Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2022.016075
Article

Tech Science Press

A Time-Efficient and Exploratory Algorithm for the Rectangle Packing Problem

Mohammad Bozorgi[1], Morteza Mohammadi Zanjireh[1,*], Mahdi Bahaghighat[1] and Qin Xin[2]

[1]Computer Engineering Department, Imam Khomeini International University, Qazvin, Iran
[2]Faculty of Science and Technology, University of the Faroe Islands, Faroe Islands
*Corresponding Author: Morteza Mohammadi Zanjireh. Email: Zanjireh@eng.ikiu.ac.ir
Received: 22 December 2020; Accepted: 29 April 2021

Abstract: Today, resource waste is considered as one of the most important challenges in different industries. In this regard, the Rectangle Packing Problem (RPP) can affect noticeably both time and design issues in businesses. In this study, the main objective is to create a set of non-overlapping rectangles so that they have specific dimensions within a rectangular plate with a specified width and an unlimited height. The ensued challenge is an NP-complete problem. NP-complete problem, any of a class of computational problems that still there are no efficient solution for them. Most substantial computer-science problems such as the traveling salesman problem, satisfiability problems (sometimes called propositional satisfiability problem and abbreviated SAT or B-SAT), and graph-covering problems are belong to this class. Essentially, it is complicated to spot the best arrangement with the highest rate of resource utilization by emphasizing the linear computation time. This study introduces a time-efficient and exploratory algorithm for the RPP, including the lowest front-line strategy and a Best-Fit algorithm. The obtained results confirmed that the proposed algorithm can lead to a good performance with simplicity and time efficiency. Our evaluation shows that the proposed model with utilization rate about 94.37% outperforms others with 87.75%, 50.54%, and 87.17% utilization rate, respectively. Consequently, the proposed method is capable to of achieving much better utilization rate in comparison with other mentioned algorithms in just 0.023 s running-time, which is much faster than others.

Keywords: Rectangle packing problem; best-fit algorithm; lowest front-line strategy

1 Introduction

Decreasing wasting of resources affecting industries and businesses can be very advantageous [1–7]. The rectangle packing problem (RPP) is one of the most practical issues in the industrial sector. The RPP has many uses in various industries, including cutting leather, wood, metals, paper, glass, cloth, and paperboard [8–11], for designing and manufacturing cars, fighters, and ships [12–14], very-large-scale integration (VLSI) design [11,15–17], and even in similar issues like timing and replacement [18,19]. It also has uses in the assignment and timing of the radio frequency spectrum [18].

In the RPP, there exists a set of rectangles with specific dimensions, and the aim is to arrange these rectangles in a rectangular page with a fixed width and an unlimited height without overlapping, provided that the packing is orthogonal [20]. It means that the sides of the rectangles that are inserted into the rectangular page are parallel to the sides of the central page's rectangle [16,21]. Also, the problem may not be guillotine-cut. That means that the replaced rectangles cannot be placed in several groups separated from the others by vertical or horizontal lines [8,17]. Rotation of the rectangles is acceptable, but the rotation angle could be only 90 degrees [22,23].

In addition to waste reduction, increasing the utilization rate is one of the fundamental challenges in the industry. The utilization rate equals the sum of the area of the input rectangles ($\sum_{i=1}^{m} w_i h_i$) divided by the area of the main page ($width \times height$). It is represented by U [22,24,25] and can be calculated by Eq. (1):

$$\tau_\tau = \frac{\sum_{i=1}^{m} w_i h_i}{(width \times height)} \tag{1}$$

In this equation, *width* represents width of the main page, and *height* represents height of the main page, (w_i) and (h_i) are the width and height of the replaced rectangles, respectively, and m is the number of these rectangles [22].

In the past decades, various related algorithms have been presented due to its crucial role and importance in the widespread applications. Despite a wide range of the RPP applications, the efficient implementation is still a complicated task. The most fundamental challenge is that we should solve an NP-complete set problem [20,24,26–29]. Therefore, with an increase in the problem scale, achieving a practical solution with an appropriate run time could be difficult or even impossible.

The rest of this article is organized as follows. In the second section, the related works about the rectangle packing context were discussed. In the third section, the methodology with the proposed algorithm was presented in full detail. In the fourth section, achieved results were presented, and ultimately the fifth section dealt with the conclusion.

2 Related Works

To solve the RPP, one of the most favorite algorithms that has attracted many researchers is the Best-Fit algorithm. It is a multi-layered algorithm in which the layers are created gradually as the rectangles are inserted. The rectangle places into the higher level, if it cannot be placed in a layer. In conclusion, the rectangle will always be located in a layer that has less waste after insertion [8].

Hu et al. [11] have presented an algorithm based on Best-Fit and BL algorithms. Their algorithm has all of the properties of the two mentioned algorithms. This algorithm categorizes the rectangles first and then places the created categories one by one. This method is suitable for rectangles with different sizes but with some restrictions to the number of input rectangles. The time complexity of this algorithm is O ($mMlogM$), in which m and M could be equal to the number of input rectangles.

Liu et al. [22] presented a solution for the RPP by combining the lowest front-line algorithm with the genetic optimization algorithm. Arrangement of the rectangles from the lowest point possible to the highest point is a prominent feature of this algorithm that reduces space waste. It also utilizes a smart function to cross local optimum modes. The long-running time of this algorithm for identifying and improving various modes is its fundamental challenge.

