TEXTURE PACKING

Group 14——

Wang Yiheng [3160104987]

Wang Yinzhao [3160101569]

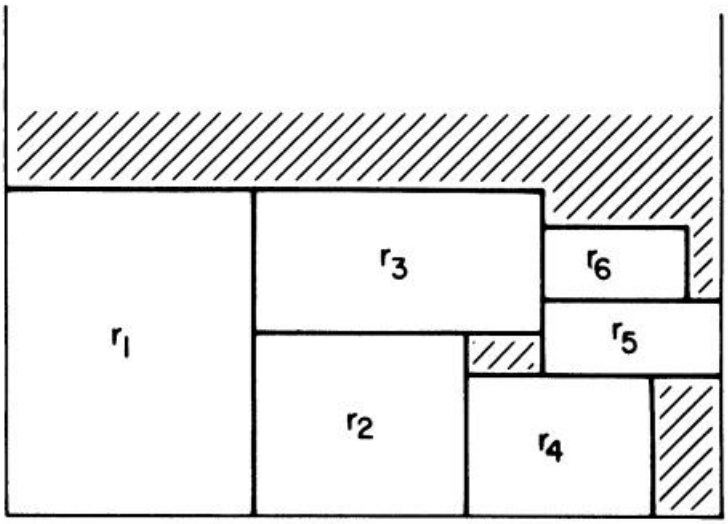
Zou Ziyu [3170103064]

Chapter 1: Introduction

Texture Packing is to pack multiple rectangle shaped textures into one large texture. The resulting texture must have a given width and a minimum height.

This project requires you to design an approximation algorithm that runs in polynomial time. You must generate test cases of different sizes (from 10 to 10,000) with different distributions of widths and heights. A thorough analysis on all the factors that might affect the approximation ratio of your proposed algorithm is expected.

The items may neither overlap nor be rotated. We describe here a list of efficient (off-line) packing algorithms. A common approach is level-oriented, the items are packed from left to right, in rows forming levels. Within the same level, all items are packed so that their bottoms align. The first level is the bottom of the strip and subsequent levels are defined by the height of the tallest item on the previous level. Some algorithms start by sorting the items by non-increasing height, they are usually named Decreasing Height (DH). The first three DH algorithms reviewed below are level-oriented. Given an approximation algorithm A, let A(I) and OPT(I) denote the height used by A and the optimal algorithm, respectively, for an instance I. The asymptotic bounds stated below assume that the width of the strip and the items is normalized so that the strip is of width 1.



Dimensions for r_i, $1 \leqq i \leqq 6$: $\frac{7}{20} \times \frac{9}{20}$, $\frac{3}{10} \times \frac{1}{4}$, $\frac{2}{5} \times \frac{1}{5}$, $\frac{1}{4} \times \frac{1}{5}$, $\frac{1}{4} \times \frac{1}{10}$, $\frac{1}{5} \times \frac{1}{10}$

Fig：one example of texture packing

Chapter 2: Algorithm Specification

Now we will specifically describe the implementation of the above algorithms.**(the symbol '*' represents our own algorithm,they are NFFDH and all online algorithm)**

OFFLINE

1.First-Fit Decreasing Height (FFDH) algorithm

At any point in the packing sequence,the next rectangle to be packed is placed left-justified on the first (i.e., lowest) level on which it will fit. If none of the current levels will accommodate this rectangle, a new level is started as in the NFDH algorithm.

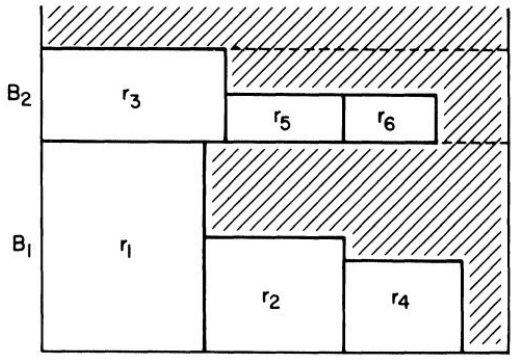


Fig: FFDH algorithms applied to the list L (r1, r2, r3, r4, r5, r6,)

Time complexity of FFDH: O(n·log n)

Approximation ratio: FFDH(I)<=(17/10)·OPT(I)+1; the asymptotic bound of 17/10 is tight.

Proof : The proof is based on the analogous proof for the First-Fit algorithm of one-dimensional bin-packing [5], [9]. We begin by defining the following weighting function.

$$W(x)=\begin{cases}\frac{6}{5}x & \text{if } 0\leqq x\leqq\frac{1}{6},\\ \frac{9}{5}x-\frac{1}{10} & \text{if } \frac{1}{6}<x\leqq\frac{1}{3},\\ \frac{6}{5}x+\frac{1}{10} & \text{if } \frac{1}{3}<x\leqq\frac{1}{2},\\ \frac{6}{5}x+\frac{4}{10} & \text{if } \frac{1}{2}<x\leqq 1.\end{cases}$$

We extend this function to rectangles r by writing W(r)= W(w(r)), and set

$$A=\sum_{r\in L} h(r)\cdot W(r).$$

It is proved that no collection of numbers x summing to 1 or less can have W(x) summing to more than 1.7. We can apply this result to our case by cutting the optimal packing into horizontal slices, formed by drawing a line through the top and bottom of each rectangle. Summing over all the slices, we have A <- 1.7 OPT(L).

2.Next-Fit Decreasing Height (NFDH) algorithm

With this algorithm, 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 thecurrent level is discontinued, and packing proceeds on the new level.

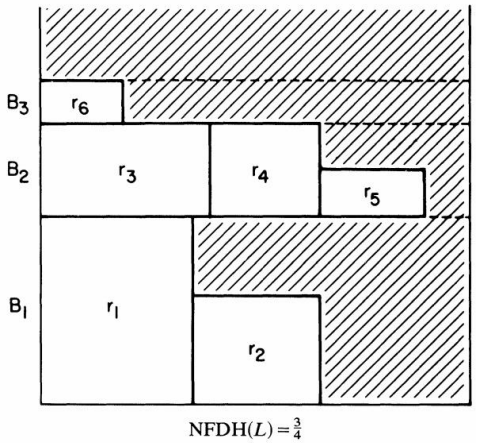


Fig: NFDH algorithms applied to the list L (r1, r2, r3, r4, r5, r6,)

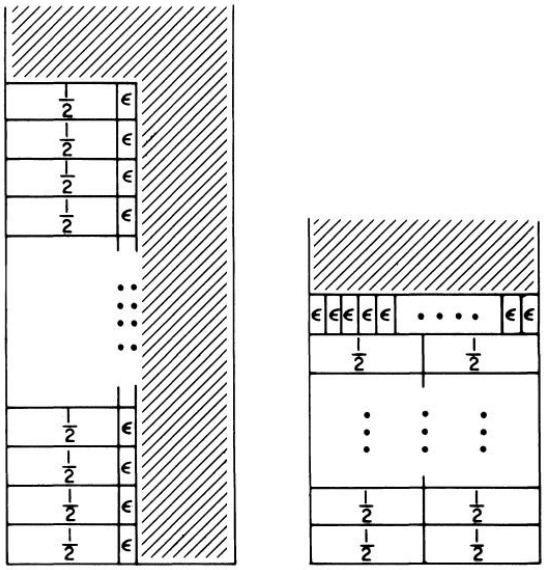
Time complexity: O(n • log n)

Approximation ratio: NFDH(I) <= 2 • OPT(I)+1; the asymptotic bound of 2 is tight.

