**0.1**	**0.1**
nice92t	2000	1000	1001	1.5	0.7	0.6	0.5	**0.2**	**0.2**	**0.2**
nice102t	2000	1000	1001	1.5	0.8	0.6	0.4	**0.2**	**0.2**	0.3
nice15t	5000	1000	1000	1.0	0.3	0.3	0.3	**0.1**	**0.1**	**0.1**
nice25t	5000	1000	1001	1.0	0.4	0.2	0.2	**0.0**	**0.0**	**0.0**
nice35t	5000	1000	1001	0.9	0.1	0.2	0.2	0.1	**0.0**	**0.0**
nice45t	5000	1000	1000	0.9	0.5	0.2	0.3	0.1	**0.0**	0.1
nice55t	5000	1000	1001	1.0	0.5	0.2	0.3	**0.1**	**0.1**	**0.1**
nice65t	5000	1000	1000	0.9	**0.1**	0.2	**0.1**	**0.1**	**0.1**	**0.1**
nice75t	5000	1000	1001	1.0	0.3	0.2	0.2	**0.0**	**0.0**	0.1
nice85t	5000	1000	1000	1.1	0.4	0.2	0.3	**0.1**	**0.1**	**0.1**
nice95t	5000	1000	1001	0.9	0.3	0.2	0.2	**0.0**	**0.0**	**0.0**
nice105t	5000	1000	1000	1.0	0.5	0.3	0.3	**0.1**	**0.1**	**0.1**
Aaverage				1.84	1.22	1.00	0.69	**0.54**	0.56	0.56

of 1.60GHz), and 8GB RAM. Three parameters: δ, k, and *childNum* involved in experimentation are respectively set as 0.03, 100, and 20.

C. BENCHMARK DATA

The effectiveness of the proposed algorithm HHA was tested on 8 widely-used benchmark datasets, including 737 instances of 2DSP problems. This data is composed of two types of problems: zero-waste and nonzero-waste. The details of this data are shown in Table 3, where *Dataset* denotes the specific benchmark dataset, *Instances* denotes the number of problem instances, n denotes the problem size, *zero-waste* denotes whether dataset is of type zero-waste, and *Data Source* denotes the source of dataset. Notably, it contains large datasets with $n > 10,000$, such as CX and ZDF.

D. RESULTS FOR 2DSP

The well-known algorithms, GRASP [16], SVC [17], ISA [26], SRA [27], HDA [35], HA [28], and CIBA [34], are selected for the performance comparison with EHA, because these algorithms perform well on most benchmark datasets. All of these algorithms were run 50 times on each instance, and each running time is limited to 1 minute. The running results are shown in Table 4 - Table 12. Instance represents the benchmark dataset. *bestH* represents the height found in each instance in a run, *LB* represents the lower bound of the height of the instance. *gap* is defined as: $gap = (bestH - LB)/LB * 100$, then *average_gap_rate* denotes the average value of gap over 50 runs. The best results obtained by all algorithms for each instance are shown in bold letters.

Table 4 - Table 7 is the experimental results of dataset C, N, NT, and CX. In the C dataset, the average gap rate of HHA was reduced by 32.76% compared to the previous optimal result. In the N dataset, the HHA obtains the optimal solution on 10 data. HHA achieves 49 best results on NT dataset, while GRASP, SVC, ISA, SRA, HDA, HA, and CIBA received 5, 5, 7, 17, 22, 31 and 26 best results, respectively. The HHA average gap is lower than other algorithms. From the performance results of these datasets, our algorithm performs well on the dataset of zero-waste, which also proves the effectiveness of the algorithm.