Huang et al. [30] presented an algorithm based on two basic algorithms, including the Best-Fit algorithm and particle swarm optimization (PSO) algorithm. They compared their method with the presented classical methods and achieved more efficient results. However, the fundamental challenge of their approach is the long-running time of the algorithm.

Chazelle [31] introduced an algorithm called Bottom Left Fill (BLF). The number of possible modes for inserting a rectangle was decreased in the BLF algorithm. While in the BL algorithm, it is impossible to use the leftover spaces from placing the rectangles, in the BLF algorithm, the rectangle is placed at the lowest point possible. The BLF algorithm uses unused spaces. The time complexity of this algorithm is $O(n2)$, where n is the number of inputs. This means that this algorithm is not scalable well.

Liu et al. [32] developed an algorithm for the RPP based on two algorithms, including the BL algorithm and the genetic algorithm (GA). They improved the initial BL algorithm because it does not identify some of the modes. As the mode in which the rectangles are alternately big and small, the big rectangles have ascending sizes, and the small ones have descending scales. The decisive point of this algorithm is the use of an intelligent crowd function for the GA (genetic algorithm). The running order of this algorithm is $O(n2)$.

3 Structure

In this section, at first, two algorithms including the Best-Fit decreasing height algorithm and the lowest front-line strategy are introduced. Then, the researchers present the proposed model in this research in its details.

3.1 The Lowest Front-Line Strategy

An acceptable paper size is US Letter ($8.5'' \times 11''$ or $21.59\ \text{cm} \times 27.94\ \text{cm}$). All margins—top, bottom, left, and right—are set to $1''$ (2.54 cm). The paper must be single column, single-spaced, except for the headings as outlined below. The acceptable font is Times New Roman, 11 pt., except for writing special symbols and mathematical equations.

The lowest front-line strategy is designed based on the BL algorithm and is defined as follows [22]:

- The first step: The algorithm starts with the initial quantification of the front line. In the beginning, the front lines set contained only one line, and that is the horizontal line at the bottom of the page.
- The second step: Locating the rectangle in a place using the lowest front-line strategy. p_i is the position where the $i-th$ rectangle is placed. First, we select and check one line of the front line set to see whether the width of the rectangle that is going to place is equal to or smaller than the width of the selected line. If the conditions are true, that rectangle will be placed at the top left of the page. If the conditions are not true, the height of the lowest line is increased until it reaches the second lowest front line, and the conditions will be rechecked. This process repeats until one line of the front line set gets selected. In this step, if several lines exist with the lowest heights, the selection will be based on the X-axis. If height is increased from a lower line to a higher line, they will be merged.
- The third step: In this step, the front line will be updated. Some of the old lines will be converted to new lines, and new lines will be added to the front line set. As the points mentioned in the second step, the adjacent lines with equal height must be merged to form a single line.
- The fourth step: If all rectangles are placed, the algorithm ends, otherwise a new rectangle will be selected, and the process will continue to the second step.

Fig. 1 demonstrates an example of the lowest front-line strategy. P_1 will be selected for $\{A_1A_5\}$, and after that, the front line will be $\{E_1E_2, A_2A_5\}$. In the same way, p_2 will be selected for $\{A_2A_5\}$, and after the placement, the front line will be $\{E_1E_2, B_1B_3, A_3A_5\}$. Similarly, after placing the p_3 rectangle in $\{A_3A_5\}$, the front line will be $\{E_1E_2, B_1B_3, C_1C_2, A_4A_5\}$. In this point, the lowest line is $\{A_4A_5\}$, and it does not have enough space to place P_4. So, the height of $\{A_4A_5\}$ was increased until it reaches $\{B_1B_3\}$, the

second-lowest front line. Finally, the $\{B_1B_3\}$ line will be selected to place the P_4 rectangle, and the front line is $\{E_1E_2, D_1D_2, B_2B_3, c_1c_2, B_4B_5\}$ at the end [22].

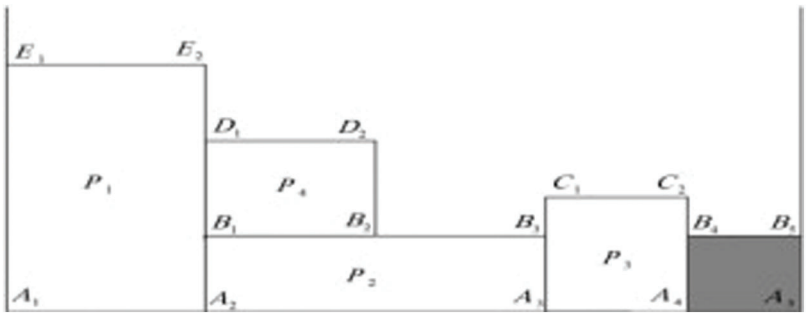

Figure 1: The lowest front-line strategy

The lowest front-line strategy does not sort the input rectangles before placing them. Actually, this is a challenge for this algorithm. It increases page height and decreases the utilization rate. This algorithm always tries to find the lowest and the leftmost point possible to place the rectangles. The way the height increases in this algorithm is its positive point.

3.2 The Best-Fit Decreasing Height Method

In the Best-Fit decreasing height algorithm, the rectangles are first sorted in decreasing order. Then, the biggest rectangle will be placed at the bottom-left corner, and the first layer will be created. To place the next rectangles, if the rectangles could not be placed in the created spaces of the created layer, we create a layer above the highest layer and place the rectangle in it. If there are several layers with enough space to place the rectangles, a layer will be selected with less waste after the rectangle placement [33,34]. The time complexity of this algorithm is O(nlogn) [35].

