Explanation and Analysis of our solution (Including Solution of Test Cases provided by college)

Authors

Deep Raval
(Enroll: 171310132048)
(deepraval.ict17@gmail.com)

Jaymin Suhagiya
(Enroll: 171310132056)
(jayminsuhagiya.ict17@gmail.com)

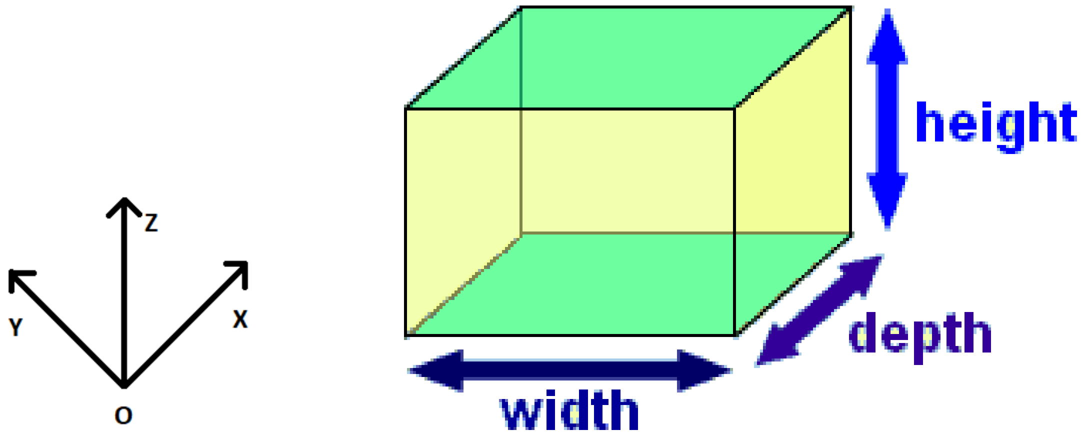
Table of Contents

Introduction

Let's start with short description of given problem: We are given rack with some capacity (i.e. length, breadth, height). We also Have some number of switches (5 in this case) with some length, breadth, height, value and Instances. We are required to place switches in rack such that volume as well as value in rack is maximized. (Note that we have also solved problem where Switches can be placed in different orientations also.)

Given Problem is special case of container loading problem i.e. KCLP (Knapsack container Loading Problem). We can view this problem as sort of 3D Knapsack problem. Classical Knapsack Problem (i.e. 1D) is $\boldsymbol{NP\ Complete.}$ Hence given problem is $\boldsymbol{NP\ Hard.}$ So, there may not be any solution with polynomial time complexity unless $\boldsymbol{P = NP}$ (Which seems unlikely).

Finally, before starting let's set the record straight by defining notations and coordinate system we will use. Below is the coordinate system and measuring convention that we will use.



[Fig 1]

Assumptions

Before discussing solution let's see assumptions assumed by us:

1. Switches can't overlap each other.
2. Switches can be placed on one another without any balancing or weight issues.
3. Density is even across whole switch volume.
4. Switches can't be braked or bended.

Approaching Solution

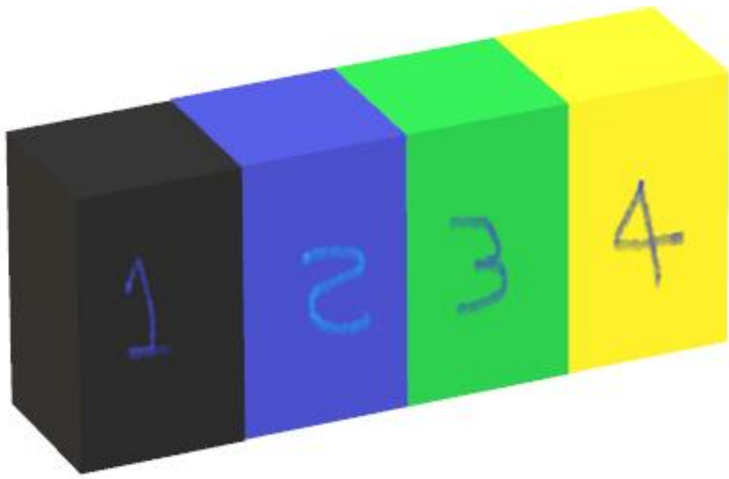
As mentioned earlier given problem is $\boldsymbol{NP\ Hard.}$ Naïve solution is extremely time and memory consuming hence it is not practical. For ex. Let's say we have rack size of $1000 \times 1000 \times 1000$. We may need 3-4 GB memory (assuming integer requires 4 bytes memory) only for representing rack. Moreover, amount of recursive calls in naïve method take memory usage to extreme level. Hence, we decided that for now implementing naïve solution is not practical (Although we can represent rack as 3D array of bits but still recursion overhead is way too large). Dynamic programming in this case also won't work. So, we decided that we will use heuristic approach to get solution (which may not be optimal).

A Heuristic Solution

First let's denote all frequently used symbols:

$$H = Height\ of\ rack$$
$$W = Width\ of\ rack$$
$$D = Depth\ of\ rack.$$

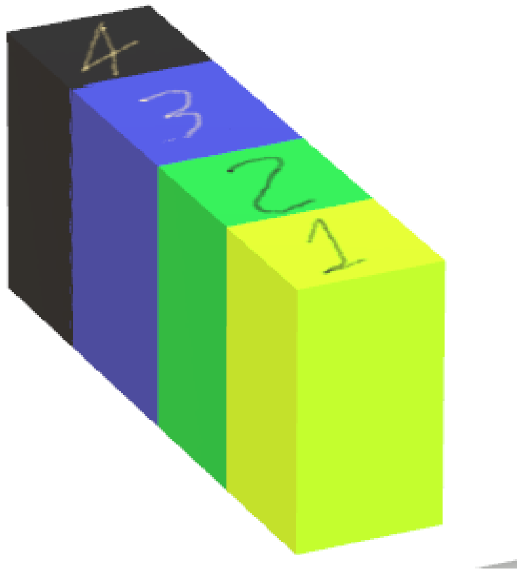
We used wall building approach (i.e. packing part of rack at a time). You can imagine rack as union of racks with smaller depths. We will take each depth and solve problem for smaller rack and we will move further with another rack of another depth. For example consider rack in fig 2. We will solve each smaller rack in the same order as their respective number.



[Fig 2]

Now the question is how to decide depths and how to solve each depth. We predetermined the depth list (as well as width list which is mentioned later) which we want to try for solution. We determine average of all dimensions, average of all minimum dimensions, and average of all maximum dimensions. We push these average values in our list. Again we iterate over all dimensions and push each unique dimension in our list.

Now we have smaller rack with some depth $d \leq D$. Now let's solve this smaller rack with dimension $H \times W \times d$ .Again we can imagine this smaller rack as union of small stripes which is shown in fig 3.



[Fig 3]

We will solve these stripes. Let's consider a stripe with some width w (chosen from width list- It is similar to depth list which was discussed earlier. The only difference is that width list contains $rack\_width$) which have dimensions $H \times w \times d$. Note that now our width and depth is fixed for currently being considered smaller rack.

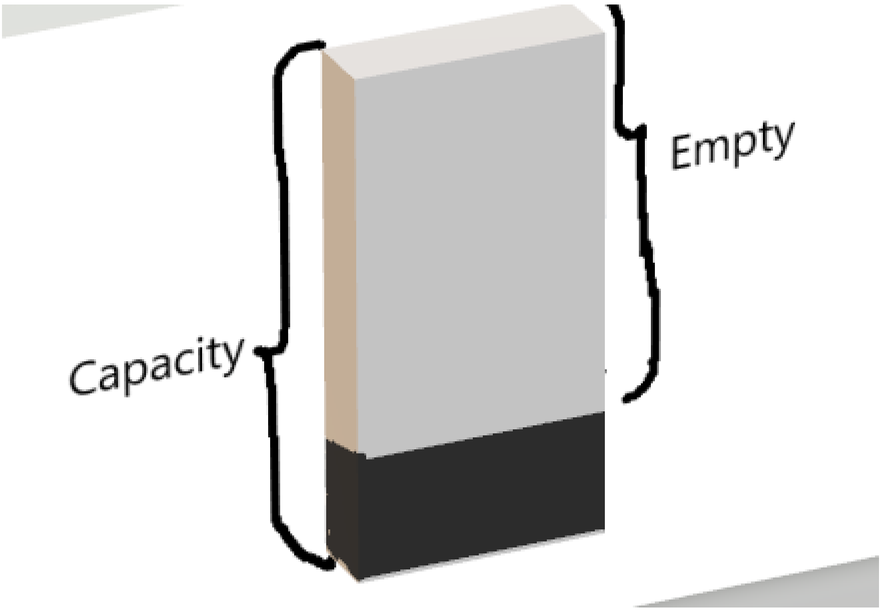
Now let's take such switches which can be placed in $w \times d$ (including possibility of different orientations and pairing) grid (Let's ignore height for the moment). Let's define our symbols as:

$$S = Collection\ of\ all\ \ switches\ which\ can\ be\ placed\ (in\ w \times d).$$
$$V = Vector\ containing\ value\ of\ each\ switch.$$
$$C = Vector\ containing\ height\ of\ each\ switch.$$

Now we need to fit switches from S in $H \times w \times d$ such that height does not exceed H. Note that switches will be placed on one another. This problem is classical 1D knapsack problem with:

$$H \ as\ Capacity\ of\ bag.$$
$$C \ as\ cost\ of\ each\ item\ (here\ cost\ is\ switch's\ height).$$
$$V \ as\ value\ of\ each\ item.$$

We can solve this 1D knapsack problem with Dynamic Programming. Thus, we arrived at 1D knapsack from 3D knapsack. Now we solve this knapsack problem and calculate value as well as determine which switches to take. Selected switches than can be removed from global switch pool. See fig 4 for knapsack representation. This was the most basic introduction of our method.