TABLE 11. Experimental results on instance Path [46].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
path1	25	1000	1001	4.1	4.1	4.1	4.1	4.2	**4.0**	3.4
path2	50	1000	1000	1.9	1.4	1.5	1.2	0.9	**0.3**	0.9
path3	100	1000	1000	2.7	2.2	2.3	2.5	2.3	**1.8**	**1.8**
path4	200	1000	1002	2.1	1.6	1.6	1.1	1.4	**1.2**	**1.2**
path5	500	1000	1000	3.4	2.2	2.0	1.6	1.4	**1.1**	1.2
path6	1000	100	1002	2.4	1.6	0.9	0.8	0.6	0.5	**0.4**
path11t	1000	1000	999	2.0	1.2	0.8	0.4	**0.2**	0.3	0.3
path21t	1000	1000	1001	1.7	0.9	0.5	0.4	**0.2**	**0.2**	**0.2**
path31t	1000	1000	1001	1.7	1.2	0.7	0.5	**0.2**	0.3	0.3
path41t	1000	1000	1000	1.6	0.9	0.6	0.3	**0.1**	**0.1**	0.2
path51t	1000	1000	1003	2.1	1.4	0.7	0.7	**0.3**	**0.3**	**0.3**
path61t	1000	1000	1002	1.6	1.1	0.8	0.3	**0.2**	**0.2**	0.3
path71t	1000	1000	999	2.0	1.3	0.9	0.5	**0.3**	0.4	0.5
path81t	1000	1000	1000	2.0	1.2	0.8	0.6	**0.2**	0.4	0.3
path91t	1000	1000	999	2.0	1.3	0.7	0.4	**0.3**	**0.3**	**0.3**
path101t	1000	1000	1002	1.6	0.9	0.6	0.4	0.2	**0.1**	**0.1**
path12t	2000	1000	1000	1.5	0.9	0.6	0.2	**0.1**	**0.1**	**0.1**
path22t	2000	1000	1002	1.4	0.8	0.5	0.2	0.1	**0.0**	**0.0**
path32t	2000	1000	1000	1.5	1.1	0.6	0.3	0.1	0.1	**0.0**
path42t	2000	1000	999	1.5	0.8	0.4	0.1	**0.0**	**0.0**	**0.0**
path52t	2000	1000	1002	1.6	1.0	0.6	0.4	**0.1**	**0.1**	**0.1**
path62t	2000	1000	1002	1.4	0.9	0.5	0.1	**0.0**	0.1	0.1
path72t	2000	1000	998	1.3	0.9	0.6	0.3	**0.1**	0.2	**0.1**
path82t	2000	1000	998	1.6	1.2	0.6	0.5	0.2	0.2	**0.1**
path92t	2000	1000	1001	1.6	0.9	0.7	0.2	**0.0**	0.1	0.1
path102t	2000	1000	1003	1.5	0.6	0.6	0.2	**0.0**	**0.0**	**0.0**
path15t	5000	1000	1000	1.0	0.2	0.3	0.1	0.1	**0.0**	**0.0**
path25t	5000	1000	998	1.1	0.5	0.3	0.3	0.1	**0.0**	**0.0**
path35t	5000	1000	1000	1.1	0.2	0.3	0.2	0.1	**0.0**	**0.0**
path45t	5000	1000	995	1.1	0.3	0.2	0.2	0.1	**0.0**	**0.0**
path55t	5000	1000	1004	1.2	0.1	0.2	0.2	**0.0**	**0.0**	**0.0**
path65t	5000	1000	1000	0.9	0.3	0.2	0.1	**0.0**	**0.0**	**0.0**
path75t	5000	1000	998	1.1	0.3	0.3	0.2	**0.1**	**0.1**	**0.1**
path85t	5000	1000	997	1.0	0.2	0.2	0.1	**0.0**	**0.0**	**0.0**
path95t	5000	1000	998	1.1	0.2	0.3	0.2	0.1	0.1	**0.0**
path105t	5000	1000	1002	1.1	0.2	0.2	0.3	0.1	**0.0**	**0.0**
Average				1.68	1.00	0.77	0.56	0.40	**0.35**	**0.35**

Table 8 is the experimental results of the 2SP dataset, which contains 38 instances consisting of ngcut, gcut, cgcut, and beng. HHA is better at gcut and beng. CIBA performs best in cgcut. CRASP performs best in ngcut and beng. The average gap of HHA is slightly larger than CRASP by 0.02. The reason for this may be that the selection strategy adopted by our algorithm does not perform well in small-scale instances. Some instances of 2SP have few rectangles, and the finer strategy performs better on this dataset.

Table 9 is the experimental results of the BWVM dataset. The dataset contains 500 examples, and the instance contains a number of rectangles ranging from 20 to 200. BWVM is a nonzero-waste dataset, which is divided into 10 classes. Each class contains 5 groups, each group with 10 instances, and each instance containing the same number of rectangles. HHA achieves the best results of the 25 groups, and the average gap is also optimal. HHA performed better on the dataset BWVM than other algorithms.

