**Automated Floor Plan Generator**

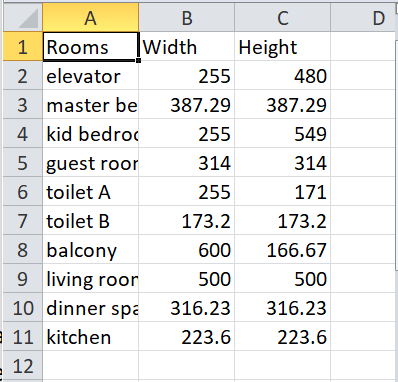
**Task 1**

**1 Introduction**



Fig. 1

A house is the most important purchase one might make in life, and we all want to live in a safe, comfortable, and beautiful environment. However, designing a house that fulfills all the functional requirements with a reasonable budget is challenging. Only a small fraction of the residential building owners have enough budget to employ architects for customized house design. House design is an expensive and time-consuming iterative process. A standard workflow is to 1) sketch a “bubble diagram” illustrating the number of rooms with their types and connections; 2) produce corresponding floorplans and collect clients feedback; 3) revert to the bubble diagram for refinement, and 4) iterate. Given limited budget and time, architects and their clients often need to compromise on the design quality. Therefore, automated floorplan generation techniques are in critical demand with immense potentials in the architecture, construction, and real-estate industries.

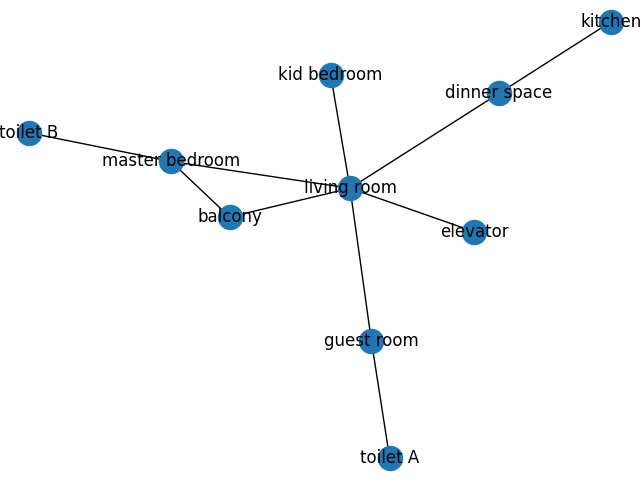


Fig. 2

This paper proposes a novel house layout generation problem, whose task is to take room sizes as an input, convert them into bubble diagram, and generate a diverse set of realistic and compatible house layouts (See Fig. 1). Room sizes are a list of the number of rooms and their corresponding dimensions of width and height. A bubble diagram is represented as a graph where 1) nodes encode rooms with their room types and 2) edges encode their spatial adjacency. A house layout is represented as a set of axis-aligned bounding boxes of rooms (See Fig. 2).