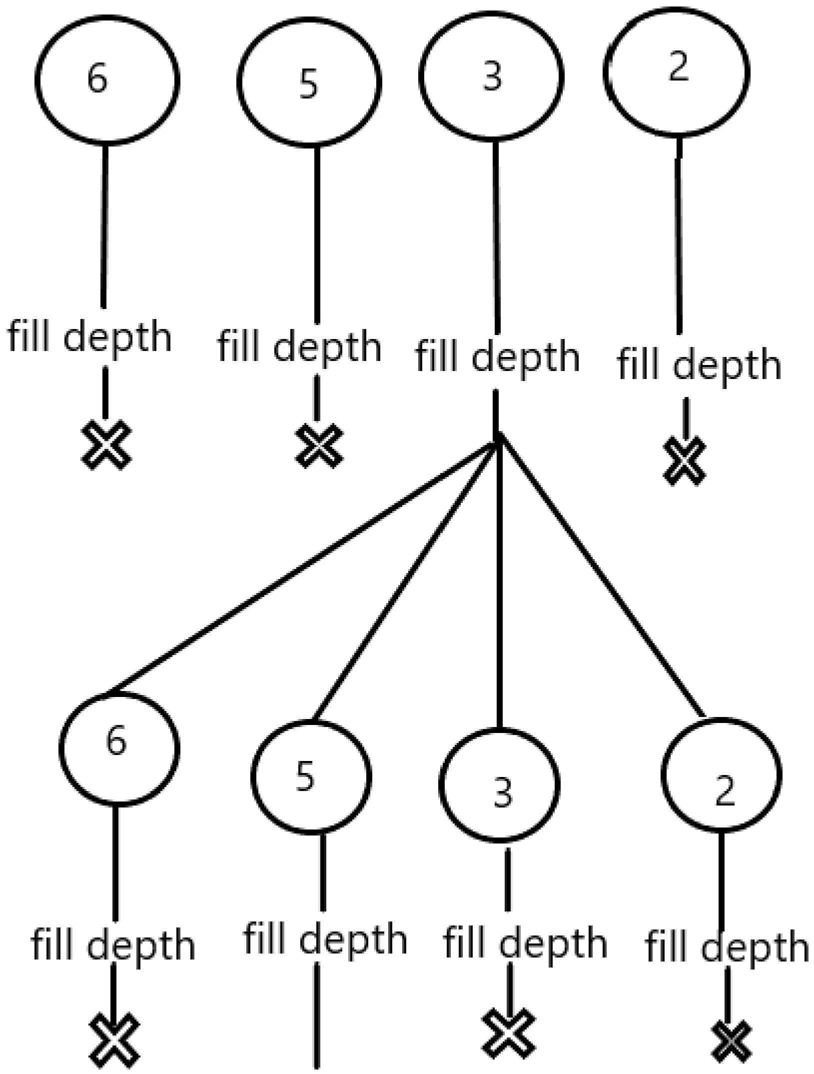


[Fig 4]

Some Important Optimizations

To Compute all possibilities with each depth is computationally expensive. So, what we do is compare solutions from each depth and only move further with most promising solution. Let's take an example. Let's say we have $depth\ list = \{6, 5, 3, 2\}$. Now let's assume that solution with depth 3 was looking good at that time. We will not explore other depths further and only explore 3 with again same depth list (Similar thing is also done with widths for each depth). Now assume while exploring further we find that after 3, 5 is most promising than further only 5 is explored (See fig 5). How to decide most promising depth/width is discussed later.

This optimization makes program run really fast. But we are discarding future calls seeing current situation (In other words we're making greedy choice).



[Fig 5]

In each $fill\_depth$ similar graph is formed but with $fill\_stride$. So, at each $fill\_depth$ this whole process is done but with $fill\_stride$. In $fill\_stride$ for each width knapsack problem is solved. Now let's look at actual algorithm.

Another optimization which we can do is pairing. Pairing is putting two switches together whose height is same and can be placed in knapsack. If this new switch can fit in knapsack than this new switch is also considered for solution (ofcourse after removing switches which made this new switch from consideration). This decreases probability of creation of small empty pockets. If the new switch is picked in knapsack two switches making it is placed in rack accordingly.

Algorithm

Global Variables:

$$N = Number\ of\ Switches\ (5\ in\ this\ case).$$

$$reck\_height\ = Height\ of\ the\ rack.$$

$$reck\_width\ = Width\ of\ the\ rack.$$

$$reck\_depth\ = Depth\ of\ the\ rack.$$

$$switches\_value\ = Vector\ with\ value\ of\ all\ switches.$$

$$switches\_inst\ = Vector\ with\ instances\ of\ all\ switches.$$

$$switches\_cord\ = 2D\ Vector\ with\ dimenssion\ of\ all\ switches$$

$$depth\_list\ = Vector\ with\ promissing\ depths$$

$$width\_list\ = Vector\ with\ promissing\ widths$$

Before starting let's see solution format which some functions take as an argument and also return it.

$Solution\ format\ (2D\ vector)$
$= \{\{current\ value\}, \{current\ volume\}, \{\ Avaialable\ switches\}, \{\ switches\ info\}\}$

$\boldsymbol{solve\_KCLP}()$:

1. $take\ input\ and\ initilaze\ global\ variables\ accordingly$
2. $Initilaze\ best\ solution, best\ local\ solution$
3. $While(rack\ has\ depth\ remaining \leq some\ depth\ d\ in\ depth\_list)$
4. $\quad best\ local\ solution = best\ solution$
5. $\quad for(each\ depth\ in\ depth\_list)$
6. $\quad\quad if(is\_better\_depth(solution\ for\ current\ depth, best\ local\ solution))$
7. $\quad\quad\quad best\ local\ solution = current\ solution$
8. $\quad best\ solution = best\ local\ solution$

The solution for current depth is found by $\boldsymbol{fill\_depth}$ function which takes previous solution (and current x coordinate – determined by current depth d) as an argument and returns best solution for that depth combined width previous solution. $\boldsymbol{is\_better\_depth}$ takes two solutions and tells which is better.

$\boldsymbol{fill\_depth(x, d, previous\ solution)}$:

1. $Initilaze\ best\_solution, best\_local\_solution$
2. $While(rack\ has\ width\ remaining \leq some\ width\ w\ in\ width\_list)$
3. $\quad best\ local\ solution = best\ solution$
4. $\quad for(each\ width\ in\ width\_list)$
5. $\quad\quad if(is\_better\_width(solution\ for\ current\ width, best\ local\ solution))$
6. $\quad\quad\quad best\ local\ solution = current\ solution$
7. $\quad best\ solution = best\ local\ solution$
8. $\boldsymbol{return}\ best\ solution$

$\boldsymbol{fill\_depth}$ function is really similar to $\boldsymbol{fill\_depth}$ function. Now solution for current depth is found by $\boldsymbol{fill\_stride}$ function which takes previous solution (and current x, y coordinates – determined by current depth and width and current depth d and current width w) as an argument and returns best solution for that width combined width previous solution. $\boldsymbol{is\_better\_width}$ takes two solutions and tells which is better.

$\boldsymbol{fill\_stride}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{d}, \boldsymbol{w}, \boldsymbol{previous\ solution})$:

1. $Determine\ all\ switches\ which\ can\ fit\ in\ d\ \times w(considering\ all\ orienations\ (or\ not) and\ pairing)$.
2. $Solve\ Knapsack\ problem\ for\ Determined\ switches\ and\ H$.
3. $append\ value, volume\ and\ coordinates\ of\ switches\ to\ previous\ solution\ (Determined\ by\ knapsack's\ solution)$.
4. $\boldsymbol{return}\ previous\ solution$

$\boldsymbol{fill\_stride}$ function actually places switches after solving knapsack problem for them.