Table 10 and 11 are experimental results of the Nice&path dataset. HHA obtains 17 best results on the Nice dataset. The average gap HHA is superior to other algorithms with the exception of SRA. HHA's performance on this data is slightly inferior to SRA, probably because the shape and size of the rectangles in the Nice dataset are similar, and HHA's selection strategy may be slightly worse on this type of dataset. HHA gets 24 best results on the Path dataset, and its average gap is the best. Overall, HHA's performance on the Nice&path dataset is in line with the optimal algorithm.

Table 12 is the experimental results of the ZDF dataset. HHA achieves the best results on nine instances. The average gap is better than other algorithms except CIBA. The ZDF dataset contains many large instances, and CIBA performs better for instances of this type. On large datasets, the computation time of HHA's hierarchical search algorithm increases, so the search ability is slightly inferior to CIBA, which is where we need to improve in the future.

Table 13 compares the average gaps of the eight algorithms. From Table 13, we can see that HHA performs best in the datasets of C, NT, CX, BWVM, and Nice&Path. GRASP performs best in the 2SP dataset and CIBA performs best in the Nice&Path and ZDF datasets. Although HHA performs slightly worse in 2SP and ZDF datasets, from the average

TABLE 12. Experimental results on instance ZDF [47].

Instance				*average_gap_rate (%)*						
	n	*W*	*LB*	GRASP	SVC	ISA	HAD	SRA	CIBA	HHA
zdf1	580	100	330	0.9	0.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
zdf2	660	100	357	0.8	0.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
zdf3	740	100	384	0.8	0.3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
zdf4	820	100	407	0.7	0.2	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
zdf5	900	100	407	0.7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
zdf6	1532	3000	434	7.8	4.4	4.3	4.0	3.7	3.7	**3.4**
zdf7	2432	3000	5055	6.4	4.8	4.8	**3.4**	4.1	3.9	4.1
zdf8	2532	3000	5054	7.2	4.1	7.3	4.4	5.7	**2.5**	3.3
zdf9	5032	3000	5438	5.9	5.7	4.5	4.6	7.0	**0.5**	0.9
zdf10	2064	6000	5398	7.7	5.6	4.8	5.0	3.9	**0.3**	3.6
zdf11	7564	6000	5361	7.5	6.7	4.8	5.2	4.7	**0.4**	2.3
zdf12	10064	6000	5286	-	9.3	5.5	4.5	4.7	**0.0**	0.9
zdf13	15096	9000	5284	-	8.3	4.7	4.7	4.5	**0.0**	0.3
zdf14	25032	3000	5172	-	5.7	2.2	3.5	1.8	**0.0**	**0.0**
zdf15	50032	3000	5127	-	15.2	**0.0**	2.0	0.7	**0.0**	**0.0**
zdf16	75032	3000	5172	-	14.7	**0.0**	0.6	**0.0**	**0.0**	**0.0**
Average					5.34	2.67	2.62	2.56	**0.71**	1.17

TABLE 13. Comparison of average gap of 8 algorithms.

Algorithm	C	N	NT	CX	2sp	bwmv	Nice&path	ZDF	Average
GRASP	0.95	0.95	2.32	0.88	**2.68**	1.77	1.65	-	-
SVC	1.03	0.63	2.27	0.57	2.80	1.80	0.96	5.34	1.93
ISA	0.76	0.41	2.24	0.88	3.02	1.66	0.72	2.67	1.55
HAD	0.71	0.32	1.91	0.61	3.04	1.63	0.63	2.62	1.43
SRA	0.69	0.23	1.60	0.52	3.07	1.63	0.47	2.56	1.35
HA	0.51	0.14	1.37	0.45	3.04	1.55	0.48	1.44	1.12
CIBA	0.58	0.13	1.30	0.41	2.87	1.53	**0.46**	**0.71**	1.00
HHA	**0.39**	**0.12**	**0.97**	**0.18**	2.70	**1.45**	**0.46**	1.17	**0.93**

gap, the HHA algorithm performs better on most benchmark datasets, which indicates that the HHA algorithm is effective.