**2 Parameters of house layout generation problem**

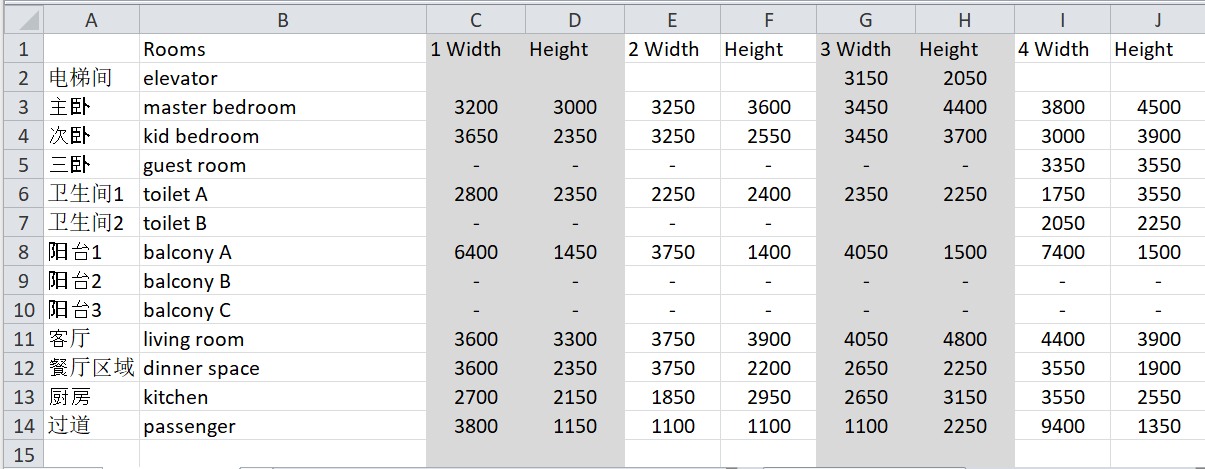


Fig. 3

**2.1. Dataset:** CSV database containing China’s common room dimensions. The database contains room type and their corresponding sets of width and height (See Fig. 3). First used module panda’s data-frame processing functions to convert the input file containing all samples into individual csv files containing single sample. Next we use module networkx to convert the csv file into a bubble diagram. Networkx provides a fast implementation of graph data structure in python. A bubble diagram is a graph, where a node is a room with a room type as its property. Two rooms are connected based on a predefined set of preset functions and adjacency table. An output house layout is axis-aligned bounding boxes.

**2.2. Assumptions:** In contrast to the real design process, a restrictive assumption to simplify the problem setting: A room shape is always a rectangle. This is the first research step in tackling the problem, where these extensions are our future work.

**3. Creating Floor Plans:**

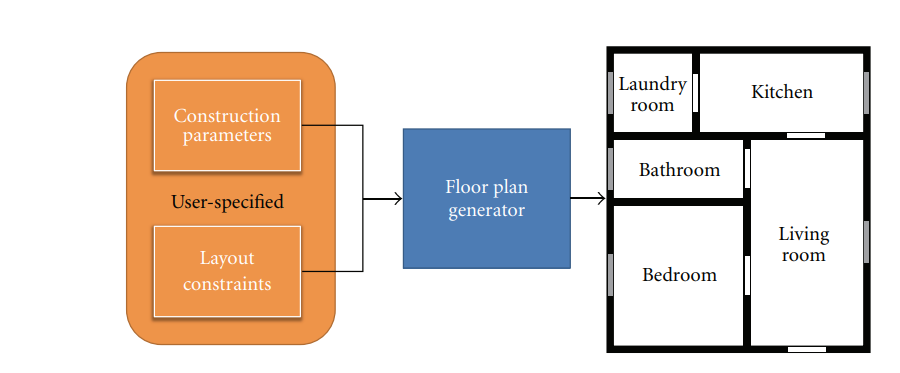
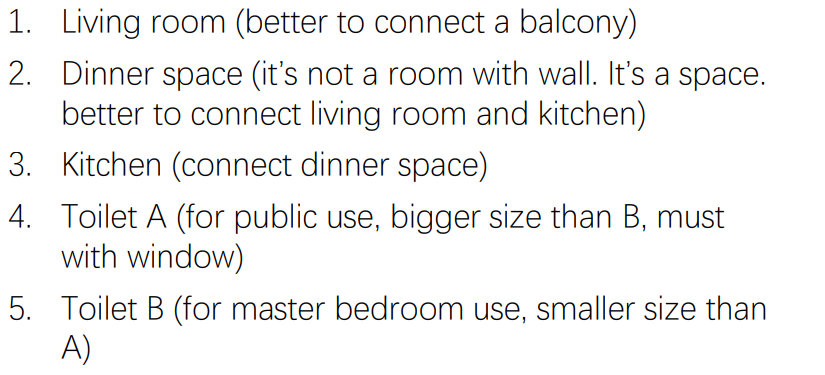


Fig. 4

**3.1. The Proposed Model:** The pipeline of the proposed model to build floor plans is presented in Figure 4. The process begins with the definition of construction parameters and layout constraints. To create a floor plan, some parameters such as, height, length, and width of the building are required. It is also necessary to know the list of desired dimensions for each room and their functionalities. The functionality specifies how a particular area of residence should be used. These functionalities or layout constraints are a set of limitations listed in Chinese design code. Fig.5 shows some of the layout constraints this model follows.



