

CARNEGIE MELLON UNIVERSITY

School of Architecture
College of Fine Arts

Thesis

Submitted in Partial Fulfillment of the requirements for the degree of

Master of Science in Computational Design

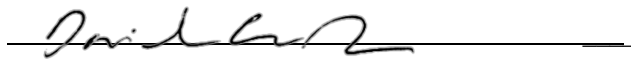
TITLE:

**Architecture Data Processing: Using Machine Learning to
Analyze Floor Plan Layout Designs**

AUTHOR:

Yufei Cheng

ACCEPTED BY:



Prof. Daniel Cardoso Llach

Advisor

May 29, 2020

DATE

Architecture Data Processing: Using Machine Learning to Analyze Floor Plan Layout Designs

Yufei Cheng
School of Architecture
Carnegie Mellon University
Pittsburgh, PA 15213
yufeic2@andrew.cmu.edu

Abstract

Thanks to a growing body of research into the application of machine learning in architecture, architecture is moving into the digital realm. One obstacle is the lack of large datasets and uniform data encoding. To address this question, this thesis establishes an architecture data processing framework for using machine learning to analyze floor plan layout designs. It proves preliminarily that the framework, from data extraction to data encoding validation, has the potential to provide a large amount of data for quantitative analysis of architectural spatial quality.

In this thesis, the technical aspects of this newly proposed architecture data processing framework are presented and discussed. For data extraction, graph theory is applied to extract a graph-based dataset. For feature engineering, isovist analysis is applied to convert spatial information into features. For data encoding, the graph embedding technique is used to explore a variety of data encoding methods. For data validation, the spatial features are linked to architectural qualities, specifically, five criteria for evaluating apartment rooms' spatial quality are applied. In this framework two software programs for extracting and converting the graph-based dataset are outlined. From a proof-of-concept experiment, it could be concluded that the choice of different spatial features and data encoding methods will affect the accuracy and efficiency of floor plan analysis. And it's necessary to select a reasonable combination according to the analysis and prediction objectives.

By presenting a novel approach to architecture data processing for analyzing floor plans, this thesis opens the gate for data-driven architectural analysis and consolidates the foundation for the architecture field to enter the big data era.

Acknowledgments

I wish to express my gratitude to the people who helped and supported my research.

First and foremost, I want to thank my advisor, Daniel Cardoso Llach for his guidance and feedback throughout the development process of this thesis. Daniel's knowledge and expertise in the field helped me to improve this thesis and think critically in the process. Without his insightful guidance, this thesis cannot be developed.

I would also thank Ardavan Bidgoli for reading this thesis and giving insightful advice in my thesis's development.

I wish to thank all people in the MS in Computational Design program for their help and encouragement.

Finally, my gratitude goes to my family and my girlfriend who supports my graduate studies at Carnegie Mellon University. None of this would happen without your love and support.

Contents

1. Introduction	3
1.1 Preface	
1.2 Research statement	
1.3 Methodology	
1.4 Thesis structure	
2. Background	7
2.1 Literature review	
2.2 The gap	
2.3 Why choose the graph representation?	
2.4 Graph embedding	
3. First Look at the Dataset	20
3.1 Source of data	
3.2 Feature Engineering	
3.3 Floor plan analysis task setting	
4. Methods	27
4.1 Graph data encoding	
4.2 Expanding features	
5. Extracting Input Graph Data from Raw Image	36
5.1 Graph data processing framework	
5.2 Graph data extraction process	
6. Validation Experiments	47
6.1 Experiment on feature selection	
6.2 Experiment on data formats	
7. Contribution, Possible Applications, and Future Work	52
8. References	56
9. Appendices	58

Chapter 1. Introduction

1.1 Preface

In the last decade, with the fast advancement of computational capabilities, a growing body of research into the application of machine learning in architecture has emerged (Pal et al., 2018; Harding & Derix, 2010). One of the applications is to use machine learning to statistically analyze existing designs and to build a learning model that can emulate the design process. When a designer is designing the layout of a floor plan, normally, he attends to apply various evaluation criteria to evaluate the quality of the designed space and repeatedly adjusts the design according to the characteristics or qualities each room needs to have. Therefore, to make machine learning to understand and emulate architecture design better, it is particularly important to investigate how to use machine learning to analyze and evaluate the spatial quality of design results.

As the fuel of machine learning, data is of great research value. The training process of machine learning requires a well-defined problem, that is, both the data inputs that describe the problem and the data output that defines the training objectives need to be quantified and clearly defined (Cai et al. 2017). When it comes to architectural design, the problem becomes how to extract the spatial information from design outcomes as data input, and how to turn the spatial quality evaluation criteria into quantitative output targets. On the other hand, when it comes to machine learning, it's also important to solve the problem of training data shortage and the lack of a unified standard of data encoding format.

Therefore, it's of great significance to research on architecture data processing. A complete data processing process from the extraction of spatial information from architectural designs to proper

encoding data into machine learning input will contribute to build databases dedicated to architectural research. And it can support research such as the development of automated design tools and large-scale architectural analysis.

1.2 Research statement

Considering the problems described above, this thesis tries to find answers to the questions: Will different approaches to architecture data processing affect the final analysis outcomes? If so, how to conduct a rigorous data processing procedure for floor plan layout analysis?

Therefore, this thesis brings the discussion of how to extract spatial information from designs and encode them as standard vectorized data when applying machine learning statistical analysis on the evaluation of room's qualities via floor plans. The proposed architecture data processing framework opens the gate for data-driven architectural analysis. A large amount of architectural design data processed by this framework will make it possible to generate an input database for machine learning and enable applications such as rapid mass evaluation of spatial environment quality and automated floor plan design based on specific spatial evaluation indicators. For example, a spatial environment quality evaluation system can be used in real estate to quickly rate the house for sale. And a generative machine learning framework can be developed on a database driven by this framework to automate the design process of function-driven tasks. All in all, this thesis on architecture data processing consolidates the foundation for the architecture field to enter the big data era.

1.3 Methodology

This thesis focuses on architecture data processing and proposes a data processing framework for floor plan design analysis using machine learning, including data extraction, feature engineering, data encoding, and performance validation. The framework takes plan images as input, and then, outputs standardized vector data available for machine learning. In the validation experiment, the dataset used is the images of apartment floor plans collected from an apartment rental website (<http://www.ziroom.com>), and the analysis is based on five different evaluations of rooms' spatial quality, from privacy to sleep-ability. And to avoid model bias, six different statistical machine learning algorithms are applied.

By elaborating on the technical details of the architecture data processing framework and the proof-of-concept experiment, this thesis demonstrates the framework's potential to provide a large amount of data for quantitative analysis of architectural space. Firstly, it proves the possibility of extracting spatial measures of rooms in a floor plan using plan images as input, and proposes a general graph-based data model as well as its extraction process in which the feature engineering based on spatial syntax and graph theory is proved to be effective by experiments. It also proposes three methods based on graph embedding to convert the original data model into standard vector data which is used as machine learning input. And through experiments of predicting five qualities of apartment rooms, it's proved that different encoding methods will affect the accuracy and efficiency of the predicting. The detailed experiment recording and comparative analysis can guide future research in selecting different spatial feature variables and data encoding methods according to different analysis objectives. And through the statistical analysis on a sample dataset, it shows that some qualitative aspects of spatial configurations can be retained and statistically learned from the dataset processed by this framework.

1.4 Thesis structure

Structurally, this thesis is organized with four parts. Firstly, a literature review of existing research from the intersection of architectural design and machine learning, by which, the theoretical background of the framework is understood. Specifically, graph theory and isovist analysis are applied as the foundation of graph-based data extracting and spatial feature engineering. Also, the graph embedding theory is applied to provide methods for converting graph data and encoding it in different formats. Secondly, the dataset used in this thesis and proposed architecture data processing framework are elaborated in detail. By looking into the dataset, the specific floor plan layout analysis objectives are discussed along with their quantitative data representation. In addition to the technical details of the framework, two software programs for extracting and converting the graph-based dataset are outlined. In the third part, a proof-of-concept experiment is introduced to demonstrate the data processing workflow and provide a comparative analysis of the accuracy and efficiency of data processed by different feature engineering and different encoding methods. The final part discusses the contribution of this thesis, as well as the limitations and potential extensions.

Chapter 2. Background

This chapter will focus on several backgrounds, including the reason for focusing on data processing.

First, a literature review will be made to briefly introduce related fields. Second, after the elaboration of the research backgrounds, the gaps which this study intends to close, along with the motivation and limitation of this thesis will be discussed. Third, given that the analysis on floor plan layouts requires clearly structured data input with quantitative target parameters for training, the methods to use the graph theory to extract information based on the graph model will be explored. And fourth, the methods for converting raw graph modeled data into standardized vector data (called graph embedding) are analyzed.

2.1 Literature review

A lot of existing researches are focused on the combination of machine learning, graph theory and architecture design. Although these studies have produced interesting results from the practical application of leading technologies in the machine learning field, such as deep learning neural networks, to the field of architectural design, there are some arbitrary and direct judgments and steps in the process of practice. Especially the steps related to the dataset and data processing is an interesting and important topic.

Several studies have addressed some related topics. E.g., “Artificial intelligence in architecture: Generating conceptual design via deep learning” (Pal et al., 2018) (see Figure 1) uses machine learning to recognize the functions of different inner spaces from an image-based floor plan layout; and further, it

uses a neural network to transform the function table into graph structure data and to generate new design outcomes. Also, other researches, such as “Architectural Drawings Recognition and Generation through Machine Learning” (Zhen et al., 2018) and “Associative Spatial Networks in Architectural Design -- Artificial Cognition of Space Using Neural Networks with Spectral Graph Theory” (Harding & Derix, 2010), explore how to use machine learning methods, including Generative Adversarial Network (GAN) and deep learning neural networks, to generate graph-like structure data, so as to represent designs and find out intriguing results of spatial analyses.

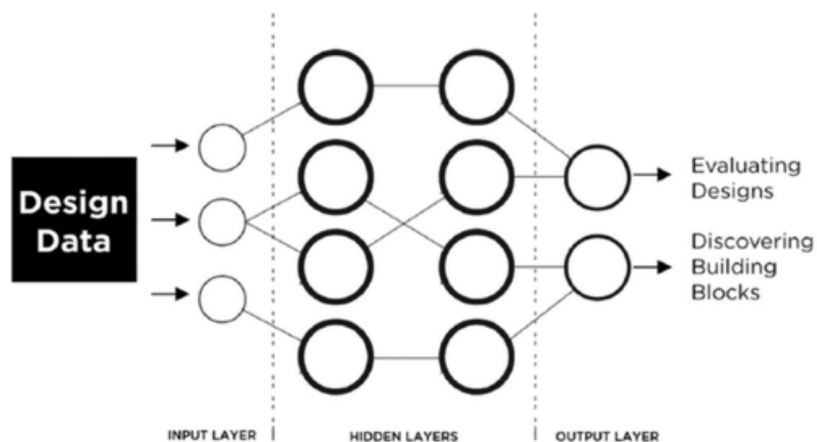


Figure 1. A workflow of deep learning networks used for analyzing design data. From Imdat et al. 2018.