V. CONCLUSION

In this paper, a hybrid heuristic algorithm has been proposed for solving the 2DSPP that is based on the OF constraint. This algorithm includes three main improvements. Primarily, the scoring rule is built upon corner increments in order to increase the capability of selecting suitable rectangles. Secondly, two red-black trees have been created for the storage of rectangles in different orders, which aims to improve the unpacked rectangles retrieval. At the end, a hierarchical search has been embedded into the random local search with the goal of finding a desirable solution. Although experimental results have proved that the proposed algorithm outperformed the existing ones regarding strip wasting space, this algorithm is only for the 2DSPP with the OF constraint. In the future, constraints such as rectangle rotation, object shape guillotine, and many practical others will be considered into the 2DSPP. More interestingly, this approach can be modified to solve the variants of the 2DSPP.

REFERENCES

[1] J. F. Oliveira, A. Neuenfeldt, Jr., E. Silva, and M. Carravilla, "A survey on heuristics for the two-dimensional rectangular strip packing problem," *Pesquisa Operacional*, vol. 36, no. 2, pp. 197–226, Aug. 2016.

[2] R. Alvarez-Valdes, M. A. Carravilla, and J. F. Oliveira, *Cutting and Packing*. Cham, Switzerland: Springer, 2018, pp. 1–46.

[3] A. Lodi, S. Martello, and M. Monaci, "Two-dimensional packing problems: A survey," *Eur. J. Oper. Res.*, vol. 141, no. 2, pp. 241–252, Sep. 2002.

[4] M. Hifi, "Exact algorithms for the guillotine strip cutting/packing problem," *Comput. Oper. Res.*, vol. 25, no. 11, pp. 925–940, 1998.

[5] S. Martello, M. Monaci, and D. Vigo, "An exact approach to the strip-packing problem," *Informs J. Comput.*, vol. 15, no. 3, pp. 310–319, 2003.

[6] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi, "Exact algorithms for the two-dimensional strip packing problem with and without rotations," *Eur. J. Oper. Res.*, vol. 198, no. 1, pp. 73–83, 2009.

[7] M. A. Boschetti and L. Montaletti, "An exact algorithm for the two-dimensional strip-packing problem," *Oper. Res.*, vol. 58, no. 6, pp. 1774–1791, 2010.

[8] G. Belov and H. Rohling, "LP bounds in an interval-graph algorithm for orthogonal-packing feasibility," *Oper. Res.*, vol. 61, no. 2, pp. 483–497, 2013.

[9] J.-F. Côté, M. Dell'Amico, and M. Iori, "Combinatorial benders' cuts for the strip packing problem," *Oper. Res.*, vol. 62, no. 3, pp. 643–661, 2014.

[10] B. S. Baker, E. G. Coffman, Jr., and R. L. Rivest, "Orthogonal packings in two dimensions," *SIAM J. Comput.*, vol. 9, no. 4, pp. 846–855, 1980.

[11] B. Chazelle, "The bottomn-left bin-packing heuristic: An efficient implementation," *IEEE Trans. Comput.*, vol. C-32, no. 8, pp. 697–707, Aug. 1983.

[12] E. Hopper and B. C. H. Turton, "An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem," *Eur. J. Oper. Res.*, vol. 128, no. 1, pp. 34–57, 2001.

[13] E. K. Burke, G. Kendall, and G. Whitwell, "A new placement heuristic for the orthogonal stock-cutting problem," *Oper. Res.*, vol. 52, no. 4, pp. 655–671, 2004.

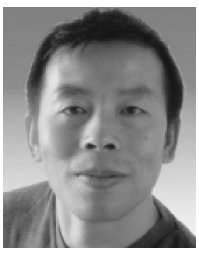
[14] D. Zhang, Y. Kang, and A. Deng, "A new heuristic recursive algorithm for the strip rectangular packing problem," *Comput. Oper. Res.*, vol. 33, no. 8, pp. 2209–2217, 2006.

[15] D.-F. Zhang, C. Sheng-Da, and L. Yan-Juan, "An improved heuristic recursive strategy based on genetic algorithm for the strip rectangular packing problem," *Acta Automatica Sinica*, vol. 33, no. 9, pp. 911–916, 2007.

