浙江大學

Strip Packing (with bonus)

ADS Group 4

陈科睿 Sleator/SP algorithm Report+PPT

裘海怡 BL/NFDH/FFDH algorithm Report+PPT

董冬 Test program Report+PPT

Abstract. In this experiment,the strip packing problem, we try different approximation algorithms ——Bottom-up left-justified (BL)、Next-fit decreasing-height (NFDH)、First-fit decreasing-height (FFDH)、Sleator's algorithm and the split algorithm (SP) to get the approximate solutions. We also generate different test cases to analyze the approximation ratio and the running time of the algorithms.we generated random, square, width scale factor sequence, hight scale factor sequence(optimal-cut) and other test data sets between 10-10000 to explore the influence on each algorithm approximation ratio .Finally, we also find some existing paper data sets for testing and comparison. In the Bonus, we do the experiment with tetrominos, and make some targeted adjustments for the structural characteristics of the input polygons to improve our algorithms.

1 Problem Overview

The strip packing problem is a 2-dimensional geometric minimization problem. Given a set of axis-aligned rectangles and a strip of bounded width and infinite height, determine an overlapping-free packing of the rectangles into the strip minimizing its height. It is strongly-NP hard and from now there exists no polynomial time approximation algorithm with a ratio smaller than $\frac{3}{2}$ unless P = NP.

2 Algorithm Presentation

2.1 Bottom-up left-justified (BL)

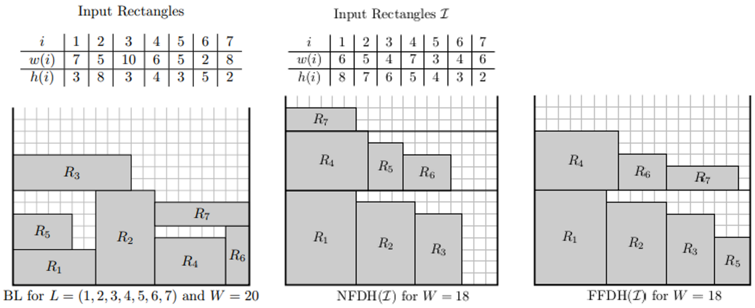
For a sequence of rectangular items, the algorithm searches for the bottom-most position to place each item and then shifts it as far to the left as possible. In the improved version, the items are ordered by decreasing widths.

2.2 Next-fit decreasing-height (NFDH)

First, the algorithm sorts the items by order of nonincreasing height. Then rectangles are packed left-justified on a level until there is insufficient space at the right to accommodate the next rectangle. At that point, the next level is defined, packing on the current level is discontinued, and packing proceeds on the new level.

2.3 First-fit decreasing-height (FFDH)

It is similar to NFDH algorithm, but when placing the next item, this algorithm scans the levels from bottom to top and places the item in the first level on which it will fit. A new level is only opened if the item does not fit in any previous ones.



Input Rectangles

i	1	2	3	4	5	6	7
$w(i)$	7	5	10	6	5	2	8
$h(i)$	3	8	3	4	3	5	2

Input Rectangles $\mathcal{I}$

i	1	2	3	4	5	6	7
$w(i)$	6	5	4	7	3	4	6
$h(i)$	8	7	6	5	4	3	2

R_3
R_7
R_5
R_2
R_1
R_4
R_6
BL for $L = (1, 2, 3, 4, 5, 6, 7)$ and $W = 20$
R_7
R_4
R_5
R_6
R_1
R_2
R_3
NFDH($\mathcal{I}$) for $W = 18$
R_4
R_6
R_7
R_1
R_2
R_3
R_5
FFDH($\mathcal{I}$) for $W = 18$

Figure 1: BLvsNFDHvsFFDH

2.4 Sleator's algorithm

First, the algorithm find all the items with a width larger than $\frac{W}{2}$ and stack them one by one at the bottom of the strip whatever the order is. Then sort all the remaining items in nonincreasing order of height. The items will be placed in this order. Draw a vertical line at $\frac{W}{2}$, which cuts the strip into two equal halves. Let h_l be the highest point covered by any item in the left half, and h_r be the highest point covered by any item in the right half. Place the items one by on in sorted order. If $h_\mathrm{l} \leq h_\mathrm{r}$, place the item on the left, otherwise on the right.