The Best-Fit decreasing height algorithm is layered well, and it is suitable for non-guillotine cut issues. Finding the best position with the lowest waste is waste is a noticeable advantage of this algorithm. Also, due to sorting the inputs, this algorithm has a better utilization rate compared to the Best-Fit algorithm without sorting. In this algorithm, we cannot place a rectangle between two layers, and this causes wasting. Therefore, we can place the rectangle between the layers in the proposed algorithm.

This study presents an algorithm with the properties of both Best-Fit decreasing height algorithm and the lowest front-line strategy. This algorithm, similar to the Best-Fit method, always tries to find the best position for the rectangle. However, this algorithm aims to decrease the total height by increasing the height of the layers, inspired by the lowest front-line strategy.

3.3 The Proposed Algorithm

The steps of the proposed algorithm are as follows:

The first step: At first, the algorithm starts with initial quantification. The height of this page is zero initially, and its final value is specified at the end of the algorithm. In this algorithm, there exist a set of lines called the front line for saving empty spaces. In the beginning, the front line is the bottom borderline of the page. The information of the input rectangles such as width, height, and the number of each one will be received at this stage of the algorithm.

In this algorithm, an array was used to reserve empty spaces. The first row of this front-line array is the starting point of the line, the second row of the front-line shows the endpoint of the line, the third row of the front-line array represents the distance of this line from the lowest front line, which is the bottom borderline of the page, and the fourth row of the front-line array shows the height of the next front-line after this line. With these four points, we can imagine a rectangle. If the value of the fifth row of the front-line array is 1, it means that this line is not occupied, and when a rectangle is placed, this value will become −1. The fifth row of the array is added to adhere to the not-overlapping assumption. When placing rectangles into a line using the fifth row of the front-line array, it will be checked whether it is full or empty.

The second step: In this step of the algorithm, the rectangles with more height than width will be rotated 90 degrees. It means that the height and the width will be replaced.

The third step: In this step, the rectangles will be sorted in decreasing height size order.

The fourth step: A rectangle will be selected from the beginning of the inputs list, and then, we search among the front-lines set for a place with enough space for the rectangle. Then, among the selected lines, we choose a line that has less waste. At this point, there are three modes:

- If a line is found among the front-lines set with sufficient width and height for placing the rectangle, we select it; and if there are several lines for placing, we choose the one that wastes less space.
- If we cannot find a line with the required height and width, we start searching among the longest front-lines for a line that has the required width for placing the rectangle. Also, if there are several lines, we choose the one that has less width and increases the height of the page lesser.
- If we cannot find a position to place the rectangle in the two predicted modes mentioned above; and if we did not find any line that has enough space for placing the rectangle, then we create a new layer above the longest front-line and place the rectangle in the bottom-left of the created layer and increase the height of the page.

The fifth step: In this step, we update the front-line and empty spaces, and convert some of the old lines into new lines and also add new lines. In addition, we must identify all of the empty spaces. Then, we try to identify all of the empty houses that have common borders. In order to make the most of the wasted space available, if one or more than two houses have common borders, it must be merged with the house that creates larger free space after merging compared to other houses. This step plays an essential role in reducing waste.

The sixth step: In this step, we check the ending of the algorithm. If there is not a rectangle in the input list to place, the algorithm ends. Otherwise, we go to the beginning of the fifth step and continue the algorithm again. In the end, we calculate the utilization rate of the algorithm. Fig. 2 shows the flowchart of the algorithm process.

```
Read(n);
Read(width);
Read(rectangle(3,n));
Height=0;
Sum=0;
Create(frontLine(5,1));
frontLineCounter=1;
rotation(rectangle(3,n));
Sort(rectangle(3,n));
for i=1:1:n
    FindFrontLine(frontLine(5,frontLineCounter));
    Insert(rectangle(3,i));
    Update(frontLine(5,frontLinecounter),frontLineCounter);
    sum=sum+rectangle(1,i)*rectangle(2,i);
end
```

Figure 2: The semi-code of the proposed algorithm

4 The Experimental Results

In this study, the MATLAB programming language was used in a PC with Intel(R) Core(TM) i7 and 12 GB internal storage. The data was deployed using the Liu algorithm [22] for the evaluation of the algorithms. Tab. 1 shows these data, and also, the width of the rectangle page is 400 units. Then, the achieved results of the proposed algorithm were compared with the Liu study [22], Best-Fit, and front-line.

Table 1: Basic data presented in [22]

Number	1	2	3	4	5	6	8	9	10	11	12	13	14	15	16	17	18	19	20
Width	25	18	79	121	29	64	48	11	46	4	5	6	7	8	9	10	11	10	50
Height	36	24	84	30	48	98	59	17	121	22	41	72	25	65	24	11	36	30	61
Quality	4	5	3	4	11	2	3	2	2	1	2	2	2	2	3	2	2	3	2

Tab. 2 shows the results of all mentioned algorithms. It indicates that the proposed algorithm has the best performance about 94.37% compared to 87.75%, 50.54%, and 87.17% for other algorithms. Besides, our method reached the utilization rate of 94.37% in just 0.023 s, well below others. Whereas the method presented in [22] reached the utilization rate average of 85.51% after 300 times repeating the algorithm and 26.3 s, and at best, it has reached the rate of 87.75%.