Strategy for $\boldsymbol{is\_better\_depth}(\ \boldsymbol{and\ width})$ function:

$\boldsymbol{is\_better\_depth}(\ \boldsymbol{and\ width})$ is one of the important aspects of our algorithm. Determining which solution is better is really important. We tried several strategies (like greater value, greater volume, greater value per volume etc..). But it looks like $width:\ \ greater\ value\ ||\ volume\ and\ depth: value/volume$ works better in majority of cases.(If we want to try any other better strategy we only need to change this function). We included three strategies in our submission.

This was the simplest overview of algorithm. Because if we discuss every single implementation detail here this document will be very long. For more insights see our implementation in $\boldsymbol{C}++$. We also wrote a small $\boldsymbol{python}$ script to visualize our solution. GUI based solver is also available.

Demo run

Solve with console run

Before looking at demo test case let’s see our Input and Output format. Input format for 5 switches (Space separated values in following format).

$$reck\_length\ reck\_breadth\ \ reck\_height$$

$$dim1(l)\ dim2(b)\ dim3(h)\ value\ instances$$

$$dim1(l)\ dim2(b)\ dim3(h)\ value\ instances$$

$$dim1(l)\ dim2(b)\ dim3(h)\ value\ instances$$
$$dim1(l)\ dim2(b)\ dim3(h)\ value\ instances$$
$$dim1(l)\ dim2(b)\ dim3(h)\ value\ instances$$

Note that program automatically resolves:

$(length, breadth, height) \rightarrow\ (height, width, depth)$.

Output will have these details: execution time, Value gained, Volume packed in %, Remaining switches and list of all switches taken, their orientation (if allowed) and their position (x, y, z). (Note that the largest dimension among $rack\_width$ and $rack\_depth$ is seen as $rack\_depth$.- When orientation is not allowed)

$\boldsymbol{Demo\ test\ case}$:

$$\begin{array}{l}\mathbf{2\ 2\ 2}\\ \mathbf{1\ 1\ 1\ 2\ 2}\\ \mathbf{1\ 1\ 1\ 4\ 2}\\ \mathbf{1\ 1\ 1\ 3\ 2}\\ \mathbf{1\ 1\ 1\ 5\ 1}\\ \mathbf{1\ 1\ 1\ 1\ 1}\end{array}$$

$\boldsymbol{Demo\ Output}$:

$*** \ SUMMARY\ OF\ PLACED\ SWITCHES\ ***$

$Sr.TYPE\ ORIENTATION\ POSITION$

1.	1	$(1,1,1)$	$(0,0,0)$
2.	3	$(1,1,1)$	$(0,0,1)$
3.	1	$(1,1,1)$	$(0,1,0)$
4.	2	$(1,1,1)$	$(0,1,1)$
5.	0	$(1,1,1)$	$(1,0,0)$

6.	2	$(1,1,1)$	$(1,0,1)$
7.	0	$(1,1,1)$	$(1,1,0)$
8.	4	$(1,1,1)$	$(1,1,1)$

$Execution\ Completed\ in$: $0.005112\ Seconds$

$Format\ for\ coordinates:\ (height, width, depth)$

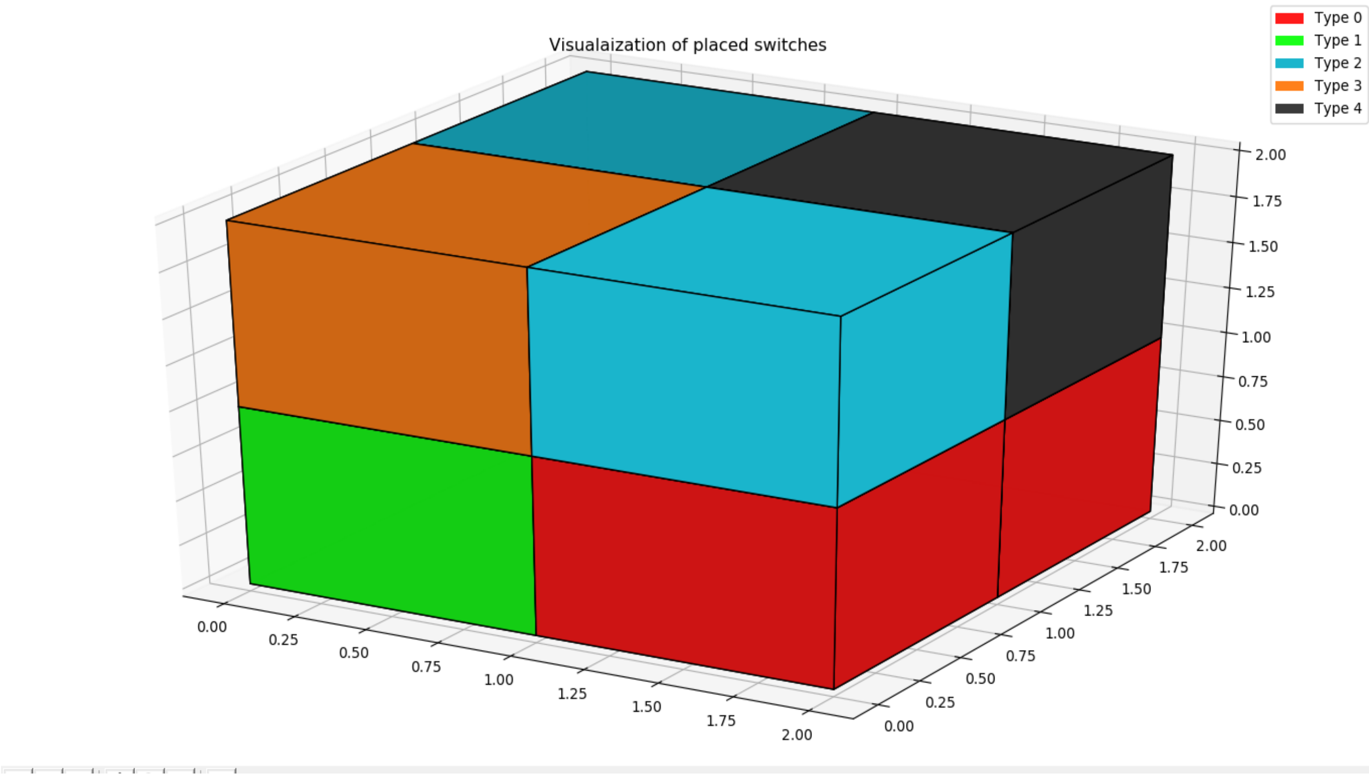
$See\ rack\ dimenssion\ as:\ (2,2,2)$

$Remaining\ Switches$: $0\ 0\ 0\ 0\ 0$

$Total\ Value\ gained$: 24

$\%\ of\ total\ volume\ packed$: $100\ \%$

Our program will also output visualization of placed switches. In this case see fig 6. In this case we can fit all switches in given rack dimension hence volume packed is 100%. Total value gained is total of all values of each switches (multiplied by theirs instances).