More recently, some researchers use machine learning to extract design logics and function relationships behind the floor plan layout and study how to apply Generative Adversarial Network (GAN) to achieve image recognition, especially to denote and label different functional areas in layout plans (Pal et al., 2018). Due to the rapid development of machine learning and deep learning, several outstanding results are generated in their conjunction, such as “Sneaky Spatial Segmentation. Reading Architectural Drawings with Deep Neural Networks and Without Labeling Data” (Harding., 2010) and “Machines’ Perception of Space: Employing 3D Isovist Methods and a Convolutional Neural Network in

Architectural Space Classification” (Peng et al., 2017) (see Figure 2), shedding new lights on tackling the existing problems in automating the labeling and recognition of plan layouts and generating the corresponding programmable models.

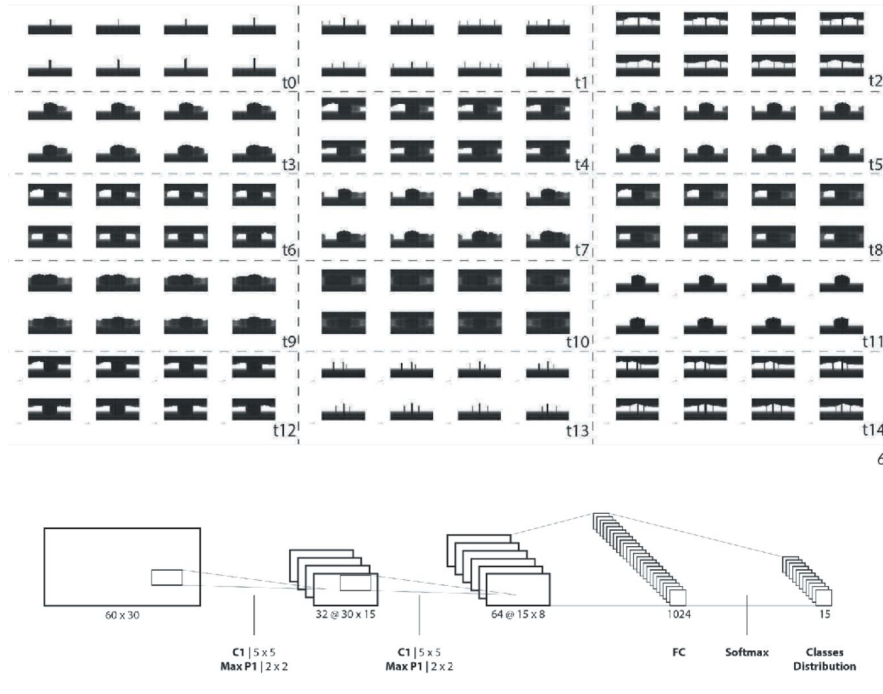


Figure 2. The CNN architecture where inputs are space samples. From Peng et. al 2017

On the other hand, several other researches, such as “Towards a machine learning framework in space syntax” (Ferrando et al., 2018), have discussed how to use extracted graph models to make further examination on the design process and how such models can be enhanced by using the graph theory, so as to propose a generative learning framework capable of performing design tasks. While these researches emphasize less on examination of specific design types and architecture types but on a machine learning framework for general floor plan layout designs, certain other researches pay more attention to multiple instances of a specific building type, such as “Architectural Distant Reading Using Machine Learning to Identify Typological Traits across Multiple Buildings” (Kvochick, Tyler. 2018).

2.2 The gap

Although the existing studies provide valuable insights into how to find patterns behind design outcomes and automate the design process by combining machine learning and the graph theory, they lack in-depth research on an important issue -- data, especially about the preparation of datasets and data processing before input into machine learning frameworks.

As to data processing, several studies have addressed databases in the design field. E.g., “Database of the Modern Architecture of the City of Maracaibo 1920-1990” (Espina, Jane J.B. 2002) implements a specific database based on a specific city. And the other study, “An Interactive Database (HizmO) for Reconstructing Lost Modernist Izmir” (Akgun, Yenal. 2003), provides a new interactive database to represent the design.

Focusing on the limitation of data processing in the previously addressed research, two papers can be analyzed. Firstly, in “Artificial intelligence in architecture: Generating conceptual design via deep learning” (Pal et al. 2018), the author proposed a graph-based machine learning system to extract information from three-dimensional space data and generated a deep learning model capable of performing conceptual design, which evolves into unique and new designs. The focus of the research is to recognize the functional combination of elements represented as a sub-graph and to restructure these sub-graphs. However, its dataset only covers 15 projects and its extracted graph data are limited to the basic graph elements, such as node, edge, the node’s basic property, and the area size of the room. Secondly, in “Towards a machine learning framework in space syntax” (Ferrando et al. 2018), the author proposed a graph-based neural network framework for spatial analysis. In the paper, the extraction of graph data from image-based raw data is discussed, which is a clear and straightforward process.

However, the cleanup and normalization steps for raw rough data are skipped by moving directly to testing and validating the framework.

Thus, gaps clearly existing in the previous related research: the data processing was ignored, and in-depth analyses are lacked, with the reasons including: the dataset is too small, or a proper means of extracting graph data from a design document is missing, or the transformed data input format varies from project to project, without comprehensive understanding of the influence of these different data formats. Therefore, although these studies use machine learning to explore the logics behind the design, their irregular data processing and lack of validation of the data converting process leave their final research results to be partially questioned.

This study tries to fill these gaps by answering the following questions: Will different approaches to data processing affect the final analysis outcomes? If so, how to conduct a rigorous data processing procedure for floor plan layout analysis? Of course, it must be admitted that the literature review can't be claimed as exhaustive and thorough; rather, it sets a starting point on the way of tackling these questions with a well-defined searching scope.

2.3 Why choose the graph representation?

Conventionally, architecture design is represented through two-dimensional drawings, for example, plan drawings, sections and elevations. Although over recent years, most architecture firms and archives have replaced traditional image documents with CAD/BIM files, the image documents still serve as an easy-to-access data storage format to preserve a large number of traditional design documents. On the other hand, no matter what kind of data sources is used, a method of extracting sufficient information

from the sources while preserving enough spatial properties and relationships between spatial components is important. This study will explore the graph theory as one of such approaches, with the graph model as the foundation of the data encoding.

The origin of graph theory can be traced back to the seventeenth century when the problem of “Bridges of Königsberg” was studied (Michael 2011, 445-470) (see Figure 3). Later, through years of development as a mathematical domain, the graph theory itself has evolved into a broad set of fields. However, it had not been used as a method to deal with architectural and geographical problems until the late 1960s (Harary, 1969). After that, the graph theory and graph model have been used as a way of encoding architectural design or geographical design into a topological models suitable for spatial analysis while preserving enough spatial properties, thanks to the help of such tools as convex map, visual fields and isovists (Michael, 2011, 445-470).

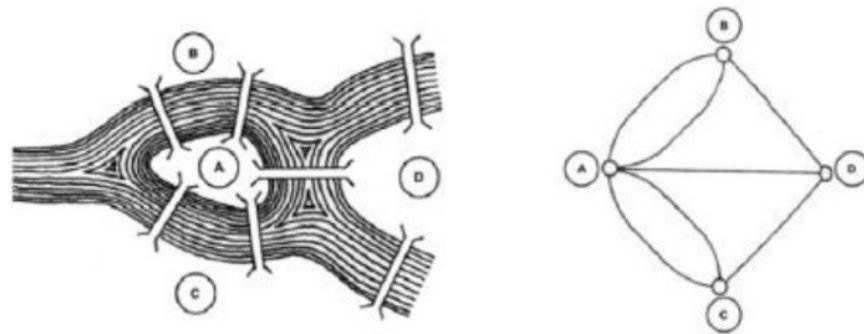


Figure 3. The graph representation of the problem “Bridges of Königsberg”. From Michael 2011, 445-470.

By employing graph theory, the plan layout design can be converted into a formatted dataset, as in the form of n-dimensional vectors. However, the raw graph models consist of values such as relationships between different spatial components which are multi-dimensional data, unlike the rest values such as component sizes, which are one dimensional. So the raw graph models cannot be directly used as standard

input of a machine learning process. Thus, a proper data encoding and converting process are needed. Given that there are many methods to achieve this purpose and different approaches to the process will result in different data formats, it's important to study the differences between these methods and validate their performance through experiments, which is just the focus of this study.

There are some limitations when using graph theory as a means of extracting spatial information into a kind of formatted data, e.g., the shape and the perception of space may be partially lost in the process, or some spatial ambiguities may disappear as a result. However, this method has the potential to serve as a foundation for developing a data processing framework, or even as an injection tool or a database in the future, for the purpose of floor plan layout design analysis using machine learning.

In the previous researches on using machine learning to analyze floor plan layout design and generate new design based on convoluted methods, a clear and obviously overlooked issue is identified: although these researches apply the newly emerging machine learning techniques into design field, they fell short on data extraction and data processing. In the field of machine learning, data processing is the key to the success of models or products. Different methods of data collecting and different formats of data input can result in a huge deviation in the outcomes.

When machine learning is used to analyze architecture design, a clearly defined problem requires a clear dataset for extracting and recording the types, attributes and relationships of various elements in design documents in a uniform format. And in graph theory, the representation of the node, edge and embedded property info in those elements in a graph model is useful, and it is a proven tool to serve as a solution to above problems (see Figure 4 and Figure 5).



Figure 4. An instance in the apartment floor plan layout dataset.

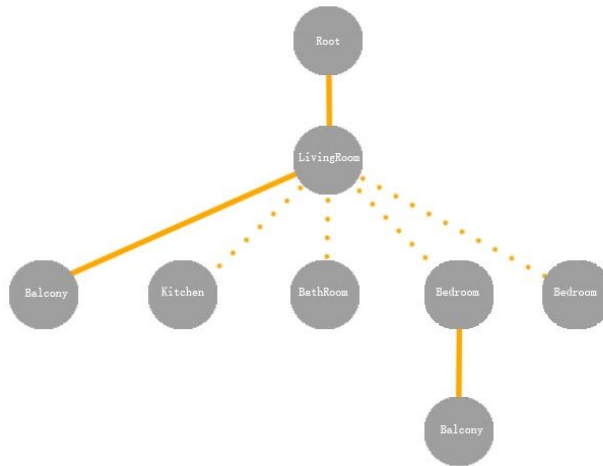


Figure 5. The corresponding graph model of the floor plan layout above.