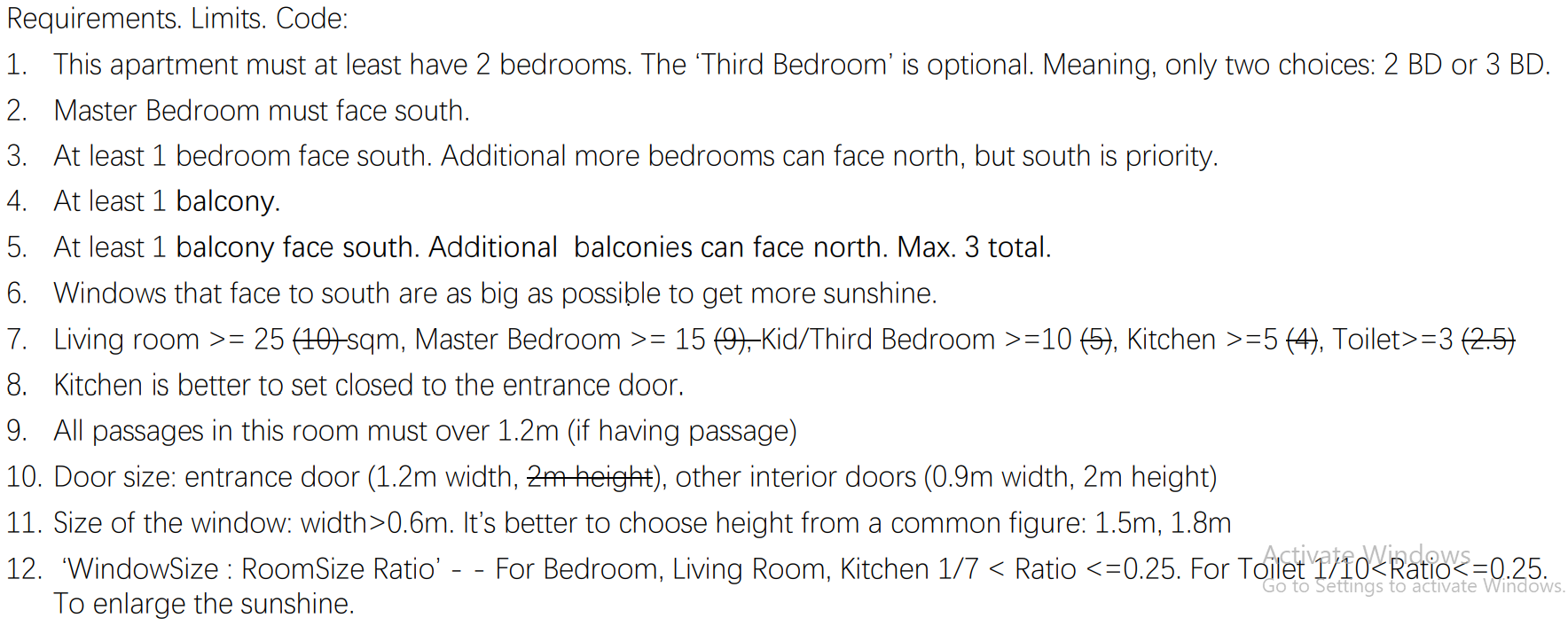


Fig. 5

Construction parameters are provided in the input csv file. The layout constraints are predefined in another csv file that defines how rooms should be clustered. Each entry of the construction parameter becomes a node in the bubble diagram with node attributes: 1) Room type; 2) Width and 3) Height.