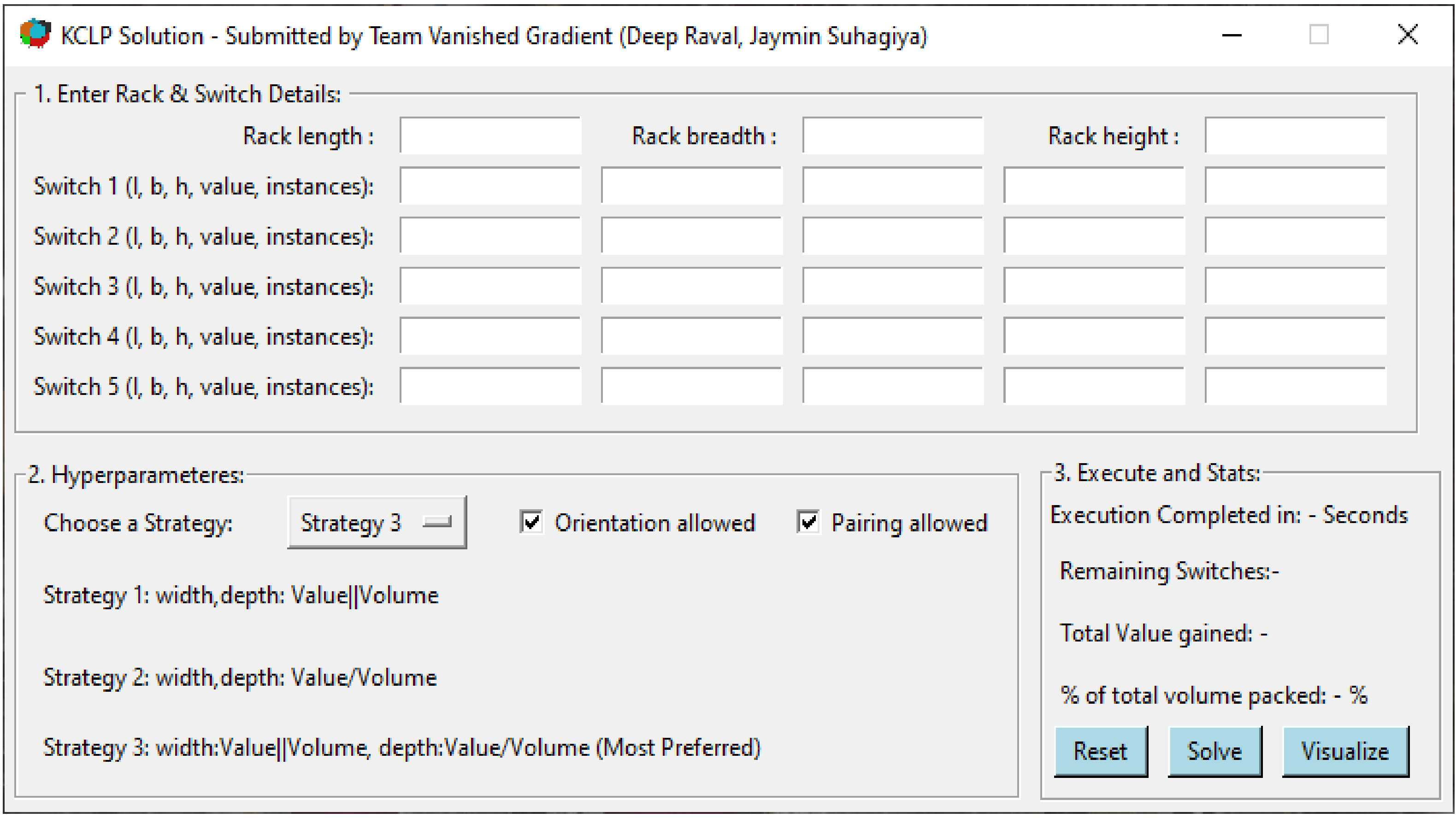


[Fig 6]

As you can see 8 switches are placed in figure. For each type there is different color which is shown in legend.

Solve with GUI

Apart from running solution in console you can also use GUI which we made (Instructions to run both console and GUI solution are in processing manual). You can see GUI in fig 7.



[fig 7]

Answers of Test Cases given by college

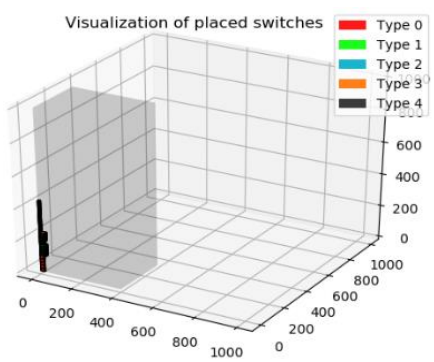
(With their empirical analysis)

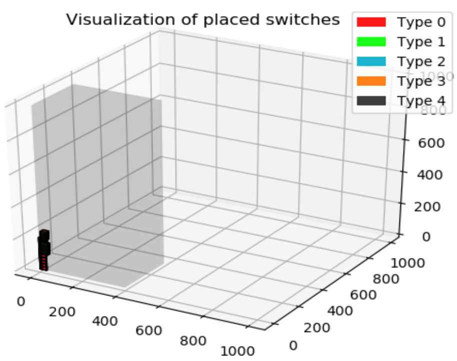
Note that all test case's running time is measured on laptop with $\boldsymbol{Intel(R)\ core\ i5-7200U\ @\ 2.50\ GHz\ 2.71\ GHz}$ processor and $\boldsymbol{8GB}$ of RAM.

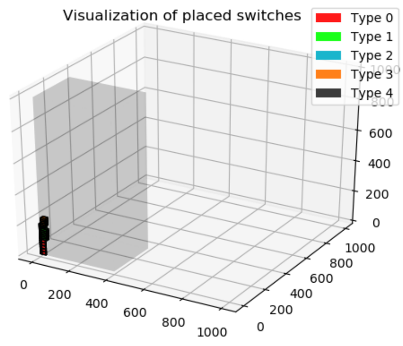
Orientation not allowed. Pairing not allowed.

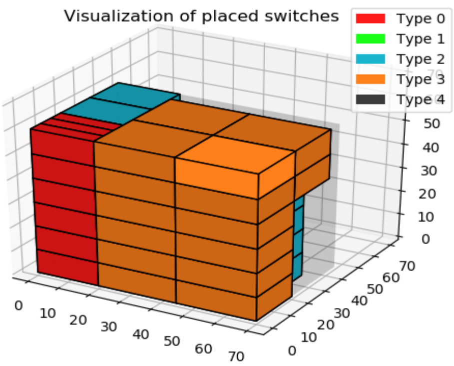
Sr.	Test Case	Strategy								
		S1			S2			S3		
		Time	Value	Volume(%)	Time	Value	Volume(%)	Time	Value	Volume(%)
1	300 400 1000 20 5 20 25 5 30 15 10 30 7 20 15 7 20 3 25 20 10 35 5 10 8 10 15 20	5.22819 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	5.89044 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	5.02684 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining
2	70 45 60 20 5 10 25 20 30 15 10 30 20 20 15 10 20 20 25 20 10 35 20 10 8 10 15 20	0.05345 Seconds	1480	85.7143% (0, 20, 6, 0, 20) Remaining	0.05871 Seconds	1200	50.7937% (0, 20, 0, 20, 0) Remaining	0.046929 Seconds	1420	66.6667% (0, 20, 10, 8, 0) Remaining
3	70 42 60 20 5 20 25 20 30 15 10 30 15 20 15 7 20 22 25 20 10 35 18 10 8 10 15 30	0.045881 Seconds	1390	62.4717% (0, 15, 0, 18, 0) Remaining	0.078786 Seconds	1270	55.3288% (0, 15, 6, 18, 0) Remaining	0.031919 Seconds	1390	62.4717% (0, 15, 0, 18, 0) Remaining
4	50 30 40 20 20 5 25 5 30 15 10 30 7 20 15 17 20 3 20 20 10 35 5 10 8 10 15 20	0.00744 Seconds	455	85.8333% (1 ,0, 3, 3, 15) Remaining	0.011964 Seconds	470	53.3333% (1, 7, 3, 3, 0) Remaining	0.010958 Seconds	470	53.3333% (1, 7, 3, 3, 0) Remaining
5	60 50 75 20 20 5 25 20 30 15 10 30 30 20 15 17 20 40 20 20 10 35 25 10 8 10 15 30	0.033913 Seconds	1755	79.3333% (0 ,9, 40, 20, 0) Remaining	0.04987 Seconds	1370	49.7778% (0, 30, 40, 13, 0) Remaining	0.032873 Seconds	1615	62.2222% (0, 30, 40, 6, 0) Remaining

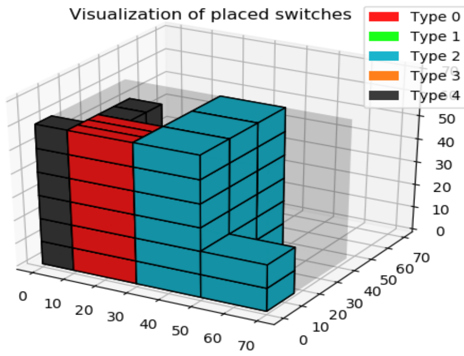
Below are the visualizations of all above cases in table (in same order as in table).