2.5 The split algorithm(SP)

This algorithm places the items in nonincreasing order of width. The intuitive idea is to split the strip into sub-strips while placing some items. Whenever possible, the algorithm places the current item i side-by-side of an already placed item j. In this case, it splits the corresponding sub-strip into two pieces: one containing the first item j and the other containing the current item i. If this is not possible, it places i on top of an already placed item and does not split the sub-strip.

3 Analysis of the Approximation Ratio

3.1 Bottom-up left-justified (BL)

$$\frac{h_{BL}}{h_{OPT}} \leq 3$$

(For any δ>0 , there exists a list L of rectangles ordered by decreasing widths such that $BL(L)/OPT(L) > 3 - \delta$)

Proof. Let h^* denote the height of the lower edge of a tallest piece whose upper edge is at height h_BL. If y denotes the height of this piece, then $h_\mathrm{BL} = y + h^*$. Let A denote the region of the bin up to height h^*. **(See Figure 1)**

Suppose we can show that A is at least half occupied. Then we have $h_\mathrm{OPT} \geq \max\{y, h^*/2\}$; hence, $y > h^*/2$ implies

$$\frac{h_\mathrm{BL}}{h_\mathrm{OPT}} \leq \frac{y + h^*}{y} < \frac{y + 2y}{y} = 3$$

and if $y \leq h^*/2$, we have

$$\frac{h_\mathrm{BL}}{h_\mathrm{OPT}} \leq \frac{h^*/2 + h^*}{h^*/2} = 3.$$

The result will thus be proved. It remains to show that A is at least half occupied:

Suppose there are many lines horizontal cut lines in A, which can be divided into alternating segments representing unoccupied and occupied areas of the BL packing. Now we'll show the sum of occupied segments is always at least equal to the sum of unoccupied segments, which holds true for lines that do not coincide with the upper or lower edges of any piece.

Initially, consider the partition of a given line just prior to when the first rectangle, say q, is assigned with a lower edge at a height exceeding the height, h, of the line. It is guaranteed that q exists since there is a piece packed above A. We will show at that point, the line is at least half occupied.

First, bottom-up packing implies that all lines must cut through at least one piece. Second, all lines must cut through a piece abutting the left bin edge. For suppose not; then the left-most piece, say q' , cut by the line must abut another piece, say q", to the left and entirely below the line. Thus, the length, x, of the unoccupied, initial segment of the line must be at least the width of q”. But since q” was packed prior to q, the width of q must be less than that of q” and hence less than x. Since at the point in time we are considering, no piece has been assigned entirely above the line, the space vertically above the initial segment must be completely unoccupied. Thus, we have the contradiction that q would have fit into the space above q” in such a way that its lower edge is at a height less than h.**(See Figure 2)**

Now consider any segment, S, of the line which cuts through an unoccupied space. Let p be the piece bordering S on the left. Since when q is assigned, it is placed above the line, q must be wider than the length of S. And for the nonincreasing order of assignmemt, q is packed later than p so p is at least as wide as q. Hence, for every unoccupied segment, there is a longer occupied segment immediately to its left. This pattern continues throughout the packing sequence, ensuring that the line remains at least half occupied. Integration over the height of A confirms that A is at least half full. **(See Figure 3)**

