06

DESIGN COMPUTATION

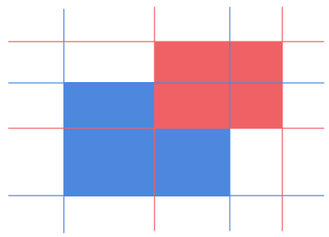
OVERALL FRAMEWORK

EXCEL DATA

An excel file with multiple spreadsheets serve as the frontend interface to interact with and also store a library of reference data derived from prior thesis study.

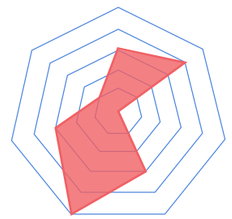
PYTHON DATA

Translating the data consolidated in excel to a custom python class as easily accessible data to inform of the design generation and evaluation phase.



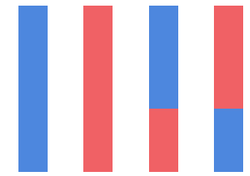
DESIGN GENERATION

An algorithm that generates a set of iterations encompassing a complex network and arrangement of programmatic spaces informed by the specified programmatic attributes.



EVALUATION

A fitness score is calculated with information from the python data library as the benchmark for various scoring, coupled with a scoring for the network connectivity.



GENETIC ALGORITHM

Through the selection, crossover and mutation functions, the genome of candidates are evolved through generations, leading to the emergence the fittest candidate.

ADJACENCIES

Adjacency matrix consolidating relationship between all programs.

ATTRIBUTES

List of physical attributes characterising spatial needs of the various programs.

TIME ACTIVATION

Expected time program will be in active state on weekdays/weekends.

PROGRAMS LIBRARY

A library of information specific to each program, translated from excel.

PROGRAM LIST

A sequence of programs with randomised attributes within range specified in library.

VOXELISATION

Grid of pixels make up floors, within each, rectangular sets of pixels define spaces.

SPATIAL RECOGNITION

Calculate the biggest possible unoccupied spaces relative to occupied spaces.

PROGRAM ASSIGNMENT

A selective assignment of programs to available unoccupied spaces.

GRAPH NETWORK

Graph network analysis is used to evaluate the connectivity and relative position of programs.

EXCEL DATA

Resulting allocation of programs to physical spaces are evaluated based on targets set.

GENOMES

The program list, lift placement etc. are evolved through generations.

Figure 82: Overall framework for the design computation of the tower.

EXCEL & PYTHON DATA

DATA CONSOLIDATION

Qualitative and quantitative information from the previous chapters were translated and consolidated into an excel sheet that serves as an easy way to manipulate the base data that drives the design generation. Adopting excel spreadsheets to serve as the frontend allows for a visual interaction and expression of information, making it potentially a more intuitive way for designers to manage the design computation tool without having computation knowledge and background. It also builds on the expandability of the methodology to be easily adapted to suit the programmatic and spatial needs of different projects.

TYPE		PROGRAMS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	1	co-working office	-1																			
	2	private office	1	0																		
	3	meeting facilities	2	2	-1																	
	4	informal working	2	2	0	0																
	5	auditorium	0	0	2	0	-1															
	6	education centre	1	1	1	0	2	-1														
	7	library	1	0	0	1	0	1	-1													
	8	meditation pods	2	1	0	2	0	0	2	0												
	9	childcare centre	1	1	0	1	0	0	1	-1	-1											
	10	pets daycare / training centre	1	1	0	1	0	0	0	-1	1	-1										
	11	hawker centre	0	0	0	1	0	0	-1	-1	0	-1	-1									
	12	night market	0	0	0	1	0	0	-1	-1	0	0	0	-1								
	13	cafe	0	0	0	1	0	1	2	0	0	0	0	0	-1							
	14	playscape	1	0	0	1	0	0	0	-1	2	0	1	1	1	-1						
	15	retail	0	0	0	0	0	0	0	-1	0	1	1	0	2	2	0					
	16	clinic	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1				
	17	gym	1	1	0	0	0	0	-1	1	0	0	0	0	0	0	0	0	-1			
	18	fitness corner	1	1	0	0	0	0	-1	1	0	0	0	0	0	2	0	0	0	-1		
	19	park	1	1	1	2	0	0	1	1	2	2	1	1	1	2	0	1	1	0	1	
	20	multi-purpose / sports hall	0	1	0	0	0	1	0	0	1	0	0	2	0	0	0	0	1	0	0	-1

Figure 83: Programmatic adjacency relationships consolidated into a table format.