Table 2: Comparison of the proposed algorithm implementation results with the Best-Fit, front-line, and Liu algorithms

Algorithm	The proposed	Liu	Front line	Best-Fit
Utilization	94.37%	87.75%	50.54%	87.17%
Run time(s)	0.023	26.3	0.042	0.031

Fig. 3 plots the utilization rate of the three Best-Fit decreasing height, the lowest front-line strategy, and the proposed algorithms. In this figure, the utilization rate percentage of every algorithm has been drawn after placing each rectangle. The horizontal axis represents the number of placed rectangles, and the vertical axis represents the utilization rate percentage.

Fig. 4 shows the run time *vs.* number of rectangles of the three algorithms, including Best-Fit decreasing height, the lowest front-line strategy, and the proposed algorithm. In this figure, the best running time belongs to the proposed algorithm.

4.1 The Effect of the Sorting and Rotation on the Utilization Rate

In this section, we analyze the effects of the sorting and rotation on the utilization rate. Sorting the input rectangles has a significant impact on the utilization rate. We sorted the input rectangles in descending order of height, width, and area of the rectangles, and then we ran the algorithm. The results of running the algorithm show that sorting in descending order of height has better results. Tab. 3 shows the results of this comparison, and Fig. 5 shows the utilization rate for various sorting modes.

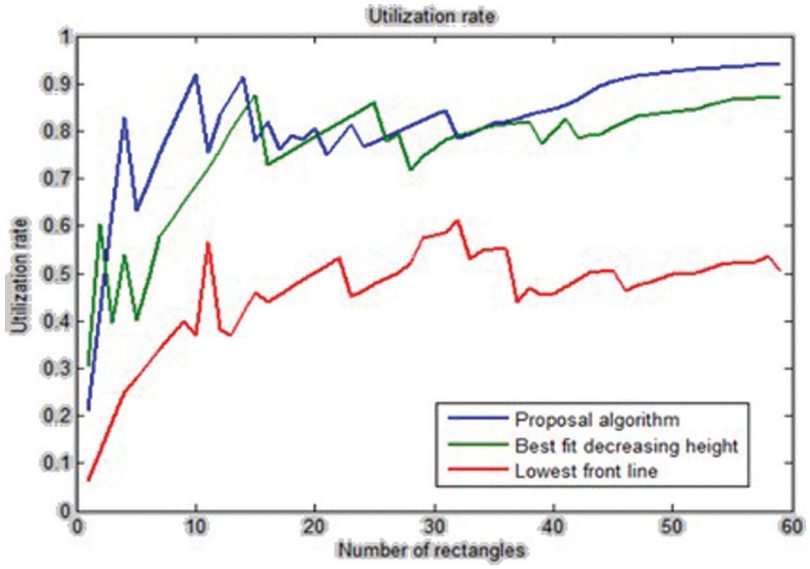

Figure 3: The utilization rate of the three Best-Fit decreasing height, the lowest front-line strategy, and the proposed algorithms

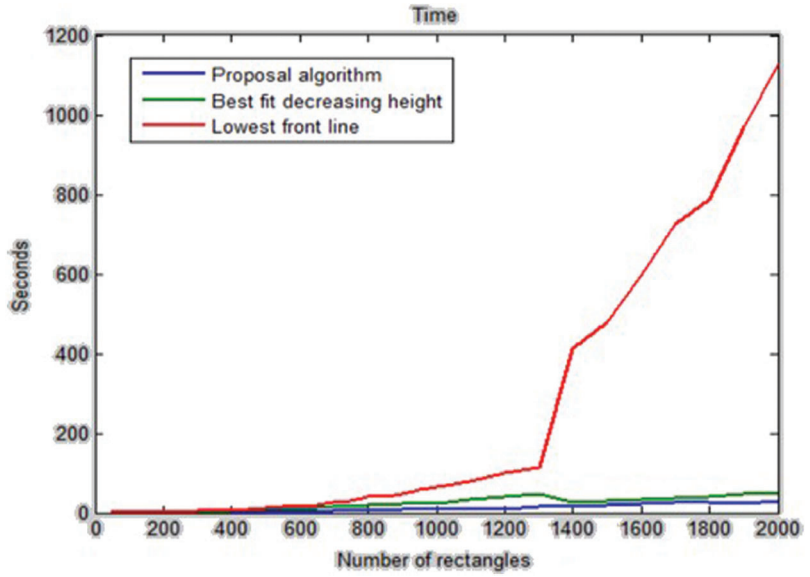

Figure 4: The running time of the Best-Fit decreasing height algorithm, the lowest front-line strategy, and the proposed algorithm

The rotation step was removed to run the algorithm again. Then, the new results were compared with the algorithm while considering the rotation. The results showed that the rotation has a positive effect on the utilization rate. Tab. 4 shows the results of this experiment. Fig. 6 draws a comparison between running the algorithm with and without the rotation.