Proof : Consider the NFDH packing of such a list L, with blocks B1, B2,..., Bt. For each i, let Xi be the width of the first rectangle in Bi, and Yi be the total width of rectangles in Bi. For each i< t, the first rectangle in B[i+l] does not fit in Bi. Therefore Yi + X[i+1] > 1, 1 < i < t. Since each rectangle in Bi has height at least H[i+1], and the first rectangle in Bi+l has height Hi+l, Ai+A[i+l]>=H[i+l](Yi+X[i+l])>H[i+l]. Therefore, if A denotes the total area of all the rectangles,which is the desired bound.

$$\begin{aligned}\mathrm{NFDH}(L) &= \sum_{i=1}^{t} H_i \leqq H_1 + \sum_{i=1}^{t-1} A_i + \sum_{i=2}^{t} A_i \\ &\leqq H_1 + 2A \\ &\leqq 1 + 2\,\mathrm{OPT}(L),\end{aligned}$$

Examples showing that the coefficient of 2 is smallest possible are derived trivially from the corresponding examples for the Next-Fit algorithm of one-dimensional bin-packing. The list L has n rectangles, where n is a multiple of 4. All the rectangles have height 1, the odd numbered ones have width 1/2, and the even numbered ones have width e, for a suitably small e > 0. The optimum and NFDH packings of L are shown in Fig. 3. In this case we have NFDH(L)= n/2 and OPT(L)= n/4+ 1, so the ratio NFDH(L)/OPT(L) can be made arbitrarily close to 2 by choosing n suitably large and esuitably small.



Fig：the worst example and the best example

3.Best-Fit Decreasing Height (BFDH) algorithm

BFDH packs the next item R (in non-increasing height) on the level, among those that can accommodate R, for which the residual horizontal space is the minimum. If no level can accommodate R, a new level is created.

Time complexity of BFDH: O(n·log n)

Approximation ratio: L(I) < 1.7 • OPT(I)+1(but it is not strict)

BFDH is an optimization on FFDH that adds an extra space check to the insertion. Due to the existence of this check, we need an extra space overhead S(n) to store the remaining space for each level, and an extra time overhead T(n) to determine the insertion.

*4. Next-First-Fit Decreasing Height(NFFDH) Algorithm

Firstly, sort the items by non-decreasing order. When in a level, we define rest-width as the total width subtracting the width already taken. We choose the first fitting item at the bottom of the level and as left as possible. And update the rest-width. Then for this item, we define rest-height as the level-height subtracting the height of the item. So the area upon the item in the level forms a column, and while there exists an item which can be put in the column, put it in and update the rest-height. Repeat the operation on the bottom of the level. Start a new level.

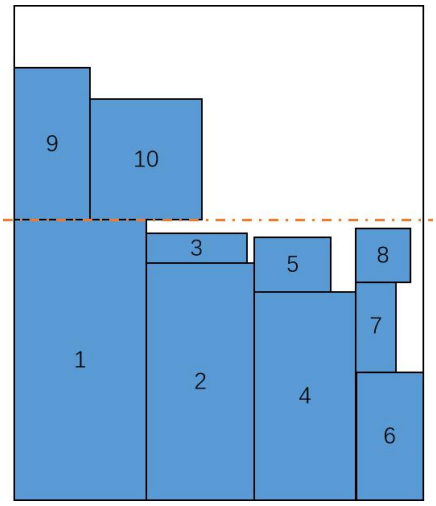


Fig1：Next-First Fit Decreasing Height Without rotation

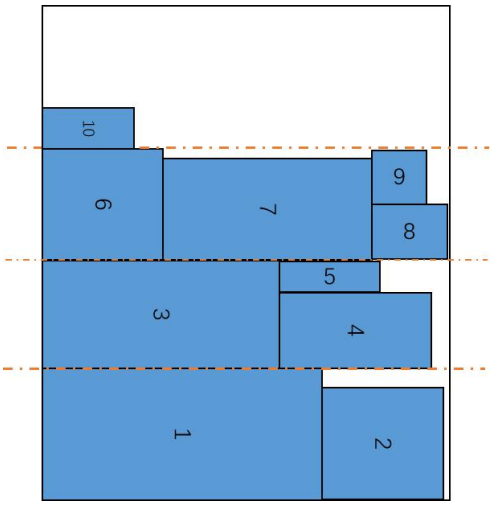


Fig2：Next-First Fit Decreasing Height With rotation

Time complexity: $O(n^2)$

Approximation ratio: NFFDH(I)<=3 • OPT(I)

ONLINE

*1.First-Fit online algorithm（ordinary）

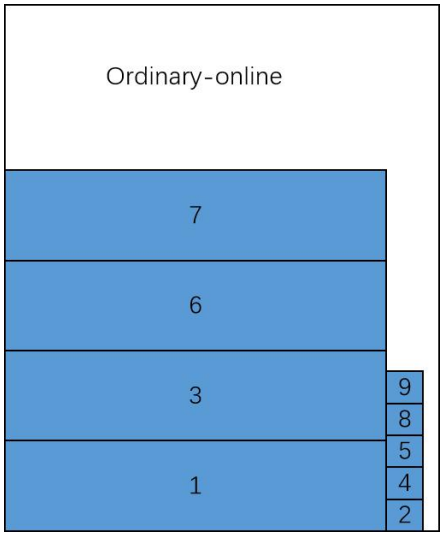
Compared with the offline FFDH algorithm, the underlying principle is basically the same. After selecting the online algorithm, its algorithm efficiency is much higher than offline (the efficiency here refers to space utilization), but at the cost of it, its time and space complexity is higher because it is completed. The inspection will be more complicated.

The difference between the online algorithm and the previous offline FFDH is:

1. You can choose whether the box can be rotated. If it can be rotated, it can be rotated when the width of the box is read. The longer one is wide, so that the width of the box must be greater than or equal to the height, and the height is the lowest.

2. When loading, if there is room for the space to meet the rotation of the box, and the total height after the rotation is satisfied, we will rotate the box and load

3. After a box is filled with one layer, the following Space can no longer be installed. That is,After a box is loaded, the space below it is equivalent to being occupied!

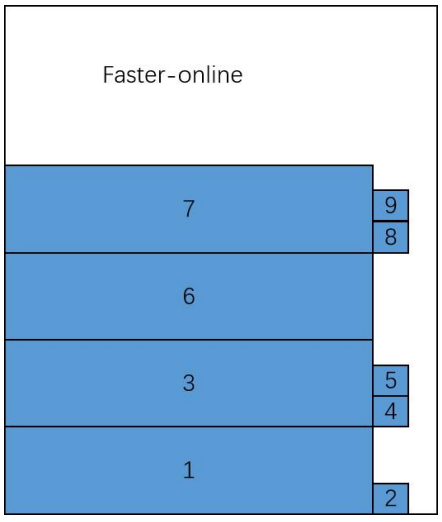


Time complexity of FFDH: O(n·log n)

***2.First-Fit online algorithm（faster）**

The algorithm is an improvement of the previous algorithm. It does not start from the bottom layer, but starts from the higher level of the baseline.

The main idea of online-faster is that when a box is loaded, the space utilization of the line loaded by the box, if it exceeds a threshold, then the line is considered as baseline, that is, the space below this line is considered to be fully utilized! Then, when you traverse the search, you only need to start from the baseline, which can save a lot of time. For example, when the box is loaded, the utilization of the layer is calculated. If it is greater than 0.9, the baseline is updated.



Time complexity of FFDH: O(n·log n)

In the online algorithm, we specifically added rotation to improve the space utilization of the algorithm.In this way, some locations that could not be inserted can be put down after rotation, further improving space utilization.

If it can be rotated, first adjust all the boxes to be wider than high, then the first position must be satisfied before the rotation can be installed, but only after the rotation is loaded without increasing the minimum height will be considered for rotation loading, otherwise it will not be considered.