For the design of floor plans, a general approach is to use machine learning to train a machine intuition to generate and qualify potential design options based on their embedded logics. It is clear that, driven by certain logics of corresponding dependency and independence, the relative placement of design elements such as rooms is settled along with their connections, if it's suitable for the node element to represent the rooms and for the edge element to represent the connections. And aided by additional embedded values,

such as the number of adjacent rooms and the rooms' isovist value, the varying qualities of different rooms can be stored in the graph as well. More specifically, a single room presented in the floor plan layout is defined by the presence of the surrounding rooms and the condition of their connections, along with the degree of openness between them and the position of the opening elements. The analysis on floor plan layout is to qualify the room itself and the adjacencies within the plan, and a graph-based data model is a logical choice for serving as the fundamental layer of information. By using graph-based data, querying the topological relationship in a floor plan can be transformed into querying the values in the graph-based dataset, where the properties of the elements in the plan are mostly preserved.

To get an understanding of the use of graphs in the spatial analysis, a mathematical comprehension of graph theory itself is needed. In a restricted sense, a graph is an ordered pair $G = (V, E)$ (Bender & Williamson. 2010), where V is a set of vertices (also called nodes or points) and E is a set of edges ($E \subseteq \{\{x, y\} \mid (x, y) \in V^2 \wedge x \neq y\}$ (also called links or lines), which consist of unordered pairs of vertices, meaning that every edge is associated with two distinct vertices. To avoid ambiguity, this type of object may be precisely called an undirected simple graph. Under the definition, the vertices x and y in the edge $\{x, y\}$ are called the endpoints of the edge, where the edge is said to join x and y and is called an incident on x and y . And a vertex may exist in a graph and not belong to an edge. Multiple edges are two or more edges that join the same two vertices. In this thesis, an undirected simple graph is chosen as the fundamental layer of information, given the direction of the openings between rooms is not directed.

To query the tangible performance criteria in a graph representation of floor plan design, some definitions are needed. In the graph theory, the relationships and properties of a set of objects can be united in a single graph model (see Figure 6):

- Nodes represent rooms consisting of such attributes as type, area, volume and perimeter.
- Edges represent the topological relationship between rooms: directly linked (doors), partially open (windows), or vertically open (stairs).

All the information in the graph model can be easily stored in a set of n-dimensional vectors, so as to prepare a dataset. Additionally, to preserve as much spatial information as possible, the isovist value and connection value are included in the node's attributes as well. And to encode the target features for training, some numeric values, including the evaluation of privacy, sleep-ability, livability and accessibility, are included in the node's attributes as well.

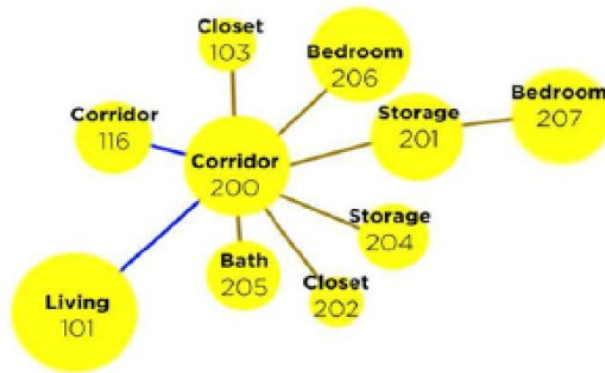


Figure 6. A simple graph model representing an apartment design. From Imdat et al. 2018.

2.4 Graph embedding

Graphs exist naturally in a wide variety of scenarios in real life, such as social graphs in social media networks, citation graphs in research reference areas, network station graphs in network communication, etc. And a rich set of insights can be queried through analyzing these graphs, thus providing researchers with methods to use the information hidden in graphs and raising their significant attention as well. And

many applications, such as node clustering, node classification, node recommendation, edge prediction, etc., can benefit a lot from an effective graph analysis. Considering the corresponding graph-modeled data extracted from the floor plan layout, those applications can be function grouping of rooms, or a room's function recommendation, or a suggestion of a connection's openness degree, or others. Given one of the most important reasons for choosing machine learning as a tool for floor plan layout analysis is to take advantage of the ability to access large scales of datasets, a controllable and predictable method of formatting datasets from raw graph-modeled datasets is needed.

A lot of efforts have been devoted to the graph analytics and converting process, with examples including the distributed graph data processing framework, the new space-efficient graph storage which can accelerate the I/O and computation cost, and so on (Cai et al. 2017). To address the computation and space lost in the existing graph analytics methods, another method called graph embedding has been proposed and it can provide an efficient and effective way of solving this problem. More specifically, graph embedding can convert a high-dimensional and irregular dimensional graph into a low dimensional space, while preserving information in the original graph. Thus, a graph can be represented as a set of low dimensional vectors, thus facilitating machine learning algorithms (see Figure 7).

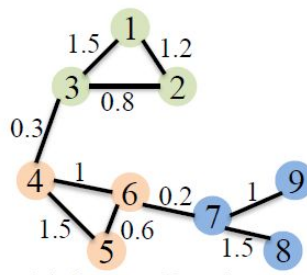


Figure 7. An example of input graph data. From Hongyun et al. 2017.

Embedding graphs into low dimensional spaces is a significant challenge, because it requires clear problem settings, including the embedding input and the embedding output. The former varies depending on scenarios and can be categorized as homogeneous graph, heterogeneous graph, graph with auxiliary information, and graph from non-relational data. The latter can lead to a different output based on different needs (see Figure 8).

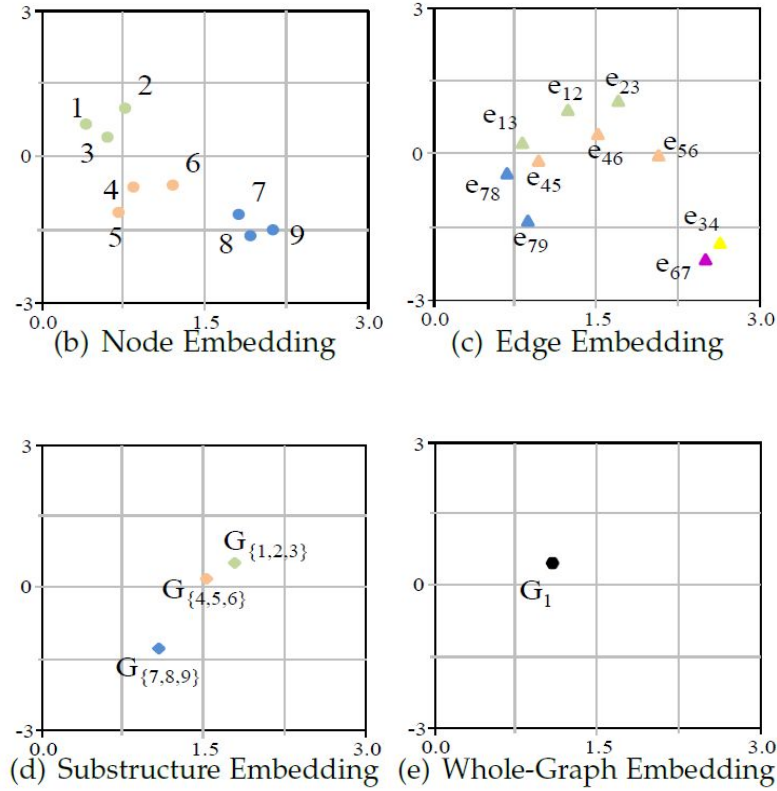


Figure 8. Differences in 4 graph embedding methods. From Hongyun et al. 2017.

Under various embedding inputs, different types of graphs carry different information to be preserved in the embedded space. Thus, the corresponding embedding methods need to be discussed separately. Take a graph with additional attributes added to the node for example: because of such auxiliary information, the graph can preserve information from other perspectives, such as the room's isovist value, to indicate a

part of spatial shape information in the floor plan. And considering a different graph input which only contains the structural information, only the connections between nodes are the target information can be preserved in the embedding process. However, unlike the embedding input which is fixed once extracted, the embedding output is a task-driven process. Take the node embedding for example: as the most common type of embedding output, it represents nodes as vectors and can be used in the node-related tasks, such as node clustering and node classification. In this thesis, in order to preserve as much spatial information from the floor plan layout as possible, the graph with auxiliary information may be the suitable choice.

Chapter 3. First Look at the Dataset

In this chapter, the data processing, optimization and differentiation will be explored. More specifically, the data processing framework proposed in this thesis covers the data extraction, feature engineering, data converting and performance validation, so as to find out what features are suitable to represent the spatial information in the floor plan layout and what data formats perform well in terms of accuracy and time performance in the training process for floor plan layout analysis. The dataset used as a proving material in the experiments and framework demonstration will be the focus. The source of the data, the feature selection, and the spatial analysis task setting, will be discussed separately below.

3.1 Source of data

As discussed in previous chapters, to test the data processing framework on several spatial analysis tasks, the apartment floor plan design as a function-driven design type shall be chosen to avoid the uncertainty of spatial ambiguity of artistic perception. The dataset to be used contains 100 instances, which are images of apartment floor plan layout design collected from an apartment rental website (<http://www.ziroom.com>) (see Figure 9). First of all, the capacity of this dataset is not big, but enough to serve as a proof-of-concept demonstration of the framework. And to eliminate as much bias as possible against a certain apartment floor plan type, the dataset covers the all apartment types, ranging from “one bedroom and one bathroom” to “four bedrooms and two bathrooms”. Secondly, although nowadays most design firms and archives have CAD/BIM files to document their designs in 2D or 3D, conventional image-based documents are usually easier to access, with the advantages of holding a huge number of architectural designs generated in history. Moreover, thanks to the uniform representation methods, the drawing styles of the images from the website are consistent, leaving the extracting process easier to

tweak, so that attention can be focused on the data encoding and validation.



Figure 9. An example of floor plan layout image in dataset.

A detailed look at the dataset can observe that it is stored in two forms: the original images of the floor plan layout design, and the CSV files to store the graph data extracted. Within the CSV files, each line of data can be viewed as a set of rooms' property features and target features for spatial analysis tasks, with such values as room number, connection information, area size, room's function and isovist value serving as rooms' property features. Rooms' privacy level, accessibility level, sleep-ability level and livability serve as the target features for spatial analysis tasks (see Figure 10).

name	connection	area_size	function	privacy_level	isovist_value	accessibility_level	sleepability	liveability
1	(2_0;3_0;4_0;5_0;6_2)	24.98	Livingroom	0	28.44	1	0	1
2	(1_0)	7.81	Kitchen	0.2	9.31	0.5	0.2	0.4

3	(1_0)	3.84	Bathroom	0.3	14.19	0.4	0.2	0.3
4	(1_0)	13.4	Bedroom	0.6	19.72	0.5	0.8	0.5
5	(1_0;7_2)	13.89	Bedroom	0.6	19.48	0.5	0.8	0.5
6	(1_2)	3.04	Balcony	0.1	24.95	1	0	1
7	(5_2)	3.26	Balcony	1	21.5	0.2	0.2	0.4

Figure 10. A raw graph data stored in a csv file.

