IJNRD.ORG ISSN : 2456-4184
INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD) | IJNRD.ORG
An International Open Access, Peer-reviewed, Refereed Journal

Generative Floor Plan Design Using Deep Learning: An AI-Powered Approach

Balraj Vaidya, Padmakar Pimpale, Guruprasad Khartadkar

International Institute of Information Technology, Pune

Savitribai Phule University

Pune, India

Abstract : This research paper examines the application of Artificial Intelligence (AI) in architectural design and spatial planning, with a specific focus on AI-driven floor plan generation. The research aims to develop a user-friendly system that enables individuals to effortlessly create floor plans by inputting dimensions and preferences. Employing advanced AI algorithms, the system processes user-provided data to produce tailored floor plans, considering factors like room layout and space optimization. The paper explores the potential of this innovative approach to streamline architectural and interior design processes, serving as a valuable tool for professionals and enthusiasts. This research contributes to the ongoing efforts to enhance spatial design and planning through the integration of cutting-edge technology.

Index Terms: Artificial Intelligence, Floor Plan Generation, Architectural Design, Space Planning, User-Provided Dimensions

1. INTRODUCTION

Designing homes is a multifaceted challenge that hinges on the creation of floor plans, representing rooms and their spatial relationships within a building. These rooms serve specific purposes, such as kitchens, dining areas, or bedrooms. The process of creating floor plans is complex, governed by explicit and implicit rules to ensure functionality and aesthetics. While the rules are flexible, there are essential guidelines that must be adhered to. For instance, it is not practical for a bedroom to be situated within a kitchen. The process of designing floor plans often involves an iterative collaboration between architects and clients in professional settings. Architects produce initial drafts, gather feedback, and make revisions, factoring in considerations like lighting, heating, and aesthetics. However, this process can be time-consuming and labour-intensive.

Creating numerous designs for a single project is inefficient and costly. Automating or streamlining this process could significantly reduce the time and cost of home design. Furthermore, many individuals who are not professional architects seek to remodel or construct their homes but lack the expertise or financial means. This creates a demand for democratizing home design skills, which could be achieved through software automation. Such software could provide non-professionals with cost-effective architectural services and streamline the iterative design process.

While tools like AutoCAD, Revit, and SketchUp exist to assist architects in creating floor plans, none actively collaborate with the creator to intelligently design the plan. Creating software capable of doing so would require a substantial level of intelligence to understand both explicit and implicit floor plan design rules. Hard-coding these rules is impractical, as they are multifaceted and often subjective. Machine learning offers a potential solution, using techniques such as computer vision, sequential models, and adversarial networks. However, adapting these techniques to the intricacies of architectural design is a challenging task.

One approach is to leverage models designed for computer vision tasks. Floor plans can be represented as images, but recognizing walls and room arrangements presents significant challenges for leading computer vision methods. Another approach involves representing floor plans as a list of vectors, reducing the dimensionality of the object recognition problem. However, this representation must address the variable length of floor plans and the positional relationships between objects.

In this paper, we explore machine learning approaches to automate floor plan design. We test sequence-predicting models, generative graph models, and a hybrid model that combines both approaches. Our goal is to develop software that reduces the workload on architects and provides a co-creative platform for floor plan development, benefiting both professionals and non-professionals in the field of home design.

2. Related Work

The scope of this paper encompasses two primary domains: automated floorplan generation and the application of machine learning in creative processes. Automated floorplan generation can be categorized into graph and combinatorial methods, as well as machine learning applications. Machine learning has also found utility in diverse creative domains, such as text, music, and drawing.
The challenge of automating floorplan design has persisted for over half a century [7]. Initially, the early solutions involved graph theory applications and combinatorial techniques [7]. As genetic algorithms and machine learning methods advanced and gained popularity, researchers began to explore the problem using these cutting-edge approaches [10]. Notably, contemporary deep learning methods like recurrent neural networks have not been employed in this context. We contend that these advanced methods offer promising potential due to their capacity to identify patterns within existing elements and their successful application in analogous creative domains.

2.1.1 Graph methods:

Levin made a significant breakthrough by introducing the concept of abstracting floorplans into graphs, where functional areas are nodes and connections between them are edges. However, early attempts at this method, also used by Cousin, struggled to revert the abstraction into practical floorplans. This limitation persisted for nearly a decade until Grason proposed a solution that restricted room shapes to simple rectangles. This constraint allowed the development of algorithms capable of generating potential floorplans from a given graph, although it also limited the variety of floorplans that could be created.