Table 3: The results of running the proposed algorithm with different sorting

Sort type	None	Width	Area	Height
Utilization rate	69.25%	72.22%	87.89%	94.37%
Runtime(s)	0.30	0.036	0.036	0.023

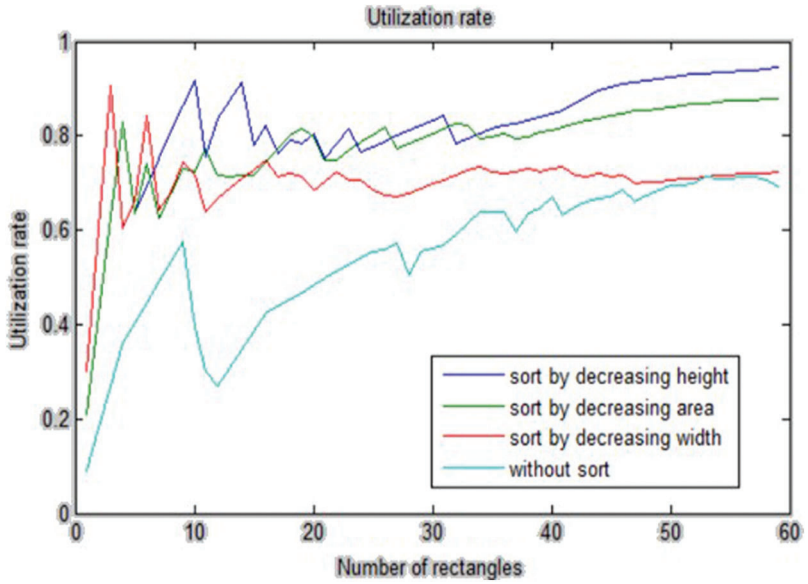

Figure 5: The utilization rate of the proposed algorithm with different sorting

Table 4: The results of the proposed algorithm implementation with removing the rotation step and sorting

Impact of step	Run-time(s)	Utilization rate	Deleted step
14.75%	0.032	82.24%	**Rotation**
36.27%	0.030	69.25%	**Sort**

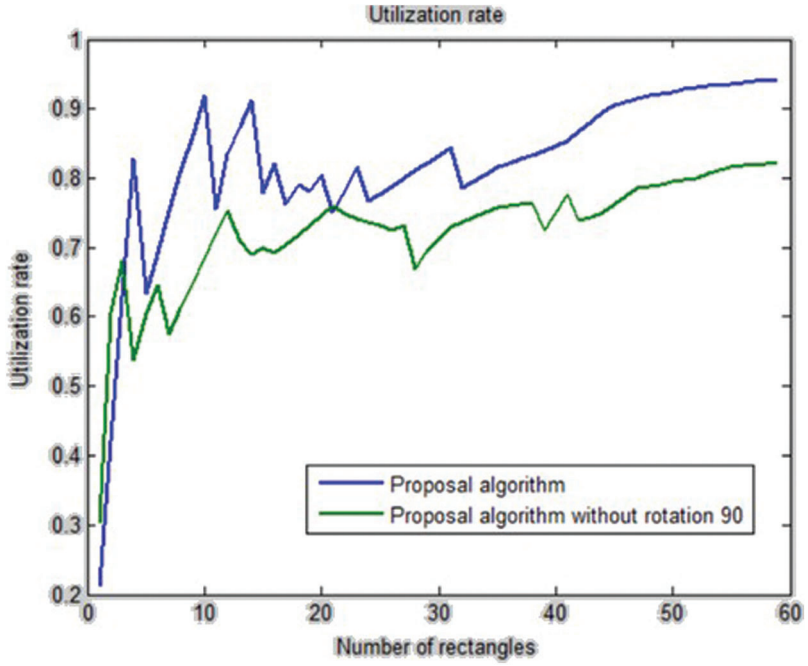

Figure 6: The utilization rate of the proposed algorithm with & without the rotation step

4.2 Comparing the Algorithm with Random Data

Comparing the algorithm with low-volume data and also, in some cases, comparing the algorithm with high-similarity data, is one of the challenges of the presented algorithms. Practically, there are various data with different sizes in the industry. Therefore, in this section, the proposed algorithm, the lowest front-line strategy, and the Best-Fit algorithm were analyzed by producing random data. The obtained results show that the utilization rate of the proposed algorithm is better than the other two algorithms. In addition, it has a shorter running time. Tab. 5 shows the utilization rate and the running time of these algorithms. Fig. 7 indicates the utilization rate of the Best-Fit, front-line, and proposed algorithms with random data. Also, Fig. 8 shows the running time of these three algorithms.

4.3 The Time Complexity

The proposed method has the polynomial-time complexity. As a result, it needs maximum O(m) time for searching between m front-lines and finding the proper line to place the rectangle. This task must be done for all of the rectangles, and the total number of rectangles is n. Therefore, the time complexity of this algorithm is generally linear in time order and equals O(nm). The results of the implementation indicate that the value of m is always less than n. It means that at the end, the total number of the front-lines is always less than the total number of the rectangles. In Tab. 5, the total number of rectangles column is n, the total number of front-lines column is m, and m is always less than n. Fig. 9 demonstrates how the number of front-lines increases per number of input data.

Table 5: The results of the utilization rate of the proposed algorithm by producing random data