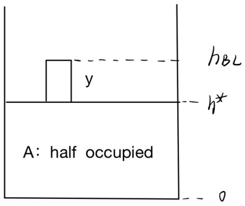


Figure 1

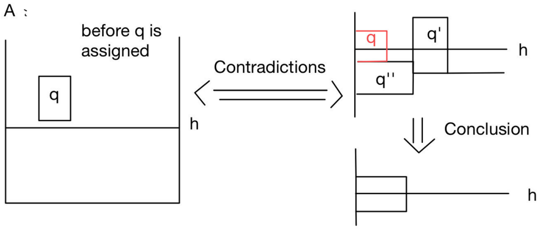


Figure 2

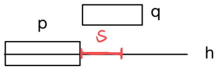


Figure 3

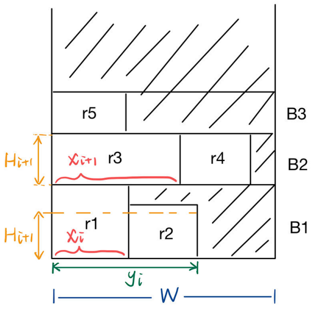


Figure 4

3.2 Next-fit decreasing-height (NFDH)

For any list L ordered by nonincreasing height,

$$\mathrm{NFDH}(L) \leq 2 \cdot \mathrm{OPT}(L) + h_{max} \leq 3 \cdot \mathrm{OPT}(L)$$

(For every $\varepsilon > 0$ there exists a set of rectangles L such that $NFDH(L) > (2 - \varepsilon)OPT(L)$)

Proof. Consider the NFDH packing of such a list L, with blocks $B_1, B_2, \cdots, B_t$. For each i, let x_i be the width of the first rectangle in B_i, and y_i be the total width of rectangles in B_i. For each $i < t$, the first rectangle in B_{i+1} does not fit in B_i. Therefore $y_i + x_{i+1} > W, 1 \leq i < t$, W is the width of strip. Since each rectangle in B_i has height at least H_{i+1}, and the first rectangle in B_{i+1} has height H_{i+1}, $A_i + A_{i+1} \geq H_{i+1}(y_i + x_{i+1}) > H_{i+1} \times W$. Therefore, if A denotes the total area of all the rectangles, we have **(See Figure 4)**

$$\begin{aligned}\mathrm{NFDH}(L) = \sum_{i=1}^{t} H_i &\le H_1 + \frac{\sum_{i=1}^{t-1} A_i + \sum_{i=2}^{t} A_i}{W} \\ &\le H_1 + 2 \times \frac{A}{W} \\ &\le h_{max} + 2\mathrm{OPT}(L) \\ &\le 3\,\mathrm{OPT}(L),\end{aligned}$$

3.3 First-fit decreasing-height (FFDH)

$$\mathrm{FFDH}(L) \le 1.7 \cdot \mathrm{OPT}(L) + h_{max} \le 2.7 \cdot \mathrm{OPT}(L)$$

Constricted by page requirements, the proof is a bit complex so we omit it.

3.4 Sleator's algorithm

$$\mathrm{Sleator}(I) \le 2 \cdot \mathrm{OPT}(I) + \frac{h_{max}}{2} \le 2.5 \cdot \mathrm{OPT}(I)$$

The proof is a bit complex so we omit it.

3.5 The split algorithm(SP)

$$\mathrm{SP}(I) \le 2 \cdot \mathrm{OPT}(I) + h_{max} \le 3 \cdot \mathrm{OPT}(I)$$

Constricted by page requirements, the proof is a bit complex so we omit it.

4 Analysis of the Time Complexity

4.1 Bottom-up left-justified (BL) - $O(n^3)$

1. Sorting: Sort the items in non-increasing order of widths with the time complexity of O($nlogn$), where n is the number of items.

2. Placing items: For each item, we need to find a suitable position for placement. In the worst case, it may iterate through all previously placed items to check whether overlaps. If we're unable to place in the current height, we need to find next lowest height of the previously placed items. Therefore, the overall time complexity for placing items is $O(n^3)$.

4.2 Next-fit decreasing-height (NFDH) - $O(nlogn)$

1. Sorting: O($nlogn$)

2. Placing items: We iterates through the items, checking if the current level can accommodate the item. If not, we creates a new level. The time complexity of placing each item is O(1), in total is $O(n)$.

4.3 First-fit decreasing-height (FFDH) - $O(n^2)$

1. Sorting: O($nlogn$)

2. Placing items: We iterates through the items, and uses a nested for loop to iterate through the levels. In the worst case, the outer loop iterates n times, and the inner loop iterates up to the number of levels (which can be up to n). Therefore, the time complexity of the placement operations is $O(n^2)$.

4.4 Sleator's algorithm - $O(nlogn)$

1. Sorting: O($nlogn$)

2. Placing items: We can determine whether each item is placed on the left or right side within $O(1)$ time. The time complexity of placing n items is $O(n)$.

