Software Assignment-1

Name	Entry No
Ahilaan Saxena	2023CS10076
Popat Nihal Alkesh	2023CS10058

Design Decisions:

1) Sleator's Algorithm:

Motivation: As a starting point, we implemented the First-Fit Decreasing Height (FFDH) algorithm (code not included), which is as follows:

- Sort the gates in decreasing order of heights
- Iterate over possible width values of bounding box
- Place the gates one by one in a level, move to the next height level if the gate width sum overshoots the chosen box width value
- Find corresponding height for every width and store the heights and widths which give min. area

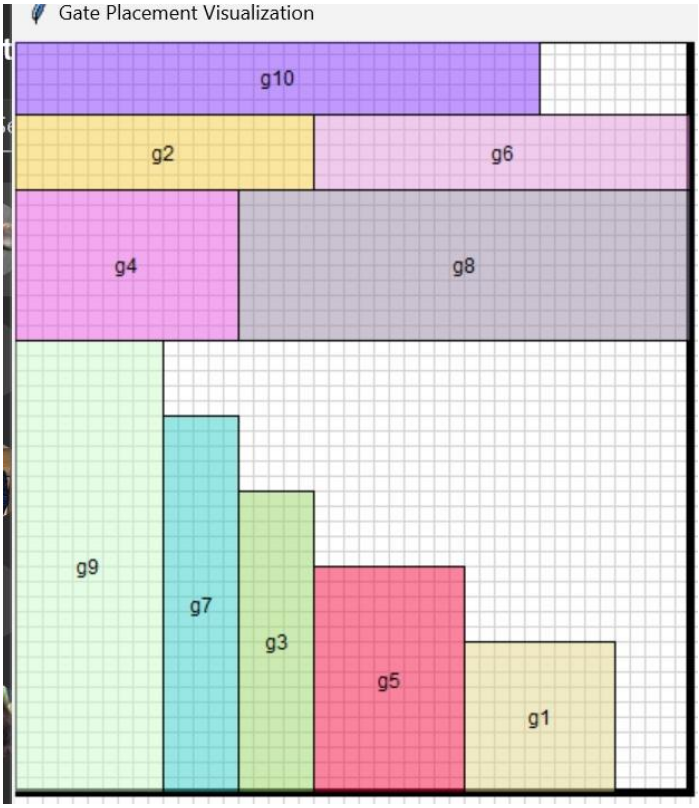


Figure 1: FFDH algo; 72.22% packing efficiency

Can this be modified? We explored various modifications of FFDH and decided to implement Sleator's algorithm, proposed by Daniel D.K.D.B Sleator, which is as follows:

- Iterate over possible width values of bounding box
- Choose all gates with width greater than half the chosen box width ($w_{gate}>w_{box}//2$) and stack them along the height at the bottom of the box till h0
- Sort the remaining gates ($w_{gate}<=w_{box}//2$) in decreasing order of heights and partition the box at $w_{box}//2$
- Now store the maximum heights of the left and right sides, h1 and h2 (both initialized to h0). If h1>=h2, then start placing the gates on the left side till you overshoot the $w_{box}//2$ partition, else place the gates on the right side till you overshoot w_{box}, update h1 and h2 accordingly.
- Repeat the process till you run out of gates. Store the height of the box as max (h1, h2) at the end of the process.