**3.2. Including Connections among the Rooms:**

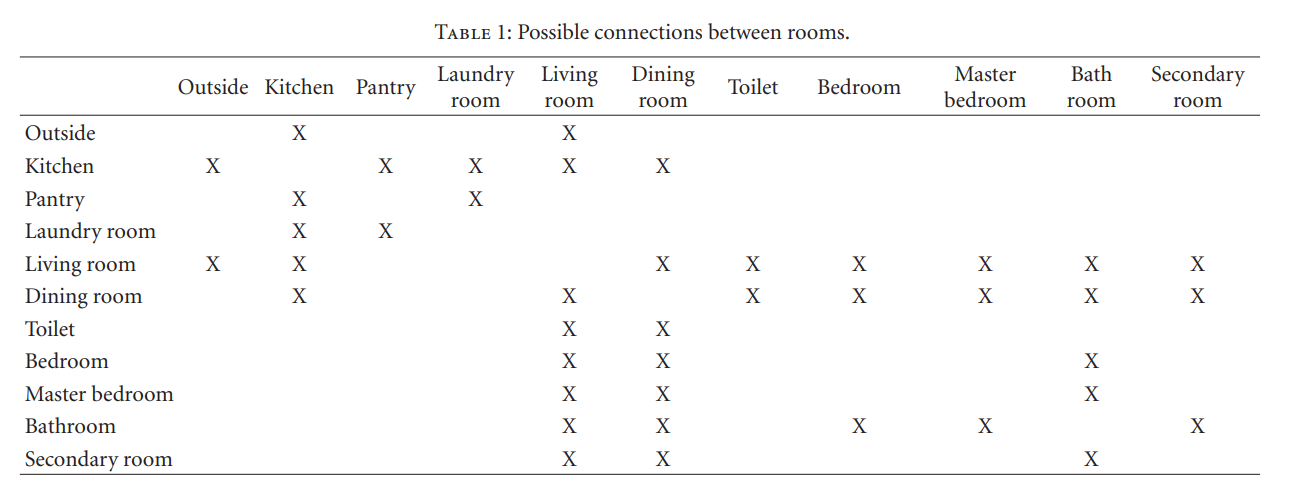


Table 1

With all the room nodes generated as previously described, connections should be created among them. These connections are created using as criteria the functionalities of each room, that is, some rooms are usually not connected, for example, the kitchen and the bedroom. All the possible connections are presented in Table 1, which has been modeled based on a previous analysis of various floor plans commercially available. In this table, we consider the entry door of the floor plan as a connection from outside and two possible rooms: kitchen and living room. However, it is important to note that another set of connections can be defined by the user, to represent another style of architecture.

Firstly, the nodes in bubble diagram are clustered into connected groups of rooms representing each of the functional presets that are defined in the Chinese design code (See Fig. 6).