It's important to note that early efforts primarily focused on designing floorplans for functional structures like office buildings and factories. These settings are well-suited for a graph or mathematical solutions, as their evaluation is based on their ability to fulfil specific functional requirements. In contrast, residential home layouts need to harmonize functionality and aesthetics, taking into account various home styles and personal preferences. This complexity cannot be adequately captured by mathematical functions or graph theory alone, rendering an approach relying solely on these methods inadequate for addressing the diverse challenges of residential design.

2.1.2 Machine Learning Techniques:

In the paper "Architectural Layout Design Optimization," Michalek et al. [10] introduced two decision-making processes. One process focuses on optimizing the overall plan's geometry, while the other concentrates on optimizing the plan's topology or room relationships. The topology model essentially achieves the same goal as the graph representations used in prior solutions, while the geometry model aids in translating room relationships into geometric representations. Michalek et al. [10] applied simulated annealing and genetic algorithms to optimize mathematically defined design objectives and constraints in both models. However, the model's effectiveness is contingent upon predefined mathematical functions. We believe that an approach capable of learning from existing examples might be less rigid and yield more practical results.

3. Background

In this paper, we use a combination of a sequential and graph model to generate floorplans. Recurrent neural networks are central to our implementation of the sequential model. Generative adversarial networks, and the graphGAN variant, are central to our graph model. Below we provide background information on these networks.

3.1 Recurrent Neural Networks:

Recurrent neural networks are particularly well-suited for handling sequential data, which is crucial in the context of AI floor plan generation. These networks excel in processing various types of sequential data, such as text, time-dependent information, or musical notes. Unlike feed-forward neural networks that process all data simultaneously, recurrent neural networks operate sequentially. They base their decisions about the next item in the sequence on the context of previously encountered inputs, which are stored as a hidden state.

In the architecture of a recurrent neural network, cells play a key role. Each cell performs calculations using learned weights and specific input values. In a vanilla recurrent neural network cell, the input comprises an item from the input sequence and the hidden state received from the preceding cell. These inputs undergo multiplication by learned weights and are then processed through an activation function. The resulting output is passed to the next cell as a hidden state or is used as an output of the network. This process can be mathematically represented as $ht = \varphi (Wxt + Uht-1)$,
where 't' indicates the nth term in the sequence, reflecting the network's sequential processing nature.

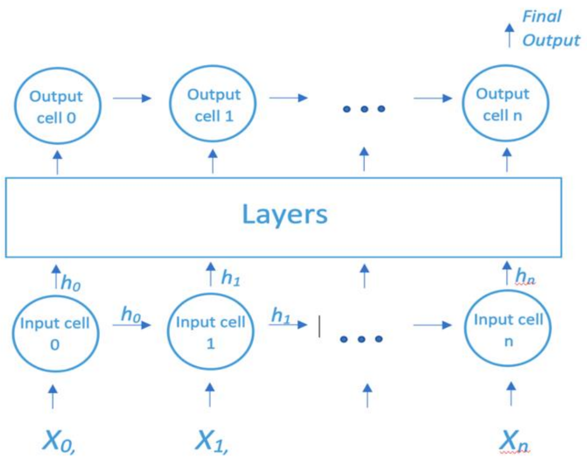


Figure 3.1: Recurrent Neural Network Architecture

The architecture of a recurrent neural network is depicted in Figure 3.1, where cells are arranged sequentially, and layers are visible. The output of one cell can serve as the input for another layer of cells, enabling the model to capture increased complexity. It's important to note that recurrent neural networks are adept at recognizing patterns, which is a valuable attribute for AI floor plan generation. However, they can sometimes struggle to maintain information over extended periods.

3.1.1 Long-Short-Term-Memory Nodes:

A Long-Short-Term-Memory (LSTM) network is a specialized type of recurrent neural network designed to retain information over an extended duration. The LSTM network incorporates two key components for context management. Similar to a standard recurrent neural network, it possesses a hidden state that traverses through the network. In addition to the hidden state, an LSTM also features an extra memory cell, which serves the purpose of preserving pertinent information for use in subsequent iterations. The flow of information into and out of this memory cell is meticulously controlled by input, output, and forget gates. These gates are responsible for learning what kind of information should be retained, retrieved, or discarded. This unique architecture empowers the LSTM network to effectively "remember" critical information over numerous data points in the future.