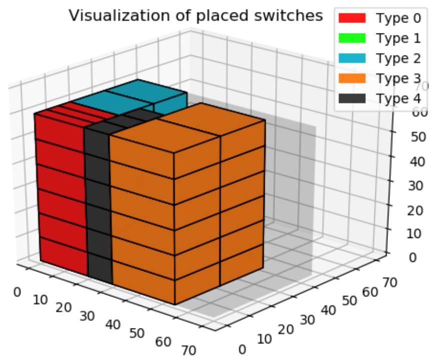

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

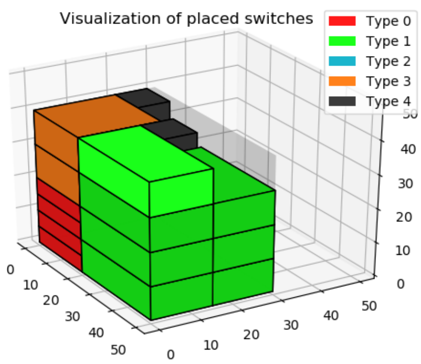

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

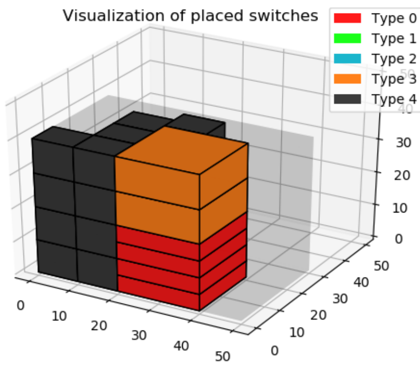

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

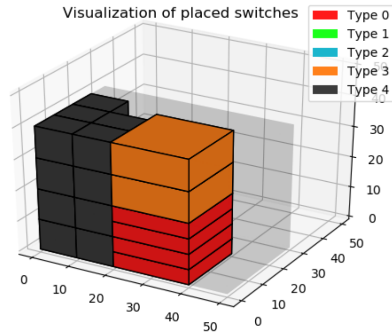

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

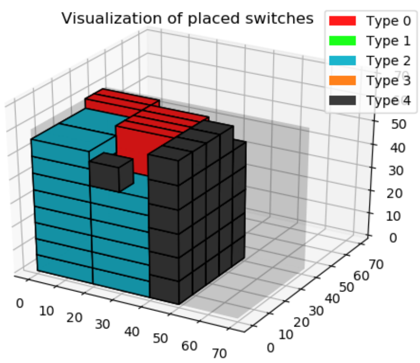

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

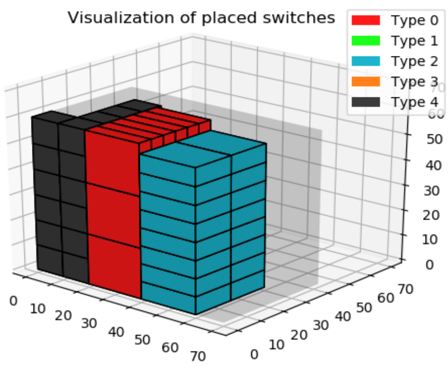

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

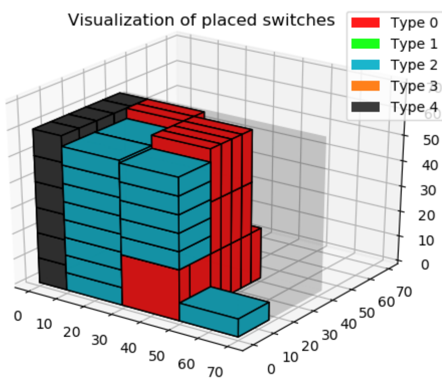

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

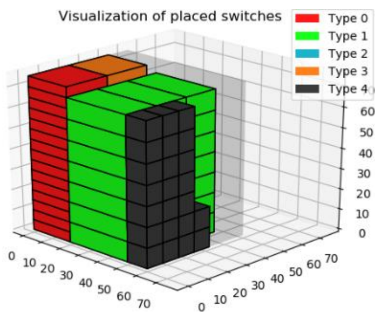

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

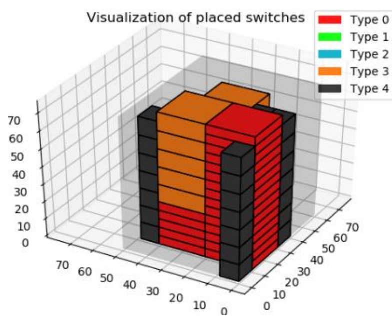

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

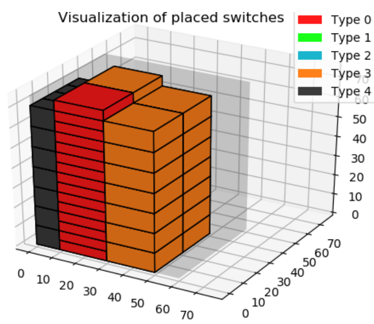

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

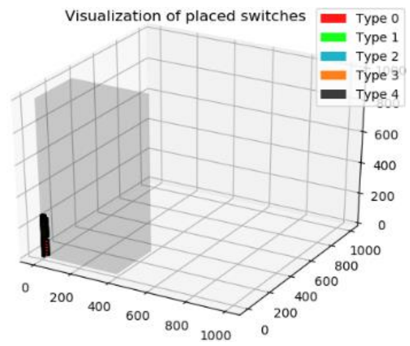

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

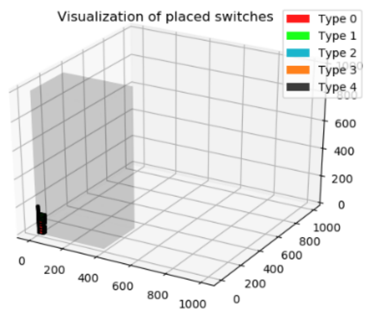

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

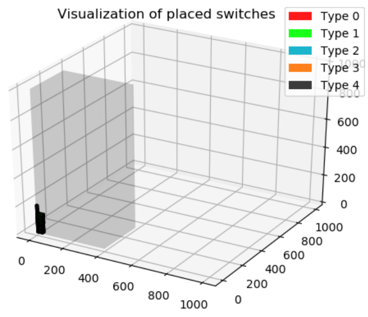
Orientation not allowed. Pairing allowed

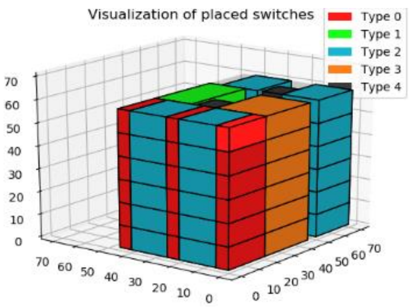
Sr.	Test Case	Strategy								
		S1			S2			S3		
		Time	Value	Volume(%)	Time	Value	Volume(%)	Time	Value	Volume(%)
1	300 400 1000 20 5 20 25 5 30 15 10 30 7 20 15 7 20 3 25 20 10 35 5 10 8 10 15 20	6.04786 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	6.0629 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	5.97211 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining
2	70 45 60 20 5 10 25 20 30 15 10 30 20 20 15 10 20 20 25 20 10 35 20 10 8 10 15 20	0.02397 Seconds	1500	75.3439% (0, 16, 0, 14, 2) Remaining	0.039956 Seconds	1520	77.672% (0, 20, 2, 8, 4) Remaining	0.036885 Seconds	1460	69.8413% (0, 20, 8, 8, 0) Remaining
3	70 42 60 20 5 20 25 20 30 15 10 30 15 20 15 7 20 22 25 20 10 35 18 10 8 10 15 30	0.04592 Seconds	1390	62.4717% (0, 15, 0, 18, 0) Remaining	0.072804 Seconds	1270	55.3288% (0, 15, 6, 18, 0) Remaining	0.04389 Seconds	1390	62.4717% (0, 15, 0, 18, 0) Remaining
4	50 30 40 20 20 5 25 5 30 15 10 30 7 20 15 17 20 3 20 20 10 35 5 10 8 10 15 20	0.007984 Seconds	425	83.1667% (1 ,0, 3, 3, 17) Remaining	0.011973 Seconds	470	53.3333% (1, 7, 3, 3, 0) Remaining	0.002194 Seconds	470	53.3333% (1, 7, 3, 3, 0) Remaining
5	60 50 75 20 20 5 25 20 30 15 10 30 30 20 15 17 20 40 20 20 10 35 25 10 8 10 15 30	0.026934 Seconds	1825	72.8889% (0 ,30, 40, 0, 0) Remaining	0.034865 Seconds	1615	62.2222% (0, 30, 40, 6, 0) Remaining	0.0034504	1615	62.2222% (0, 30, 40, 6, 0) Remaining

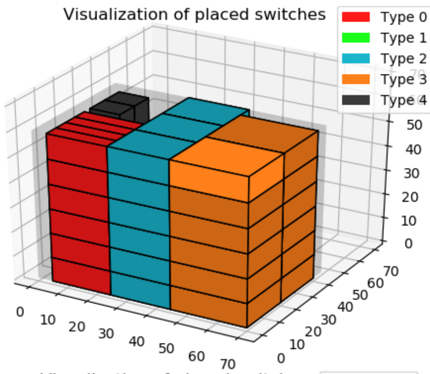
Below are the visualizations of all above cases in table (in same order as in table).