Chapter 3: Testing Results

OFFLINE PART

Case1:Exactly identical rectangles

Texture width:100

The number of rectangle:10000

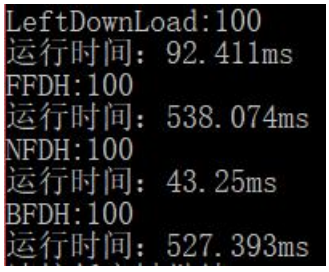
The size of every rectangle(height*width):1*1

The optimal storage height = 10000

The result storage height = 10000

The space utilization(Ratio of all rectangles to total utilization space) = 1

The approximate ratio = 1



Texture width:100

The number of rectangle:10000

The size of every rectangle(height*width):7*7

The optimal storage height = 5005

The result storage height = 5005

The space utilization = 7*7*10000/(5005*100) = 0.9790

The approximate ratio = 1

```
Please input the numb
LeftDownLoad:5005
运行时间：100.226ms
FFDH:5005
运行时间：3402.96ms
NFDH:5005
运行时间：42.914ms
BFDH:5005
运行时间：3503.95ms
```

Texture width:100

The number of rectangle:10000

The size of every rectangle(height*width):50*50

The optimal storage height = 250000

The result storage height = 250000

The space utilization = 100%

The approximate ratio = 1

```
LeftDownLoad:250000
运行时间：90.26ms
FFDH:250000
运行时间：24044.2ms
NFDH:250000
运行时间：47.6188ms
BFDH:250000
运行时间：24070.7ms
```

Case 2:Special staggered loop rectangles

Texture width:100

The number of rectangle:10000

The size of every rectangle(height*width):30*30、20*20...(loop)

The optimal storage height = 70010(actually,the minimum is 66690)

The result storage height = 70010(20*20 is a group of 5, and 30*30 is a group of 3)

The space utilization = (20*20*5000+30*30*5000)/(70010*100) = 0.9284

The approximate ratio = 1

```
LeftDownLoad:70010
运行时间：153.919ms
FFDH:70010
运行时间：14060.9ms
NFDH:70010
运行时间：44.0636ms
BFDH:70010
运行时间：14027.5ms
```

The following is a special example that will make different algorithms make different responses.

Texture width:100

The number of rectangle:7

The size of every rectangle(height*width):30*20 30*20 10*50...(loop)

The optimal storage height = 40

The result storage height = 40(30*20 is a group of 5, and 10*50 is a group of 2)

The space utilization = 1

The approximate ratio = 1

The result is very abnormal, different algorithms actually made the same reaction.We will put the specific analysis in a later module.

```
Please input the numb
LeftDownLoad:40
运行时间：5.18194ms
FFDH:40
运行时间：0.40041ms
NFDH:40
运行时间：1.01374ms
BFDH:40
运行时间：0.398359ms
```

Texture width:100

The number of rectangle:10000

The size of every rectangle(height*width):30*20 10*50 20*25...(loop)

The optimal storage height = 53350 (666*30+1666*10+833*20+30+20)

The result storage height = 53350

The space utilization ≈ 1

The approximate ratio = 1

```
LeftDownLoad:53350
运行时间：89.1856ms
FFDH:53350
运行时间：11810.5ms
NFDH:53350
运行时间：47.0547ms
BFDH:53350
运行时间：11714.5ms
```

Case 3:Random rectangles

To test the accuracy of the set, we tested each N at least 10 times (smaller N tests more than 10 times) to minimize accidental errors. Below we only show some time consuming during the test.

(a)Given width =100 while the random range is (0,100) for height and width

N=10

```
[2.1] Rotation denied:
N-FFDH: 367  Average Testing Time: 0.002ms Appoximate-ratio: 1.37
FFDH: 367  Average Testing Time: 0.004ms Appoximate-ratio: 1.37
NFDH: 384  Average Testing Time: 0.003ms Appoximate-ratio: 1.44
BFDH: 358  Average Testing Time: 0.004ms Appoximate-ratio: 1.34
[2.2] Rotation permited:
N-FFDH: 328  Average Testing Time: 0.002ms Appoximate-ratio: 1.23
FFDH: 328  Average Testing Time: 0.004ms Appoximate-ratio: 1.23
NFDH: 333  Average Testing Time: 0.003ms Appoximate-ratio: 1.25
BFDH: 328  Average Testing Time: 0.004ms Appoximate-ratio: 1.23
```

N=100

```
[2.1] Rotation denied:
N-FFDH: 3185  Average Testing Time: 0.114ms Appoximate-ratio: 1.19
FFDH: 3185  Average Testing Time: 0.034ms Appoximate-ratio: 1.19
NFDH: 3521  Average Testing Time: 0.024ms Appoximate-ratio: 1.31
BFDH: 3185  Average Testing Time: 0.046ms Appoximate-ratio: 1.19
[2.2] Rotation permited:
N-FFDH: 3223  Average Testing Time: 0.098ms Appoximate-ratio: 1.20
FFDH: 3223  Average Testing Time: 0.037ms Appoximate-ratio: 1.20
NFDH: 3364  Average Testing Time: 0.024ms Appoximate-ratio: 1.25
BFDH: 3223  Average Testing Time: 0.049ms Appoximate-ratio: 1.20
```

N=500

```
[2.1] Rotation denied:
N-FFDH: 13666  Average Testing Time: 1.877ms Appoximate-ratio: 1.07
FFDH: 13670  Average Testing Time: 0.298ms Appoximate-ratio: 1.07
NFDH: 16926  Average Testing Time: 0.159ms Appoximate-ratio: 1.32
BFDH: 13497  Average Testing Time: 0.398ms Appoximate-ratio: 1.05
[2.2] Rotation permited:
N-FFDH: 15110  Average Testing Time: 1.604ms Appoximate-ratio: 1.18
FFDH: 15110  Average Testing Time: 0.358ms Appoximate-ratio: 1.18
NFDH: 15980  Average Testing Time: 0.149ms Appoximate-ratio: 1.25
BFDH: 15103  Average Testing Time: 0.542ms Appoximate-ratio: 1.18
```

N=1000

```
[2.1] Rotation denied:
N-FFDH: 27225  Average Testing Time: 5.423ms Appoximate-ratio: 1.05
FFDH: 27234  Average Testing Time: 0.749ms Appoximate-ratio: 1.05
NFDH: 33822  Average Testing Time: 0.302ms Appoximate-ratio: 1.30
BFDH: 26983  Average Testing Time: 1.060ms Appoximate-ratio: 1.04
[2.2] Rotation permited:
N-FFDH: 30454  Average Testing Time: 4.852ms Appoximate-ratio: 1.17
FFDH: 30458  Average Testing Time: 0.970ms Appoximate-ratio: 1.17
NFDH: 32097  Average Testing Time: 0.289ms Appoximate-ratio: 1.23
BFDH: 30457  Average Testing Time: 1.552ms Appoximate-ratio: 1.17
```

N=2000

```
[2.1] Rotation denied:
N-FFDH: 52535  Average Testing Time: 17.534ms Appoximate-ratio: 1.04
FFDH: 52545  Average Testing Time: 2.379ms Appoximate-ratio: 1.04
NFDH: 65749  Average Testing Time: 0.647ms Appoximate-ratio: 1.30
BFDH: 52198  Average Testing Time: 3.293ms Appoximate-ratio: 1.03
[2.2] Rotation permited:
N-FFDH: 59013  Average Testing Time: 16.372ms Appoximate-ratio: 1.17
FFDH: 59021  Average Testing Time: 3.218ms Appoximate-ratio: 1.17
NFDH: 62489  Average Testing Time: 0.633ms Appoximate-ratio: 1.24
BFDH: 58987  Average Testing Time: 4.903ms Appoximate-ratio: 1.17
```