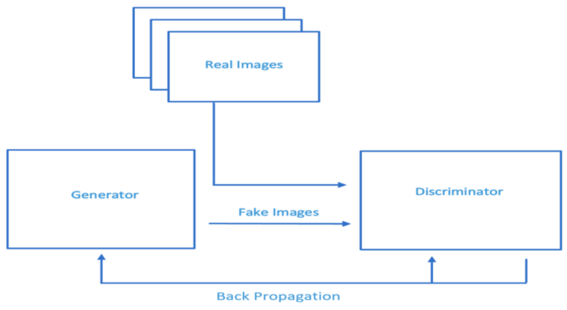


Figure 3.2: Generative Adversarial Network schema

3.2 Generative Adversarial Networks :

In the context of AI floor plan generation, Generative Adversarial Networks (GANs) have become a pivotal tool. Introduced in 2014, GANs consist of two fundamental components: the generator and the discriminator. While many machine learning models aim to approximate labels (y) based on variables (x), GANs are tailored to work conversely. The generator's primary function is to approximate x given y, representing a specific data distribution. In the context of AI floor plan generation, this means that the generator is tasked with creating floor plans based on specific design criteria and user inputs. To effectively train the generator, a discriminator is employed. The discriminator evaluates the floor plans generated by the generator and distinguishes them from real floor plans. This process serves as a critical feedback loop, enabling the generator to improve its ability to create realistic and functional floor plans.

The GAN algorithm operates on a minimax function, where the generator's reward is tied to the discriminator's loss, and vice versa. Over time, both the generator and discriminator become increasingly adept at their respective roles. The generator's goal is to produce floor plans that are indistinguishable from real-world designs. The minimax function mathematically encapsulates this dynamic, ensuring that the generator strives to create high-quality floor plans that meet user-defined criteria.

3.3 GraphGANs :

Since the inception of GANs in 2014, numerous variations and adaptations have been proposed. Researchers have harnessed the adaptability of GANs, experimenting with different components and machine learning models. Notably, GANs have achieved remarkable success in generating synthetic images that closely resemble real-world counterparts. Consequently, much research has been dedicated to enhancing GANs for image synthesis.

An interesting departure from this image-centric focus is the Graph Generative Adversarial Network (GraphGAN), introduced by Wang et al [11]. GraphGAN employs a generator, denoted as G(v|vc), to determine the probability distribution of connectivity (p(v|vc)), with vc representing a chosen node. The primary objective is for the generator to predict the most probable edges to emerge from vc. Simultaneously, the discriminator's role is to differentiate between authentic vertex pairs and those generated by the model. While GraphGAN excels in generating edges within an existing graph, it does not facilitate the creation of new nodes, limiting the potential for graph expansion.

4. METHODS

In addressing the floorplan problem, it can be dissected into two primary sub-problems.

1. Learning to generate individual room types within the context of previously generated rooms.

2. Learning to assemble these rooms cohesively into a floor plan.

To tackle the first sub-problem, we propose the utilization of a sequence model. This model is trained to produce a sequence of rooms, each associated with their respective feature vectors. Given an initial list of rooms, the sequence model generates the next room, adding it to the list. This iterative process continues as needed.

For the second sub-problem, we adopt a graph-based representation of the floor plan. In this representation, rooms are treated as nodes, and the connections between rooms are depicted as edges. By identifying realistic edges in this graph, we glean insights into how rooms should be positioned and connected. Starting from the generated list of rooms, we explore all possible edges connecting any two rooms. These edges are subjected to evaluation by a discriminator, which filters out implausible connections. The discriminator's output yields the final set of edges that define the generated floorplan. Combining the outcomes of these two models, we obtain both a feature vector and a set of edges, effectively constituting our graph-based representation. The conversion from this graph representation to a comprehensive floorplan is achieved through a graph-to-floorplan algorithm.