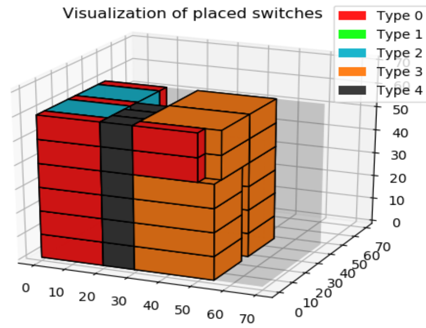

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

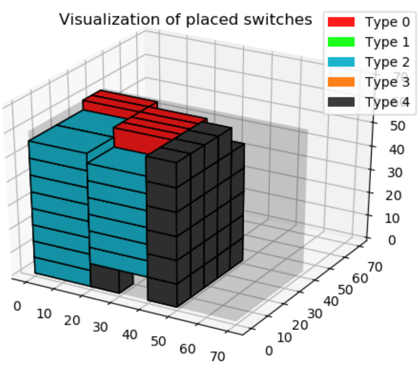

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

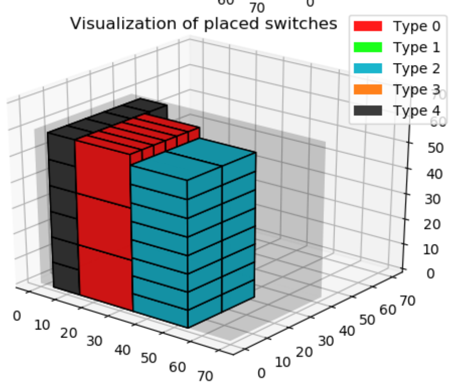

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

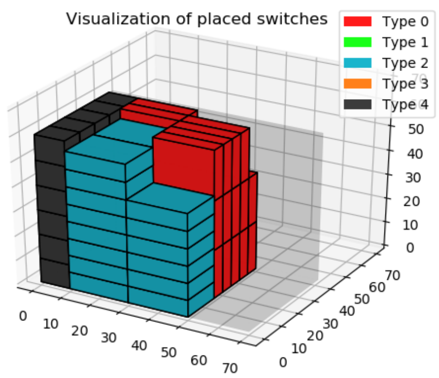

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

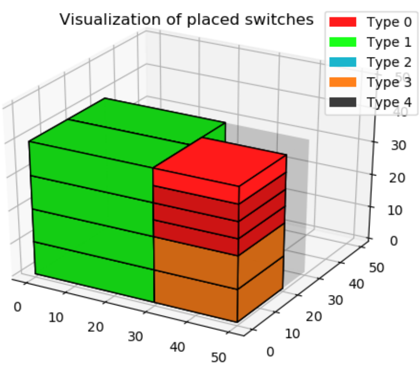

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

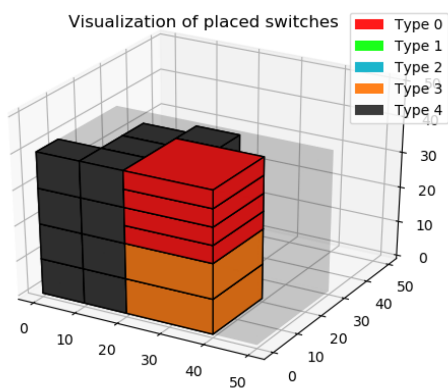

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

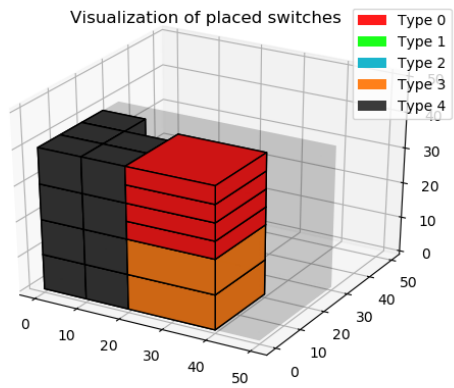

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

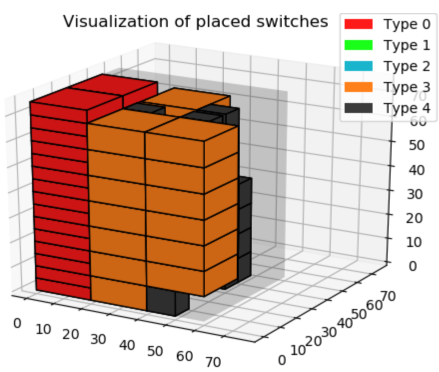

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

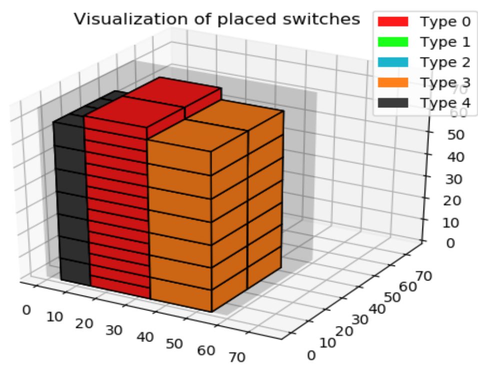

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

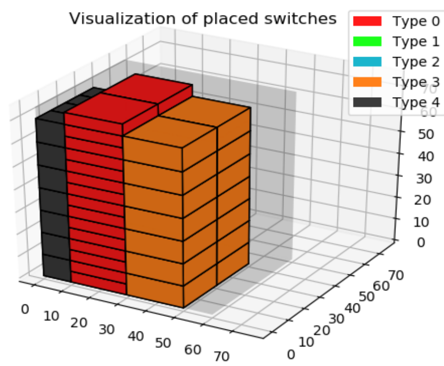

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

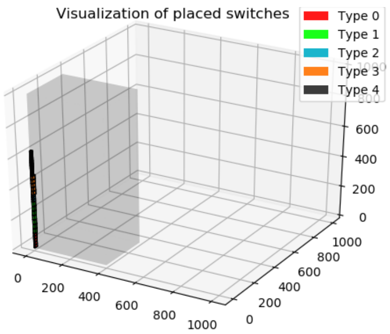

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

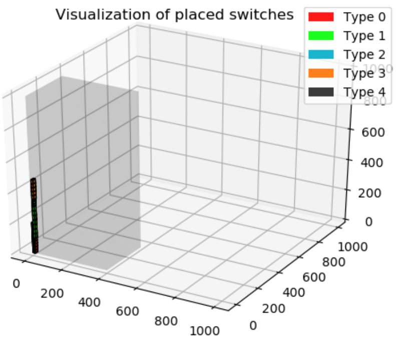

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

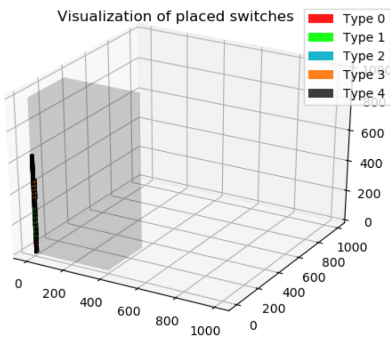
Orientation allowed. Pairing not allowed.

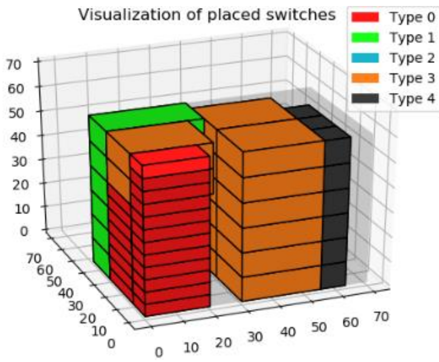
Sr.	Test Case	Strategy								
		S1			S2			S3		
		Time	Value	Volume(%)	Time	Value	Volume(%)	Time	Value	Volume(%)
1	300 400 1000 20 5 20 25 5 30 15 10 30 7 20 15 7 20 3 25 20 10 35 5 10 8 10 15 20	8.77165 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	10.284 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	9.30416 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining
2	70 45 60 20 5 10 25 20 30 15 10 30 20 20 15 10 20 20 25 20 10 35 20 10 8 10 15 20	0.05186 7 Seconds	1470	70.3704% (0, 14, 20, 6, 0) Remaining	0.19250 5 Seconds	1705	88.8889% (0, 16, 12, 8, 0) Remaining	0.09396 4 Seconds	1690	89.1534% (0, 1, 11, 16, 0) Remaining
3	70 42 60 20 5 20 25 20 30 15 10 30 15 20 15 7 20 22 25 20 10 35 18 10 8 10 15 30	0.04985 3 Seconds	1560	86.1678% (0, 11, 22, 4, 0) Remaining	0.241 Seconds	1555	76.0771% (0, 13, 0, 15, 0) Remaining	0.10472 5 Seconds	1730	91.8934% (0, 3, 1, 18, 0) Remaining
4	50 30 40 20 20 5 25 5 30 15 10 30 7 20 15 17 20 3 20 20 10 35 5 10 8 10 15 20	0.01197 2 Seconds	615	86.667% (0, 3, 3, 3, 0) Remaining	0.02598 1 Seconds	565	70% (0, 7, 3, 1, 0) Remaining	0.01595 5 Seconds	605	80.8333% (1, 6, 3, 0, 0) Remaining
5	60 50 75 20 20 5 25 20 30 15 10 30 30 20 15 17 20 40 20 20 10 35 25 10 8 10 15 30	0.05688 4 Seconds	1860	8.2667% (0, 4, 40, 20, 3) Remaining	0.10970 5 Seconds	1855	74.8889% (0, 29, 40, 0, 0) Remaining	0.05385 3 Seconds	2035	86.8889% (0, 23, 40, 0, 0) Remaining

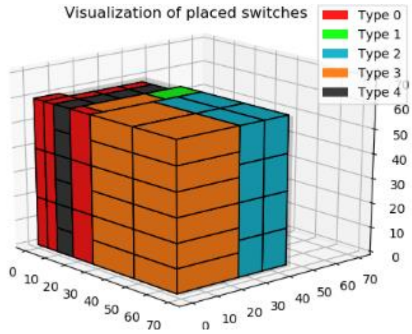
Below are the visualizations of all above cases in table (in same order as in table).