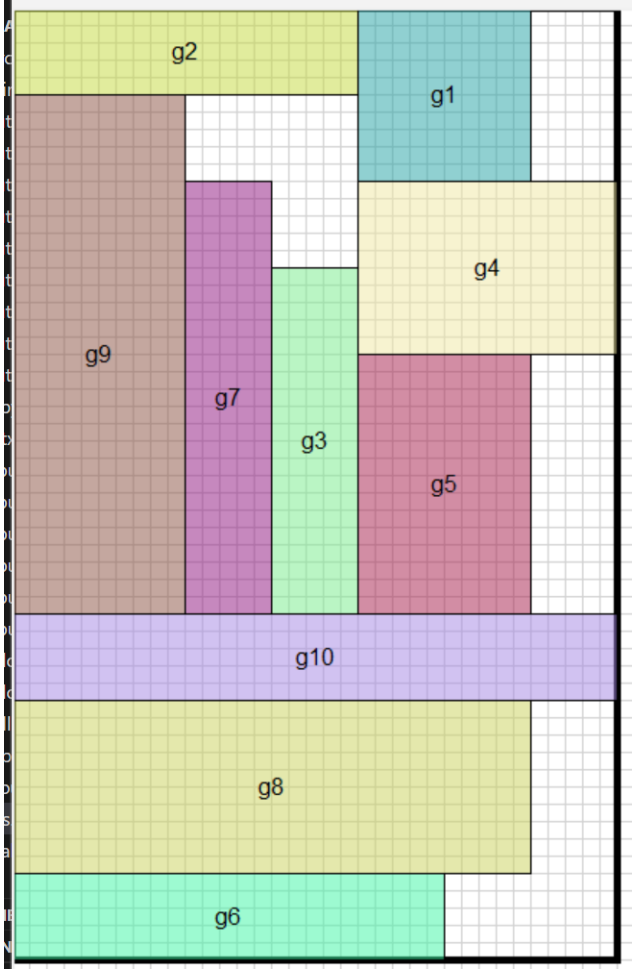


Figure 2: Sleator algo; 84.42% packing efficiency

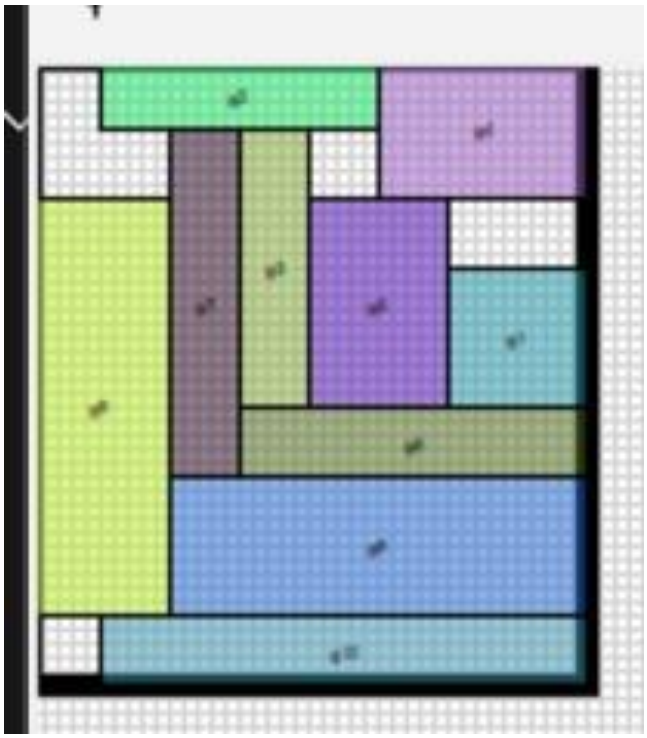


Figure 3: Tetris algo; 90.28% packing efficiency

2) Tetris algorithm:

Intuition: The improved results in Sleator algo motivated us to brainstorm more greedy algorithms by ourselves. It seemed natural that if we are filling the rectangles in height reverse sorted order, after first level is filled from left to right, right region will be lower

- First sort the rectangles based on height in descending order
- Iterate through widths of bounding box
- For each width, make independent "levels" in alternate side order
- A "level" is made by arranging the gates side by side, till we can do it in given width limit
- First level is made from left to right, second level from right to left and so on
- Now, start from bottom most level, put each block as low as it can get. For next level, for individual block, check what is the maximum height already filled in the range of its width and arrange it as low as possible accordingly

This algorithm worked well on some cases. However, it didn't give satisfactory result for cases having wide blocks. So, we came up with a modification to this algorithm:

- First, take all the rectangles with width >= (width of bounding box)/2
- Stack them at bottom right, with decreasing order of width
- Then sort remaining blocks in increasing order of height and perform the same algorithm as discussed earlier

Intuition for this modification: In initial algorithm, all the wide blocks will generally be in the upper region, where a whole level will be occupied by those single wide blocks. This would take up extra space of the whole level. Instead, if we put wide blocks in bottom right initially, then high (and probably thin) blocks will fit the vacant space left by wide blocks later and same algo continues. This will give better result.

- In the submitted file, function of first algorithm is named "direct" and modification is named "new". Every time, both functions will execute and the one giving better answer (less area) will be considered.

Time complexity analysis:

1) **Sleator algorithm:**
   - No. of gates (n)<=1000; Max width <= 100
   - Max possible sum of widths = 100*1000 = 100000
   - Hence, iteration through possible widths takes O(∑w) time
   - For each w:
   - Iterating through the gates to select gates having $w_{gate} > w_{box}//2$ takes O(n) time
   - Sorting the remaining gates takes O(nlogn) time
   - Placing the remaining rectangles takes O(n) time
   - Overall time complexity = O(∑w * (nlogn+n)) = O(∑w*nlogn) which is of the order (10^5 * 10^3) ~ 10^8 operations
   - To remove the summation, we can approximate ∑w = n*w when all gates have same width 'w'. Hence, O(n^2 * w)
   - This is verified by the following graphical analysis:

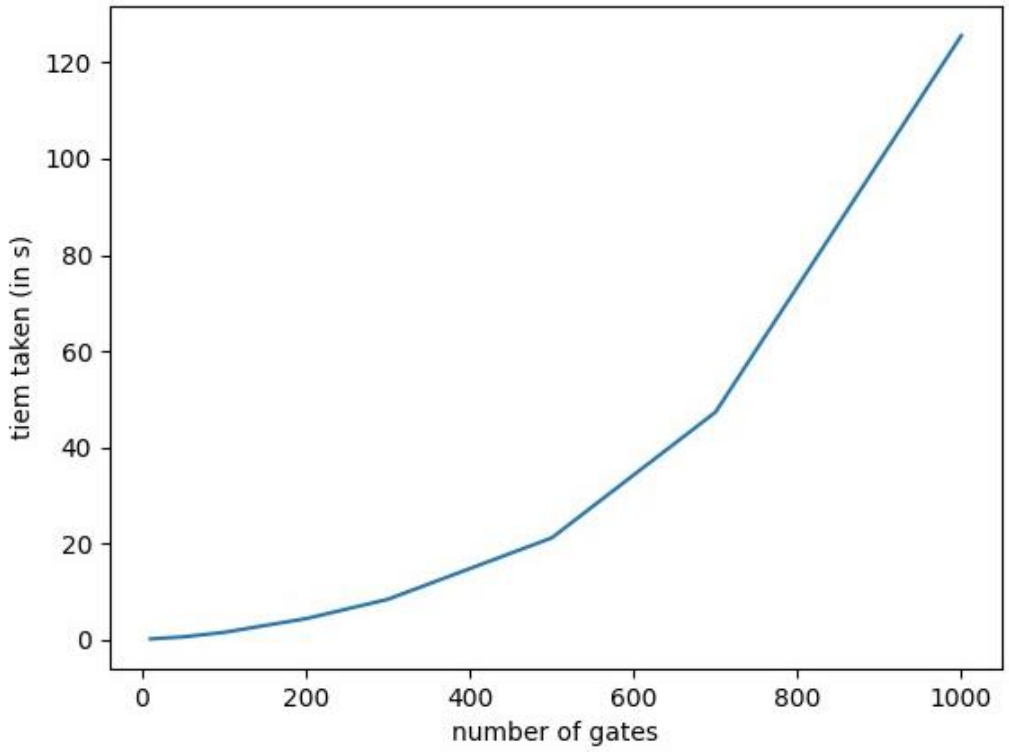


Fig. 4: A parabolic shape is obtained for the n vs t graph which verifies our hypothesis

2) Tetris algorithm:

- The list "heights" is initialized with the width limit of bounding box. It will be modified later. **[O(wlim)]**
- Iterating over all blocks to create levels. **[O(n)]**
- Iterate through each level, inside each level, iterate through each block of that level. For each block, increment the "heights" of x-region of that block. Overall, **[O(n*w)]** (w is avg width of a block)
- This whole thing is done for a particular wlim, iterating wlim from wmin=max width of a single block to wmax=sum of widths of all blocks, **[O(Σw)]**

➢ Overall, **O((Σw)*(n*w))** = **O((Σw)^2)**

➢ According to given constraints, worst case ≈ O(10^{10})

➢ But in practical cases, when worst case input (named constraint_tc.txt in zip file) was given, both functions took around 47 minutes each, which is not feasible.

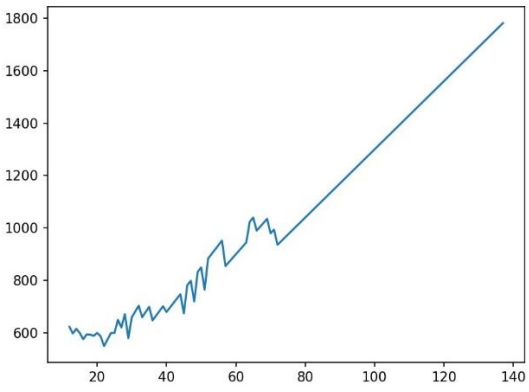
```
2744.5493513999972 2826.1574792000465
```

Figure 3:Time taken by each function (in s)

Optimization:

- First, we observed that this algorithm, almost always, gave a grid very close to a square
- For solid conclusion, we plotted a few results

Here, x axis represents width of bounding box and y axis represents respective minimum area given by our algorithm.

Total area (sum of individual areas of each rectangle)	**Plots**
500	

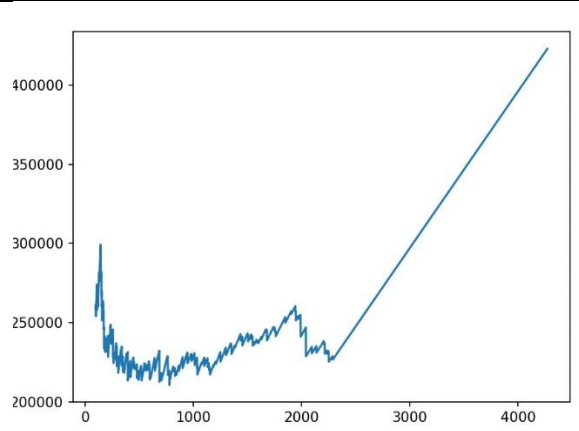
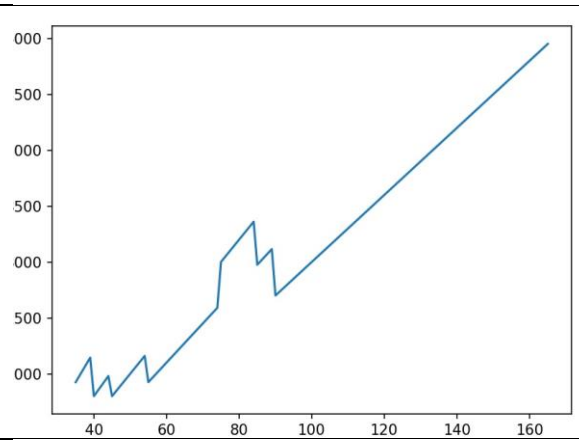
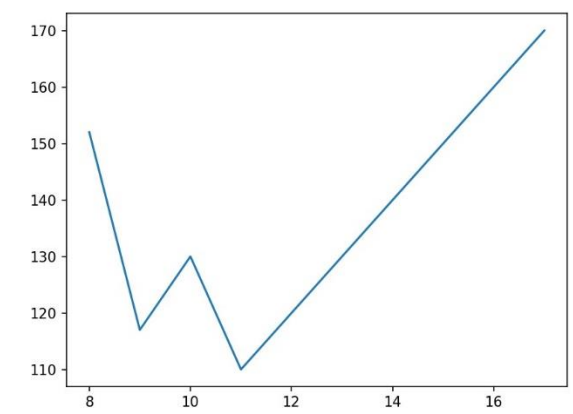
4000	
1625	
90	

Table 1: Plots for various test cases compared with total area

All these plots show that minimum area is always obtained at the width close to √(total area). i.e. side length of hypothetical square that PERFECTLY covers all rectangles.