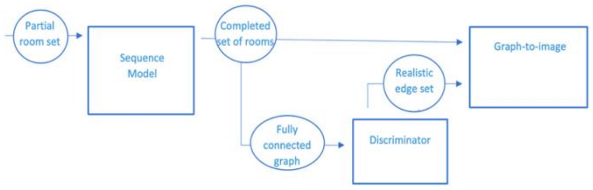


Figure 4.1: Hybrid Model Architecture

4.1 Room Feature Generation via Sequential Inference

In our approach, we conceptualize the rooms within a floor plan as part of a sequential structure. We believe that the creation of a floorplan should be viewed as a sequence where each room's design heavily depends on the context of the existing rooms. Furthermore, adhering to certain patterns and spatial relationships is essential to ensure the final layout is practical and visually coherent. To capture and learn these intricate patterns and relationships, we employ sequential machine learning models, which can effectively predict the next room in the sequence.

In more intricate floorplan designs that encompass high-dimensional representations of rooms, including characteristics like size, material, shape, and lighting, recurrent neural networks have demonstrated their competence in predicting the next items in a sequence. Specifically, long short-term memory networks (LSTMs), a variant of recurrent neural networks, excel in retaining significant contextual information across longer sequences, making them particularly suitable for larger and more complex building layouts.

To facilitate the model's ability to learn these patterns and relationships, maintaining a consistent sequence order is imperative. In our methodology, we order the floor plan based on room position. The sequence commences with the top-left room and follows a logical progression to the right, then down, and finally to the left, creating a "snaking" pattern. Formally, when determining the sequence of rooms with shared walls forming connections between them, a depth-first-search algorithm is employed. This algorithm initiates with the top-left room and assigns priority to the directions "right," "left," "down," and "up" when ordering neighbouring rooms. Each room within the sequence is represented as a two-dimensional vector of essential characteristics. For our experiments, we identify three pivotal features: width, height, and room type. These attributes are fundamental for visually constructing a room within the floor plan. The model processes the critical features of the existing rooms and predicts the corresponding attributes of the next room in the sequence. The newly generated room is also represented as a vector of characteristics and is seamlessly integrated into the sequence. This iterative process continues until the desired number of rooms is achieved.

4.2 Graph-Based Room Assembly:

The generation of individual rooms using the sequential model is only part of the floor plan creation process. To form a complete floor plan, it is essential not only to have a list of rooms but also to understand how these rooms are interconnected. We propose a graph-based approach for capturing the spatial relationships and connections between rooms within a floor plan. By representing floor plans as graphs, we can directly encode their structural knowledge.

In this graph representation, each room is treated as a node, and shared walls between rooms are represented as edges. We denote this graph as G(V | E), where V represents the set of vertices (rooms) and E represents the set of edges (shared walls). By learning to create edges between nodes in this graph representation, our model gains insights into the adjacencies and connectivity between rooms. The generation of edges between nodes is achieved by training a discriminator capable of distinguishing between real and artificial edges. This discriminator takes into account critical features of two rooms, including their width, height, and room type, to predict the likelihood that these two rooms are adjacent. Upon generating all the rooms using the sequential model, we transform these rooms into nodes within a fully connected graph. Each edge within this graph undergoes evaluation by the discriminator, and edges that are considered artificial are pruned from the graph. This process results in a graph with only the edges that are deemed realistic.

The discriminator was trained as part of a generative adversarial network (GAN). We made slight modifications to the existing GraphGAN framework to create a model capable of evaluating the likelihood of a connection between any two nodes. The generator, denoted as G, is designed to approximate an existing node vc such that it is likely to be connected to another node v. On the other hand, the discriminator, denoted as D, calculates the probability that an edge [vi, vj] belongs to the set of edges E. Both the generator and discriminator are involved in a minimax game, where the generator strives to create an edge between an existing node and a generated node, while the discriminator aims to distinguish between real and artificial edges. In the training process, the discriminator and generator work in opposition, as improvements in one model's performance lead to corresponding losses in the other. However, when it comes to generating a graph, both models play essential roles. The generator creates realistic nodes, adds them to the graph, and connects them by choosing an existing node and predicting a new node (VC). The discriminator, trained to identify realistic edges, tests all pairs [vi, vc] for all existing nodes vi in the graph. If the D(vi, vc) value surpasses a certain threshold, an edge (vi, vc) is created in the graph.