N=5000

```
[2.1] Rotation denied:
N-FFDH: 130202  Average Testing Time: 90.369ms Appoximate-ratio: 1.03
FFDH: 130213  Average Testing Time: 13.729ms Appoximate-ratio: 1.03
NFDH: 163084  Average Testing Time: 2.092ms Appoximate-ratio: 1.28
BFDH: 129602  Average Testing Time: 17.825ms Appoximate-ratio: 1.02
[2.2] Rotation permited:
N-FFDH: 147818  Average Testing Time: 97.875ms Appoximate-ratio: 1.16
FFDH: 147824  Average Testing Time: 19.475ms Appoximate-ratio: 1.16
NFDH: 156578  Average Testing Time: 2.169ms Appoximate-ratio: 1.23
BFDH: 147799  Average Testing Time: 27.592ms Appoximate-ratio: 1.16
```

N=10000

```
N-FFDH: 260466  Average Testing Time: 268.760ms Appoximate-ratio: 1.02
FFDH: 260483  Average Testing Time: 45.017ms Appoximate-ratio: 1.02
NFDH: 327860  Average Testing Time: 3.948ms Appoximate-ratio: 1.28
BFDH: 259682  Average Testing Time: 54.317ms Appoximate-ratio: 1.02
[2.2] Rotation permited:
N-FFDH: 297301  Average Testing Time: 322.912ms Appoximate-ratio: 1.16
FFDH: 297308  Average Testing Time: 63.439ms Appoximate-ratio: 1.16
NFDH: 314678  Average Testing Time: 3.849ms Appoximate-ratio: 1.23
BFDH: 297292  Average Testing Time: 88.452ms Appoximate-ratio: 1.16
```

N=50000

```
[2.1] Rotation denied:
N-FFDH: 1285388  Average Testing Time: 5071.642ms Appoximate-ratio: 1.01
FFDH: 1285418  Average Testing Time: 1035.156ms Appoximate-ratio: 1.01
NFDH: 1635453  Average Testing Time: 20.236ms Appoximate-ratio: 1.28
BFDH: 1284442  Average Testing Time: 1155.304ms Appoximate-ratio: 1.01
[2.2] Rotation permited:
N-FFDH: 1483657  Average Testing Time: 7586.910ms Appoximate-ratio: 1.16
FFDH: 1483663  Average Testing Time: 1517.159ms Appoximate-ratio: 1.16
NFDH: 1571538  Average Testing Time: 17.695ms Appoximate-ratio: 1.23
BFDH: 1483655  Average Testing Time: 1938.350ms Appoximate-ratio: 1.16
```

N=100000

```
[2.1] Rotation denied:
N-FFDH: 2548622  Average Testing Time: 18962.602ms Appoximate-ratio: 1.00
FFDH: 2548657  Average Testing Time: 4181.722ms Appoximate-ratio: 1.00
NFDH: 3249267  Average Testing Time: 36.988ms Appoximate-ratio: 1.28
BFDH: 2547550  Average Testing Time: 4441.069ms Appoximate-ratio: 1.00
[2.2] Rotation permited:
N-FFDH: 2953752  Average Testing Time: 30679.115ms Appoximate-ratio: 1.16
FFDH: 2953764  Average Testing Time: 5964.426ms Appoximate-ratio: 1.16
NFDH: 3130174  Average Testing Time: 35.785ms Appoximate-ratio: 1.23
BFDH: 2953764  Average Testing Time: 7632.178ms Appoximate-ratio: 1.16
```

(b)Given width =1000 while the random range is (0,30) for height and width

N=10

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 26  Testing time: 0.009ms Appoximate-ratio: 101.13
FFDH: 26  Testing time: 0.038ms Appoximate-ratio: 101.13
NFDH: 26  Testing time: 0.009ms Appoximate-ratio: 101.13
BFDH: 26  Testing time: 0.008ms Appoximate-ratio: 101.13
[2.2] Rotation permited:
N-FFDH: 23  Testing time: 0.005ms Appoximate-ratio: 89.46
FFDH: 23  Testing time: 0.023ms Appoximate-ratio: 89.46
NFDH: 23  Testing time: 0.007ms Appoximate-ratio: 89.46
BFDH: 23  Testing time: 0.012ms Appoximate-ratio: 89.46
```

N=100

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 30  Testing time: 0.098ms Appoximate-ratio: 11.83
FFDH: 30  Testing time: 0.084ms Appoximate-ratio: 11.83
NFDH: 30  Testing time: 0.045ms Appoximate-ratio: 11.83
BFDH: 30  Testing time: 0.046ms Appoximate-ratio: 11.83
[2.2] Rotation permited:
N-FFDH: 28  Testing time: 0.082ms Appoximate-ratio: 11.04
FFDH: 28  Testing time: 0.047ms Appoximate-ratio: 11.04
NFDH: 28  Testing time: 0.041ms Appoximate-ratio: 11.04
BFDH: 28  Testing time: 0.053ms Appoximate-ratio: 11.04
```

N=500

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 30  Testing time: 1.072ms Appoximate-ratio: 2.48
FFDH: 30  Testing time: 0.200ms Appoximate-ratio: 2.48
NFDH: 30  Testing time: 0.182ms Appoximate-ratio: 2.48
BFDH: 30  Testing time: 0.182ms Appoximate-ratio: 2.48
[2.2] Rotation permited:
N-FFDH: 29  Testing time: 0.714ms Appoximate-ratio: 2.40
FFDH: 31  Testing time: 0.195ms Appoximate-ratio: 2.56
NFDH: 31  Testing time: 0.181ms Appoximate-ratio: 2.56
BFDH: 31  Testing time: 0.186ms Appoximate-ratio: 2.56
```

N=1000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 30  Testing time: 2.763ms Appoximate-ratio: 1.23
FFDH: 41  Testing time: 0.422ms Appoximate-ratio: 1.68
NFDH: 41  Testing time: 0.347ms Appoximate-ratio: 1.68
BFDH: 41  Testing time: 0.411ms Appoximate-ratio: 1.68
[2.2] Rotation permited:
N-FFDH: 30  Testing time: 1.876ms Appoximate-ratio: 1.23
FFDH: 42  Testing time: 0.336ms Appoximate-ratio: 1.72
NFDH: 42  Testing time: 0.353ms Appoximate-ratio: 1.72
BFDH: 42  Testing time: 0.350ms Appoximate-ratio: 1.72
```

N=2000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 63  Testing time: 11.818ms Appoximate-ratio: 1.30
FFDH: 65  Testing time: 0.759ms Appoximate-ratio: 1.34
NFDH: 65  Testing time: 0.727ms Appoximate-ratio: 1.34
BFDH: 65  Testing time: 0.700ms Appoximate-ratio: 1.34
[2.2] Rotation permited:
N-FFDH: 58  Testing time: 7.154ms Appoximate-ratio: 1.19
FFDH: 66  Testing time: 0.738ms Appoximate-ratio: 1.36
NFDH: 66  Testing time: 0.690ms Appoximate-ratio: 1.36
BFDH: 66  Testing time: 0.736ms Appoximate-ratio: 1.36
```

N=5000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 132  Testing time: 77.655ms Appoximate-ratio: 1.08
FFDH: 136  Testing time: 2.360ms Appoximate-ratio: 1.12
NFDH: 136  Testing time: 2.503ms Appoximate-ratio: 1.12
BFDH: 136  Testing time: 2.285ms Appoximate-ratio: 1.12
[2.2] Rotation permited:
N-FFDH: 129  Testing time: 72.298ms Appoximate-ratio: 1.06
FFDH: 138  Testing time: 1.990ms Appoximate-ratio: 1.13
NFDH: 138  Testing time: 1.907ms Appoximate-ratio: 1.13
BFDH: 138  Testing time: 2.027ms Appoximate-ratio: 1.13
```