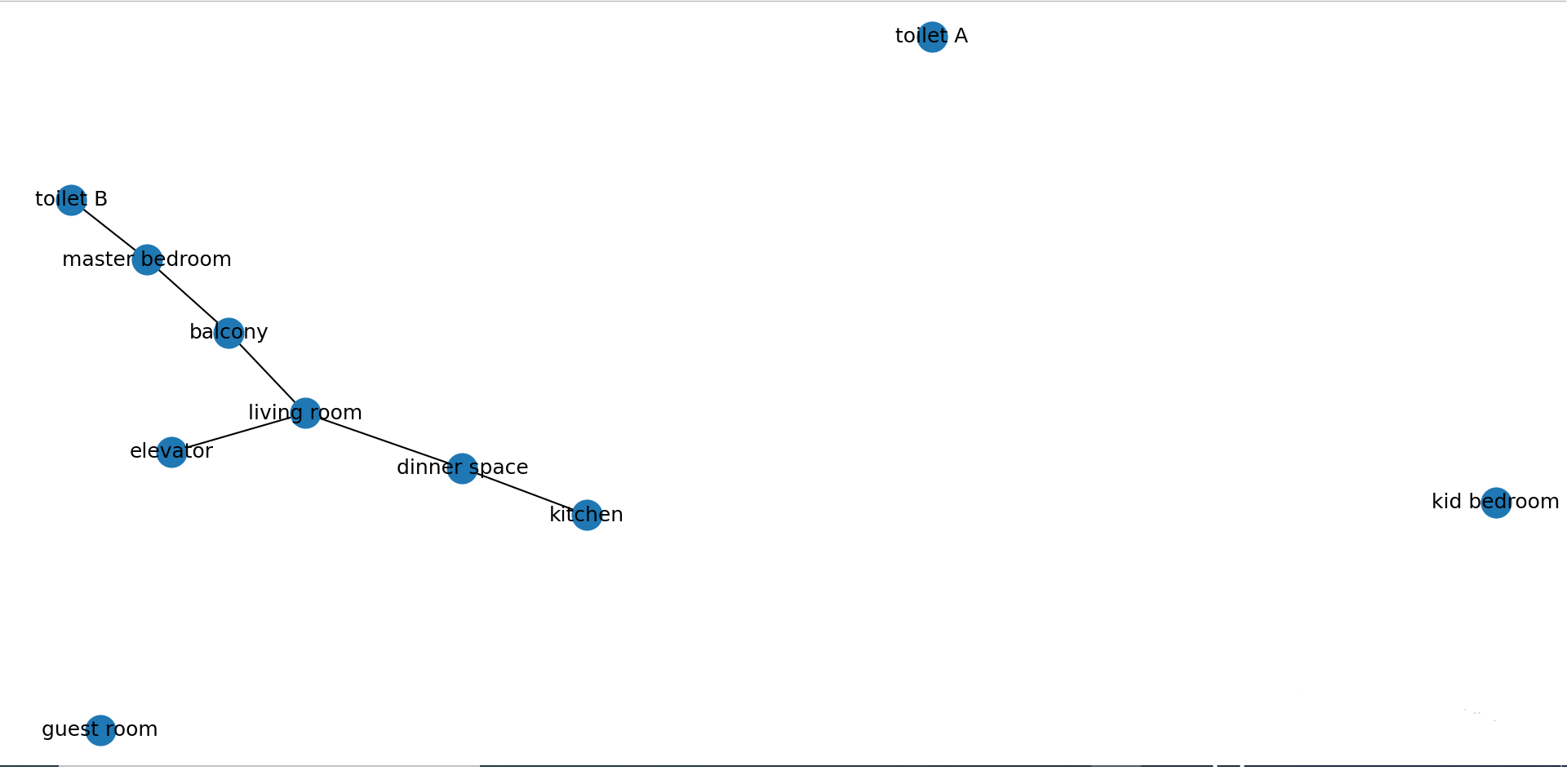


Fig. 6

Then the clusters are connected by traversing a csv file representation of Table 1, containing a list of all possible edges. The final bubble diagram can be seen in Fig. 7

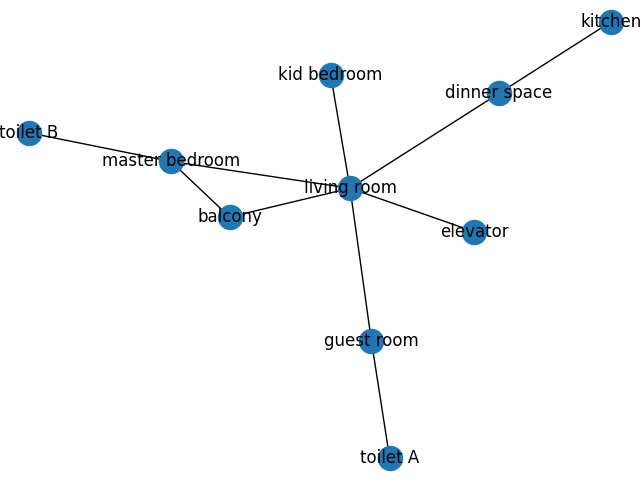


Fig. 7

**3.3. Generating Room Layout:** The bubble diagram is only a representation of the actual layout, the main goals this model needs to accomplish is converting it into a set of best possible layouts; according to the metrics specified in section 2.2.

The approach this model uses is simple yet effective. Each room is classified into three distinct categories: social area, service area, and private area. The social area can include the living room, the dining room, and the toilet. In the service area we can have the kitchen, the pantry, and the laundry room. At last, the private area can embrace the bedroom, the master bedroom, the intimate bathroom, and a possible secondary room that can be used in different ways, for example, as a library. This list is not fixed and can be customized by the user, and the proposed categorization is made to group the common areas.

First, the model defines rooms that belong to social area (living room, dinner space and public toilet) and plot them on a rectangle that is mapped by maximum width and height of the apartment. Next it plots the private area (bedrooms) around the living room, while maintaining all functional presets. For example: Master bedroom is connected to living room and faces south. Similarly, the service area (kitchen) is placed around the dinner space. Lastly, the attached rooms (balcony, toilets) are plotted with respect to their adjacent rooms. For example: Intimate toilet is connected to master bedroom; balcony is connected to living room; etc.