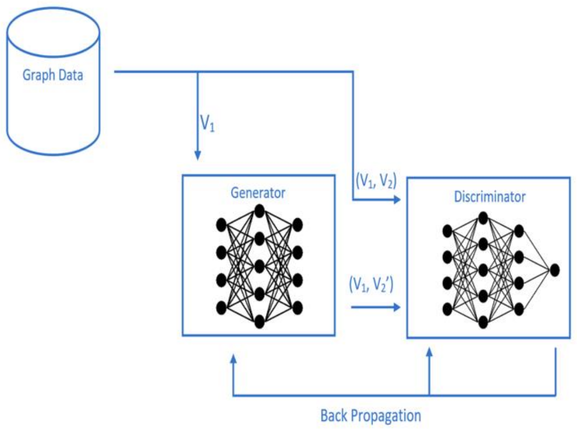


Figure 4.2: Graph Generative Adversarial Network

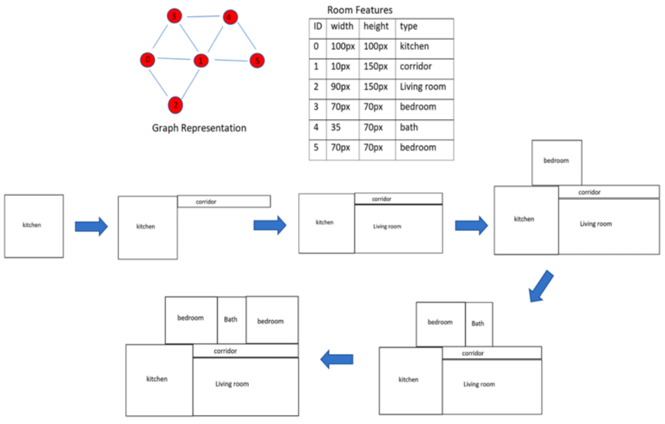
4.3 Converting the Graph to an Image:

Our graph-based representation captures room information and adjacencies but lacks spatial relationships. To convey spatial relationships, we assign coordinates to rooms using their width and height features. A room is represented as a rectangular shape with two coordinate points: minimum (top-left) and maximum (bottom-right) corners. The goal is to convert the graph representation into a set of coordinates, effectively creating an "image" of the floorplan.

The algorithm generates all possible images based on the room set and adjacencies provided by the models. There are numerous image possibilities from one graph, with more edges leading to a smaller solution set due to increased constraints. The algorithm iterates through the graph, converting nodes into coordinates within the image. A room can be added in different positions within the image, so the algorithm maintains a set of all potential images.

The process begins with an external node and adds neighbours in DFS order, ensuring that each node adheres to adjacency requirements with existing rooms in the image. Each node can be placed in up to four positions relative to its neighbours: up, down, left, or right.

The number of possible images can be extensive, up to O(4n). However, the average case is typically much lower, especially when rooms have multiple edges, which limits placement options. The chosen image is often the most compact one, with compactness determined by the ratio of the total area covered to the total area utilized. If this ratio is 1, the floorplan forms a perfect rectangle.



Room Features

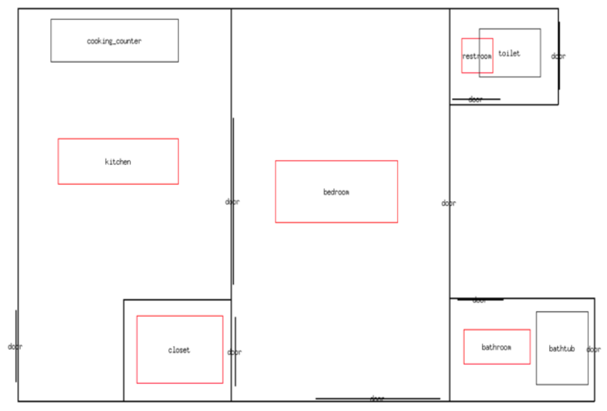
ID	width	height	type
0	100px	100px	kitchen
1	10px	150px	corridor
2	90px	150px	Living room
3	70px	70px	bedroom
4	35	70px	bath
5	70px	70px	bedroom

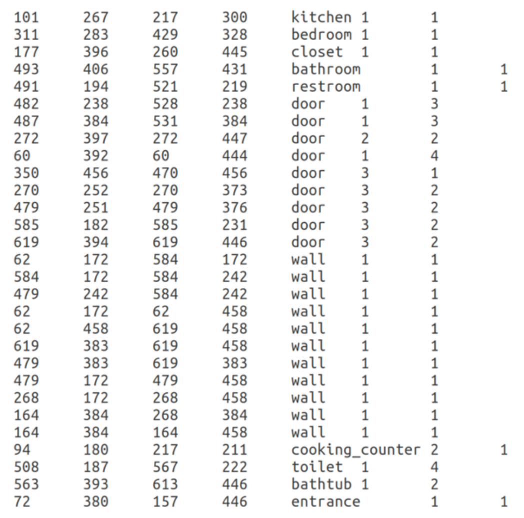
Figure 4.3: Graph to image process

5. Evaluation

In our evaluation, we conducted a comparative analysis involving our hybrid model and three distinct types of baseline models. We individually employed the sequence and graph models to construct floorplans, allowing us to establish baselines for each component of the hybrid model and assess the advantages of their combination. Additionally, we devised an average-based model, generating floorplans based on aggregated statistics extracted from our dataset.