[16] R. Alvarez-Valdés, F. Parreño, and J. M. Tamarit, "Reactive GRASP for the strip-packing problem," *Comput. Oper. Res.*, vol. 35, no. 4, pp. 1065–1083, 2008.
[17] G. Belov, G. Scheithauer, and E. A. Mukhacheva, "One-dimensional heuristics adapted for two-dimensional rectangular strip packing," *J. Oper. Res. Soc.*, vol. 59, no. 6, pp. 823–832, 2008.
[18] S. Jakobs, "On genetic algorithms for the packing of polygons," *Eur. J. Oper. Res.*, vol. 88, no. 1, pp. 165–181, 1996.
[19] D. Liu and H. Teng, "An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles," *Eur. J. Oper. Res.*, vol. 112, no. 2, pp. 413–420, 1999.
[20] A. Bortfeldt, "A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces," *Eur. J. Oper. Res.*, vol. 172, no. 3, pp. 814–837, 2006.
[21] J. A. Bennell, L. S. Lee, and C. N. Potts, "A genetic algorithm for two-dimensional bin packing with due dates," *Int. J. Prod. Econ.*, vol. 145, no. 2, pp. 547–560, 2013.
[22] K. A. Dowsland, "Some experiments with simulated annealing techniques for packing problems," *Eur. J. Oper. Res.*, vol. 68, no. 3, pp. 389–399, 1993.
[23] E. K. Burke, G. Kendall, and G. Whitwell, "A simulated annealing enhancement of the best-fit heuristic for the orthogonal stock-cutting problem," *Informs J. Comput.*, vol. 21, no. 3, pp. 505–516, 2009.
[24] S. Hong, D. Zhang, H. C. Lau, X. Zeng, and Y.-W. Si, "A hybrid heuristic algorithm for the 2D variable-sized bin packing problem," *Eur. J. Oper. Res.*, vol. 238, no. 1, pp. 95–103, 2014.
[25] M. Iori, S. Martello, and M. Monaci, *Metaheuristic Algorithms for the Strip Packing Problem*. New York, NY, USA: Springer, 2003, pp. 159–179.
[26] S. C. H. Leung, D. Zhang, and K. M. Sim, "A two-stage intelligent search algorithm for the two-dimensional strip packing problem," *Eur. J. Oper. Res.*, vol. 215, no. 1, pp. 57–69, 2011.
[27] S. Yang, S. Han, and W. Ye, "A simple randomized algorithm for two-dimensional strip packing," *Comput. Oper. Res.*, vol. 40, no. 1, pp. 1–8, 2013.
[28] D. Zhang, Y. Che, F. Ye, Y.-W. Si, and S. C. H. Leung, "A hybrid algorithm based on variable neighbourhood for the strip packing problem," *J. Combinat. Optim.*, vol. 32, no. 2, pp. 513–530, 2016.
[29] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3D bin packing problem with deep reinforcement learning method," 2017, *arXiv:1708.05930*. [Online]. Available: https://arxiv.org/abs/1708.05930
[30] L. Wei, W.-C. Oon, W. Zhu, and A. Lim, "A skyline heuristic for the 2D rectangular packing and strip packing problems," *Eur. J. Oper. Res.*, vol. 215, no. 2, pp. 337–346, 2011.
[31] L. Wei, Q. Hu, S. C. H. Leung, and N. Zhang, "An improved skyline based heuristic for the 2D strip packing problem and its efficient implementation," *Comput. Oper. Res.*, vol. 80, pp. 113–127, Apr. 2017.
[32] D. Zhang, L. Wei, S. C. H. Leung, and Q. Chen, "A binary search heuristic algorithm based on randomized local search for the rectangular strip-packing problem," *Informs J. Comput.*, vol. 25, no. 2, pp. 332–345, 2013.
[33] Y. Wang and L. Chen, "Two-dimensional residual-space-maximized packing," *Expert Syst. Appl.*, vol. 42, no. 7, pp. 3297–3305, 2015.
[34] Z. Chen and J. Chen, "An effective corner increment-based algorithm for the two-dimensional strip packing problem," *IEEE Access*, vol. 6, pp. 72906–72924, 2018.
[35] B. Chen, Y. Wang, and S. Yang, "A hybrid demon algorithm for the two-dimensional orthogonal strip packing problem," *Math. Problems Eng.*, vol. 2015, Dec. 2014, Art. no. 541931.
[36] A. M. Gomes and J. F. Oliveira, "Solving irregular strip packing problems by hybridising simulated annealing and linear programming," *Eur. J. Oper. Res.*, vol. 171, no. 3, pp. 811–829, 2006.
[37] D. Zhang, Y. Peng, and S. C. H. Leung, "A heuristic block-loading algorithm based on multi-layer search for the container loading problem," *Comput. Oper. Res.*, vol. 39, no. 10, pp. 2267–2276, 2012.
[38] E. Hopper, "Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods," Ph.D. dissertation, School Eng., Univ. Wales, Cardiff, NS, USA, 2000.
[39] E. Pinto and J. F. Oliveira, "Algorithm based on graphs for the non-guillotinable two-dimensional packing problem," in *Proc. 2nd ESICUP Meeting*, Southampton, U.K., 2005.
[40] N. Christofides and C. Whitlock, "An algorithm for two-dimensional cutting problems," *Oper. Res.* vol. 25, no. 1, pp. 30–44, Jan./Feb. 1977.
[41] J. E. Beasley, "Algorithms for unconstrained two-dimensional guillotine cutting," *J. Oper. Res. Soc.*, vol. 36, no. 4, pp. 297–306, Apr. 1985.
[42] B.-E. Bengtsson, "Packing rectangular pieces—A heuristic approach," *Comput. J.*, vol. 25, no. 3, pp. 353–357, 1982.
[43] J. E. Beasley, "An exact two-dimensional non-guillotine cutting tree search procedure," *Oper. Res.*, vol. 33, no. 1, pp. 49–64, 1985.
[44] J. O. Berkey and P. Y. Wang, "Two-dimensional finite bin-packing algorithms," *J. Oper. Res. Soc.*, vol. 38, no. 5, pp. 423–429, May 1987.
[45] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," *Manage. Sci.*, vol. 44, no. 3, pp. 388–399, 1998.
[46] C. L. Mumford-Valenzuela, J. Vick, and P. Y. Wang, "Heuristics for large strip packing problems with guillotine patterns: An empirical study," in *Metaheuristics: Computer Decision-Making*. Boston, MA, USA: Springer, 2004, pp. 501–522.
[47] S. C. H. Leung and D. Zhang, "A fast layer-based heuristic for non-guillotine strip packing," *Expert Syst. Appl.*, vol. 38, no. 10, pp. 13032–13042, 2011.