3.2 Feature Engineering

Rooms' property features will be extracted to preserve the topological relationship of rooms and the spatial information from the original floor plan layout images. By using graph theory and spatial analysis, such as isovist analysis, these features are chosen and extracted. Although some of the spatial information may be lost in the process, most of it will be stored in the selected features.

The rooms' sizes are extracted to preserve the spatial properties of the rooms, because the area sizes can reflect the function of the rooms to some extent. For example, a living room tends to have a larger size than bedrooms or bathrooms. Also, the function is recorded as both a property value of rooms and a spatial analysis task to predict. Furthermore, to encode the topological relationship of rooms and store the structure in a graph model, the connection value is used as a special value for this purpose. In the graph model, the node and edge are two key elements to represent the structure of the relationship network, as nodes represent the rooms and edges represent the connections between rooms. In addition to the number of rooms used to separate nodes, the type of edges represents the openness of connections. For example, the type '0' means the rooms are connected by doors that can close the connection completely; the type "2" means the communication between the rooms is free and people can move without interruption; and

the type “1” means the connection between two rooms is in a semi-open state, such as an indoor transparent partition. More specifically, the connection value “(2_2;3_0;4_1)” in line “1” means that in the graph model, the room “1” is connected to the room “2” in an open state encoded as “2”, to the room “3” by a door encoded as “0”, and to the room “4” by a semi-open structure encoded as “1”.

However, if preserving the topological relationship as a graph model and recording the rooms’ area sizes alone, massive spatial shape information may be lost. Thus, a means of preserving spatial shape information and the spatial relationship is needed. As pointed out in analyses of spatial properties from 2D plans, several techniques are used to analyze the spatial visual properties of elements in plans. Convex map and axial maps are two common methods to use in graph theory when preserving spatial and shape information. In addition to these two methods, isovist analysis is a more advanced approach, as it can represent the relative spatial-visual properties in the whole environment by constructing an isovist field through polygons encoding all visually accessible regions with the central point of the current spatial element as the origin. Some previous researches involve isovist analysis. E.g., “Prospect-Refuge theory and the textile-block houses of Frank Lloyd Wright: an analysis of spatial-visual characteristics using isovists” (Dawes & Ostwald. 2014) addresses the isovist area, radial length and radial line as possible isovist properties; and “Graph-based models of space in architecture and cognitive science -- a comparative analysis” (Franz et al. 2017) discusses different spatial-visual analytics methods along with models of spatial cognition. Thus, an additional feature called isovist value is recorded in the dataset to store the size of each room’s isovist field, which preserves the spatial characteristics from the perspective of the room's spatial relationship. And the benefit of adding this isovist feature can be easily observed by considering the following three instances. All the three instances are living rooms and have a similar topological relationship in the graph, i.e., they all connect to the adjacent kitchen, bathroom, and bedroom in a similar manner and they all have similar area sizes, but their shapes and actual spatial characteristics

have some subtle differences. By considering their different isovist field sizes, their spatial differences can be observed (see Figure 11, Figure 12, and Figure 13).



Figure 11. Isovist field of a living room example with 22.51m^2 area size and 37.04m^2 isovist value.

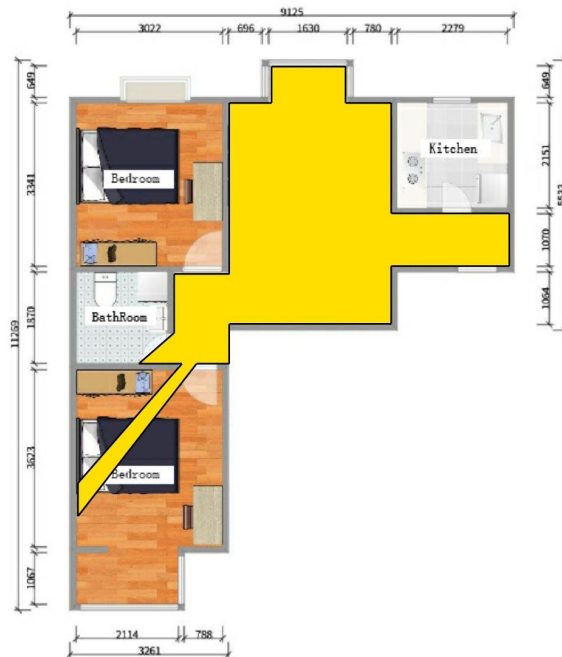


Figure 12. Isovist field of a living room example with 20m^2 area size and 21.4m^2 isovist value.

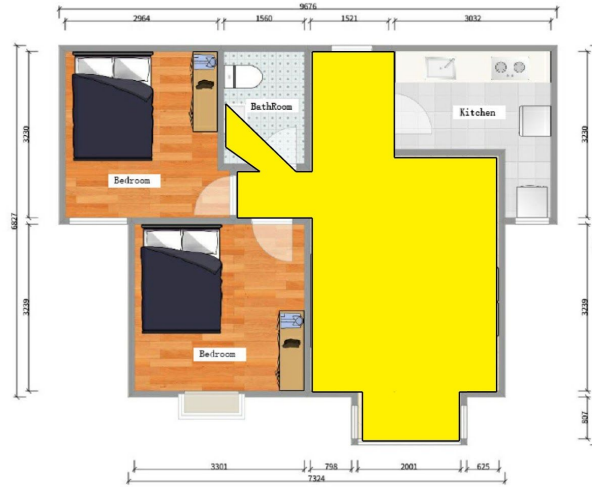


Figure 13. Isovist field of a living room example with 23m^2 area size and 24.95m^2 isovist value.

3.3 Floor plan analysis task setting

As mentioned in the previous chapter, a clear setting of problems is of great significance. Therefore, a clear set of features is added to the dataset to serve as the target value for the spatial analysis tasks. Given the apartment floor plan chosen as the source, some measuring values of space's quality can be used as the target values for training. More specifically, rooms' function is the target non-numeric value for predicting rooms' use; rooms' privacy value is the target quantitative value for measuring the private or public degree; and accessibility is the target quantitative value for measuring the easy-to-access degree of a space. And sleep-ability is the target quantitative value for measuring how comfortable the sleeping related activities are in a room, while livability is the target quantitative value for measuring how comfortable the social activities are. Except that the first feature above is a non-numeric value, the rest four features are all numeric values on a scale of 0-1, which may influence the selection of machine

learning algorithms in experiments (see Figure 14).

name	connection	area_size	function	privacy_level	isovist_value	accessibility_level	sleepability	liveability
1	(2_0;3_0;4_0;5_0;6_2)	24.98	Livingroom	0	28.44	1	0	1
2	(1_0)	7.81	Kitchen	0.2	9.31	0.5	0.2	0.4
3	(1_0)	3.84	Bathroom	0.3	14.19	0.4	0.2	0.3
4	(1_0)	13.4	Bedroom	0.6	19.72	0.5	0.8	0.5
5	(1_0;7_2)	13.89	Bedroom	0.6	19.48	0.5	0.8	0.5
6	(1_2)	3.04	Balcony	0.1	24.95	1	0	1
7	(5_2)	3.26	Balcony	1	21.5	0.2	0.2	0.4

Figure 14. A raw graph data with spatial analysis task features.

Chapter 4. Methods

This chapter will explain the methods adopted to extract spatial properties from the floor plans, including the graph model and the isovist analysis, and will also explore the measures of validating the spatial qualities of the floor plan design. Further, it will describe and discuss the method to merge the non-vector data in the spatial properties into vector data, because the latter is more suitable for machine learning algorithms. The objective of the experiment is to demonstrate the data extraction process and the potentials of merging non-vector data into standardized vector data suitable for the validation of the performance and accuracy of the training and predicting process. And the guidelines of the steps are shown as follows:

- Formalize the graph model representation of the floor plan based on the example case.
- Document the area property of the room.
- Draw the spatial property of the room, such as isovist value and function.
- Define the method of marking target training parameters based on the spatial analysis tasks.
- Define the mathematical formula of merging non-vector data into vector data.
- Define the proper way of extracting derived data from the raw spatial property data.
- Verify the detection of the differences in target training results after using different data formats.

Based on the research topic, this chapter is divided into two parts. The first part is about graph data encoding and the methods for encoding spatial property features and target training features. And the second part will focus on the methods for expanding original raw graph data features into independent features in the graph data conversion process.

4.1 Graph data encoding

From the same raw graph data, various data representation forms can be extracted, whether they are centered on a node or an edge. The different data representation forms can all contribute to the data input in the machine learning analysis process. A floorplan can be represented as a list of vectors, where each vector contains information about an object within the plan (see Figure 15, Figure 16, and Figure 17). Such a kind of representation can reduce the dimensionality of the object recognition problem. Rather than learning object characteristics through hundreds of pixels, a model can learn object characteristics through the information contained in vectors, such as size, end-point coordinates, type of object, etc. This representation method is promising, but contains further challenges when considering the variable length of each floor plan and how the model will need to learn the positional relationships between all objects, as such relationships may not be explicitly represented through a list of vectors. Therefore, using graphs to encode structural information of floorplans is the first step of data processing. And then using graph embedding to convert graph data into standardized vector data is the second.



Figure 15. An instance of the apartment floor plan design layout.

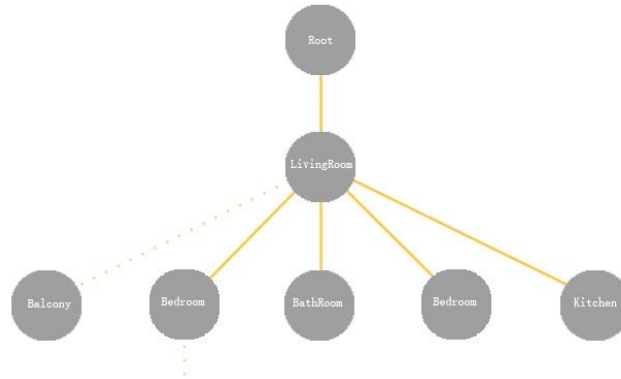


Figure 16. The graph model of the floor plan layout above.

name	connection	area_size	function	privacy_level	isovist_value	accessibility_level	sleepability	liveability
1	(2_0;3_0;4_0;5_0;6_2)	24.98	Livingroom	0	28.44	1	0	1
2	(1_0)	7.81	Kitchen	0.2	9.31	0.5	0.2	0.4
3	(1_0)	3.84	Bathroom	0.3	14.19	0.4	0.2	0.3
4	(1_0)	13.4	Bedroom	0.6	19.72	0.5	0.8	0.5
5	(1_0;7_2)	13.89	Bedroom	0.6	19.48	0.5	0.8	0.5
6	(1_2)	3.04	Balcony	0.1	24.95	1	0	1
7	(5_2)	3.26	Balcony	1	21.5	0.2	0.2	0.4

Figure 17. A raw graph data extracted from the floor plan layout above.