**3.4. Adding doors and windows:** Geometrically, the door is created on the edge shared by two rooms that keep a possible connection. For instance, it is possible to have a connection between the kitchen and the living room, where a door can be created. The size of the doors is predefined, and their center on the edge is randomly defined. A similar process happens with the generation of windows, but ensuring that they are placed on the external edges. Figure 1 illustrates a generated floor plan, containing windows (white rectangles) and doors (black triangles).

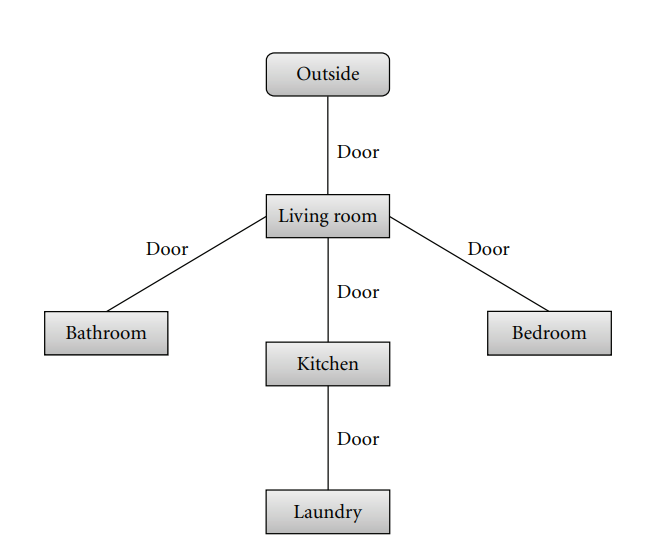
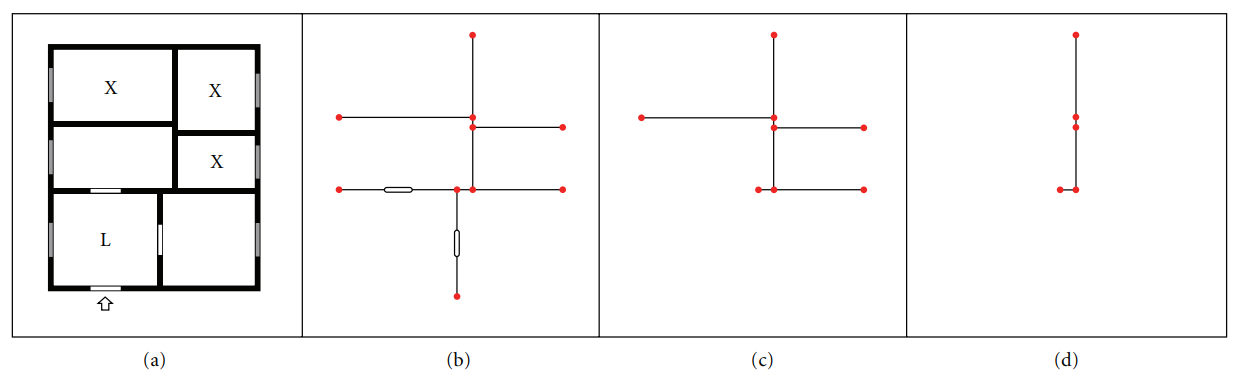


Fig. 8

After the connections between rooms are processed, a connectivity graph is automatically created (Figure 8), representing the links among the rooms. It allows checking if there is any room that is not accessible. Also, buildings and houses created with our method can be used to provide environment for user simulation. The graph always starts from outside and follows all possible connections from the accessed room. If any room is not present in the graph, it is necessary to include corridors. This situation can happen when there are no possible connections between neighboring rooms (rooms which share edges). This process is described in the next section.