However, this optimization cannot be done for every case. As, if we have a very few gates, the difference in packing efficiency in every packing would be significantly different. So even a possibly slightly worse packing (square) would result in a large drop in packing efficiency.

- Considering this, we added this optimizing line in our function:

```
if(len(gatees)>100):
    side=math.sqrt(totarea)
    minw=max(int(2*side/3),minw)
    maxw=min(maxw,int(4*side/3))
```

Figure 5: Modification for optimization

That is, if number of gates given is more than 100, then iteration is reduced to iteration from 2s/3 to 4s/3, where s is the side length discussed earlier. The range of 2s/3 and 4s/3 is selected by checking the trade-off of time taken and accuracy, of various ranges for big test cases.

New time complexity (for n>100): **O((Σw)*√(total-area))**

Practically, in worst possible case, each function took about 40 seconds, and, in most cases, this optimization did not change the total area the function was giving.

Graphical comparison:

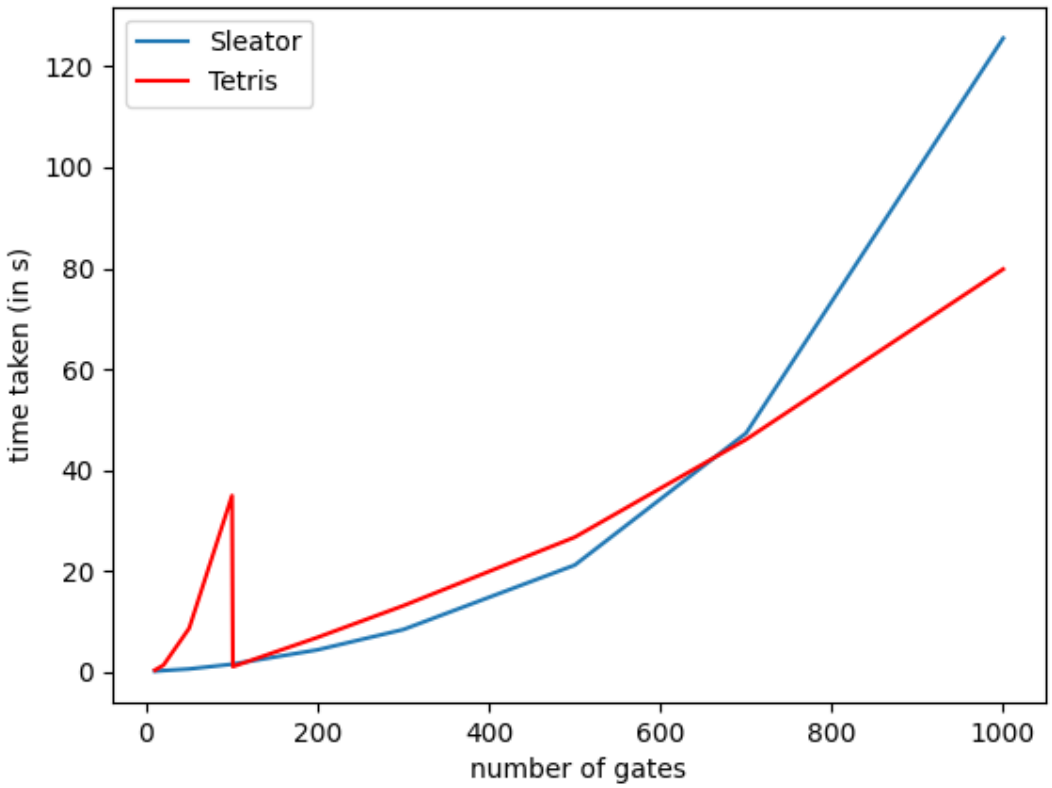
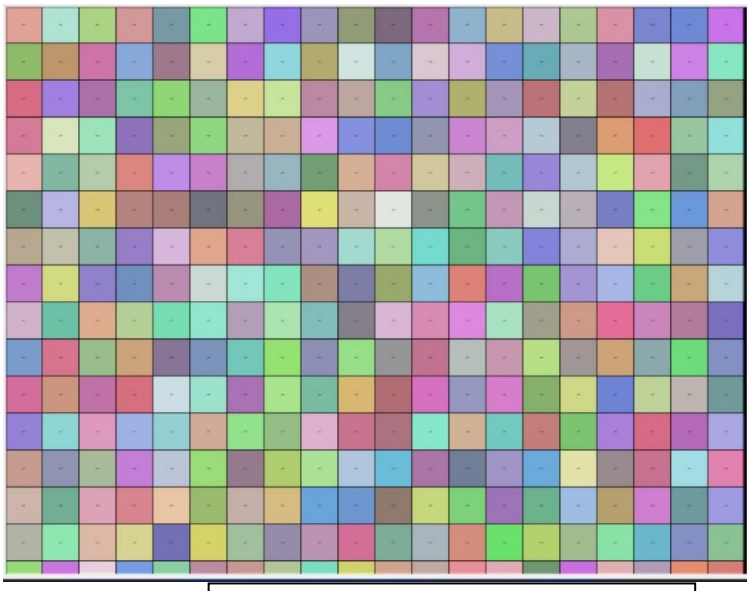