Several aspects need to be discussed as to the raw graph representation data of the example floor plan. First of all, for each room in the plan, its function and its size are documented as a non-numeric and a numeric value, respectively. Secondly, the topological relationship between a room and its adjacent rooms are stored as a specialized value -- “connection”, in which the relationship is represented as a pair of the adjacent room and the relationship between them, i.e., “adjacent room_adjacency type”. To be clearer, the adjacency value “0” stands for a door connection, the value “2” stands for a totally transparent connection, and value “1” stands for semi-transparent connection. Obviously, these connection data cannot be directly used as standard vector data, since the dimension of them depends on the number of adjacent rooms. As a result, the dimension is uncertain, and the data cannot be simply converted into vector data. In the next chapter, the conversion of the weight value, either focusing on edge or path, will be elaborated.

On the other hand, in order to validate the difference in applying different data formats to the machine learning process of spatial analysis, several additional target features are included in the data gathering process. The tasks of floor plan layout analysis chosen for the experiments include predicting the rooms’ function, privacy, accessibility, sleep-ability and livability. And these values will represent two types of spatial analysis separately by using machine learning: the usage of the room and the spatial property of the room. Among the data gathered, “function”, “privacy_value”, “accessibility_level”, “sleep-ability” and “livability” are the corresponding features mentioned before. It's worth noting that the latter four features are collected by a survey of 20 people, including architects and architecture students, who are asked about their professional opinions on the privacy, accessibility, sleep-ability and livability of each room. The final value of each feature is the average of all the values collected.

4.2 Expanding features

Before starting the graph data conversion process, it can be observed that the property features in the original raw graph data cover the information that is too dense and the lines of data have no relation to each other; as a result, these features cannot completely cover all the information that formed the original floor plan layout design. Therefore, some methods for expanding those features are needed (see Figure 18 to Figure 24).

First of all, the features “function” and “isovist” do not need to be expanded and can be directly inherited. As for the feature “connection”, it encodes the entire graph structure and the topological relationship of rooms in the plan; thus, it serves as a high-dimensional vector value. For any node in a graph, it has some connected nodes that have different types of connections to it. Therefore, two features representing the connected nodes and the type of these connections can be extracted, where a new feature “num_of_neighbors” represents the number of all connected nodes and another new feature “edge_value” represents the weighted sum of all connections. And the feature “area_size” cannot be simply inherited. A way of preserving this information is to add a new feature representing the proportion of a single room’s area size to the entire apartment’s area size, i.e., the feature of “room_area_to_total”. Moreover, as the graph is an abstraction of the topological relationship of nodes and connections, a better way of preserving the architectural meanings of linking spatial sections, a method in graph analysis called degree centrality can be used as a new feature. By its definition, the degree centrality represents how many connections a node has with other nodes in the graph, which is computed as:

$$\text{degree_centrality} = \text{Number of connections (v)} / (|V| - 1)$$

where v stands for the current node, V stands for the set of all nodes, and |V| stands for the number of all nodes in the graph.

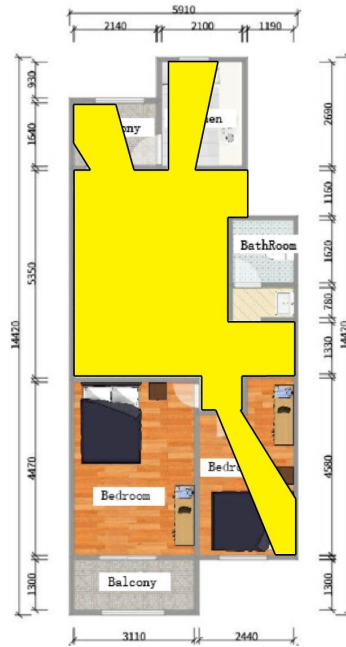


Figure 18. Node of living room: area_size - 20, isovist_value - 33.48, connection - (2_0;4_0;5_0;6_2;7_2)



Figure 19. Node of bedroom 1: area_size - 14, isovist_value - 20.04, connection - (1_0;3_1)



Figure 20. Node of balcony 1: area_size - 4, isovist_value - 17.24, connection - (2_1)



Figure 21. Node of bathroom: area_size - 4, isovist_value - 10.99, connection - (1_0)

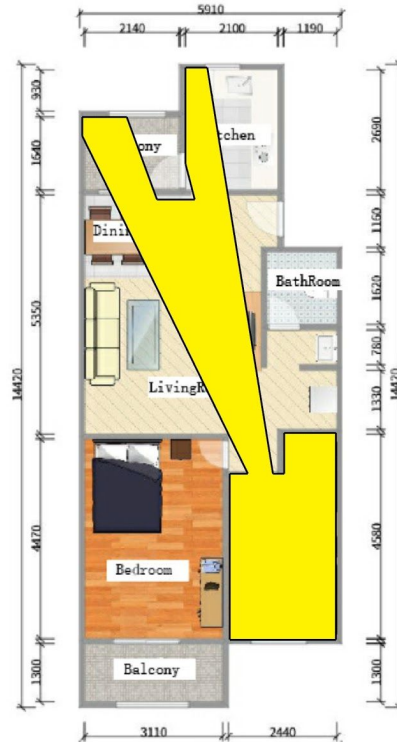


Figure 22. Node of bedroom 2: area_size - 10, isovist_value - 23.95, connection - (1_0)

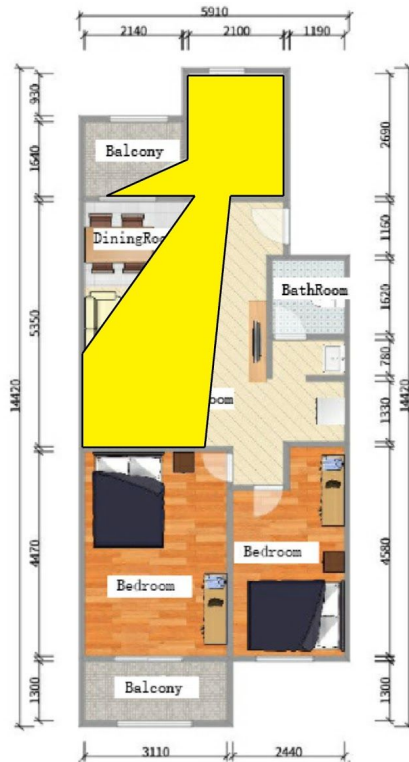


Figure 23. Node of kitchen: area_size - 5, isovist_value - 18.41, connection - (1_2;8_0)

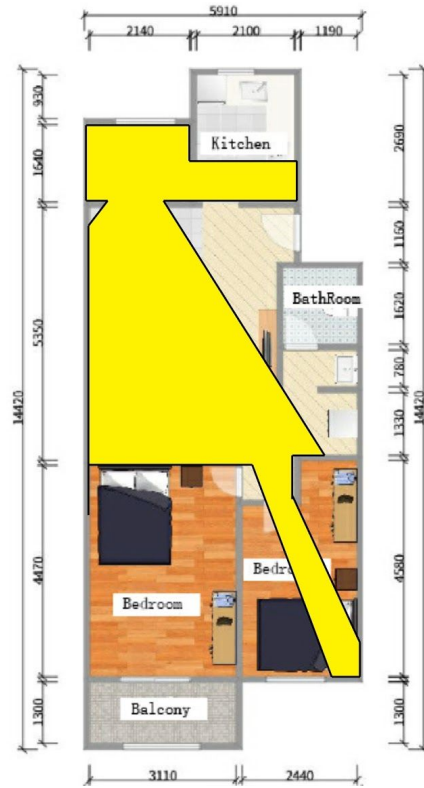


Figure 24. Node of balcony 2: area_size - 3, isovist_value - 27.57, connection - (7_0)

Chapter 5. Extracting Input Graph Data from Raw Image

In this chapter, a graph-based data processing framework will be introduced. And in the second and third parts of this chapter, two important steps in the framework, i.e., the graph model extraction from raw image and the graph data conversion process, will be discussed and demonstrated with examples. The processed dataset will be used as an input in the next chapter to validate the choice of features in the graph data, the influence of different data encoding formats, and the choice of feature settings for floor plan layout analysis tasks.

5.1 Graph data processing framework

Technically, this graph-based data processing framework is to build a framework capable of extracting spatial information from floor plan layout design and using the graph theory and spatial analysis to choose features to add to the extracted graph-based data (see Figure 25). Then, certain methods in the graph-embedding theory are used to convert and encode the raw graph data into several standardized vector data suitable for serving as input in the machine learning process. And in the final stage of constructing the framework, a proper validation process will be used to verify the impact of data in different encoding formats on floor plan layout analysis tasks as well as their adaptability to different machine learning models. The input of the framework is the raw images of the floor plan layout design, and the output is different converted graph-based data along with their performance analysis.

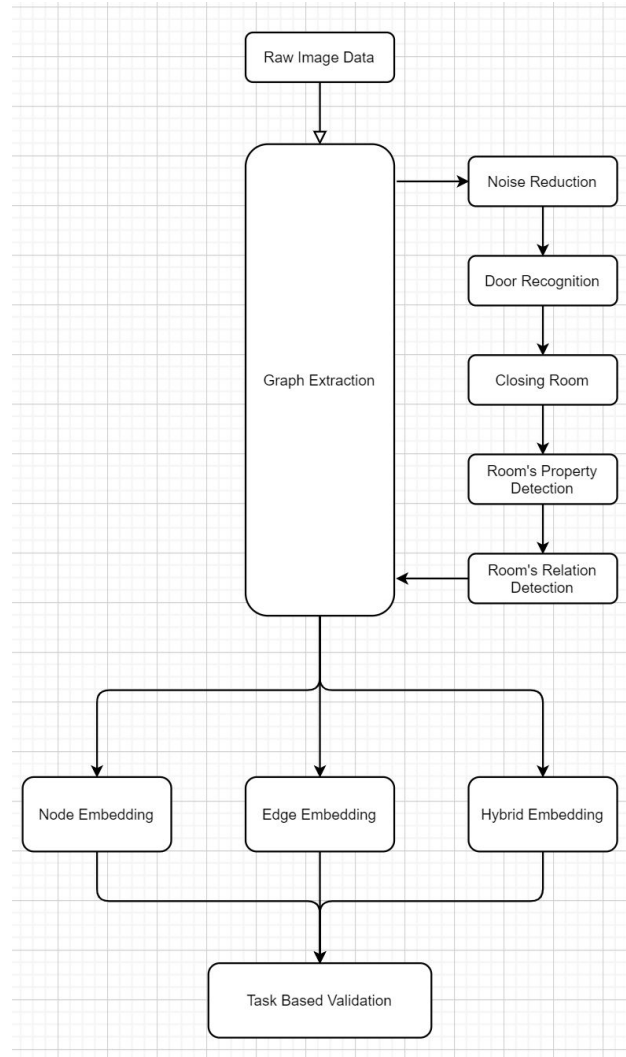


Figure 25. The UML representation of the graph data processing framework.