N=10000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 252  Testing time: 311.217ms Appoximate-ratio: 1.07
FFDH: 253  Testing time: 3.835ms Appoximate-ratio: 1.07
NFDH: 253  Testing time: 3.918ms Appoximate-ratio: 1.07
BFDH: 253  Testing time: 3.939ms Appoximate-ratio: 1.07
[2.2] Rotation permited:
N-FFDH: 247  Testing time: 257.896ms Appoximate-ratio: 1.05
FFDH: 254  Testing time: 3.756ms Appoximate-ratio: 1.08
NFDH: 254  Testing time: 3.339ms Appoximate-ratio: 1.08
BFDH: 254  Testing time: 3.679ms Appoximate-ratio: 1.08
```

N=50000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 1220  Testing time: 3919.166ms Appoximate-ratio: 1.01
FFDH: 1222  Testing time: 20.350ms Appoximate-ratio: 1.01
NFDH: 1224  Testing time: 16.531ms Appoximate-ratio: 1.01
BFDH: 1222  Testing time: 21.598ms Appoximate-ratio: 1.01
[2.2] Rotation permited:
N-FFDH: 1220  Testing time: 3500.810ms Appoximate-ratio: 1.01
FFDH: 1225  Testing time: 26.628ms Appoximate-ratio: 1.01
NFDH: 1225  Testing time: 19.226ms Appoximate-ratio: 1.01
BFDH: 1225  Testing time: 27.051ms Appoximate-ratio: 1.01
```

N=100000

```
[2] Testing result for off-line algorithm:
[2.1] Rotation denied:
N-FFDH: 2408  Testing time: 14336.037ms Appoximate-ratio: 1.00
FFDH: 2413  Testing time: 49.529ms Appoximate-ratio: 1.01
NFDH: 2418  Testing time: 33.956ms Appoximate-ratio: 1.01
BFDH: 2413  Testing time: 51.994ms Appoximate-ratio: 1.01
[2.2] Rotation permited:
N-FFDH: 2404  Testing time: 13733.789ms Appoximate-ratio: 1.00
FFDH: 2414  Testing time: 58.294ms Appoximate-ratio: 1.01
NFDH: 2418  Testing time: 36.187ms Appoximate-ratio: 1.01
BFDH: 2414  Testing time: 68.768ms Appoximate-ratio: 1.01
```

ONLINE PART

Case 1:Special staggered loop rectangles

The number of rectangle:7

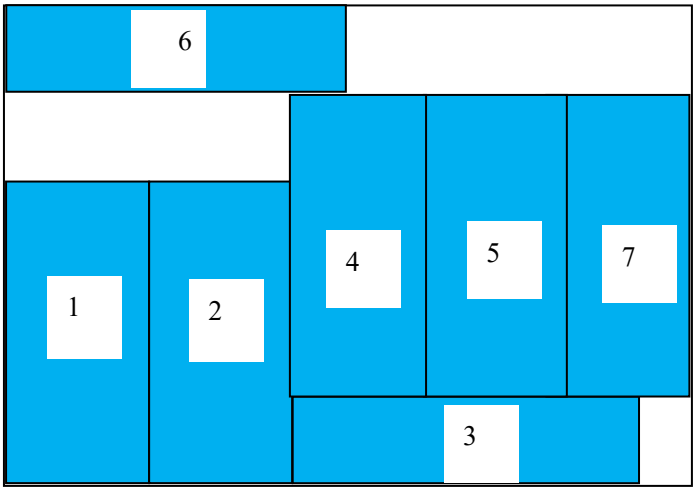
The size of every rectangle(height*width):30*20 30*20 10*50...(loop)

The optimal storage height = 40

The result storage height = 50

The space utilization = (30*20*5+10*50*2)/5000=0.8

The approximate ratio = 50/40=1.25



```
Please input box's height and width:
30 20
The box is loaded at box[0][0] to box[29][19]
30 20
The box is loaded at box[0][20] to box[29][39]
10 50
The box is loaded at box[0][40] to box[9][89]
30 20
The box is loaded at box[10][40] to box[39][59]
30 20
The box is loaded at box[10][60] to box[39][79]
10 50
The box is loaded at box[40][0] to box[49][49]
30 20
The box is loaded at box[10][80] to box[39][99]
0 0
The height is 50
```

Case 2:Random rectangles

Given width = 1000 while the random range is (0,30) for height and width

online-ordinary&online-faster

N=10

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 26  Testing time: 0.091ms Appoximate-ratio: 141.46
onLine-faster: 26  Testing time: 0.121ms Appoximate-ratio: 141.46
[1.2] Rotation permited:
onLine-ordinary: 23  Testing time: 0.011ms Appoximate-ratio: 125.14
onLine-faster: 23  Testing time: 0.120ms Appoximate-ratio: 125.14
```

N=100

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 30  Testing time: 0.555ms Appoximate-ratio: 12.31
onLine-faster: 30  Testing time: 1.880ms Appoximate-ratio: 12.31
[1.2] Rotation permited:
onLine-ordinary: 26  Testing time: 0.266ms Appoximate-ratio: 10.67
onLine-faster: 26  Testing time: 2.199ms Appoximate-ratio: 10.67
```

N=500

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 30  Testing time: 5.088ms Appoximate-ratio: 2.37
onLine-faster: 30  Testing time: 14.959ms Appoximate-ratio: 2.37
[1.2] Rotation permited:
onLine-ordinary: 30  Testing time: 3.983ms Appoximate-ratio: 2.37
onLine-faster: 30  Testing time: 15.883ms Appoximate-ratio: 2.37
```

N=1000

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 45  Testing time: 80.949ms Appoximate-ratio: 1.83
onLine-faster: 47  Testing time: 32.119ms Appoximate-ratio: 1.91
[1.2] Rotation permited:
onLine-ordinary: 46  Testing time: 34.660ms Appoximate-ratio: 1.87
onLine-faster: 46  Testing time: 32.411ms Appoximate-ratio: 1.87
```

N=2000

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 68  Testing time: 675.784ms Appoximate-ratio: 1.44
onLine-faster: 70  Testing time: 74.441ms Appoximate-ratio: 1.48
[1.2] Rotation permited:
onLine-ordinary: 67  Testing time: 581.566ms Appoximate-ratio: 1.41
onLine-faster: 69  Testing time: 67.833ms Appoximate-ratio: 1.46
```

N=5000

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 144  Testing time: 5768.153ms Appoximate-ratio: 1.21
onLine-faster: 148  Testing time: 187.581ms Appoximate-ratio: 1.24
[1.2] Rotation permited:
onLine-ordinary: 140  Testing time: 4609.778ms Appoximate-ratio: 1.17
onLine-faster: 147  Testing time: 148.878ms Appoximate-ratio: 1.23
```

N=10000

```
[1] Testing result for on-line algorithm:
[1.1] Rotation denied:
onLine-ordinary: 260  Testing time: 19946.928ms Appoximate-ratio: 1.08
onLine-faster: 269  Testing time: 323.102ms Appoximate-ratio: 1.12
[1.2] Rotation permited:
onLine-ordinary: 255  Testing time: 18298.698ms Appoximate-ratio: 1.06
onLine-faster: 265  Testing time: 300.626ms Appoximate-ratio: 1.10
```

Chapter 4: Analysis

The next two result of given_width = 100 and range of size of box is (1,1) to (100,100)