**3.5. Including Corridors on the Floor Plans:**



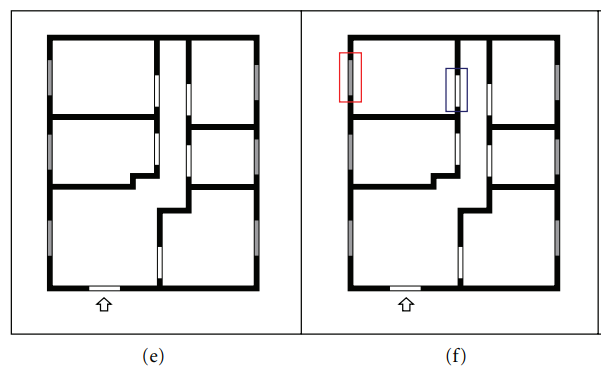


Fig. 9

The generation of corridors is necessary in order to maintain the coherence of the generated environment and to provide valid space for characters navigation. Firstly, the rooms without access from the living room are selected. These rooms are flagged with an X, as illustrated in Figure 9(a). The main idea is to find possible edges shared by nonconnected rooms to be used to create the corridor. The corridor must connect the living room (marked with an L in Figure 9(a)) with all X rooms. The proposed solution uses the internal walls of the building to generate a kind of circulation “backbone” of the space, that is, the most probable region to generate the corridor in the floor plan.

The algorithm is very simple and has three main steps. Firstly, all external walls are removed, as corridors are avoided in the boundary of the floor plan (Figure 9(b)). Secondly, we remove all internal segments (representing walls) that belong to the living room (Figure 9(c)). The remaining segments (described through their vertices) are used as input to the graph creation. Vertices are related to nodes, and segments describe the edges in the graph (Figure 9(c)). In order to deal with the graph, we use A∗ algorithm, which is an **optimal AI algorithm** widely used in path-finding and graph traversal. In our case, the graph is explored to find the shortest path that connects all rooms without connectivity to the living room. A room is considered connected if the graph traverses at least one of its edges. Finally the shortest path is chosen to create the corridor (Figure 9(d)).

**4 Running the script:**

The model is scripted in python language and requires a python interpreter to run. Any version of python can be used to run the program. It requires an environment with the following modules installed:

* pandas
* networkx
* matplotlib

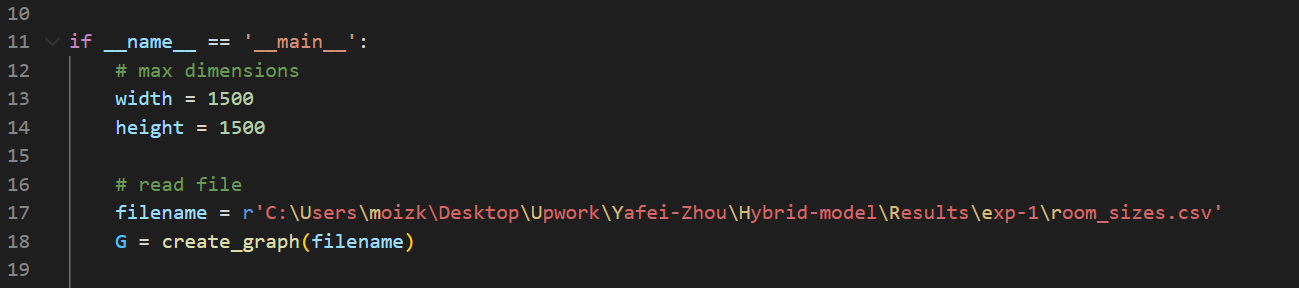
The environment can be setup in windows or linux in 2 simple steps:

* Open terminal and go to the directory where the project is saved.
* Run the command “pip install –r requirements.txt”.

The project contains 7 files:

* Graph.py: Generates bubble diagram from input csv file (construction parameters and functional presets).
* Layout.py: Generates the layout from bubble diagram
* Edges.py: Generates doors and windows.
* Render.py: Plots the layout on an image with color code.
* Main.py: Provides the driver code to run the model.

Running the script is easy.



First add the max dimensions in line 12 and 13. Then add the directory of the input csv file in line 17 (as shown above). Next open the terminal and type “python main.py”.

**5 Front end Design:**

\*Will be developed in the end…

**6 Results we get:**

\*Results for 10 experiments are already provided on the shared folder under the file name “Results\_model2.zip”. Csv file is input and images are output.