Figure 6: Graphical analysis of time complexity: Notice how in Tetris algorithm, there is a discontinuity (sudden drop) at n=100. This shows the effect of optimization discussed above

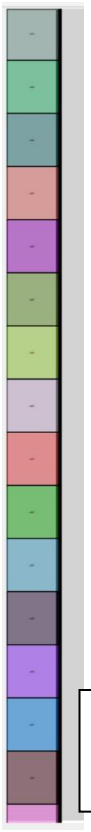
Test Cases:

1) *Equal 500*

- Checks the packing for 500 gates, each of dimensions 47*47
- Most common idealization of practical circuits, which have large number of gates having similar sizes and aspect ratio close to 1
- Both algos give 100% packing as expected, with the tetris algo giving a more square-ish output, thus making it more practical compared to Sleator algo (might be improved by setting max bound on height)



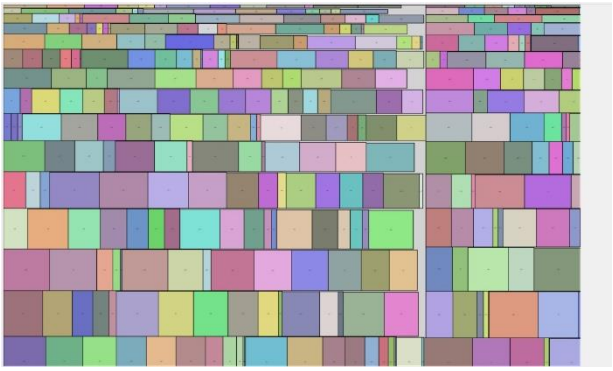
Equal 500, Tetris (incomplete visualization due to large no of gates)



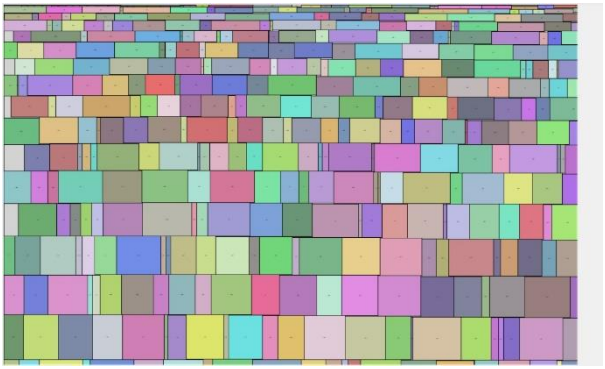
Equal 500, Sleator (incomplete visualization due to large no of gates)

2) *Random 500*

- Generates 500 gates using randint function (dimension range 1 to 100)
- Tests the ability of the algorithms to work on random circuits with no well-defined pattern of gate sizes
- Both algos give good results (94.75% and 98.24% respectively), pointing to the fact that the packing efficiency is enhanced at large number of gates



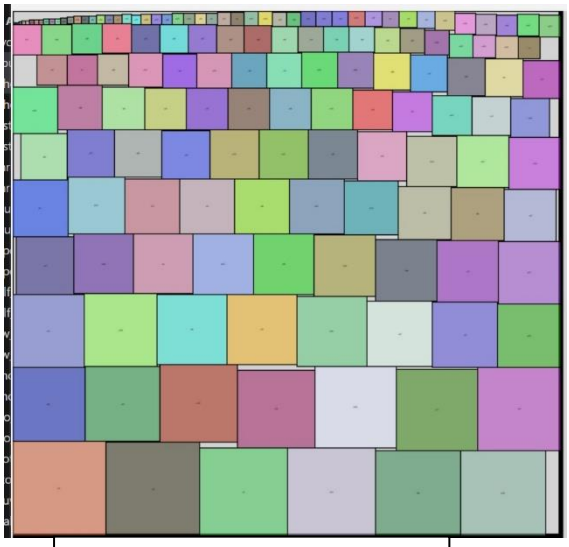
Random 500, Sleator (incomplete visualization due to large no of gates)



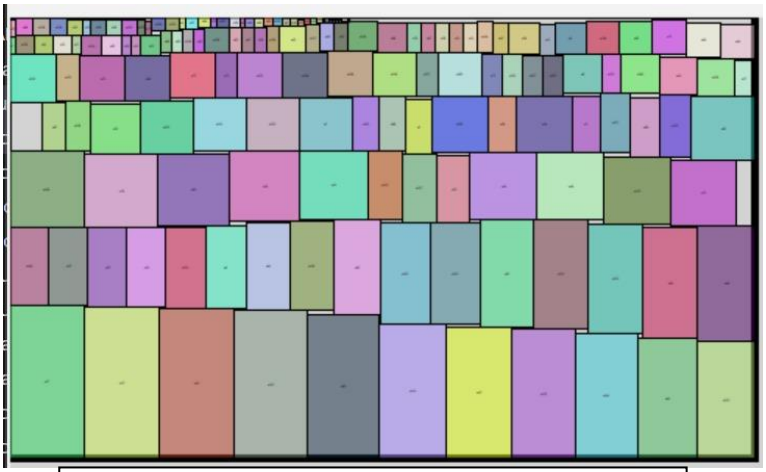
Random 500, Tetris (incomplete visualization due to large no of gates)