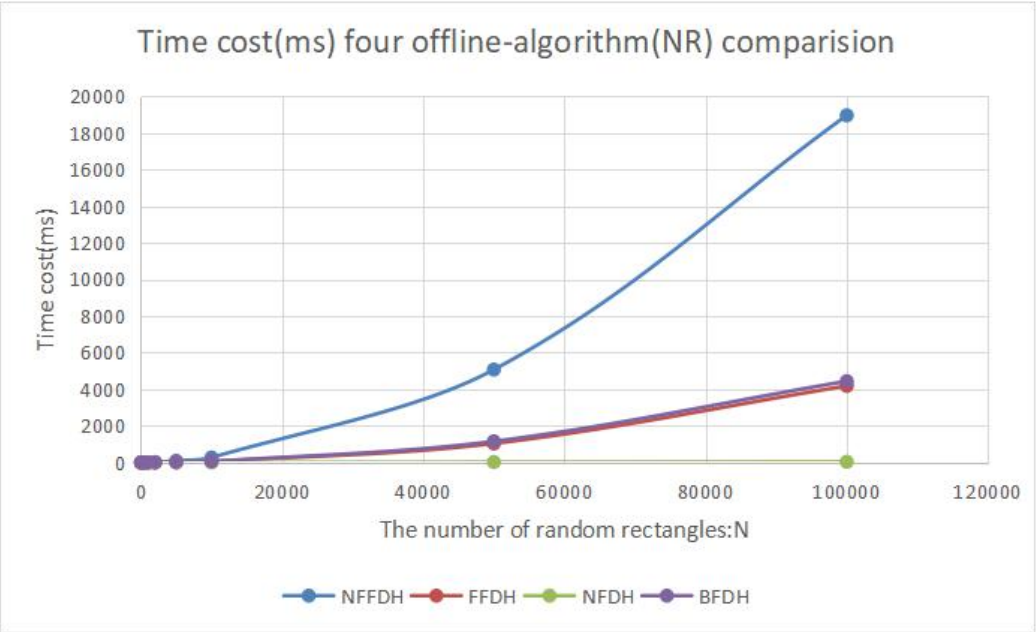
Through the testing of the examples in various cases, we found multiple impact factors that affect the efficiency and running time of certain algorithms.

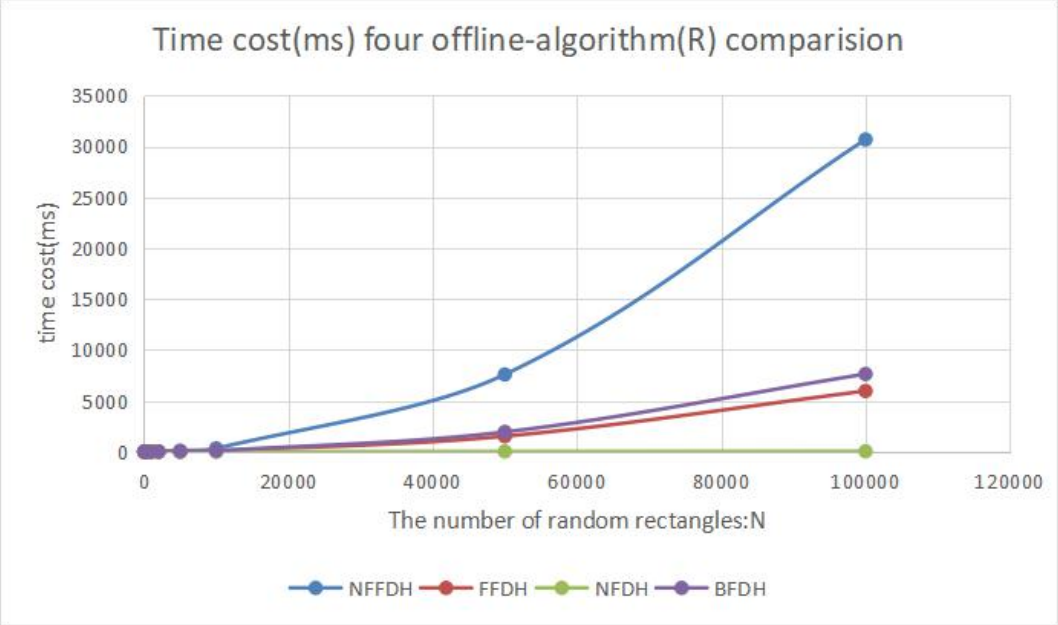
Through extensive testing and final error elimination, the NFDH algorithm runs at the fastest rate. When n is large enough (between 2000 and 10000), the running time begins to branch significantly. Although the time complexity of NFDH is the same as FFDH(O(n) = n*logn), the idea of the algorithm is to exchange time for the cost of space, which leads to it is very fast, but the space utilization is also low (the worst space utilization rate is 0.5+e).

However, we found that just changing the given width and the range of the input box width is different from the example tested next. Observing the curve, FFDH and BFDH start to rise at a certain stage, and although NFFDH is a well-defined curve (its time complexity is n^2), for the same N, its time cost is much larger.

When we checked the results, we found that the minimum heights of the outputs of all offline algorithms are consistent, which means they are always optimal. Secondly, we compare the width of each box with the maximum width given, and their average ratio is not small, which means that the possibility of bad is greatly increased, as long as a "bulk" box appears, then A level is almost occupied. So we infer that in addition to the algorithm itself, the parameters passed in will also affect the time cost.

The following two charts show the relationship between time overhead and the number of boxes in both cases of rotation(R) and non-rotation(NR).