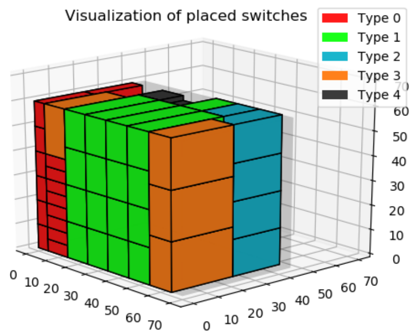

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

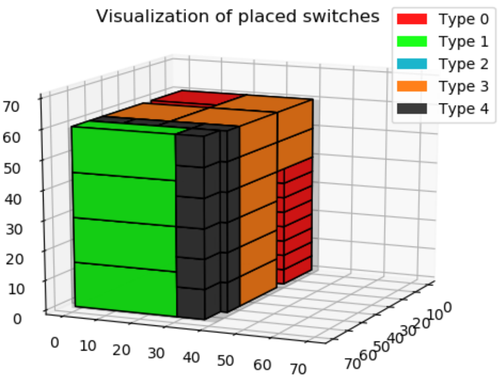

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

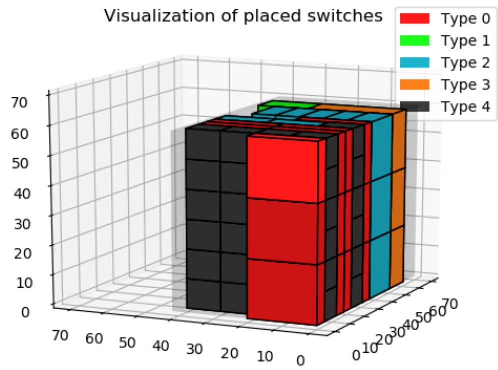

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

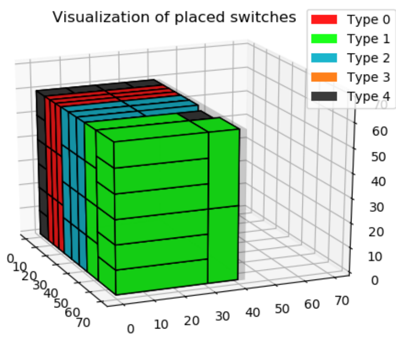

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

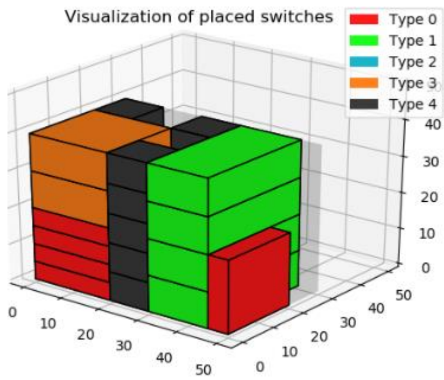

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

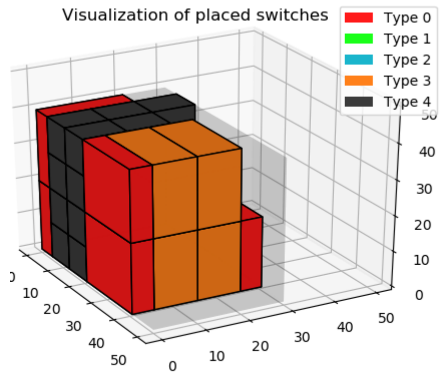

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

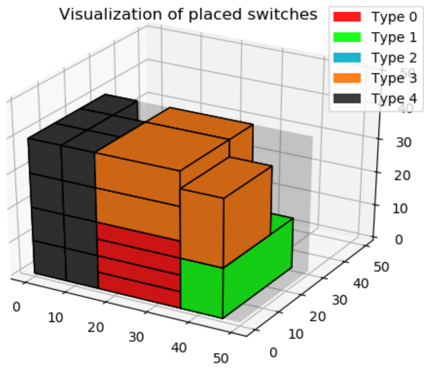

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

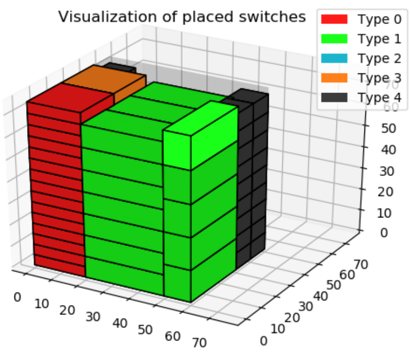

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

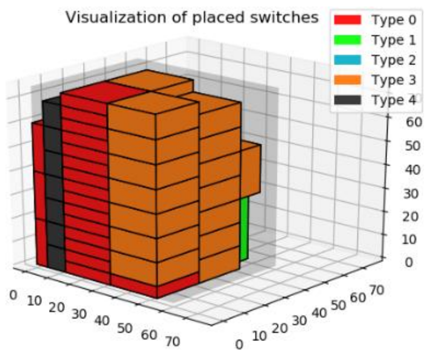

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

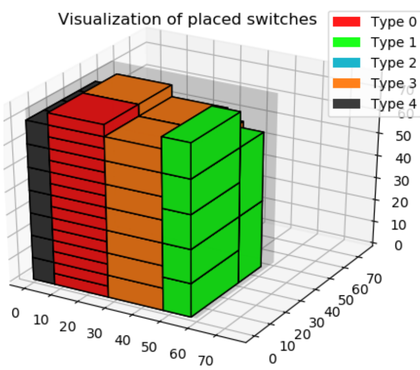

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

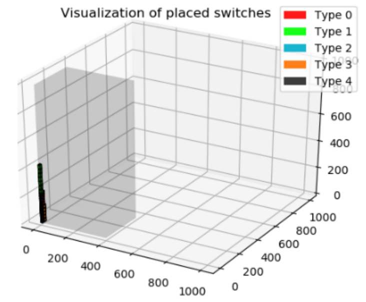

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

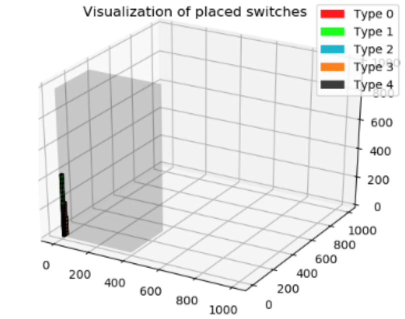

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

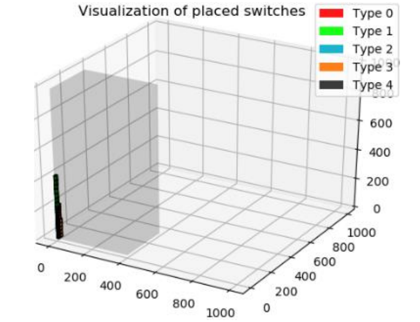
Orientation allowed. Pairing allowed.