Run-time(s)			Utilization rate			Amount of front-line in the proposed algorithm	Amount of rectangles	N
The proposed algorithm	Best-Fit	Front line	The proposed algorithm	Best-Fit	Front line			
0.061439	0.113711	0.135782	0.9680	0.9044	0.8213	142	283	50
0.179677	0.268977	0.365532	0.9759	0.9261	0.8076	294	560	100
0.268492	0.679887	0.940690	0.9722	0.9212	0.8018	379	808	150
0.561501	1.371588	1.670308	0.9720	0.9088	0.7620	498	1102	200
0.784825	1.683756	3.098013	0.9720	0.9355	0.7979	616	1444	250
1.136259	2.152571	3.692223	0.9735	0.9459	0.7850	672	1684	300
1.687331	3.185944	5.856734	0.9718	0.9281	0.8020	747	1972	350
1.792151	4.626413	6.491700	0.9737	0.9384	0.7742	836	2117	400
2.338938	6.179962	8.555913	0.9682	0.9335	0.7860	924	2380	450
2.532763	7.593715	12.461293	0.9705	0.9581	0.7893	1026	2738	500
2.609368	8.936252	16.055307	0.9695	0.9454	0.7790	1100	3012	550
3.789860	7.678264	17.209783	0.9737	0.9408	0.7863	1126	3260	600
3.503998	12.119956	21.050525	0.9750	0.9314	0.8077	1235	3497	650
5.143389	14.592887	27.870559	0.9710	0.9426	0.8253	1328	3955	700
5.692376	16.248161	30.576511	0.9693	0.9485	0.7703	1348	4110	750
6.548756	20.146549	38.626904	0.9768	0.9537	0.7935	1448	4534	800
5.552323	20.865130	41.414833	0.9673	0.9508	0.7987	1464	4592	850
7.132780	22.115729	48.515116	0.9740	0.9447	0.7807	1503	4819	900
8.174483	26.831960	57.326876	0.9691	0.9548	0.7943	1501	5244	950
9.152594	23.973368	65.242514	0.9678	0.9425	0.7896	1744	5763	1000
10.860228	34.939327	78.368714	0.9655	0.9531	0.7831	1644	6055	1100
9.508915	40.207071	99.702676	0.9675	0.9550	0.7966	1730	6615	1200
14.972989	46.728704	113.762547	0.9657	0.9625	0.7896	1909	7044	1300
16.997029	26.938338	411.017108	0.9708	0.9479	0.7981	1984	7671	1400
18.103096	28.565729	478.56225	0.9695	0.9673	0.8027	2008	8164	1500
21.743997	33.262568	598.293248	0.9681	0.9603	0.7964	2194	8796	1600
24.726140	38.004280	723.358521	0.9715	0.9596	0.7852	2232	9415	1700
25.643730	42.053160	786.529926	0.9706	0.9625	0.8030	2383	9853	1800
24.416149	45.981673	971.096964	0.9749	0.9517	0.7884	2750	10555	1900
31.034617	50.574680	1128.696616	0.9659	0.9603	0.7944	2623	11030	2000

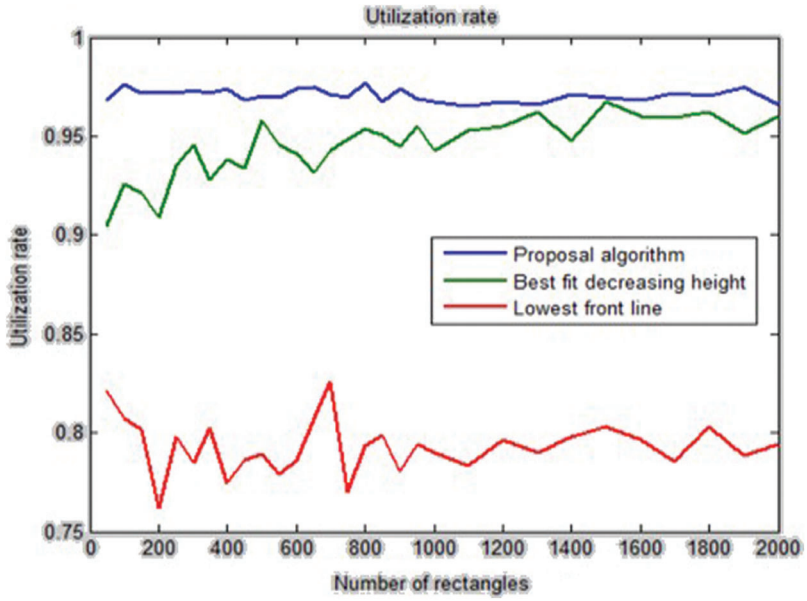

Figure 7: The utilization rate of the Best-Fit, front-line, and proposed algorithms with random data

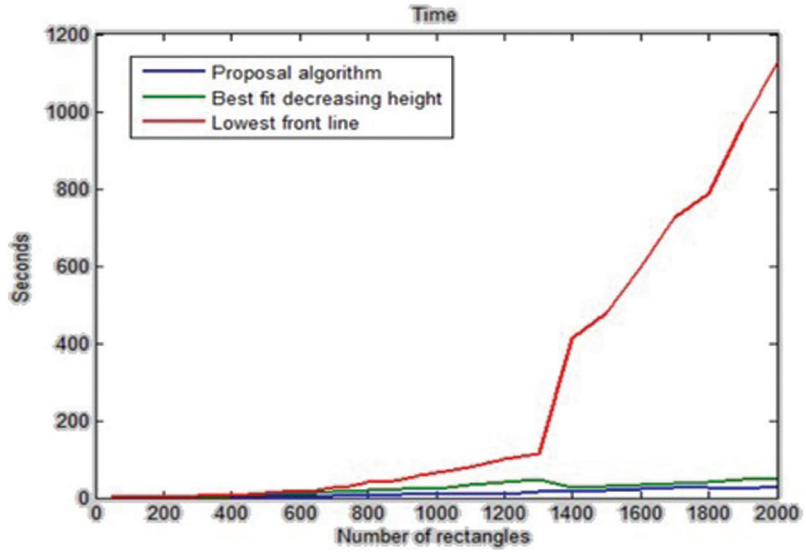

Figure 8: The running time of the Best-Fit, the front-line, and the proposed algorithms with random data