TYPE		PROGRAMS	area_lower	area_upper	nol_lower	nol_upper	asp_ratio_lower	asp_ratio_upper	height	max_count
	1	co-working office	500	1500	1	3	0.5	1	1.5	
	2	private office	200	1500	1	1	0.35	1	2.5	
	3	meeting facilities	50	120	1	1	0.5	0.85	0	
	4	informal working	50	80	1	2	0.3	1	0	
	5	auditorium	800	1200	2	2	0.6	0.7	0	2
	6	education centre	150	250	1	2	0.5	0.8	0	1
	7	library	1200	1500	2	5	0.3	0.8	1	1
	8	meditation pods	5	20	1	1	0.8	1	0	
	9	childcare centre	1200	1500	2	3	0.6	1	2.5	1
	10	pets daycare / training centre	500	1000	1	2	0.35	0.7	1	1
	11	hawker centre	1200	1500	1	2	0.7	1	1	1
	12	night market	5	30	3	5	0.2	0.4		
	13	café	100	150	1	2	0.5	0.7	1	3
	14	playscape	1000	1500	2	4	0.7	1	0	2
	15	retail	800	1500	1	3	0.3	0.8	1.5	
	16	clinic	80	200	1	2	0.5	0.7	1	1
	17	gym	1000	1500	1	1	0.6	1	0	1
	18	fitness corner	20	50	1	1	0.4	0.7	0	2
	19	park	5	30	1	20	0.1	0.4		
	20	multi-purpose / sports hall	300	500	1	1	0.6	0.8	0	2

Figure 84: Programs' physical attributes consolidated into a table format.

Based on the synergies derived from the exploring of personas in Chapter 5, the resulting adjacency relationships represented in Figure 79 was consolidated into a table format as shown in Figure 83. A score of 2 is given to pairs of programs that are believed to be strongly synergetic with one another. A score of 1 given to pairs of programs that are believed to possibly create interesting moments programmatically when placed beside on another. A score of 0 is given to pairs of programs that are believed to have no special relation with one another. Finally, a score of -1 is given to pairs of programs that are believed should be away from one another.

TYPE		PROGRAMS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	1	co-working office	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	2	private office	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	3	meeting facilities	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	4	informal working	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	5	auditorium	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	6	education centre	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	7	library	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	8	meditation pods	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	1	0	0
	9	childcare centre	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	10	pets daycare / training centre	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
	11	hawker centre	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	12	night market	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0
	13	café	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	14	playscape	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
	15	retail	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	16	clinic	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	17	gym	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1	1	1	0	0
	18	fitness corner	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	1	1	1	1	1	1	0	0
	19	park	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	20	multi-purpose / sports hall	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Figure 85: Programs' expected activity level by the hours (weekdays) consolidated into a table format.

TYPE		PROGRAMS	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	1	co-working office	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	private office	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	meeting facilities	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	informal working	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	5	auditorium	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	education centre	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
	7	library	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	8	meditation pods	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	9	childcare centre	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	pets daycare / training centre	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
	11	hawker centre	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	12	night market	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	13	café	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	14	playscape	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	15	retail	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
	16	clinic	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	17	gym	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	18	fitness corner	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
	19	park	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	multi-purpose / sports hall	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 86: Programs' expected activity level by the hours (weekends) consolidated into a table format.

Figure 84 consolidates the physical requirements specific to each program including attributes like the lower and upper bounds of the expected area, number of levels, space aspect ratio, the relative height the program is ideally placed and if there expects to be a maximum count of any program.

Most of the values assigned to the attributes are self explanatory, except the values assigned to the heigh attribute. The value 3 is given to programs that should ideally be placed on the upper floors. Value 2 given to programs that should ideally be placed on the mid-level floors. Value 1 given to programs that should ideally be placed on the lower floors. Value 0 given to programs that have no clear bias with regards to the height it is placed on. If a value is an average of two of the categories described, it would signify that it is ideally placed on heights within either of the categories.

Figure 85 and 86 consolidates the expected activity level by the hours for both the weekdays and weekends respectively. These are values that were represented in Figures 80 and 81 in Chapter 5.

EXCEL TO PYTHON

The python Panda library was used to translate the data from excel into a customized library of information easily accessible in python. For each program, the following attributes were stored in the respective data format:

PROGRAM LIST

GENERATION DRIVER

The program list is a sequence of programs and program requirements that informs the design generation process. The design generation algorithm goes down this sequence of programs paired with specific program requirements and fit them into space that best satisfy the physical requirements.

PROGRAM REQUIREMENTS

From the excel data, the lower bound and upper bound of the area and aspect ratio attributes for each program has be stored into a python class. With this information, taking a specific program as an input, the program requirements function is expected to return a randomized set of physical dimensions, inclusive of the length of two sides of a rectangular space and the resulting area, that fulfils the registered range for the specified program.

It first generates an approximate target area and aspect ratio within the ranges given, referred to as approx_area and approx_ratio respectively. The length of the two sides of a rectangular space shall be referred to as sideA, which is by convention the longer side, and sideB. The following equations will help derive with the formula to compute the length of sideA and sideB:

$$\text{sideA} \times \text{sideB} = \text{approx\_area}$$
$$\text{sideB} / \text{sideA} = \text{approx\_ratio}$$
$$\text{sideB} = \text{sideA} \times \text{approx\_ratio}$$
$$\text{sideA} \times (\text{sideA} \times \text{approx\_ratio}) = \text{approx\_area}$$

$$\textit{sideA} = \sqrt{\frac{\text{approx\_area}}{\text{approx\_ratio}}}$$

$$\textit{sideB} = \sqrt{\frac{\text{approx\_area}}{\text{approx\_ratio}}} \times \text{approx\_ratio}$$

With this, sideA and sideB can be determined. The lengths are rounded to the nearest integer while the resulting area is rounded to the nearest 50 m^2 if the area is larger than 200m^2, else it is rounded to the nearest integer.