For the input of the framework, the image dataset is an apartment floor plan layout design, and its function-driven design and evaluation nature defines the floor plan layout analysis tasks, which in turn decide the five target features for training, i.e., the prediction of rooms' function, privacy level, accessibility level, sleep-ability level and livability level. In order to avoid any mutual interference, all the five target features are separated in the final output dataset, so as to test the independent performance of all data formats.

For the second part of the framework, the extraction of graph-based data from the raw images will be discussed step by step. And this part will focus specifically on graph model extraction, which would preserve the topological relationships of the rooms in the floor plan. More specifically, this extraction process can be divided into five steps: noise reduction, door recognition, room closing, room's property detection, and room's relationship detection. And the additional spatial property feature extraction will also be demonstrated in the latter part.

The third part about the framework will discuss in detail the application of a subtopic in the machine learning field, called graph embedding. This method allows the projection and conversion of raw graph data into uniformed and standardized graph data. And this method can be divided into three approaches: node embedding, edge embedding, and hybrid embedding. In the converting process, the original property features, such as the feature "connections" which stores the graph structures, will be expanded into new independent features, in order to preserve the global information and the potential relationships between spatial elements.

5.2 Graph data extraction process

For the second part of the framework, the extraction of graph-based data is its focus and achieved in OpenCV script coded in Python 3.6. The process can be divided into five steps. In the first step, the input is the raw floor plan layout images (see Figure 25). In any image, a lot of noise elements exist, such as text labels, furniture elements, ground texture, etc, which are of no use for constructing the topological relationship of rooms in the floor plan (see Figure 26). The elements needed are walls, doors, windows, and no-color backgrounds. Therefore, a noise reduction process is applied to remove those noise elements to generate a black-white image. In the second step, the input is the black-white image, in which the

rooms are neither recognized nor closed properly. To close the rooms properly, the type of the intervals on walls is important, as the regular doors would be much smaller than the double-door glass partitions between a living room and a balcony. By tweaking the threshold for detecting the intervals, the type of the intervals can be categorized and used to close the rooms' borders (see Figure 27). In the next step, the connections of rooms are recorded as edges linking nodes in the graph. Based on the completed intervals in the previous step, a vertical reference line is used to detect which rooms this interval is connecting and add into the graph model those two related nodes' connections and a piece of edge information whose type is defined by the interval's type (see Figure 28). In the final stage, the property of the nodes is added. Relying on the closed space sections in the previous step, the area size of each room can be calculated and added as one of the node's spatial properties (see Figure 29).



Figure 26. An instance of the floor layout image.

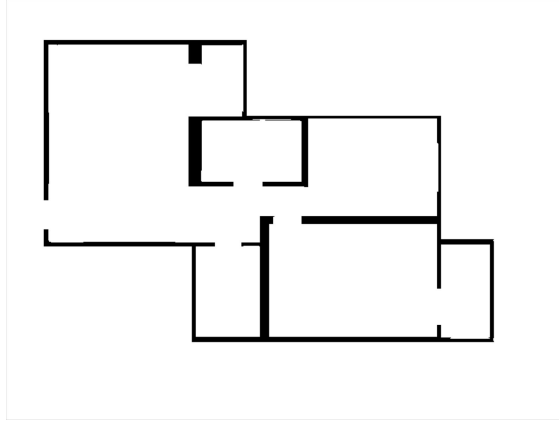


Figure 27. Processed image after noise reduction.

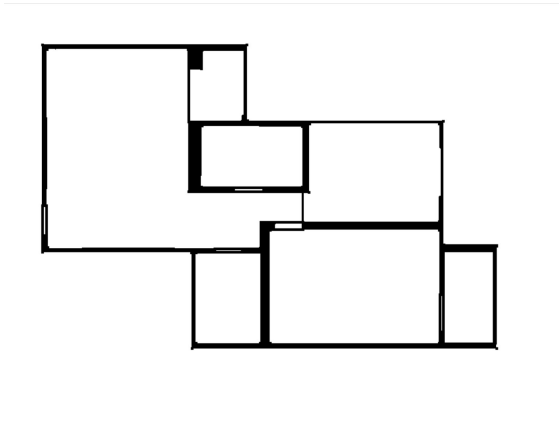


Figure 28. Processed image after door detection and section closing.

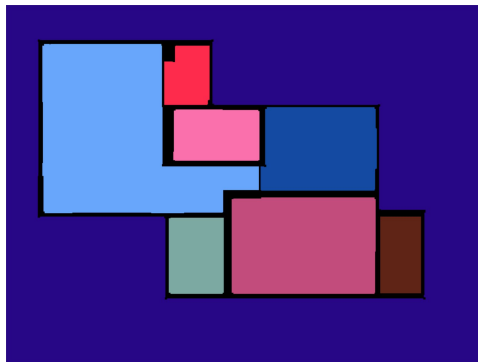


Figure 29. Processed image after room detection and tagging.

Several places are needed to be clarified and noted about this extraction process. First of all, the purpose of this extraction process is to demonstrate a general approach of extracting graph-based data from 2D images. Thus, the five steps discussed above have some limits, including the fact that the thresholds of determining the distances between doors or partitions are specially adjusted according to the pixel size of images in this dataset. Outside this dataset, several other floor plan images in different visual styles, including hotels, hospitals and schools, are tested. The results show that after adjustment to the thresholds, the extraction process still works. However, it's worth noting that the images with too many details, such as furniture and text markings, usually perform not so well. Secondly, the process is not entirely automatic. The five main steps are automated in OpenCV codes, with occasional manual correction of room connections' types, which happens mostly when rooms are connected by translucent open partitions. And the extraction of rooms' spatial properties, such as isovist field size, is manually operated. Finally, the running time of this script is usually a few seconds, depending on the complexity of the space partition in the floor plan design. In conclusion, this extraction process is not robust enough to handle all design images and not entirely automated. But it shows a general picture of how an extraction process should be conducted and it prepares a raw graph dataset for later use.

5.3 Graph embedding process

The third part of the framework is to convert and encode the raw graph data by using the graph embedding methods. As discussed in the previous chapter, the input type is a simple undirected graph with auxiliary information. In the raw graph data, the graph structure information is stored in the feature “connection” and the serial number of each line, while the auxiliary information referencing to the spatial property is stored in the features “area_size” and “isovist_value” (see Figure 30). During the embedding

process, in order to preserve the global information in graphs and the relationships between nodes' properties, features like "connections" and "area_size" will be expanded into several new features, depending on the specific embedding methods. And considering the different graph embedding methods, the three most common methods are applied in this framework, which are node embedding, edge embedding, and hybrid embedding.

name	connection	area_size	function	privacy_level	isovist_value	accessibility_level	sleepability	liveability
1	(2_0;3_0;4_0;5_0;6_2)	24.98	Livingroom	0	28.44	1	0	1
2	(1_0)	7.81	Kitchen	0.2	9.31	0.5	0.2	0.4
3	(1_0)	3.84	Bathroom	0.3	14.19	0.4	0.2	0.3
4	(1_0)	13.4	Bedroom	0.6	19.72	0.5	0.8	0.5
5	(1_0;7_2)	13.89	Bedroom	0.6	19.48	0.5	0.8	0.5
6	(1_2)	3.04	Balcony	0.1	24.95	1	0	1
7	(5_2)	3.26	Balcony	1	21.5	0.2	0.2	0.4

Figure 30. A raw graph data extracted from the floor plan layout.

The method of node embedding is one of the most common embedding output settings where the converting focus is to preserve nodes' information as much as possible and each node is represented as a vector in a low dimensional space. Centered around the node itself, each line of the converted data stores the nodes' original low-dimensional features, such as "area_size", and the newly expanded features, such as "room_area_to_total" and "degree_centrality", and is serialized by the original line number. Thus, the number of lines in the converted dataset will not change, except for more features being stored in each line (see Figure 31).

name	edge_value	area_size	isovist_value	room_area_total	num_of_neighbors	degree centrality	function	privacy_level	accessibility	sleepability	livability
1	7	24.98	28.44	0.36	5	0.83	Living room	0	1	0	1
2	1	7.81	9.31	0.11	1	0.17	Kitchen	0.2	0.5	0.2	0.4
3	1	3.84	14.18	0.05	1	0.17	Bathroom	0.3	0.4	0.2	0.3
4	1	13.4	19.72	0.19	1	0.17	Bedroom	0.6	0.5	0.8	0.5
5	1	13.89	19.48	0.2	2	0.33	Bedroom	0.6	0.5	0.8	0.5
6	4	3.04	24.95	0.04	1	0.17	Balcony	0.1	1	0	1
7	3	3.26	21.5	0.05	1	0.17	Balcony	1	0.2	0.2	0.4

Figure 31. Converted data using node embedding.

In contrast to the method of node embedding, the method of edge embedding aims to represent an edge as a low-dimensional vector. During the converting process, the spatial property features from the original raw graph data are engineered and some are expanded to new features. Unlike in the method of node embedding, all these features are used to sum nodes features on both ends of the current edge. For example, the feature “edge_area_weight” is the combination of two nodes’ original features of “area_size” to represent the space size this edge connected. So is the same with the feature

“edge_connection_weight”, which represents the number of adjacent rooms this edge has linked in the floor plan. And for the output of the edge embedded data, the lines of data usually increase as the number of connections between rooms is often larger than the number of rooms themselves (see Figure 32).

edge_area_weight	edge_node1	edge_node2	edge_isovist_weight	edge_connection_weight	edge_area_to_total	edge_num_of_neighbors	edge_degree_centrality	node1_function	node1_privacy	node1_accessibility	node1_sleepability	node1_livability
28.82	1	3	42.62	6	0.41	6	1	Living room	0	1	0	1
32.79	1	2	37.75	6	0.47	6	1	Living room	0	1	0	1
38.87	1	5	47.92	7	0.56	7	1.16	Living room	0	1	0	1
28.02	1	6	53.39	6	0.4	6	1	Living room	0	1	0	1
38.38	1	4	48.16	6	0.55	6	1	Living room	0	1	0	1

Figure 32. First 5 lines of converted data using edge embedding.