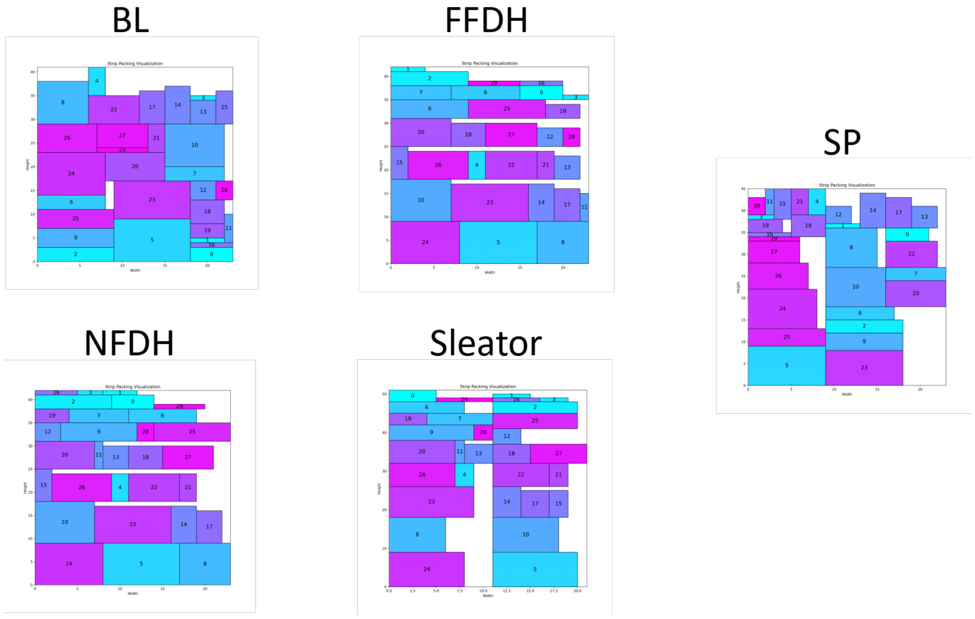
4.5 The split algorithm (SP) - $O(n^2)$

1. Sorting: O($nlogn$)

2. Placing items: For each item, we need to check all existing sub-strips once, and the number of sub-strips is at the $O(n)$ level. Therefore the time complexity of placing n items is $O(n^2)$.