PROGRAM SEQUENCE

To get a good mix of private and semi-public/public programs on every level, the program sequence first generates a base list of private programs followed by an insertion of semi-public/public programs at regular intervals.

BASE SEQUENCE (PRIVATE PROGRAMS)

As the generation process involves the removing of programs from the program list upon the fulfilment of certain conditions, which will be further elaborated in a later section, the program list generated has to sum up to an area that is larger than the maximum volume of the building. As such, the target total area of program requirements is set at five times the total building volume, in other words the mass multiplication of the width, breadth and number of levels.

To start off, a list of private programs paired with specific requirements for each program is generated. This step randomly chooses a private program that has no maximum count, with reference to Figure 84, and generates a corresponding program requirement for the specified program. Following, it involves the random inserting of private programs with restricted counts, similarly paired with a set of program requirements for each specified program. This serves as the base list of programs.

STEPPED INSERTION (SEMI/PUBLIC PROGRAMS)

Subsequently, the semi-public and public programs without restrictions placed on its count are then inserted into the base list of programs at a specified regular interval.

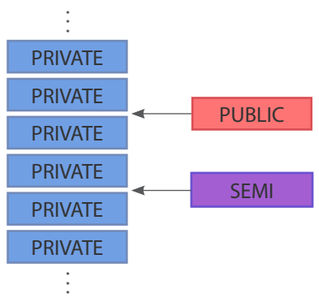


Figure 87: Regular insertion of public and semi-public programs at a sample 2 step interval.

Similarly, the programs with maximum counts are inserted subsequently.

DESIGN GENERATION

VOXELISATION

Spaces within the tower are represented by sets of individual pixels which represent 1x1m squares in physical space. The position of each pixel is represented by its row and column index. Each space is physically demarcated by a lower pixel in the lower left corner and an upper pixel in the upper right corner.

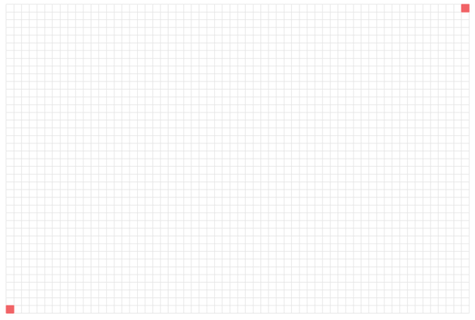


Figure 88: An example of a space representing a fully unoccupied floor with the lower and upper pixels highlighted.

LIFT INITIATION

To start off the design generation, a lift core is first initiated. The lift core has to be defined by a given dimension while its position has the flexibility of initiating randomly or alternatively, it may also be specified. This is to allow for an initial population of candidates with varying lift positions for the Genetic Algorithm, covering a wide unbiased design space, allowing the Genetic Algorithm to converge the lift position towards a favoured position.

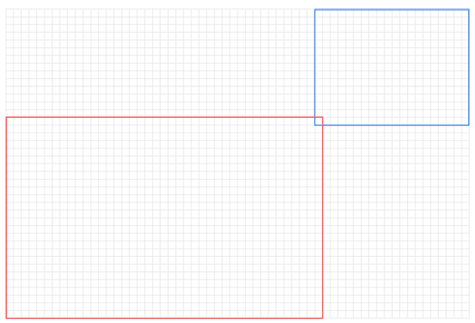


Figure 89: Red box highlights the possible pixel positions the lower pixel of the lift core may occupy.

From Figure 89, highlighted in blue is the upper right extreme position that a lift core of specified dimensions may occupy. This leaves the area highlighted in red to be the possible pixel positions the lower pixel of the lift core may occupy. For a randomized position of lift core for the initial population of tower iterations, an unbiased selection of a pixel within the red box is selected.

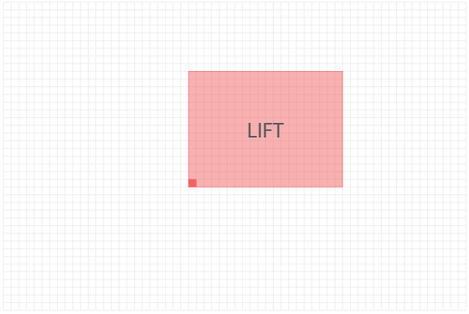


Figure 90: An example of a space representing a fully unoccupied floor with the lower and upper pixels highlighted.

Once the position of the lift is decided, a space is then being initiated, with lift as the program assigned. Along with the program decided, the lower pixel, upper pixel, width, breadth, floor, aspect ratio are stored accordingly and the space is registered as an occupied space.

SPATIAL RECOGNITION

To allow for the best fit of spaces corresponding to the ideal range of spatial requirements of each program, upon the addition of assigned programs to spaces within each floor, the largest possible rectangular spaces are being computed.