For the method of hybrid embedding, by definition, it’s the embedding of a combination of different types of graph components, e.g., node + edge (i.e., substructure), or node + community. Here in this framework, given the functional layout is generally related to the relationships between paths and sight occlusions, the substructure, i.e., node + path, is chosen as the form of hybrid embedding. More specifically, the focus of this method is to encode features along the chain of connected nodes called “path”. Similar to the method

of edge embedding, all the original low dimensional features and the converted features are added to the path in sum. The difference is that the length of the path can be tweaked depending on how far the relationship chain goes. In this demonstration, because the dataset is about an apartment floor plan, the length of the path is set to three. Furthermore, a specific algorithm called Random Walk is used for calculation. In this algorithm, the operation starts at a node in the graph, randomly chooses an edge, and then traverses the graph until a sufficiently long path is produced (see Figure 33). Achieved by breadth-first search (BFS) or depth-first search (DFS), this method turns a multi-dimensional irregular graph into a simple matrix which includes a list of fixed length paths. In the output of the method of hybrid embedding, the number of lines and the number of features in each path will increase significantly (see (Figure 34).

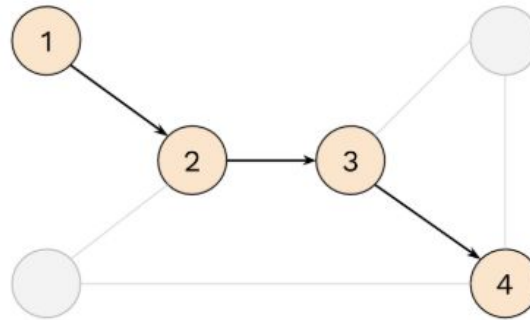


Figure 33. A demonstration of the random walk algorithm. From Hongyun el al. 2017

path_ area_ weigh t	path_i sovist _weig ht	path_ area_r o_tota l	path_ num_ of_ne ighbo rs	path_ degre e_cen trality	start_ functi on	start_ privac y	start_ acces sibility	start_ sleepa bility	start_l ivabil ity	node1	node2	node3
42.13	69.42	0.61	8	1.33	Livin groo	0	1	0	1	1	5	7

					m							
36.63	51.93	0.52	7	1.17	Kitchen	0.2	0.5	0.2	0.4	2	1	3
16.68	57.23	0.67	8	1.33	Kitchen	0.2	0.5	0.2	0.4	2	1	5
35.83	62.7	0.51	7	1.17	Kitchen	0.2	0.5	0.2	0.4	2	1	6
46.19	57.47	0.66	7	1.17	Kitchen	0.2	0.5	0.2	0.4	2	1	4

Figure 34. First 5 lines of converted data using hybrid embedding.

Chapter 6. Validation Experiments

This chapter will discuss the validation experiments in the framework, which are divided into two parts, including the experiment on the influence of selected features and the experiment on different data formats. In the first part, the experiment will focus on splitting the dataset into several sub-datasets, each of which has one spatial feature missing, aiming to test the performance contribution of each feature. In the second part, the experiment is to use different formatted datasets as inputs to test the performance of spatial analysis tasks in several different machine learning algorithms, aiming to validate the influence of different dataset encoding methods. All the experiments are applied to the entire dataset and documented in detail. And the takeaway and conclusion from the experiments are discussed at the end of each section.

6.1 Experiment on feature selection

This part is aiming to test the influence of each spatial property in the dataset, including the features of “edge_value”, “area_size”, “isovist_value”, “room_area_to_total”, “num_of_neighbors”, and “degree_centrality”. The method of testing is to split the original dataset into several sub-datasets and test their performance in the same floor plan layout analysis task. To avoid any errors, all the experiments use the same machine learning algorithm, i.e., the logistic regression, and the same predicting task, i.e., to predict a room’s function (see Figure 35).

	Original	No area_size	No degree_centrality	No edge_value	No isovist_value	No num_of_neighbors	No room_area_to_total
Room's function predicting	88.6 %	77.2 %	79.8 %	78.9 %	76.3 %	77.2 %	77.1%

Figure 35. Different Accuracy performance on predicting room's function

Some interesting observations can be made from the results above. First of all, all the performance results of the sub-datasets show that they all fall behind the original complete dataset, indicating that all the six spatial features are necessary for improving the accuracy performance. This shows that no feature extracted in the framework is useless. Secondly, the sub-dataset with the feature “isovist_value” removed scores the lowest in performance. Considering the isovist field preserves the spatial visual information of the space, as discussed in the chapter before, this finding indicates that this feature contributes the most in the original dataset. Thirdly, the results from the sub-dataset without the feature “degree_centrality” seem to indicate that this feature contributes the least in terms of accuracy performance. A possible explanation is that the degree of centrality has, to some degree, a similar effect to the number of connections, as they all try to store the information about the relative relationship between nodes and their adjacent nodes.

6.2 Experiment on data formats

This part of the experiments is to validate the influence of different converting processes and data encoding formats on the spatial analysis tasks. To avoid errors and reliance on certain machine learning models, each analysis task is tested against the five common machine learning algorithms from different categories. And the floor plan layout analysis tasks are defined as in the previous chapter, where the task

features are discussed, including the prediction of room's function, privacy level, accessibility level, sleep-ability level, and livability level. Because the first task's feature is a non-numeric value and the other four are numeric values, the chosen machine learning algorithms are different. More specifically, logistic regression, Naive Bayes, oneR, SMO and Random Forest are used in the task of predicting the room's function. And linear regression, Kstar, decision table, SMOreg and Random Forest are used in the left four tasks. And the experiments are conducted on a platform called Weka 3.8, with all the training processes going through 10-fold cross-validation (see Figure 36 to Figure 40).

	Logistic regression	Naive bayes	OneR	SMO	Random forest
Node embedded	77.2 %	87.7 %	74.6 %	75.4 %	88.6 %
Edge embedded	80.9 %	77.2 %	76.2 %	66.1 %	84.1 %
Hybrid embedded	70.9 %	79.0 %	77.4 %	66.8 %	91.6 %

Figure 36. Different Accuracy performance on predicting room's function

	Linear regression	Kstar	Decision table	SMOreg	Random forest
Node embedded	36.8 %	64.3 %	53.6 %	36.8 %	63.0 %
Edge embedded	44.7 %	46.2 %	74.8 %	52.5 %	81.4 %
Hybrid embedded	30.2 %	55.3 %	78.6 %	32.6 %	68.6 %

Figure 37. Different Accuracy performance on predicting room's privacy level

	Linear regression	Kstar	Decision table	SMOreg	Random forest
Node embedded	47.1 %	53.2 %	47.9 %	50.0 %	57.7 %
Edge embedded	55.7 %	30.2 %	74.0 %	56.2 %	79.5 %
Hybrid embedded	19.0 %	25.1 %	68.3 %	21.6 %	58.2 %

Figure 38. Different Accuracy performance on predicting room's accessibility level

	Linear regression	Kstar	Decision table	SMOreg	Random forest
Node embedded	43.1 %	81.6 %	69.2 %	43.2 %	77.8 %
Edge embedded	34.7 %	33.4 %	66.4 %	43.1 %	75.3 %
Hybrid embedded	23.2 %	55.8 %	77.2 %	24.3 %	69.8 %

Figure 39. Different Accuracy performance on predicting room's sleep-ability

	Linear regression	Kstar	Decision table	SMOreg	Random forest
Node embedded	57.0 %	68.3 %	60.2 %	58.5 %	70.7 %
Edge embedded	49.9 %	38.5 %	81.5 %	50.4 %	85.2 %
Hybrid embedded	13.6 %	25.9 %	55.4 %	14.9 %	56.8 %

Figure 40. Different Accuracy performance on predicting room's livability level

Examination of the performance data gathered in the experiments shows that several findings and considerations can be discussed. The experiment is divided into five tasks, which separately represent five space quality evaluation targets in the spatial analysis, i.e., the room's function, privacy level, accessibility level, sleep-ability level, and livability level. Among these tasks, the target value for the first task is a non-numeric value, while the latter four target values are numeric. For the first task, two statistical machine learning algorithms are used, i.e., random forest and logistic regression, because they represent two distinct basic models of machine learning algorithms, namely the traditional bifurcation model based on logical judgment and the neural model based on the convolutional network.

From a general view of all the five experiments, it can be observed that the dataset using the node embedded method can generally get a better accuracy performance. The edge embedded method comes to the second place, and in the combination of certain tasks and algorithms, this method can outperform the node embedded method. And the hybrid embedding method comes to the third in most cases. More specifically, taking the first task of predicting rooms' function for example, the node embedded method

scores the highest accuracy in Random Forest, SMO, and Naive Bayes, and it also comes very close to the best in the other two algorithms. And the edge embedded method comes as the second, indicating that the node embedded method is suitable for analyses related to single rooms' property, such as functions. For the second task, i.e., to predict the privacy level of a room, it can be observed that the edge embedded method scores the best accuracy and the node embedded method comes the second, indicating that the edge embedded method is suitable for analyses related to rooms' spatial property, such as the privacy level. The reason is that in order to analyze it, the topological relationship of the adjacent rooms and actual adjacency area values must be put into consideration. However, an interesting observation is that although the node embedded method comes second in terms of accuracy, it actually gets the highest score in terms of time performance. The main reason for this is that after the conversion process, the data capacity grows rapidly in the edge embedded method and the hybrid embedded method, especially in the latter. And this reason can also explain why the latter runs slowly in the testing. Thus, when implementing a machine learning system that requires high speeds and quick response, the hybrid embedded method may not be proper to take into consideration.

Overall, the node embedded method seems a better choice for striking a balance between accuracy and time performance. And the edge embedded method mostly scores the second, while the hybrid embedded method comes the last. However, some limitations must be considered. In the data extraction process, many of the spatial features selected are related to a single space's spatial-visual characteristics, meaning that the information related to the fluidity of space or the space quality about people flow is overlooked. But this information is more likely to be perceived while people are moving along a certain path in space. Therefore, although the hybrid embedded method fails to perform well on the selected tasks, it still has the potential for other tasks. Also some tweaks about spatial feature selection or the length degree of paths may have a positive influence.

Chapter 7. Contribution, Possible Applications, and Future Work

The contribution of this thesis consists of four parts. First of all, it's a novel idea to research the question that is how to extract effective spatial information from floor plan images as machine learning input data and verify which extracted features are effective and non-redundant. Although the proposed graph-based data model and its extraction process are not a new concept, the discussion of specific encoding formats of graph structure and experimental verification of feature engineering based on space syntax and graph theory are innovative. Through the research on the extraction and expansion of different spatial feature variables and their performance analysis in the experiments to predict room's spatial qualities, It could be revealed that some spatial feature variables such as the size of isovist field are effective and necessary while others such as the number of room's connections are redundant. It not only proves the possibility of extracting sufficient spatial information from floor plan layout images for machine learning but also provides guidance for extracting spatial feature variables.