**7 Compare model to others:**

* No other model till date incorporates both doors and windows for automated floor plan generation which makes our model most practical.
* No other model is capable of incorporating China’s design code limitations or any other sort of functional presets when generating layouts. That is why our model is best for the Chinese market.
* Other models such as HouseGAN++, GCN, Ashual et al and Johnson et al predict layouts with the accuracy of 30-80% while our hybrid model boasts an accuracy of over 90% in every case.
* No other model can generate a layout within a predefined boundary (A\* B). Our model is capable of doing that for any value of A and B.
* No other model uses corridors in their layout, which makes most of their outputs useless since there are rooms that are not connected to other areas such as living room. Our model can sense if there is a need for corridor in a given layout, then generate the shortest path corridor using AI path finding algorithm.
* Our model can generate results instantly rather than waiting for hours and even days for training unlike House-GAN.

**8 How our model improve with funding:**

**8.1. Handling non rectangular room shapes:** Using vectorization algorithm we can convert each room rectangle into a separate raster image and change its shape and dimensions according to user specifications.

**8.2. Generating thousands of layouts and choosing best one:** Use the funding to collect a large dataset of china’s high rise apartments and use that to train an even stronger AI model that will predict thousands of layouts for each input and select the best one from that almost instantly. This will be done using a combination of conditional GAN (to add presets).

**8.3. Thermal performance and simulation:** As seen above, one of the earlier and most important design phases, performed by architects, is space planning. The use of computers to speed up the drawing process, also helping the architect to cope with a much larger set of objectives, has been a computational challenge since the 1960s. However, most efforts resulted in simplified static estimation and floor plan generation approaches have never been coupled to dynamic simulation, even though the idea had been previously proposed (Zimmermann, 2005). By coupling dynamic simulation to automated generation of floor plans algorithms it is possible not only to globally evaluate and rank different design solutions, but also provide the architect with detailed information on the performance of each space, allowing him to take judicious decisions. In the space planning phase, design decisions may have significant impact on the thermal performance of a building.

Floor plans generated by our model will be assessed, ranked, and optimized for their thermal performance using the FPOP algorithm, which consists in a design variable sequential optimization conforming to thermal performance criteria. Specific design variable operators are used sequentially according to the user design strategy. Depending on which variables, the operators test all admissible values, e.g. to flip the floor plan horizontally or vertically, or uses a gradient-descent technique, e.g. to determine the best window size of a space.

The thermal performance assessment is carried out using EnergyPlus dynamic simulation program every time that a design solution is subject to any change. Using a sequential design variables procedure, it is possible for the architect to specify his design strategy, setting which objects and in what order to optimize. Even though different design strategies could produce different results, the architect may want to merely improve the window dimensions or to place overhangs, making this approach more versatile for the architect to use in different design phases. It also allows the user to step back and re-optimize some other aspects of the design solution that were changed manually by the architect. The design variables available to the user are floor plan orientation and reflection, window orientation and size, overhang size, fin size, and wall translation, which result from changing floor plan element parameters.

FPOP optimizes EPSAP’s generated designs either by using a gradient descent technique or by testing all admissible variable values with its operators. This reduces the computational time significantly in comparison to population-based approaches, in an architectural design stage where fast prototyping is important. EPSAP and FPOP algorithms allow the architect to create, compare and improve alternative floor plans. Practitioners may screen these alternative solutions and select which should be further developed in the following stages of the architectural design process.

After floor plans have been generated, thermally assessed and ranked, the FPOP algorithm will explore the improvement potential of each design solution. Starting with the preferable floor plan orientation randomly determined between -45 and 45 degrees North, FPOP will try to determine new positions for all windows (OT) in every space perimeter before reflecting the floor plan (FPR) according to North-South and East-West axes. Then, it will rotate the floor plan (FPO) until it finds the best orientation in the vicinity of possible solutions. Next, it resizes the openings (OD) and translates walls (WT) before recalling Floor plan Orientation (FPO) operator to fine tune the orientation. Finally, FPOP determines if there is a need to place and resize overhangs (OvD) and fins (FD) in spaces that have over-heating penalties.