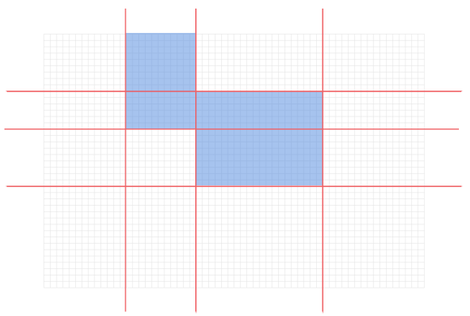


Figure 91: The subdivision of spaces relative to the occupied spaces.

SUBDIVISION

The first step of the spatial reognition process involves the sub-dividing of spaces relative to the occupied spaces. Visually, this means to extend the edges of the occupied spaces which are represented as blue rectangles in Figure 83 to split the large rectangle representing the space available on each floor. Following the example shown in Figure 91, the subdivision resulted in an irregular, 4 by 4 rectangular grid.

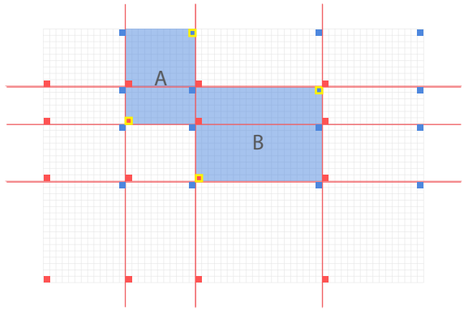


Figure 92: Breaking down the information required to retrieve the respective sub-divded spaces, red pixels highlights the lower pixels required, blue pixels highlights the upper pixels required, while pixels with yellow borders are pixels representing active spaces which are already occupied.

Computationally, to process this in a fast way, simple number operations have to be adopted to identify where exactly to split the pixels. With reference to Figure 92, in order to accurately retrieve the 16 subdivided spaces, all that is needed is the row and column indices of the pixels that are highlighted in red and blue. The pixels in red are the lower pixels while the pixels in blue are the upper pixels of the 16 spaces respectively.

As observed and also logically, the red and blue pixels are an array of pixels that are aligned along certain rows and columns. These rows are columns may be generalised and condensed into the following:

(Taking Figure 92 as a visual reference, the row and column of the lower pixel of occupied Space A will be referred to as A_{LR} and A_{LC} respectively, while for the upper pixel, it will be referred to as A_{UR} and A_{UR} respectively. Similar applies to Space B.)

For the lower pixels (red pixels):

rows	0	A_{LR}	$A_{UR}+1$	B_{LR}	$B_{UR}+1$	···	max
cols	0	A_{LC}	$A_{UC}+1$	B_{LC}	$B_{UC}+1$	···	max

For the upper pixels (blue pixels):

rows	0	A_{LR}	$A_{UR}+1$	B_{LR}	$B_{UR}+1$	···	max
cols	0	A_{LC}	$A_{UC}+1$	B_{LC}	$B_{UC}+1$	···	max

Figure 93: Generalised formulas of required rows and columns which permutates to form the required lower pixels and upper pixels that defines the subdivided spaces.

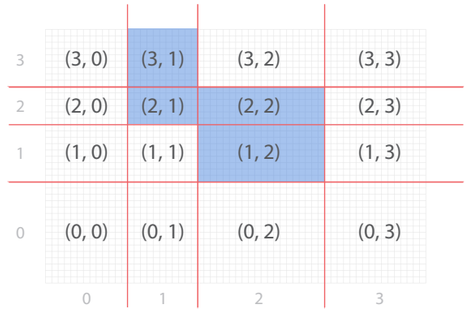


Figure 94: Identifying positions of sub-divded spaces with a temporary indexing of rows and columns in the format of (rows, columns).

SUBSET DETECTION

To aid the subsequent steps, a function to efficiently detect if Space X is a subset of another, Space Y, was to be defined. This is a relatively straightforward step of checking if the following holds true:

$$X_{LR} \geq Y_{LR}$$
$$X_{LC} \geq Y_{LC}$$
$$X_{UR} \leq Y_{UR}$$
$$X_{UC} \leq Y_{UC}$$

With the subset detection function, referring to Figure 93 and Figure 94 the sub-divdided spaces highlighted in blue: (1, 2) and (2, 2) can be detected as a subset of occupied Space A, (2, 1) and (3,1) can be detected as a subset of occupied Space B.

The separation of the sub-divided spaces into categories of occupied vs unoccupied will be helpfui for the next step.

LARGEST RECTANGLE DETECTION

With this format of sub-divided spaces set up and identified into categories of occupied vs unoccupied, the next step is the detection of the largest rectangular spaces that make up the remaining unoccupied spaces.

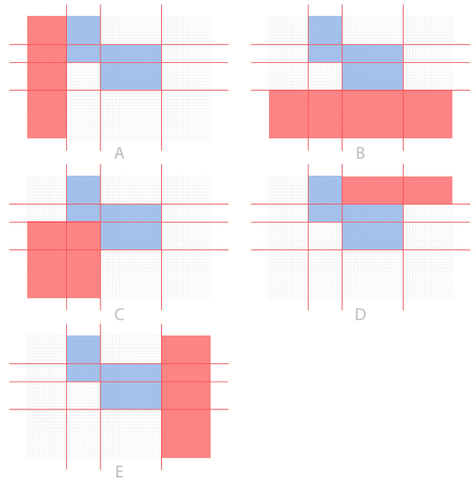


Figure 95: The largest rectangles relative to the two occupied spaces, to be detected by an algorithm.