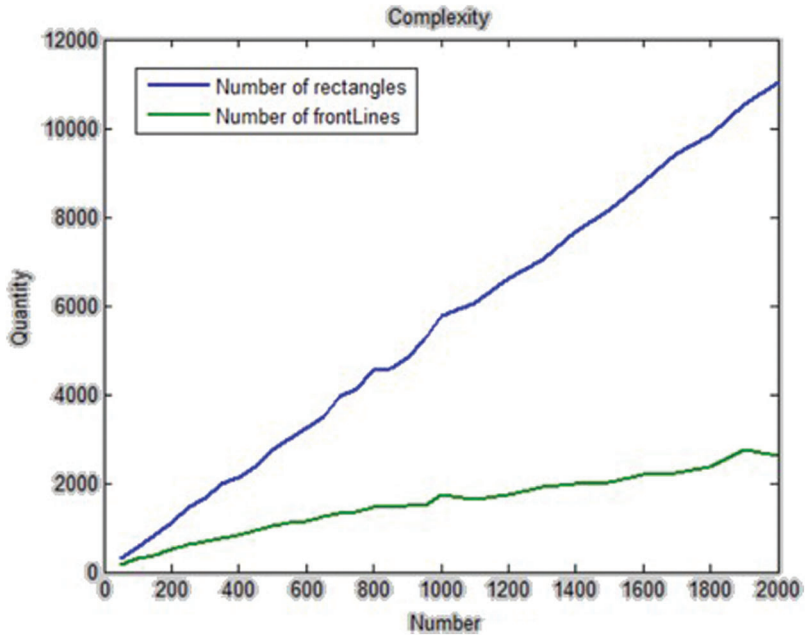

Figure 9: The comparison between the number of rectangles and the number of front-lines

5 Summary and Conclusion

In this study, an efficient algorithm was presented for the Rectangle Packing Problem (RPP). RPP struggles to insert a set of non-overlapped rectangles (with specific features) in a rectangular space (with a constant width and an unlimited height), providing that the packing is orthogonal. To this end, we have implemented the proposed algorithm along with the Best-Fit decreasing height and the lowest front-line strategy. Our evaluation shows that the proposed model with utilization rate about 94.37% outperforms others with 87.75%, 50.54%, and 87.17% utilization rate, respectively. Consequently, the proposed method is capable to of achieving much better utilization rate in comparison with other mentioned algorithms in just 0.023 s running-time, which is much faster than others.

The proposed method has the polynomial-time complexity. It is generally linear in time order and equals $O(nm)$. The results indicate that the value of "m" is always less than "n". It means that, at the end, the total number of the front-lines is always less than the total number of the rectangles.

This problem is NP-complete and has numerous local extremum points. Consequently, it is possible that the presented algorithm cannot find some global extrema. Therefore, we can use this algorithm along with GA or unsupervised learning models [36] to solve it for future studies. In the future, we first consider a solution to the problem with the Best-Fit algorithm presented in this study. Then, we improve our model every time that we repeat GA.

Funding Statement: The author(s) received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. E. Kelishomi, A. Garmabaki, M. Bahaghighat and J. J. S. Dong, "Mobile user indoor-outdoor detection through physical daily activities," *Sensors,* vol. 19, no. 3, pp. 511, 2019.

[2] M. Bahaghighat and S. A. Motamedi, "Psnr enhancement in image streaming over cognitive radio sensor networks," *Etri Journal,* vol. 39, no. 5, pp. 683–694, 2017.

[3] M. Bahaghighat, Q. Xin, S. A. Motamedi, M. M. Zanjireh and A. Vacavant, "Estimation of wind turbine angular velocity remotely found on video mining and convolutional neural network," *Applied Sciences*, vol. 10, no. 10, pp. 3544, 2020.

[4] M. Ghorbani, M. Bahaghighat, Q. Xin and F. Ozen, "ConvLSTMConv network: A deep learning approach for sentiment analysis in cloud computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–12, 2020.

[5] F. Abedini, M. Bahaghighat and M. S'hoyan, "Wind turbine tower detection using feature descriptors and deep learning," *Facta Universitatis, Series: Electronics and Energetics*, vol. 33, no. 1, pp. 133–153, 2019.

[6] M. Bahaghighat, F. Abedini, M. S'hoyan and A. J. Molnar, "Vision inspection of bottle caps in drink factories using convolutional neural networks," in *IEEE 15th Int. Conf. on Intelligent Computer Communication and Processing*, Cluj-Napoca, Romania, pp. 381–385, 2019.

[7] M. Bahaghighat, L. Akbari and Q. Xin, "A machine learning-based approach for counting blister cards within drug packages," *IEEE Access*, vol. 7, pp. 83785–83796, 2019.

[8] A. K. Virk and K. Singh, "Solving multi-objective two dimensional rectangle packing problem," in *Proc. of Sixth Int. Conf. on Soft Computing for Problem Solving*, Patiala, India, Springer, pp. 188–196, 2017.

[9] Y. Hu, H. Hashimoto, S. Imahori, T. Uno and M. J. Yagiura, "A partition-based heuristic algorithm for the rectilinear block packing problem," *Journal of the Operations Research Society of Japan*, vol. 59, no. 1, pp. 110–129, 2016.

[10] D. Zhang, Y. Che, F. Ye, Y. W. Si and S. Leung, "A hybrid algorithm based on variable neighbourhood for the strip packing problem," *Journal of Combinatorial Optimization*, vol. 32, no. 2, pp. 513–530, 2016.

[11] Y. Hu, S. Fukatsu, H. Hashimoto, S. Imahori and M. Yagiura, "Efficient overlap detection and construction algorithms for the bitmap shape packing problem," *Journal of the Operations Research Society of Japan*, vol. 61, no. 1, pp. 132–150, 2018.