To evaluate our method, we formulated an algorithm for comparing floorplans. While gauging a floorplan's ability to cater to human preferences is inherently complex, we opted for a comparative approach by assessing the overall shape of the generated plans. This was achieved by determining the degree of overlap between two floorplans when one is optimally superimposed on the other. Further details on the methodology for calculating this overlap are provided below. To execute the evaluation, we initiated each model by presenting it with the first room of a real plan from the vector graphics representation of the dataset. Subsequently, the model's task was to autonomously complete the remainder of the floorplan. The evaluation process involved quantifying the overlap between the real plan and the generated plan, allowing us to gauge the effectiveness of our hybrid model in comparison to the selected baselines.





```
101     267     217     300     kitchen 1       1
311     283     429     328     bedroom 1       1
177     396     260     445     closet  1       1
493     406     557     431     bathroom        1       1
491     194     521     219     restroom        1       1
482     238     528     238     door    1       3
487     384     531     384     door    1       3
272     397     272     447     door    2       2
60      392     60      444     door    1       4
350     456     470     456     door    3       1
270     252     270     373     door    3       2
479     251     479     376     door    3       2
585     182     585     231     door    3       2
619     394     619     446     door    3       2
62      172     584     172     wall    1       1
584     172     584     242     wall    1       1
479     242     584     242     wall    1       1
62      172     62      458     wall    1       1
62      458     619     458     wall    1       1
619     383     619     458     wall    1       1
479     383     619     383     wall    1       1
479     172     479     458     wall    1       1
268     172     268     458     wall    1       1
164     384     268     384     wall    1       1
164     384     164     458     wall    1       1
94      180     217     211     cooking_counter 2       1
508     187     567     222     toilet  1       4
563     393     613     446     bathtub 1       2
72      380     157     446     entrance        1       1
```

Figure 5.1: Data in vector form and visualized

5.1 Preprocessing:

To train our sequence model, we required room attributes, specifically the width, height, and room type, for each room within the floor plan. However, our available data did not provide room dimensions directly. Instead, it denoted room labels within the confines of each room's walls. These room labels were defined by arbitrary coordinates and did not offer explicit information about the room's dimensions. On the right side of Figure 5.1, you can observe a visualized example of a floor plan, with the room labels highlighted in red.

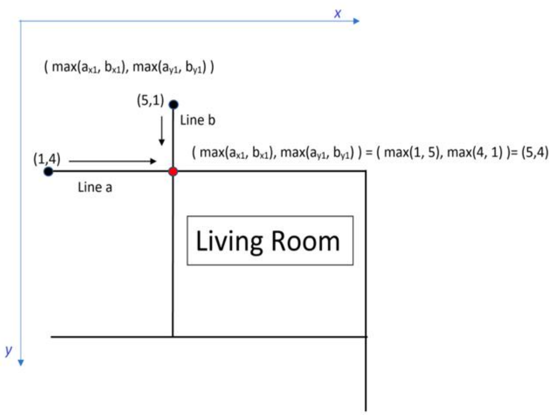

x
y
(max(a_x1, b_x1), max(a_y1, b_y1))
(5,1)
Line b
(max(a_x1, b_x1), max(a_y1, b_y1)) = (max(1, 5), max(4, 1))= (5,4)
(1,4)
Line a
Living Room

Figure 5.3: Deriving the (x1, y1) Values of a Room from the Upward and Leftward Walls

Even though certain rooms couldn't be precisely represented using only width and height dimensions, it was imperative to approximate these features. This was necessary to enable our model to learn about room adjacencies and estimate rough dimensions.