The ultimate aim of the spatial recognition process is to identify all the largest rectangles relative to the occupied spaces that have been assigned with programs, Figure 95 shows an example of the goal of the following algorithm.

There are three main cases that have been categorised into the following: single division row space (Case B and Case D of Figure 95), single division column space (Case A and Case E of Figure 95) and lastly, which is also the most complex to detect, the multi-divisions box space (Case C of Figure 95). The description and methodology adopted to implement the detection of these three cases will be elaborated on.

SINGLE DIVISION ROW DETECTION

Starting with the single division row detection, it refers to spaces formed by sub-divisions adjacent to one another in the same row.

With reference to Figure Y, for each row, the columns of the unoccupied spaces are consolidated into a list. The list contained the column indices in an ascending order (Figure 96).

R3	0	2	3	
R2	0	3		
R1	0	1	3	
R0	0	1	2	3

Figure 96: Consoildating for each row, for each unoccupied space, the column indices are noted in ascending order (with reference to Figure Y).

R3	0	2	3	
R2	0	3		
R1	0	1	3	
R0	0	1	2	3

Figure 97: Within each row, the sorted column indices in ascending order are searched to identify consecutive numbers, representing adjacent sub-divided spaces (with reference to Figure Y).

As shown in Figure 97, by identifying columns with consecutive numberings, it is analogous to determining spaces that are adjacent to one another in physical space. For each set of running numbers, the lower pixel of the space with the lowest index and the upper pixel of the space with the highest index together defines a space merged by its row. This possibly gives a few of the largest rectangles, search as Case B and Case D of Figure 95.

It is to note the resulting spaces from the above merging methodology may result in cases that are a subset of the multi-division box space typology. These cases will be taken care of once all the detections are done, and elaborated on in the section titled cull subset.

SINGLE DIVISION COLUMN DETECTION

With an identical methodology as the single division row detection, the single division column detection implements the same algorithm with columns and rows swapped.

C3	0	1	2	3
C2	0	3		
C1	0	1		
C0	0	1	2	3

Figure 98: Consoildating for each column, for each unoccupied space, the row indices are noted in ascending order (with reference to Figure Y).

C3	0	1	2	3
C2	0	3		
C1	0	1		
C0	0	1	2	3

Figure 99: Within each column, the sorted row indices in ascending order are searched to identify consecutive numbers, representing adjacent sub-divided spaces (with reference to Figure 94).

In Figure 99, we see an example of how the largest single file column spaces can be detected.

MULTI-DIVISIONS BOX DETECTION

In contrast with the two types of detection explained above, the multi-divisions box detection is relatively complicated and requires multi-step, recursive searching processes.

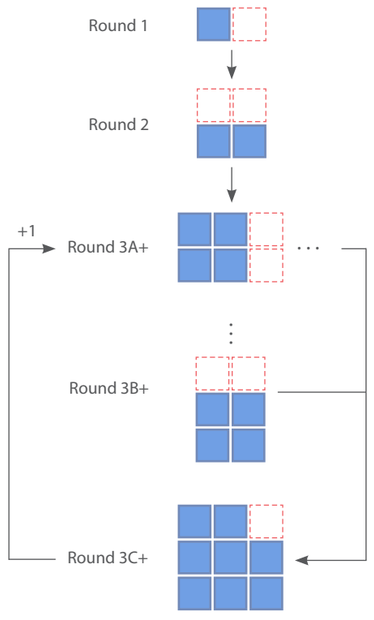


Figure 100: Overall flow visually describing the search for the next sub-divided space(s) to form a multi-division rectangular box typology space.

In Figure 100, the overall flow that highlights the next sub-divided space(s) to be searched for to form a larger multi-division rectangular box space is shown. The blue, filled squares represent an unoccupied space while the red, dotted squares represent the search to check if the space at the position highlighted are confirmed to be unoccupied. If all the spaces highlight in red are confirmed to be unoccupied, it proceeds to the next round of search.

This recursive search process is applied to every unoccupied sub-divided space found from the step in Figure 92. Each starting space that is fed into the recursive search loop will be referred to as the root space. The root space always take position at the lowest left corner of the largest rectangles found. For the sake of explaining the searching algorithm, the row index and column index of the root space will be referred to as R_{row} and R_{col} respectively.

For Round 1, the only space to search for is the space to the right of the root space. This would translate to the search for a space with the same row index as the root space and column index that is of a count higher than that of the root space: (R_{row} , R_{col} +1).

If Round 1 is fulfilled, the algorithm proceeds to Round 2. For Round 2, the spaces to search for are the two spaces above the ununoccupied spaces that have been found from Round 1: (R_{row} +1, R_{col}) and (R_{row} +1, R_{col} +1).

If Round 2 is fulfilled, the algorithm proceeds to Round 3. With Round 2 fulfilled, it would mean a square grid of 2 by 2 space have been confirmed. When a square grid is confirmed, it simultaneously triggers 3 possibilities the space could expand in.