[12] P. K. Agarwal and M. Shing, "Oriented aligned rectangle packing problem," *European Journal of Operational Research*, vol. 62, no. 2, pp. 210–220, 1992.

[13] A. K. Virk and K. J. Singh, "Solving bi-objective two-dimensional rectangle packing problem using binary cuckoo search," *International Journal of Computer Science and Information Security*, vol. 14, no. 7, pp. 165, 2016.

[14] Y. H. Zhu, "Parameter analysis of placement function for the rectangular packing problem based on GA," in *Materials Science Forum*, vol. 836, pp. 381–386, 2016.

[15] S. Wang, "Solving rectangle packing problem based on heuristic dynamic decomposition algorithm," in *2nd Int. Conf. on Electrical and Electronics: Techniques and Applications*, Beijing, China, pp. 187–196, 2017.

[16] K. Jansen and R. J. D. O. Solisoba, "Rectangle packing with one-dimensional resource augmentation," *Discrete Optimization*, vol. 6, no. 3, pp. 310–323, 2009.

[17] L. Wei, W. Zhu, A. Lim, Q. Liu and X. J. Chen, "An adaptive selection approach for the 2D rectangle packing area minimization problem," *Omega*, vol. 80, pp. 22–30, 2018.

[18] E. Huang and R. Korf, "Optimal rectangle packing: An absolute placement approach," *Journal of Artificial Intelligence Research*, vol. 46, pp. 47–87, 2013.

[19] A. Bortfeldt, "A reduction approach for solving the rectangle packing area minimization problem," *European Journal of Operational Research*, vol. 224, no. 3, pp. 486–496, 2013.

[20] R. Mohanty and P. Kiran, "New results on next fit and first fit on-line algorithms for square and rectangle packing," in *Int. Conf. on Advances in Computing, Communications and Informatics*, Udupi Karnataka, India, pp. 2201–2207, 2017.

[21] M. Chlebik and J. Chlebikov, "Hardness of approximation for orthogonal rectangle packing and covering problems," *Journal of Discrete Algorithms*, vol. 7, no. 3, pp. 291–305, 2009.

[22] H. Liu, J. Zhou, X. Wu and P. Yuan, "Optimization algorithm for rectangle packing problem based on varied-factor genetic algorithm and lowest front-line strategy," in *IEEE Congress on Evolutionary Computation*, Beijing, China, pp. 352–357, 2014.

[23] N. Bansal and A. Khan, "Improved approximation algorithm for two-dimensional bin packing," in *Proc. of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Portland Oregon, USA, pp. 13–25, 2014.

[24] J. Marberg and J. Schneider, "Rectangle packing with additional restrictions," *Theoretical Computer Science*, vol. 412, no. 50, pp. 6948–6958, 2011.

[25] A. Joos, "Perfect packing of cubes," Acta *Math. Hungar*, vol. 156, no. 2, pp. 375–384, 2018.

[26] Y. L. Wu, W. Huang, S. Lau, C. Wong and G.Young, "An effective quasi-human based heuristic for solving the rectangle packing problem," *European Journal of Operational Research*, vol. 141, no. 2, pp. 341–358, 2002.

[27] R. E. Korf, "Optimal rectangle packing: Initial results," in *Proc. of the Thirteenth International Conference on Automated Planning and Scheduling*, Trento, Italy, pp. 287–295, 2003.

[28] Q. Li, S. Yang and S. Zhu, "Solving 2D rectangle packing problem based on layer heuristic and genetic algorithm," in *4th Int. Conf. on Intelligent Human-Machine Systems and Cybernetics*, Nanchang, China, vol. 2, pp. 192–195, 2012.

[29] W. Huang, D. Chen and R. Xu, "A new heuristic algorithm for rectangle packing," *Computers & Operations Research*, vol. 34, no. 11, pp. 3270–3280, 2007.

[30] L. Huang, Z. Liu and Z. Liu, "An improved lowest-level best-fit algorithm with memory for the 2D rectangular packing problem," in *2014 Int. Conf. on Information Science, Electronics and Electrical Engineering*, Hokkaido, Japan, vol. 2, pp. 1279–1282, 2014.

[31] B. J. Chazelle, "The bottomn-left bin-packing heuristic: An efficient implementation," *IEEE Transactions on Computers*, vol. 32, no. 8, pp. 697–707, 1983.

[32] D. Liu and H. Teng, "An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles," *European Journal of Operational Research*, vol. 112, no. 2, pp. 413–420, 1999.

[33] N. Ntene and J. V. Vuuren, "A survey and comparison of guillotine heuristics for the 2D oriented offline strip packing problem," *Discerete Optimization*, vol. 6, no. 2, pp. 174–188, 2009.

[34] D. Garg, "Parallelizing generalized one-dimensional bin packing problem using mapReduce," in 2014 in *IEEE Int. Advance Computing Conf.*, New Delhi, India, pp. 628–635, 2014.

[35] A. Lodi, S. Martello and M. Monaci, "Two-dimensional packing problems: A survey," *European Journal of Operational Research*, vol. 141, no. 2, pp. 241–252, 2002.

[36] E. Amouee, M. M. Zanjireh, M. Bahaghighat and M. Gorbani, "A new anomalous text detection approach using unsupervised methods," *FACTA University, Series: Electronics and Energetics*, vol. 33, no. 4, pp. 631–653, 2020.