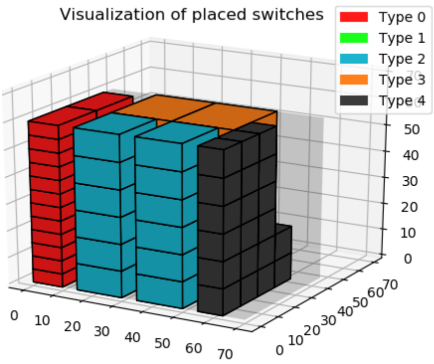
Sr.	Test Case	Strategy								
		S1			S2			S3		
		Time	Value	Volume(%)	Time	Value	Volume(%)	Time	Value	Volume(%)
1	300 400 1000 20 5 20 25 5 30 15 10 30 7 20 15 7 20 3 25 20 10 35 5 10 8 10 15 20	9.13375 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	8.7419 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining	8.74574 Seconds	870	0.074% (0, 0, 0, 0, 0) Remaining
2	70 45 60 20 5 10 25 20 30 15 10 30 20 20 15 10 20 20 25 20 10 35 20 10 8 10 15 20	0.05345 Seconds	1500	73.0159% (0, 20, 6, 8, 0) Remaining	0.16926 9 Seconds	1500	73.0159% (0, 16, 12, 8, 0) Remaining	0.078144 Seconds	1690	89.1534% (0, 1, 11, 16, 0) Remaining
3	70 42 60 20 5 20 25 20 30 15 10 30 15 20 15 7 20 22 25 20 10 35 18 10 8 10 15 30	0.04690 Seconds	1560	86.1678% (0, 11, 22, 4, 0) Remaining	0.18493 1 Seconds	1555	76.0771% (0, 13, 0, 15, 0) Remaining	0.100334 Seconds	1730	91.8934% (0, 3, 1, 18, 0) Remaining
4	50 30 40 20 20 5 25 5 30 15 10 30 7 20 15 17 20 3 20 20 10 35 5 10 8 10 15 20	0.01567 8 Seconds	485	88.3333% (0, 1, 3, 2, 25) Remaining	0.03729 9 Seconds	565	70% (0, 7, 3, 1, 0) Remaining	0.015658 Seconds	605	80.8333% (1, 6, 3, 0, 0) Remaining
5	60 50 75 20 20 5 25 20 30 15 10 30 30 20 15 17 20 40 20 20 10 35 25 10 8 10 15 30	0.04685 7 Seconds	1540	78.6222% (14, 9, 40, 11, 12) Remaining	0.10036 4 Seconds	1855	74.8889% (0, 29, 40, 0, 0) Remaining	0.06892 Seconds	1715	74.4444% (0, 15, 40, 16, 0) Remaining

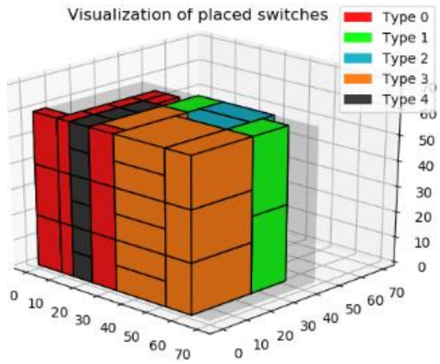
Below are the visualizations of all above cases in table (in same order as in table).

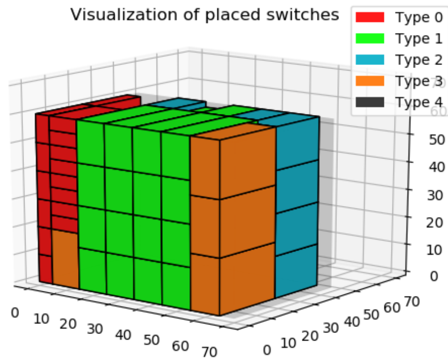

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

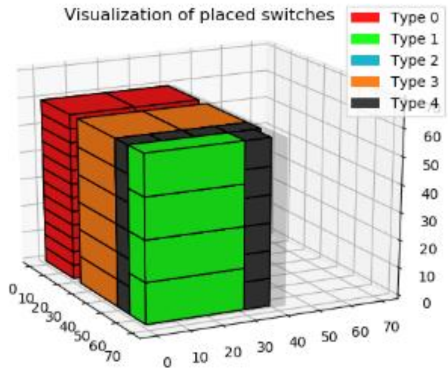

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

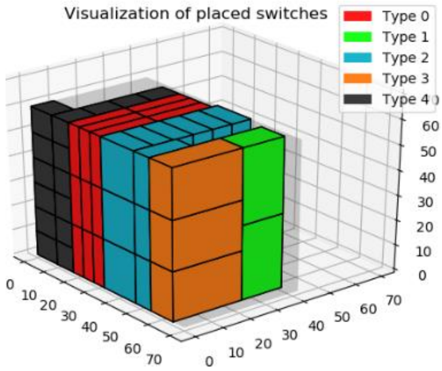

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

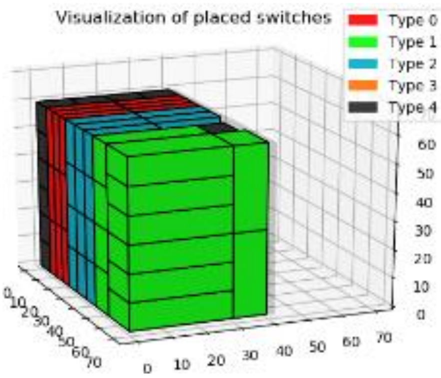

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

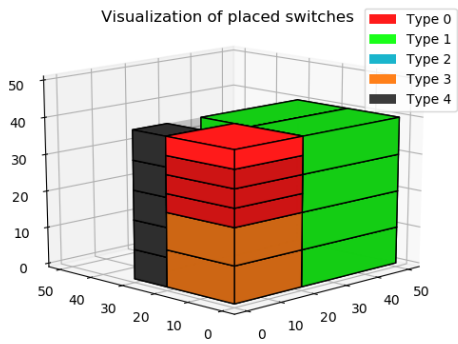

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

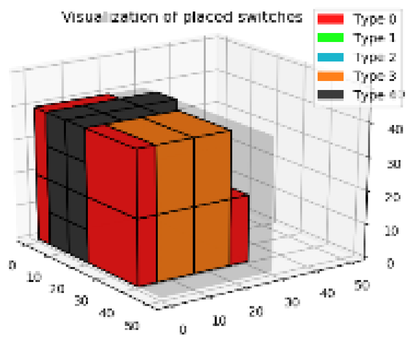

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

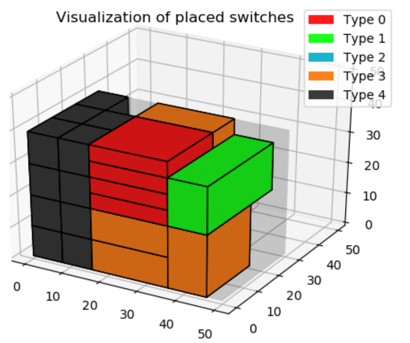

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

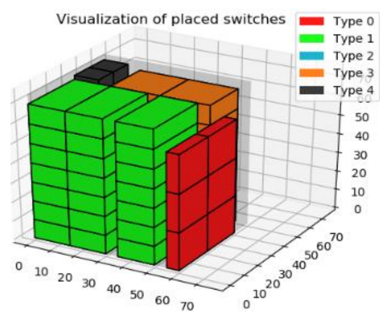

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

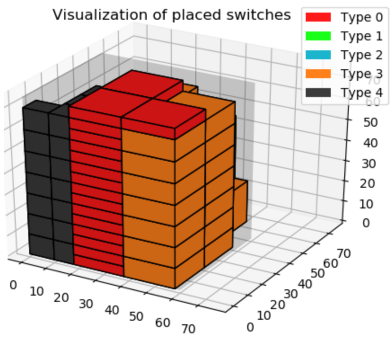

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

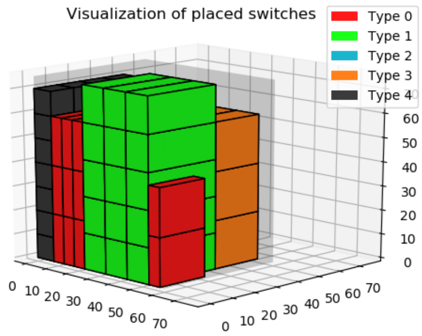

Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4


Visualization of placed switches
Type 0
Type 1
Type 2
Type 3
Type 4

Theoretical analysis

First let's see notations used:

$$N = Number\ of\ distinct\ switches\ (5\ in\ this\ case).$$
$$H = Height\ of\ rack$$
$$W = Width\ of\ rack$$
$$D = Depth\ of\ rack.$$
$$L_W = Length\ of\ promising\ width\ list.$$
$$L_D = Lenghth\ of\ promising\ depth\ list.$$
$$w_{min} = Minimum\ width\ in\ width\ list.$$
$$d_{min} = Minimum\ depth\ in\ depth\ list.$$

Our algorithm runs in:

$$\boldsymbol{O\left(\frac{N\ H\ \ W\ D\ L_D\ L_W}{d_{min}\ w_{min}}\right)}$$

Note that $N\ H$ is complexity of solving knapsack problem (using dynamic programming) which is pseudo polynomial. Complexity derived here is purely theoretical and it may change with implementation (because of usage of standard data structures provided by language – for example map, set, vector in $C++$).

Limitations of our Solution

The only limitation of our solution is it will not give optimal solution because it is based on a heuristic approach.

Conclusion

As the given problem was $\boldsymbol{NP\ Hard}$ solving it optimally using naïve approach is not practical. We found heuristic solution through which we can find solution (solution may not be optimal) in reasonable time. Another interesting factor about our algorithm is that we can try all combinations of different strategies and see which one gives best result.

******