Case 1 refers to the case of Round 3A+, which is a recursive search to expand in the direction towards the right. This translates to a search for: (R_{row} , R_{col} +2) and (R_{row} +1, R_{col} +2), followed by (R_{row} , R_{col} +3) and (R_{row} +1, R_{col} +3) and so on. The search will stop when any two of the space cannot be found as an unoccupied space.

Case 2 referes to the case of Round 3B+, which similarly is a recursive search, except it searches to expand in the direction upwards. This translates to a search for (R_{row} +2 , R_{col}) and (R_{row} +2, R_{col} +1), followed by (R_{row} +2 , R_{col}) and (R_{row} +2, R_{col} +1) and so on. The search will stop when any two of the space cannot be found as an unoccupied space.

If Round 3A+ and Round 3B+ were fulfilled, in a sense that the search algorithm was not stopped upon the first step of searching, it would trigger Round 3C+, which is the search for the next full square grid of one space larger. Round 3C+ searches for the top right corner space: (R_{row} +2, R_{col} +2).

If Round 3C+ is fulfilled as well, a full square grid of space is once again detected. This triggers a new set of detection, Round 4, which processes is identical to Rounds 3A+, 3B+ and 3C+, except with larger grid size. This continues until the search for all the unoccupied spaces have been halt due to the failure to meet the respective search conditions for each round.

CULL SUBSET

With these three methods of space detection explained, a comprehensive, overlapping set of the largest possible rectangles that make up the unoccupied spaces can be found (Figure 95). The following step will utilise the subset detection function described before to filter out all the spaces that are not truly considered to be amongst the set of largest rectangles. With this as the final step, a collection of the largest rectangles making up the unoccupied spaces can successfully be consolidated relative to the occupied spaces.

PROGRAM ASSIGNMENT

To find the best fit of program specified in the requirements, within the pool of available largest rectangular spaces found from the spatial recognition algorithm, there are three possible types of fits: exact fit, forced fit and selection fit. For all these typologies, where there is a choice, the most central positioning relative to the lift core position will be selected. This will be elaborated in the section titled central positioning.

EXACT FITTING

Exact fitting refers to the case for which the exact dimensions, according to the program requirements found in the program list, can be fitted into the chosen space.

For the case of exact fitting, the smallest unoccupied space that is able to fit the specified dimensions in either orientation is chosen. This is with the intention to reserve the bigger spaces for programs with larger area requirements. Granted that there indeed exist one of the space which is able to fit the exact dimensions, there are two possible cases. First, the case where the dimension can fit it both orientations while in the second case, the specified dimensions in the requirements can only fit in one direction.

For the case which dimension is able to fit in both orientations, the orientation which results in the tightest fit at either edge will be chosen (Figure 101).

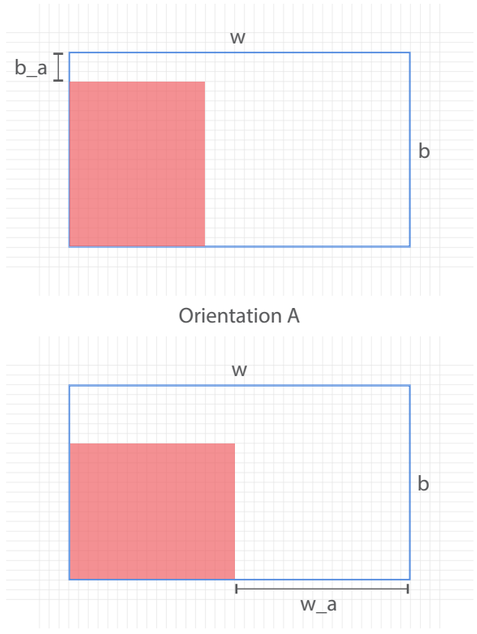


Figure 101: Finding the orientation that gives the tightest fit on either edge of the chosen space.

Referring to the width of the chosen space as space_w, the breadth of the chosen space as space_b, sideA of the programs requirement as req_sideA and sideB of the programs as req_sideB:

$$b\_a = space\_b - req\_sideA$$
$$w\_a = space\_w - req\_sideA$$

Recalling from Program Requirement section, by convention, req_sideA of a requirement dimension is the longer of the two edges. The above formula hence gives a comparison of the tightest edge of either side. If $b\_a < w\_a$, as in the example shown in Figure 101, the orientation with req_sideA as the breadth, Orientation A, will be chosen. If in another case, $w\_a \leq b\_a$, the orientation with req_sideA as the width will be chosen.

If in a case where only one orientation fits, the orientation that fits can be figured out by comparing which edge of the space is longer. If space_w > space_b, the orientation with req_sideA as the width will be the one that fits. On the contrary, if space_b > space_w, the orientation with req_sideA as the breadth will be the one that fits.