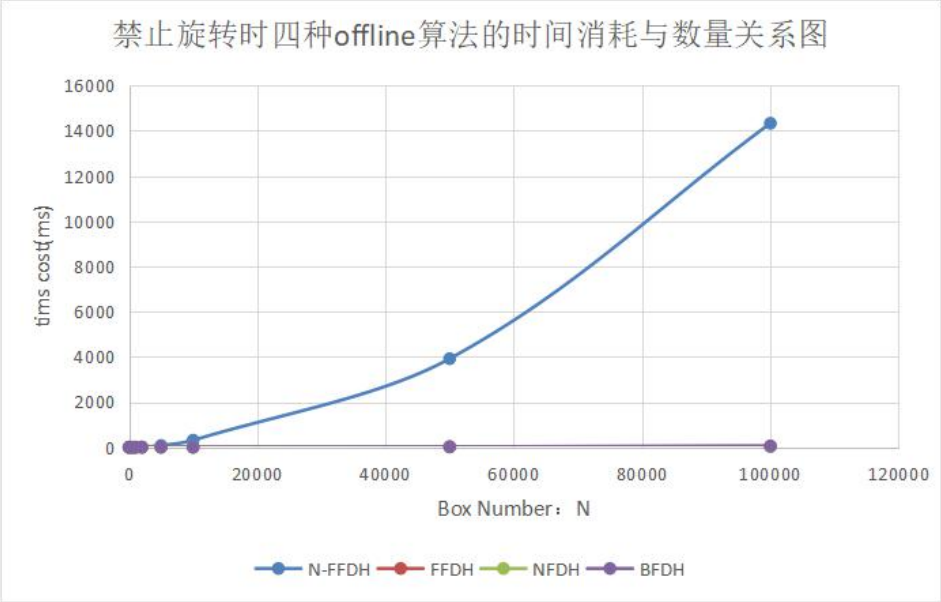


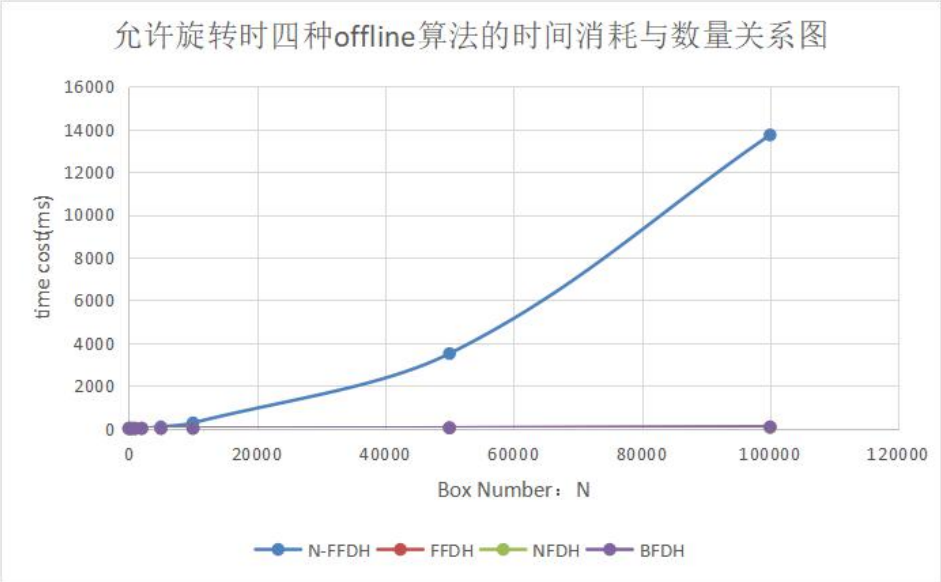


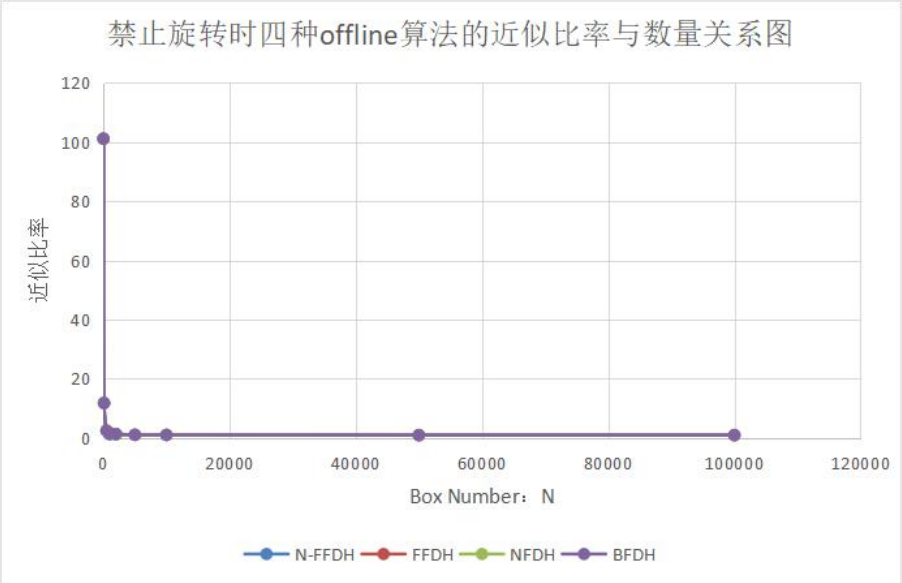
The following results' given_width = 1000 and range of size of box is (1,1) to (30,30)

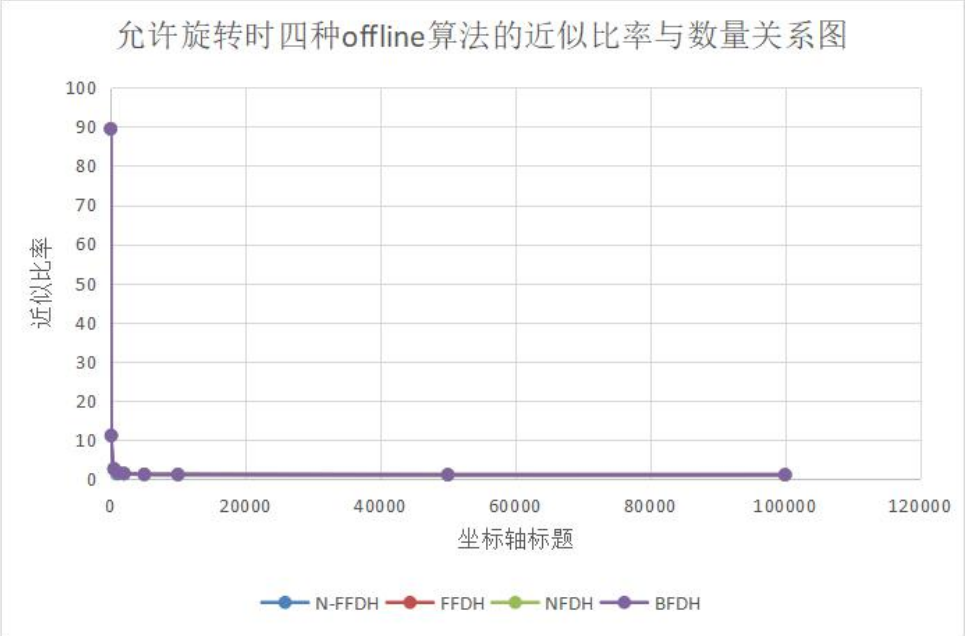
However, it is clear that there is a significant difference in time overhead and time complexity in the above tests, which is not in line with expectations. Therefore, we chose to expand the given width and further define the length and width of the box. This time is in line with expectations.

So we conclude that the second factor affecting the time overhead is the ratio of the height and width of the box to the given width.This also shows that if we tend to divide the box into smaller pieces rather than a large piece, then the algorithm will run faster and better. In fact, texture packaging uses a small set of graphs stored as large maps, and then each region is rendered multiple times.


禁止旋转时四种offline算法的时间消耗与数量关系图
tims cost(ms)
16000
14000
12000
10000
8000
6000
4000
2000
0
0
20000
40000
60000
80000
100000
120000
Box Number：N
N-FFDH
FFDH
NFDH
BFDH

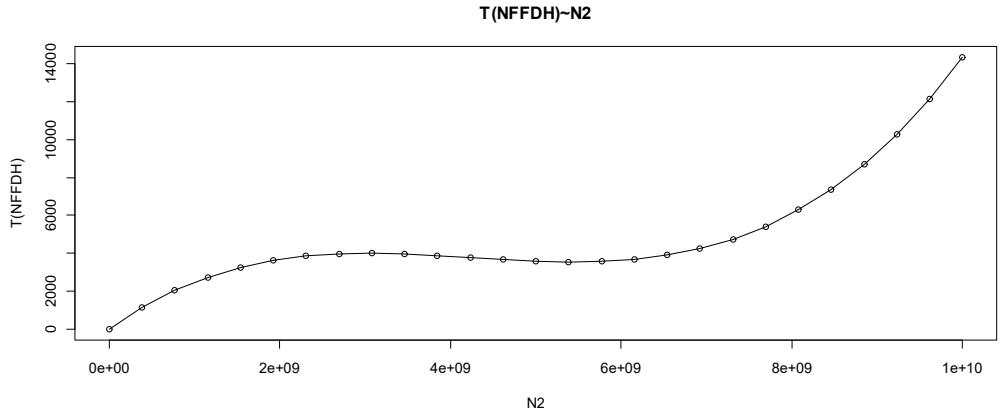

允许旋转时四种offline算法的时间消耗与数量关系图
time cost(ms)
16000
14000
12000
10000
8000
6000
4000
2000
0
0
20000
40000
60000
80000
100000
120000
Box Number：N
N-FFDH
FFDH
NFDH
BFDH


禁止旋转时四种offline算法的近似比率与数量关系图
近似比率
120
100
80
60
40
20
0
0
20000
40000
60000
80000
100000
120000
Box Number：N
N-FFDH
FFDH
NFDH
BFDH

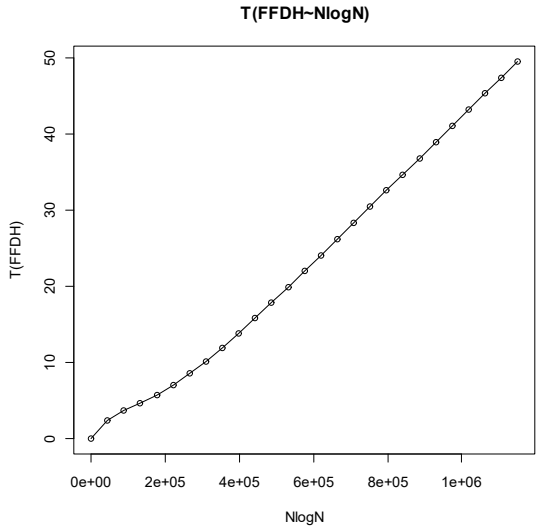


Next, we fit the test set and the theoretical time complexity of each algorithm to observe their fitness.