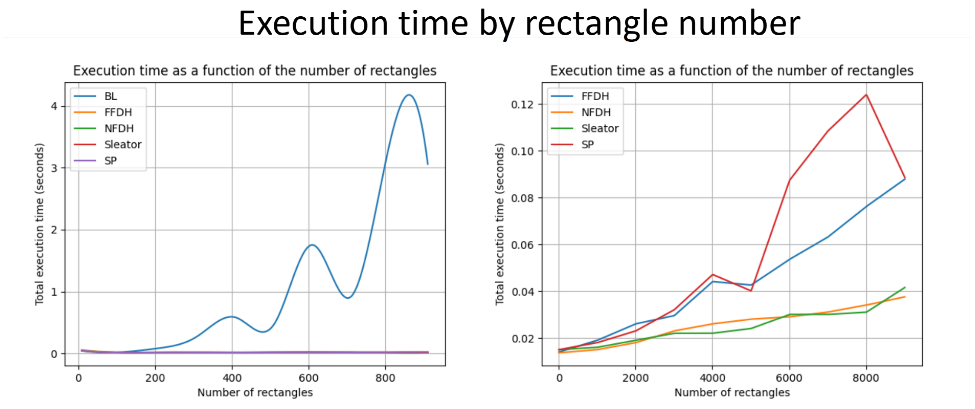
5 Experiment and Analysis

5.1 Visualization


BL
FFDH
SP
NFDH
Sleator

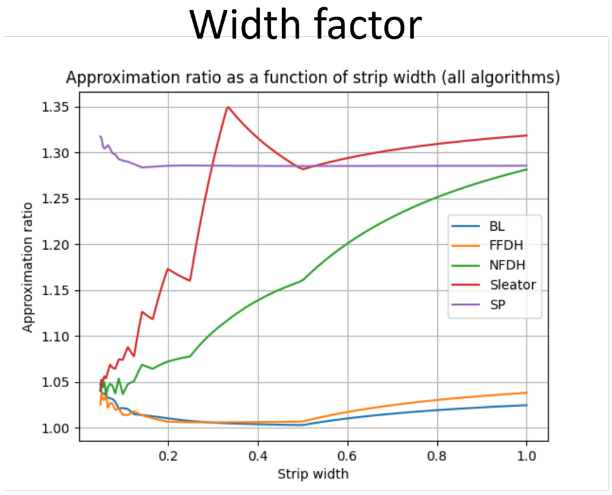
To better show the outcomes of the algorithms, we programmed a visualization of the strip results. This visualization helps to intuitively understand the packing outcomes of the algorithm, clearly showing the placement of each rectangle within the strip. Each rectangle is, labeled, and plotted according to its dimensions and position. This graphical representation can then be utilized for in-depth analysis, easy comparison between different algorithms, or effectively communicating results in presentations or publications.

5.2 Time complexity



Which is coordinate with theoretical analyse, indicating time complexity respectively: BL - $O(n^3)$, NFDH - $O(nlogn)$, FFDH - $O(n^2)$, Sleator - $O(nlogn)$, SP - $O(n^2)$.

5.3 Width factors (by max(widths)/strip_width)



For the BL algorithm, the approximation ratio gradually increases as the width factor decreases uniformly from 1 to 0.05. This indicates that the BL algorithm has a slight sensitivity to the width factor. The highest recorded approximation ratio is 1.04238, which is a sufficiently low value, indicating that the BL algorithm still performs quite well as the width factor decreases.

The performance of the FFDH algorithm varies with changes in the width factor. There isn't a consistent trend. However, the highest recorded approximation ratio is 1.04049, comparable to the BL algorithm. Despite the variability, FFDH still maintains a relatively low approximation ratio, indicating decent performance.

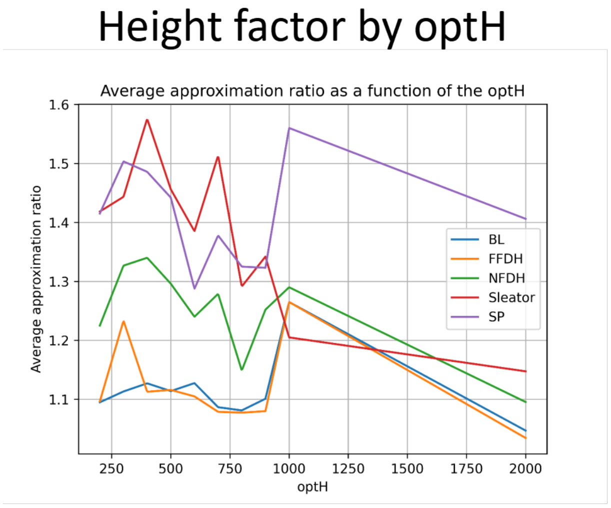
The NFDH algorithm shows a significant drop in the approximation ratio as the width factor decreases, falling from a high of 1.26143 down to about 1.04993. This indicates that the NFDH algorithm performs poorly at higher width factors but improves as the width factor decreases. Because if one item's width is a bit wide, it will waste the space of the level it locates.

The performance of the Sleator algorithm varies considerably, starting with an approximation ratio of 1.29203 and falling to about 1.05749 as the width factor decreases. Like the NFDH algorithm, Sleator performs poorly at higher width factors but improves as the width factor decreases.It because when the number of items whose width is up to $\frac{W}{2}$ is increasing, the space Sleator algorithm wasted is increasing.

The SP algorithm continually increases its approximation ratio as the width factor decreases, starting at 1.26370 and ending up around 1.29164. However, the range is pretty small, which suggests an interesting phenomenon that the SP algorithm might be slightly sensitive to changes in the width factor.

Overall, all five algorithms exhibit sensitivity to the width factor to varying degrees. The BL and FFDH algorithms perform the best, maintaining low and less fluctuating approximation ratios. The NFDH and Sleator algorithms perform poorly at higher width factors but improve as the width factor decreases. The SP algorithm might be slightly sensitive to changes in the width factor.