FORCED FITTING

If no suitable space is found from exact fitting, the condition to trigger forced fitting is tested. Forced fitting searches for a space that has area ranging within a specified threshold relative to the area specified by the requirement. Referring to the area threshold as th, the area of each space as space_area and the requirement area as req_area, here is the condition for the search of a suitable area:

$$(1 - th) \times req\_area \leq space\ area \leq (1 + th) \times req\_area$$

For example, if the specified threshold is 20%, it searches for all of the largest unoccupied spaces with areas that are within 80% to 120% of the requirement area. Of these shortlisted areas, the space with the area closest to the requirement area will have the entire space assigned with the program.

SELECTION FITTING

Selection fitting is the last possible method used to assign a program to a space. This fitting method is reserved for spaces which have areas exceed that of the upper bound specified in the forced fitting method, in other words, if:

$$space\ area \geq (1 + th) \times req\_area$$

If there are spaces which fulfil the above condition, the algorithm will attempt to compute the dimensions for a pair of subset of space within each space by maximising the width and breadth respectively while keeping approximately to the target area (Figure 102). This is an attempt to reduce the creation of awkard leftover spaces while trying to stick to the target attributes as close as possible.

For the two resulting sub-spaces for each space, the resulting aspect ratios will be computed and stored for comparison. Across all spaces that fulfil the above criteria, the sub-space that results in the aspect ratio closest to the target aspect ratio will emerge as the most suitable space to assign the program to.

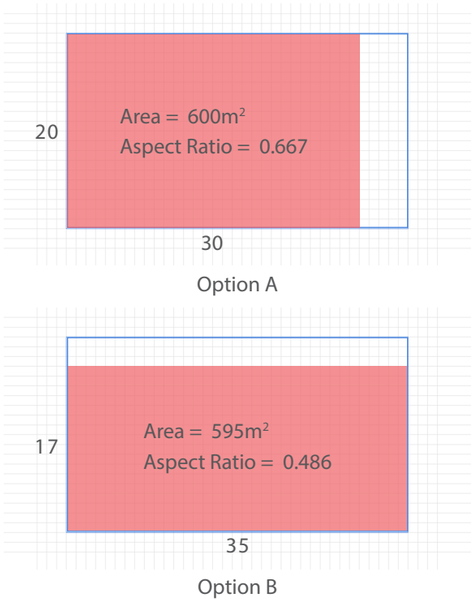


Figure 102: An example of the pair of test sub-spaces for one of the space which fulfils the criteria.

For example, if the target area of a program requirement is 600m² and target aspect ratio is 0.5, Option 2 of Figure 102 will be chose over Option A.

PROGRAM CULLING CONDITION

If all three methods of fitting is unable to find a suitable space for the next unassigned program in the program list, this particular program requirement will receive one count of warning for failure to adapt to any available space. The process continues by moving on to the next unassigned program requirement in the program list.

However, if any program requirement within the pgoram list hits a warning count that is greater than a threshold to be specified, this program requirement will be permanently removed from the program list.

CENTRAL POSITIONING

For the exact fitting and selection fitting processes, a sub-space of the unoccupied space is assigned to the program. This results in an extra step to determine which corner of the rectangular space should the new space be aligned to.

Using a simple example with only one assigned space, which is also the lift core, there are four

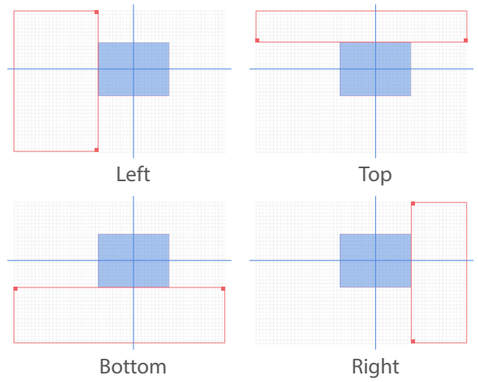


Figure 103: Determining the position of each space relative to the lift core.

largest possible rectangular spaces determined by the spatial recognition algorithm.

The first step in the central positioning algorithm is to identify the relative position of each largest rectangular unoccupied space relative to the lift core, whether it is on the left, right, top or bottom (Figure 103). Referring to the lower pixel row and column of the space as S_LR and S_LC, the upper pixel row and column as S_UR and S_UC, the lower pixel row and column of the lift core as S_LR and S_LC and the upepr pixel row and column as L_UR and L_UC respectively, this can be done with the following conditions:

$$\begin{aligned} \text{Left}: & \ S\_UC < L\_LC \\ \text{Right}: & \ S\_LC > L\_UC \\ \text{Top}: & \ S\_LR > L\_UR \\ \text{Bottom}: & \ S\_UR > L\_LR \end{aligned}$$

With the relative position figured out, the next step is to identify which of the two corner pixels closer to the lift core, highlighted in red for each of the space in Figure 103, is closest to the middle row or column of the lift core.

For the space that is towards the left of the lift core, the upper right pixel is the corner closest to the lift core. For the one on right, it is the lower right pixel. For the one on the bottom, it is the upper right pixel. For the one on the top, it is the upper left pixel. With the closest corner pixel to the lift determined, the new sub-space that the program will be assigned to be will be aligned to the determined corner accordingly.

PROCESS CONSOLIDATION

With the voxelization, lift initiation, spatial recognition and program assignment functions defined, the last part is to consolidate how the functions will work together to generate the tower.