3) *Exponential Sampling*

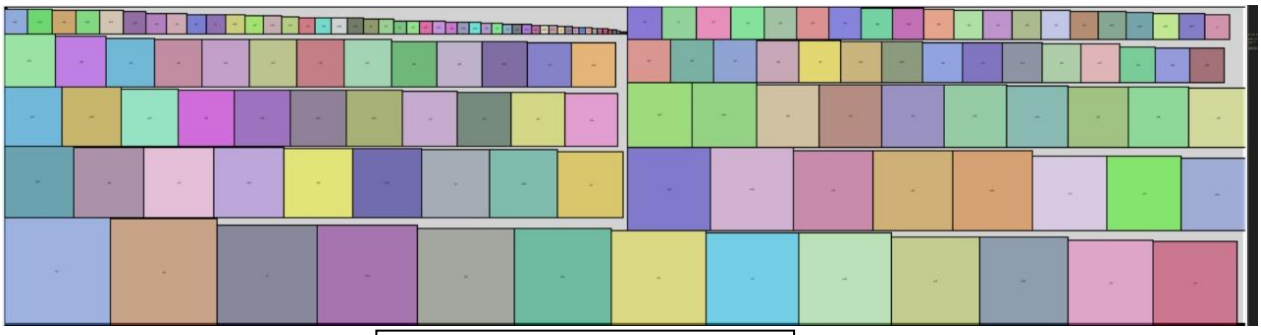
- Samples 200 gates from an exponential distribution for the first subcase and randomly doubles one of the dimensions for the second subcase
- The first subcase mirrors the frequency of large (low freq.) and small gates (high freq.) in practical circuits, while the second one additionally introduces the concept of NOT gates or inverters which usually have an aspect ratio close to 2. Sleator performs decent by its standards while Tetris outperforms, because the former evidently pushes all small gates into the top row in sorted order, leaving ample space on one side. This reflects a downside of discrete levelling in FFDH-like algorithms
- Exponentially sampled: Sleator: 90.96%; Tetris: 97.08%
  Half elongated exponentially sampled: Sleator: 90.44%; Tetris: 97.13%



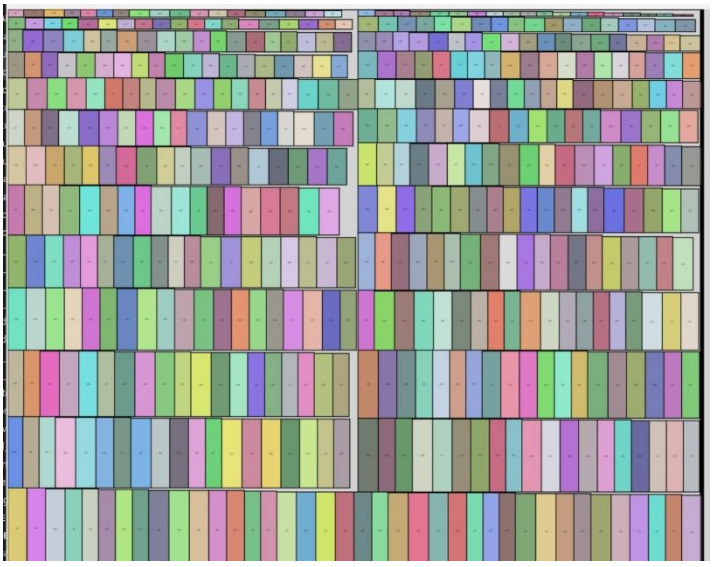
Exponentially sampled, Tetris



Half elongated Exponentially sampled, Tetris



Exponentially sampled, Sleator

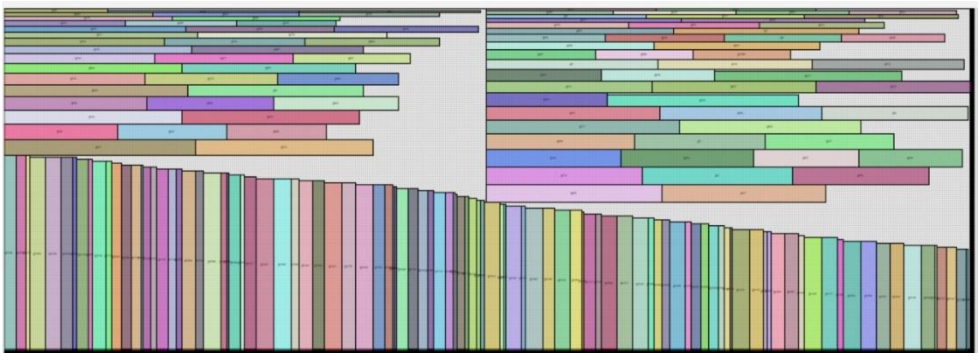


Half elongated Exponentially sampled, Sleator

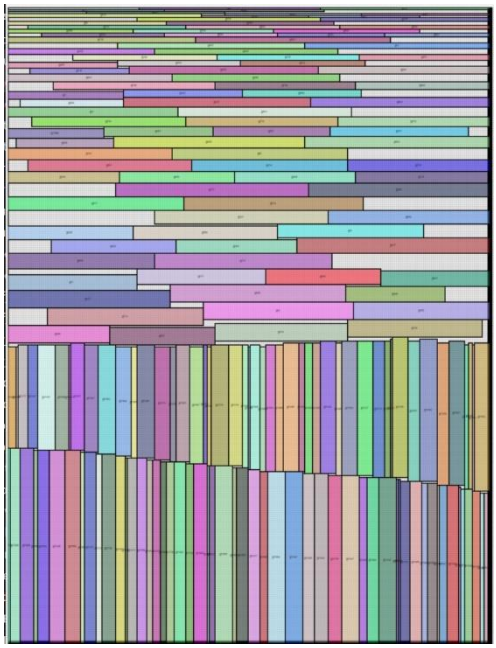
4) *Elongated Gates*

- *The dimensions are generated by setting low width for large height and vice versa. The gates are highly elongated thus. This case is impractical but still worth a mention as both algos perform subpar (83.84% and 90.44%)*