5.4 Height factors (optimal-cuts)

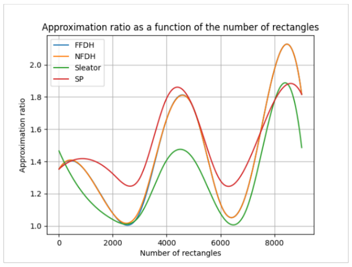


This part we use optimal-cuts to generate the test cases, which means we can know the optH and calculate the precise approximation ratio.

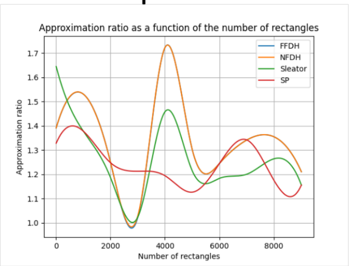
We can see all algorithms have approximation ratios around the 1.1 to 1.4 range, with Sleator and SP occasionally exceeding these values. With the increasing of optH(in other words, the numbers of rectangles), all algorithms' approximation ratios has a declining trend.

5.5 Randomly generated

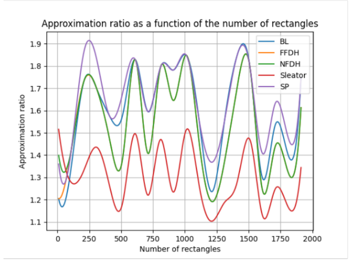
Random1



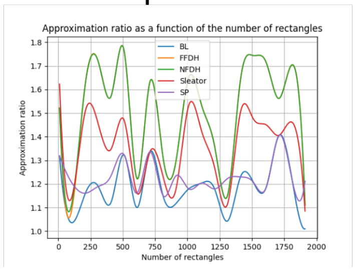
Square1



Random2



Square2

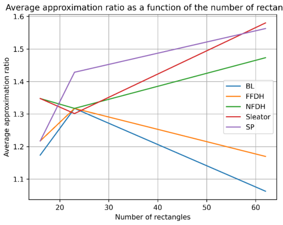


In the randomly generated test data-sets, We can not know the optH, but we can use $\frac{\text{the total areas of items}}{\text{strip\_width}}$ to gain the lower bound of it. But even we do this, all five algorithms still has a low approximation radio.

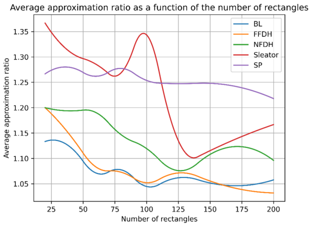
What is to be mentioned is that the BL algorithm's runtime significantly increases as the number of rectangles grows ($O(n^3)$), indicating it may not scale well with larger inputs. But it works well in packing squares, actually, the approximation radio of BL solving the square items can be proven less than 2.

5.6 Literally offered

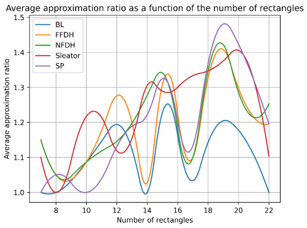
CGCUT



BENG



NGCUT



To better analyse their performances, we introduced some data-sets offered in previous papers, each of them provided with optimals. You can see the details of the introductions to these data-sets in .txt. The results are as follows.

The BL, FFDH, and NFDH algorithms all show a decrease in approximation ratio as the dataset grows larger, indicating increased efficiency when handling a larger number of rectangles. Among these, BL algorithm shows the best performance in terms of approximation ratio, and FFDH also provides rather good performance.

Sleator's algorithm exhibits a relatively higher approximation ratio, indicating that it may not be as efficient as the other algorithms when handling larger datasets. However, the performance in terms of execution time remains relatively stable across different dataset sizes.

The SP algorithm exhibits a relatively stable approximation ratio as the dataset grows larger. However, the execution time increases slightly with larger datasets. This implies that the SP algorithm maintains a relatively consistent performance across different dataset sizes, but may require more computational resources for larger datasets.