MENGFAN CHEN received the B.E. degree in computer science and technology from Fujian Normal University, in 2017. She is currently pursuing the master's degree with the School of Information and Computer Science, Xiamen University. Her current research interests include computational intelligence, data mining, big data, and combinatorial optimization.

KAI LI received the B.E. degree in computer science and technology from the Southwest University of Science and Technology, in 2017. He is currently pursuing the master's degree with the School of Information and Computer Science, Xiamen University. His current research interests include artificial intelligence, computational intelligence, and data mining.



DEFU ZHANG received the bachelor's and master's degrees in computational mathematics from Xiangtan University, in 1996 and 1999, respectively, and the Ph.D. degree in computer software and theory from the School of Computer Science, Huazhong University of Science and Technology. He was a Senior Researcher with Shanghai Jinxin Financial Engineering Academy, from 2002 to 2003. He was a Postdoctoral Researcher with the Longtop for Financial Data Mining Group, from 2006 to 2008. From 2008 to 2016, he visited Hong Kong City University, the University of Wisconsin_Madison, and Macau University. Besides, he developed an Internet Plus Big Data Platform (http://www.pzcnet.com). He is currently a Professor with the Department of Computer Science, Xiamen University. He supervised the ACM/ICPC Team, Xiamen University. He has authored over 40 journal articles. His research interests include computational intelligence, data mining, big data, cloud computing, online decision optimization, and food security. He was a recipient of three gold medals and eight silver medals, from 2004 to 2009, and he took part in the World Final Contest, in 2007.

LING ZHENG received the M.Sc. and Ph.D. degrees from the University of Aberystwyth, U.K. He is currently a Research Fellow with the School of Information Science and Engineering, Xiamen University, China. His research interests include innovative technologies for data processing and analysis, computer vision, and pattern recognition.

XIN FU received the Ph.D. degree in computer science from Aberystwyth University, U.K., in 2010. She is currently an Associate Professor with the School of Management, Xiamen University, China. Her research interests include decision support systems, fuzzy and qualitative modeling, business intelligence, and sharing economy. Her research has been published in journals, including *Information and Management*, *Decision Support Systems*, the IEEE TRANSACTIONS ON FUZZY SYSTEMS, *Pattern Recognition*, the *Journal of Cheminformatics*, and so on.