In the process of deriving the room's width and height, we initially identified the minimum and maximum x and y values that defined the room. The minimum x and y values were referred to as (x1, y1), while the maximum x and y values were represented as (x2, y2). Our preprocessing algorithm was initiated by locating the walls that enclosed the room. These walls were the ones immediately above, below, to the left, and the right of the room label provided in the dataset. It's important to note that the algorithm exclusively considered walls that spanned the full length of the room label.

Subsequently, these walls were employed to approximate the (x1, y1) and (x2, y2) corner coordinates. Remembering that each wall had its own set of (x1, y1) and (x2, y2) corner coordinates is crucial. For horizontal walls, the y1 and y2 values were identical, while for vertical walls, the x1 and x2 values matched. A room's corner was defined by the intersection of a horizontal and vertical wall. If both walls terminated at the same corner, the relevant (x, y) values of the horizontal and vertical walls coincided. However, it's typically the case that only a portion of a wall is relevant to enclosing a specific room. In other words, any given wall usually extends beyond the boundaries of a room. When determining a room's corner using horizontal and vertical walls, the (x, y) values were selected to be as close as possible to the room's centre. The (x1, y1) values were found by identifying the intersection of these two lines. This intersection occurred below the beginning of the vertical line and to the right of the beginning of the horizontal line. In this coordinate plane, the y coordinate increases in the downward direction, and the x coordinate increases in the right direction. Therefore, the appropriate y value could be determined by selecting the maximum of the two lines' y1 values (the one closest to the bottom), while the appropriate x value was determined by choosing the maximum of the two lines' x1 values (the one closest to the right). This process is depicted in Figure 5.3.

Similarly, when determining the (x2, y2) values of the room, walls often extended beyond their intersection point. In such cases, the walls extended too far to the right and too far below, so the intersection point was found by identifying the minimum values between the walls' x2 and y2 values.

6. Conclusion

The floorplan generation challenge has persisted for decades, witnessing evolving approaches alongside advancements in computing tools, both algorithmic and hardware. Our proposal represents a fusion of deep learning and graph application techniques, offering a method for floorplan generation. Our approach combines a sequential machine learning model for room generation with a graph model for predicting room adjacencies.

In our experimental assessments, our hybrid model demonstrated superior performance compared to its components when tested in isolation. Additionally, it outperformed an average-based model that relied on dataset average statistics. This success can be attributed to the hybrid model's ability to effectively leverage each component's strengths.

Despite these intriguing results, we acknowledge the presence of limitations, particularly concerning room shapes. Our current implementation enforces a rectangular shape for all rooms, which does not reflect the diversity of real-world architectural designs and may constrain the artistic aspects of floorplan generation. We believe that our Artificial Neural Network methods offer the potential to address these limitations by accommodating various input sizes, thus allowing for the generation of rooms with diverse shapes based on an accurately representative dataset.

7. References

1. Knowledge-driven description synthesis for floor plan Interpretation*: International Journal on Document Analysis and Recognition, 2021*.

2. Per Galle. "An algorithm for exhaustive generation of building floor plans". In: Communications *of the ACM* 24.12 pp. 813–825.

3. Automatic Generation of AI-Powered Architectural Floor Plans using Grid Data: International Journal of Applied Engineering Research ISSN 0973-4562 Volume 18 Number 2 (2023).

4. Tell2Design: A Dataset for Language-Guided Floor Plan Generation, StaNLP Research Group, Singapore University of Technology and Design.

5. Karol Gregor et al. "DRAW: A Recurrent Neural Network For Image Generation". In: *CoRR* abs/1502.04623 (2015). arXiv: 1502.04623. URL: http://arxiv.org/abs/1502. 04623.

6. Natasha Jaques et al. "Generating Music by Fine-Tuning Recurrent Neural Networks with Reinforcement Learning". In: *Deep Reinforcement Learning Workshop, NIPS*. 2016.

7. Peter Hirsch Levin. "Use of graphs to decide the optimum layout of buildings". In: *The Architects' Journal* 7, pp. 809–815.

8. Chen Liu et al. "Raster-to-vector: revisiting floorplan transformation". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2195–2203.

9. Paul Merrell, Eric Schkufza, and Vladlen Koltun. “Computer-generated residential building layouts”. In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 6. ACM. 2010, p. 181.

10. Jeremy Michalek, Ruchi Choudhary, and Panos Papalambros. “Architectural layout design optimization”. In: *Engineering optimization* 34.5 (2002), pp. 461–484.

11. Hongwei Wang et al. “Graphgan: Graph representation learning with generative adversarial nets”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.