[Credit: Dhruv Pawar, a friend of ours who came up with this case]

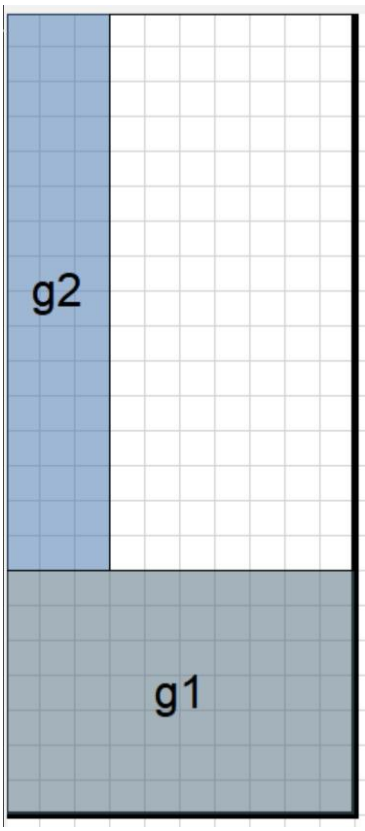


Elongated gates, Sleator

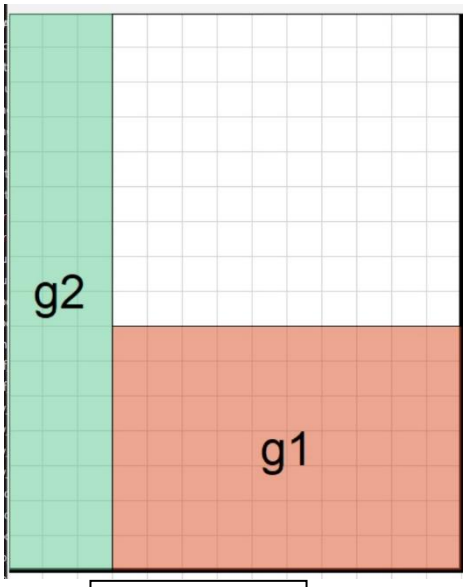


Elongated gates, Tetris

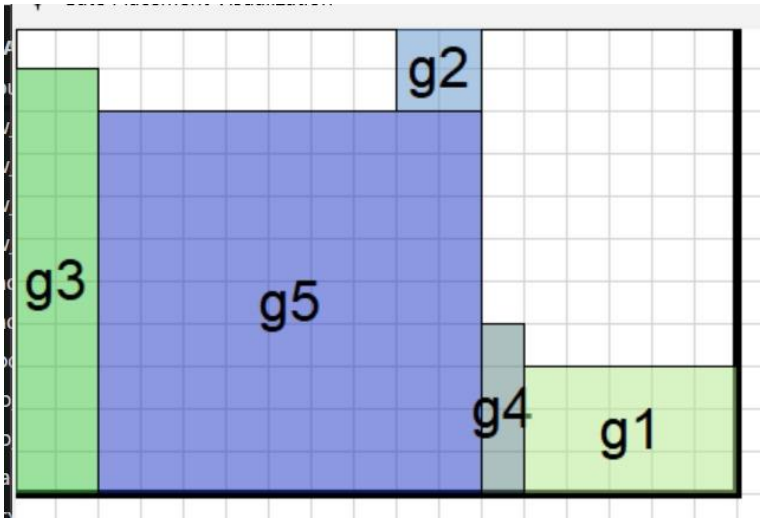
5) *Less no. of gates and high variance*
   - *Both algorithms perform poorly on a trivial case of 2 gates (51.3% and 56.73% respectively) as well as on a set of 5 gates having large size variance (66.31% and 86.71% for Sleator and tetris resp.)*