6 Conclusion

6.1 Optimal Algorithm Choice

It's important to note that the choice of algorithm depends on the specific requirements of the application. In situations where time isn't a significant constraint, both the BL algorithm and the FFDH algorithm provide excellent performance across multiple tests. However, these algorithms can be time-consuming, which makes them less practical for larger datasets in real-world situations. In such cases, the Sleator algorithm proves to be a viable alternative, as it offers a reasonable compromise between efficiency and precision, making it better suited for handling larger datasets.

6.2 Effect of Width Variations on Algorithm Performance

The performance of the strip packing algorithms seems to be substantially influenced by variations in width. As the width factor increases, the FFDH, BL, and SP algorithms demonstrate commendable resilience, with their performance largely unaffected. This suggests that these algorithms are suitable for applications where widths can vary significantly. Conversely, the Sleator and NFDH algorithms show a greater sensitivity to changes in width, leading to a noticeable drop in their performance, making them less optimal when dealing with increasing widths.

6.3 Height Factor's Influence on Algorithm Performance

Variations in the height factor appear to have a relatively minor impact on the performance of all five algorithms. Regardless of the changes in height factor, the performance of these algorithms remains reasonably stable and within acceptable bounds. This suggests that the height factor may not pose a significant challenge to the overall performance, regardless of the algorithm chosen.

6.4 Concluding Remarks

In conclusion, for scenarios without strict time limitations, the BL and FFDH algorithms consistently outperform across different datasets and should be the primary choices. However, for larger datasets, the Sleator algorithm becomes a more efficient choice. Also, width factor significantly influences the approximation rate and needs to be taken into account while designing and implementing strip packing solutions for optimal results.

7 Bonus

7.1 Method

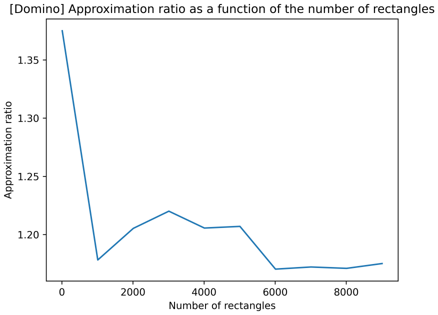
We constructed a greedy algorithm for solving this tetrominos problem. Given a set of tetrominos of different shapes (rectangles, squares, L-shapes, and T-shapes) and a strip of a specific width, the goal of the algorithm is to place the tetrominos in the strip.

For each type of shape, the algorithm scans through the current rows from left to right and top to bottom to find an empty spot that fits the shape. If a suitable spot is found, it places the tetromino there and updates the maximum number of rows used.If no empty spot is found for the tetromino in the current rows, it creates new rows and places the tetromino in these new rows.

In the main part of the program, it first reads the number of tetrominos and the width of the strip. Then for each tetromino, it reads its shape and calls a function to place it in the strip. Finally, it prints out the maximum number of rows used, which is the result of the algorithm.

For each item, we check all the position to find a place where could fill this item in. The number of position may reach $O(nW)$ level. Therefore the time complexity of placing n items is $O(n^2W)$.

7.2 Result



It's interesting to note that the approximation ratio doesn't steadily decrease or increase with the number of rectangles, but instead seems to fluctuate within a relatively narrow range. This might be due to the specific sequence of tetrominoes and the width of the strip used in these tests, which might create particular patterns that the greedy algorithm handles more or less efficiently.

8 Reference

1. Baker, Brenda S.; Coffman Jr., Edward G.; Rivest, Ronald L. (1980). ”Orthogonal Packings in Two Dimensions”. SIAM J. Comput. 9 (4): 846–855.

2. Coffman Jr., Edward G.; Garey, M. R.; Johnson, David S.; Tarjan, Robert Endre (1980). ”Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms”. SIAM J.

Comput. 9 (4): 808–826.

3. Sleator, Daniel Dominic (1980). ”A 2.5 Times Optimal Algorithm for Packing in Two Dimensions”. Inf. Process. Lett. 10: 37–40.
4. Golan, Igal (August 1981). ”Performance Bounds for Orthogonal Oriented Two-Dimensional Packing Algorithms”. SIAM Journal on Computing. 10 (3): 571–582.