LEVEL TERMINATION

The generation is done by floors, starting from the lowest ground floor and continues until the highest floor is completed.

This also arises the need for a certain condition to terminate the recursive search to assign a program for each level. This is done by defining a maximum number of program requirements the program assignment algorithm will search down the program list before it triggers the termination of the current level and initiates the next level.

For example, if the maximum number of program requirements to search through is set at 10, the program assignment algorithm will continue to try to fit the first 10 programs in the program list into available spaces. If all 10 programs are deemed unable to fit, the current level is terminated and the next level is initiated.

OVERALL FLOW

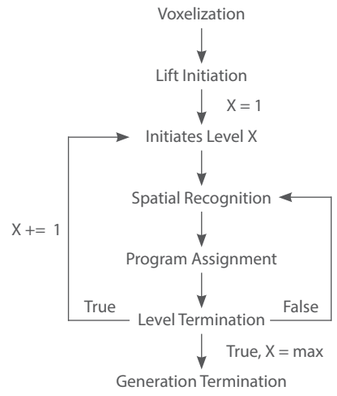
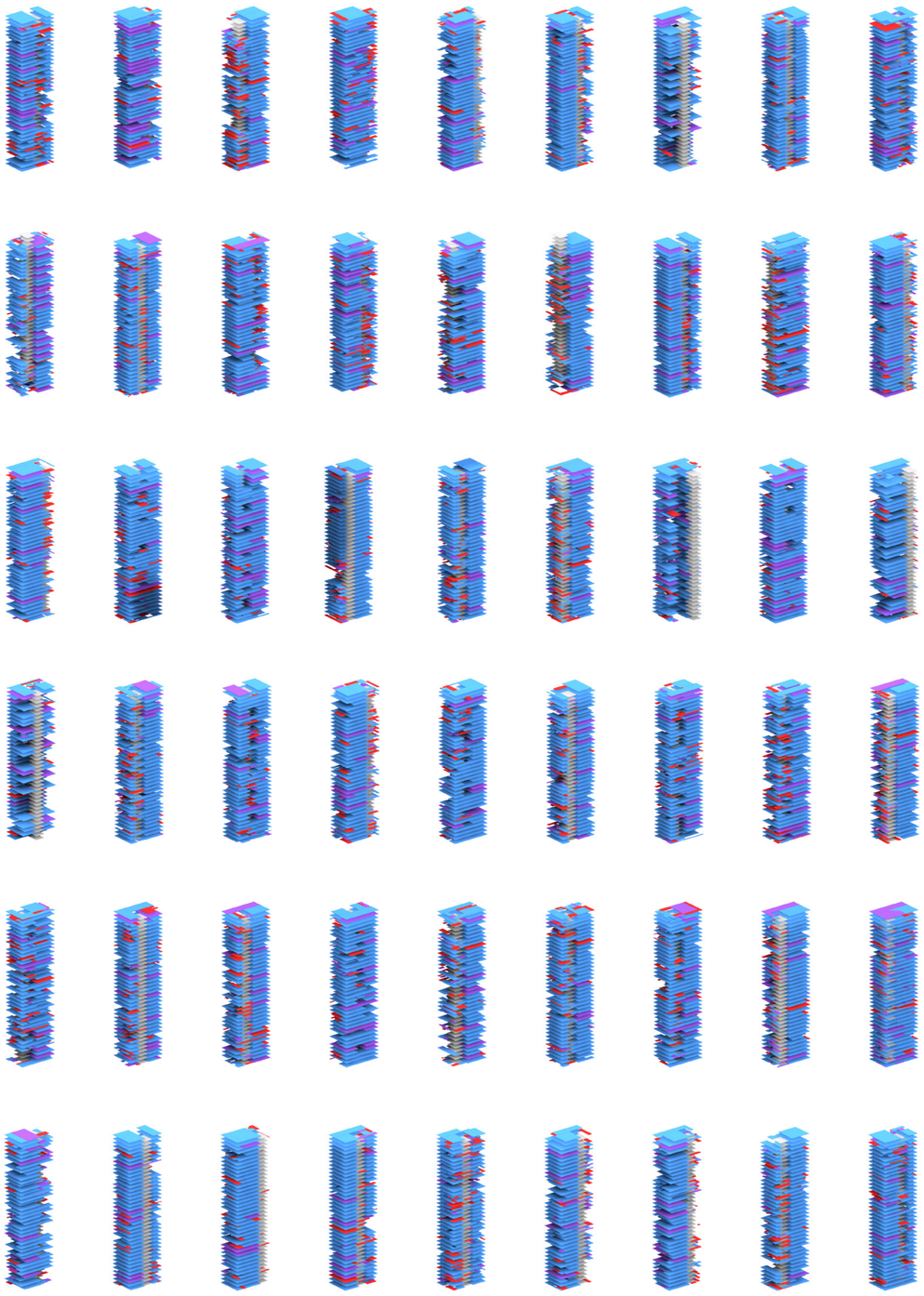


Figure 104: An example of the pair of test sub-spaces for one of the space which fulfils the criteria.



SAMPLE ITERATIONS