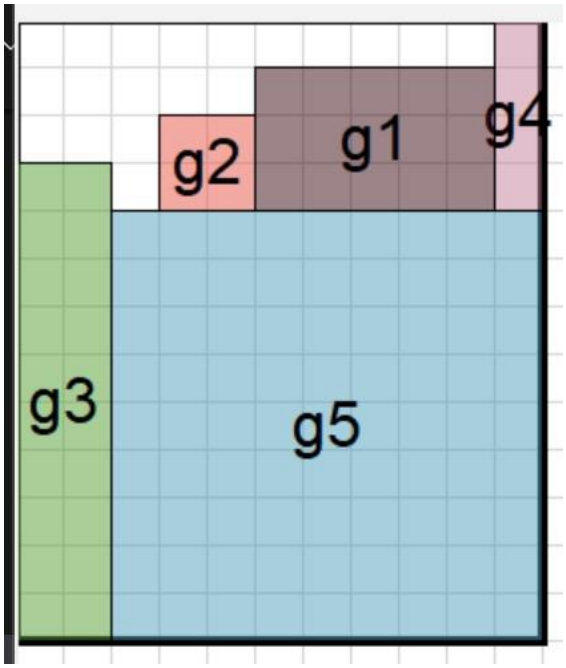
2 gates, Sleator



2 gates, Tetris



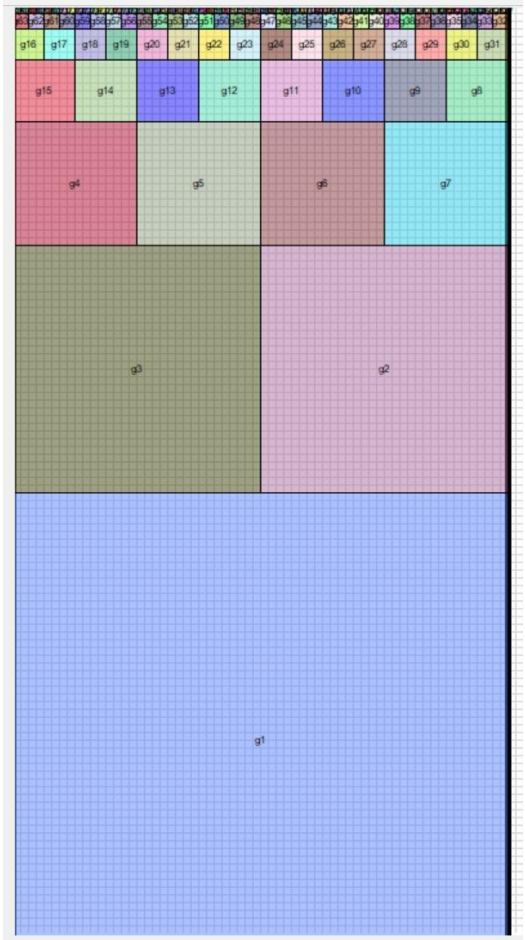
5 gates (high var), Sleator



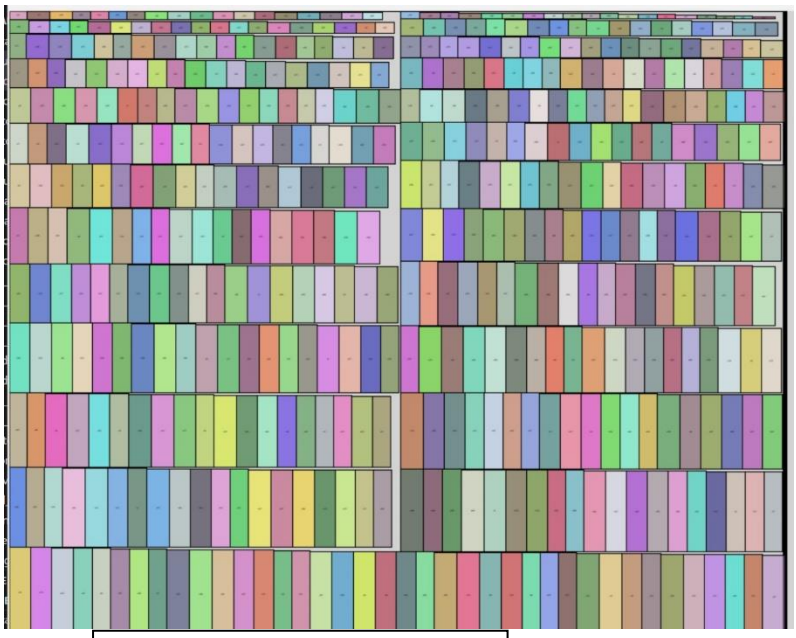
5 gates (high var), Tetris

Other cases worth mentioning are:

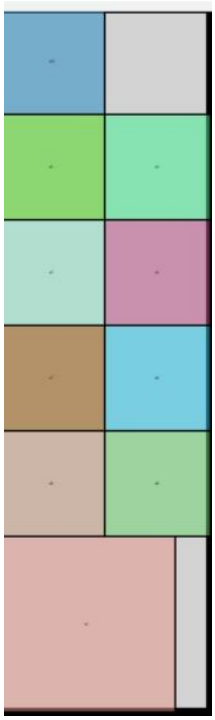
- Taking 2^(6-i) no. of gates having dimension 2^i * 2^i i.e. 1 64*64, 2 32*32, 4 16*16 gates and so on, gives 100% packing for both cases.
- Keeping width variance low and height variance high, or width variance high and height variance low: both give great results (95.38% and 98.9% for former case, for Sleator and tetris resp.)
- A notable observation for Sleator was for a test case where 1 gate had dimension 100*100 and 9 gates had dimension 60*60. It was expected that the algo would stack all of them on top of one another (as none of them is less than half the max. width). But it ended up giving 88.33% efficiency by auto-adjusting the box width to 120.



Powers of 2 (same packing for both algos)



Low variance on width, Sleator



100*100 (1 gate), 60*60 (9 gates) for Sleator

References:

- What algorithm can be used for packing rectangles of different sizes into the smallest rectangle possible in a fairly optimal way? - Stack Overflow
- Colson, David, 2020, Exploring rectangle packing algorithms (david-colson.com)
- Sleator, Daniel, 1980, A 2.5 times Optimal Algorithm for packing in two dimensions, CS department, Stanford university