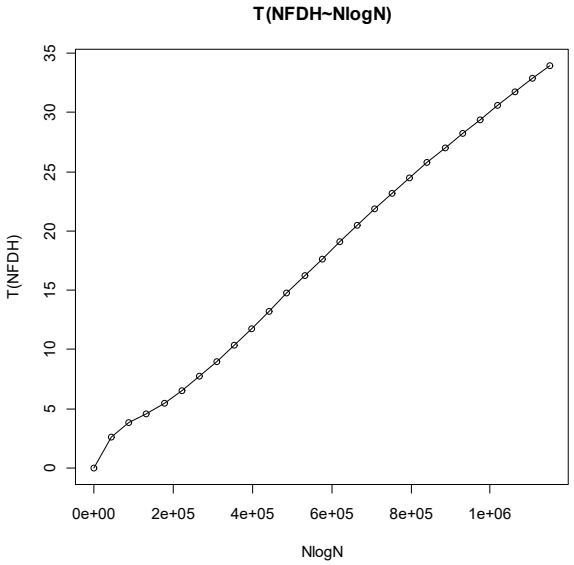
(1)NFFDH



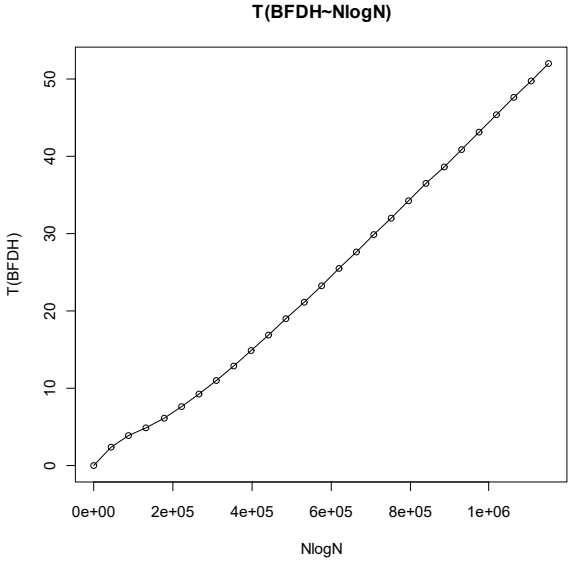
(2)FFDH



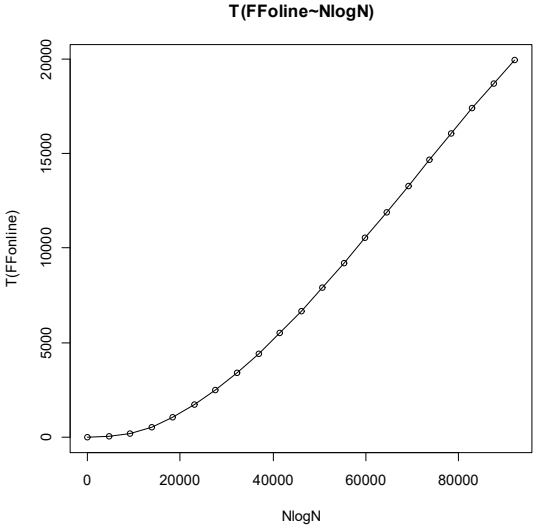
(3)NFDH



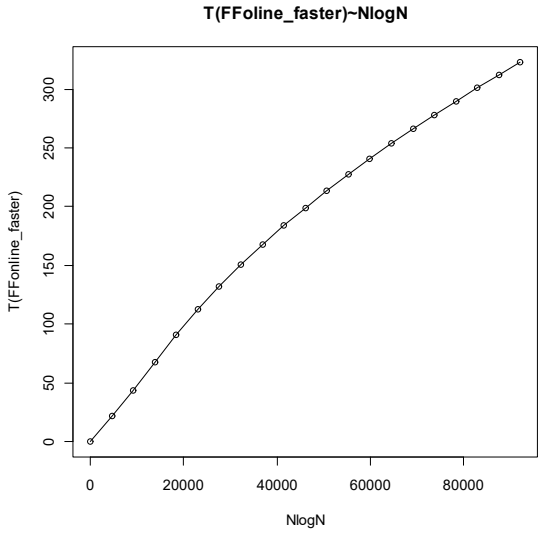
(4)BFDH



(5)FF-Online-ordinary



（6）FF-online-faster



Appendix:　Source Code (in C++)

Our source codes include *texture_packing.cpp*. Thus you can click on the above file to view the source code, so it won't go into details here.

References

[1]B.S. Baker, D.J. Brown, and H.P. Katseff. A 5/4 algorithm for two-dimensional packing. *Journal of Algorithms*, 2:348--368, 1981.

[2]J.O. Berkey and P.Y. Wong. Two dimensional finite bin packing algorithms. *Journal of Operational Research* Society, 2:423--429, 1987.

[3]B. Chazelle. The bottom-left bin packing heuristic: An efficient implementation. *IEEE Transactions on Computers*, 32:697--707, 1983.

[4]F.K.R. Chung, M.R. Garey, and D.S. Johnson. On packing two-dimensional bins. *SIAM J. Algebraic Discrete Methods*, 3:66--76, 1982.

[5]J.B. Frenk and G.G. Galambos. Hybrid next-fit algorithm for the two-dimensional rectangle bin-packing problem. *Computing*, 39:201--217, 1987.

[6]E.G. Coffman Jr and M.R. Garey and D.S. Johnson and R.E. Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms.*SIAM Journal on Computing*, 9:808--826, 1980.

[7]A. Lodi and S. Martello and D. Vigo. Neighborhood search algorithm for the guillotine non-oriented two-dimensional bin packing problem. In S. Voss and S. Martello and I.H. Osman and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for optimization*, pages 125--139. Kluwer Academic Publishers, Boston, 1998.

[8]A. Lodi and S. Martello and D. Vigo. Approximation algorithms for the oriented two-dimensional bin packing problem. *Journal of Operational Research* Society, 112:158--166, 1999.

[9]A. Lodi and S. Martello and D. Vigo. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing*, 11:345--357, 1999.

[10]I. Schiermeyer. Reverse-fit: A 2-optimal algorithm for packing rectangles. *In Proceedings of 2nd European Symposium on Algorithms*, pages 290--299, Utrecht, The Netherlands, August 1994.

[11]D.D. Sleator. A 2.5 times optimal algorithm for packing in two dimensions. *Information Processing Letters*, 10(1):37--40, 1980.

[12]A. Steinberg. A strip-packing algorithm with absolute performance bound 2. *SIAM Journal on Computing*, 9:401--409, 1997.

Author List

Programmer: Wang Yinzhao

Tester&Report Writer: Zou Ziyu

PPT&Presentation:Wang Yiheng