Secondly, this paper recognizes the problems in the process of converting high-dimensional graph data into low-dimensional vector data, that is, the lack of research on different data converting and encoding methods. To address this, this thesis explores multiple converting methods and analyzes their performance variance under different floor plan analysis objectives. In this thesis, three methods based on graph embedding theory are proposed to convert the raw graph data into standard vector data as machine learning input, and it is proved in the experiment that different encoding methods will affect the accuracy and efficiency of the analysis. The detailed experiment recording and comparative analysis can guide future research on selecting different data encoding methods according to different analysis objectives. For example, the edge embedding method is more accurate in the analysis of room's spatial qualities that

are related to room's topological relationship such as accessibility, while the hybrid embedding method is less efficient due to the increase of data capacity.

Thirdly, through the statistical analysis of five different room's spatial qualities, from privacy to sleep-ability, a dataset of 100 apartment floor plans is used as research sample which shows that these qualitative aspects of spatial configurations can be retained and learned statistically through the dataset processed by this framework.

Finally, the proposed framework provides a new scheme of complete architecture data processing, which takes floor plan images as input and the standardized vector data as outputs for machine learning analysis, including the graph-based data extraction, spatial feature engineering, multiple data encoding options, and the corresponding validation experiments.

Considering the proposed architecture data processing framework and related research, a question is raised about what main applications it would have. From a broader perspective of view, this thesis opens the gate for further analysis of architectural designs and design process optimization driven by data.

Processed by this framework, a large amount of data can push the limit of applying machine learning to analyze floor plan layout designs. Hence an input database based on previous design archives can enable innovative applications such as automated floor plan design based on specific spatial evaluation indicators and rapid mass evaluation of spatial environment quality. For example, a generative machine learning framework can be developed on a database driven by this framework to automate the design process of function-driven tasks which accelerates designers' work efficiency. A spatial environment quality evaluation system on mobile apps can be used in the real estate industry to quickly rate the house for sale, which allows consumers to have more choices and rental platforms easier to evaluate properties. And in

the context of design research, the previous design documents can be processed into a machine learning database which is categorized by different spatial quality evaluation objectives, such as privacy and livability, for future research.

However, this thesis has some limitations. First of all, the framework's software program for extracting graph data is not robust to style. In the initial experiment of trying plan images with different visual styles, it's found that the parameters and thresholds need to be adjusted accordingly, especially in the steps of noise reduction and room's connection extraction, which affects the universality of this framework. Secondly, the capacity of the dataset used in this study can be expanded. In the validation experiments, learning models, such as the deep neural network which relies on a large amount of data, are not adopted, it affects the depth of this experiment. Besides, the extraction of some spatial feature variables such as "isovist_value" has not been automated. Meanwhile, in the process of collecting architects' evaluation on room's qualities, the sample survey size is limited by the number of people. These all limit the practical application of this framework.

Due to the nature and limitations of this thesis, several open questions are left for further study. First of all, a fully-robust software capable of extracting all spatial information from plan images is yet to be achieved. Secondly, the input of the framework can be extended to other sources, such as 2D CAD files or 3D BIM files, to expand the scope of this framework and the diversity of data sources. Moreover, other methods for data modeling of spatial information in designs and converting high-dimensional data are also worthy of research.

In conclusion, this thesis proposes a data processing framework to study problems occurring in the application of machine learning to analyze floor plan layout design, that is how to extract spatial

information as input data effectively, how to convert high dimension data into low dimension vector data, and how to verify these data processing steps based on specific quantitative analysis targets. To deal with these question, this research is conducted with 3 steps. First of all, previous research in related fields and backgrounds including graph theory, isovist analysis, and graph embedding theory are reviewed. In this thesis, a dataset composed of 100 apartment floor plans is used and the analysis objective is to predict five spatial quality evaluations of each room. Then, the data processing framework including data extraction, feature engineering, data encoding, and experimental verification is proposed and its technical details are elaborated. In the experiment on the sample dataset, the performance differences of different spatial feature variables and different data encoding methods are tested and analyzed. Finally, the contribution, limitations, and future development of the thesis are discussed. By presenting a novel approach to architecture data processing for analyzing floor plans, this thesis opens the gate for data-driven architectural analysis and consolidates the foundation for the architecture field to enter the big data era.

References

- [1] Akgun, Yenal. 2003. “An Interactive Database (HizmO) for Reconstructing Lost Modernist Izmir”
- [2] Arturo F. Montagu and Juan Pablo Cieri. 2003. “Urbamedia - Development of an urban database of fragments of some Argentinian and Latin-American cities using digital technology”
- [3] Cecilia Ferrando; Daniel Cardoso Llach (advisor); Peter Scupelli(advisor); Carnegie Mellon University. Dept. of Architecture. 2018. “Towards a machine learning framework in space syntax”
- [4] Cecilia Ferrando; Niccolo Dalmaso, Jiawei Mai, Daniel Cardoso Llach. 2019. “Architectural Distant Reading Using Machine Learning to Identify Typological Traits Across Multiple Buildings”
- [5] Dawes, M.J., Ostwald, M.J., 2014a. Prospect-Refuge theory and the textile-block houses of Frank Lloyd Wright: an analysis of spatio-visual characteristics using isovists. *Build. Environ.* 80, 228–240.
- [6] Espina, Jane J.B. 2002. “Database of the Modern Architecture of the City of Maracaibo 1920-1990”
- [7] Franz, G., Mallot, H., Wiener, J., Neurowissenschaft, K., 2005. Graph-based models of space in architecture and cognitive science-a comparative analysis. In: *Proceedings of the 17th International Conference on Systems Research, Informatics and Cybernetics*. Baden-Baden, pp. 30–38.
- [8] Harding J, Derix C. 2010. “Associative Spatial Networks in Architectural Design - Artificial Cognition of Space using Neural Networks with Spectral Graph Theory”
- [9] Hongyun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. *A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications*. 2017
- [10] Huang, Weixin, and Hao Zheng. 2018. "Architectural Drawings Recognition and Generation through Machine Learning." *Proceedings of the 38th Annual Conference of the Association for Computer-Aided Design in Architecture*, Mexico City, Mexico.
- [11] Imdat As, Siddharth Pal, Prithwish Basu. 2018. “Artificial intelligence in architecture: Generating conceptual design via deep learning”.

- [12] Ju Hyun Lee, Michael J.Ostwald, Hyunsoo, Lee. 2017. Measuring the spatial and social characteristics of the architectural plans of aged care facilities.
- [13] Kuo, Chung-Jen. 2003. "Spatial Analysis of Chinese Garden Designs with Machine Learning"
- [14] Kvochick, Tyler. 2018. "Sneaky Spatial Segmentation. Reading Architectural Drawings with Deep Neural Networks and Without Labeling Data"
- [15] Magyar, Peter and Temkin, Aron. 2000. "Developing an Algorithm for Topological Transformation"
- [16] Michael J Ostwald. The mathematics of spatial configuration: revisiting, revising and critiquing justified plan graph theory. Nexus Network Journal, 13(2):445–470, 2011.
- [17] Nourian, Pirouz; Rezvani, Samaneh and Sariyildiz, Sevil. 2013. "Designing with Space Syntax"
- [18] Peng, Wenzhe, Fan Zhang and Takehiko Nagakura. "Machines' Perception of Space: Employing 3D Isovist Methods and a Convolutional Neural Network in Architectural Space Classification." (2017).

Appendices

1. Java code for graph embedding.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class GraphRead {
    public class RoomNode {
        String serialNumber;
        String function;
        double areaSize;
        double privacyLevel;
        double isovistValue;
        int connectionValue;
        double room_area_to_total;
        double accessibilityLevel;
        double sleepability;
        double liveability;
        int num_of_neighbors;
        double degree_centrality;

        Set<RoomNode> directAdjacent = new HashSet<>();
        Set<RoomNode> mediumAdjacent = new HashSet<>();
        Set<RoomNode> doorAdjacent = new HashSet<>();
        Set<RoomNode> allAdjacent = new HashSet<>();
    }

    //Construct the graph in a node list.
    private List<RoomNode> constructGraph(String source) {
        List<RoomNode> nodeList = new ArrayList<>();
        List<List<String>> nodeInfo = new ArrayList<>();

        //Construct the graph structure.
        try (BufferedReader reader = new BufferedReader(new FileReader(source))) {
            String line = reader.readLine();
            while ((line = reader.readLine()) != null) {
                String[] lineArray = line.split(",");
                List<String> lineInfo = new ArrayList<>();
                Collections.addAll(lineInfo, lineArray);
                nodeInfo.add(lineInfo);

                RoomNode node = new RoomNode();
                node.serialNumber = lineArray[0];
```

```

        nodeList.add(node);
    }
} catch (IOException e) {
    e.printStackTrace();
}

//Add the node connection relationship.
for (int i = 0; i < nodeList.size(); i++) {
    String connectionString = nodeInfo.get(i).get(1);
    connectionString = connectionString.substring(1, connectionString.length()-1);
    String[] connection = connectionString.split(";");
    RoomNode node = nodeList.get(i);
    for (String connectionPair: connection) {
        int nodeIndex = Integer.parseInt(connectionPair.split("_")[0]) - 1;
        int connectionLevel = Integer.parseInt(connectionPair.split("_")[1]);
        node.allAjacent.add(nodeList.get(nodeIndex));
        if (connectionLevel == 0) {node.doorAjacent.add(nodeList.get(nodeIndex));}
        else if (connectionLevel == 1) {node.mediumAjacent.add(nodeList.get(nodeIndex));}
        else {node.directAjacent.add(nodeList.get(nodeIndex));}
    }
}

//Add the node property.
double areaTotal = 0;
for (int i = 0; i < nodeList.size(); i++) {
    RoomNode node = nodeList.get(i);
    node.function = nodeInfo.get(i).get(3);
    node.areaSize = Double.parseDouble(nodeInfo.get(i).get(2));
    areaTotal += node.areaSize;
    node.pivacyLevel = Double.parseDouble(nodeInfo.get(i).get(4));
    node.isovistValue = Double.parseDouble(nodeInfo.get(i).get(5));
    node.connectionValue = node.directAjacent.size() + node.doorAjacent.size() + node.mediumAjacent.size();
    node.accessbilityLevel = Double.parseDouble(nodeInfo.get(i).get(6));
    node.sleepability = Double.parseDouble(nodeInfo.get(i).get(7));
    node.liveability = Double.parseDouble(nodeInfo.get(i).get(8));
    node.num_of_neighbors = node.allAjacent.size();
    node.degree_centrality = node.num_of_neighbors * 1.0 / (nodeList.size() - 1);
    node.degree_centrality = Double.parseDouble(String.format("%.2f", node.degree_centrality));
}
for (int i = 0; i < nodeList.size(); i++) {
    RoomNode node = nodeList.get(i);
    node.room_area_to_total = node.areaSize / areaTotal;
    node.room_area_to_total = Double.parseDouble(String.format("%.2f", node.room_area_to_total));
}

return nodeList;
}

//Construct the Node_Embedded CSV.
private void nodeEmbedded(List<RoomNode> nodeList, String path) throws IOException {

```

```

FileWriter csvWriter = new FileWriter(path + "/Node_Embedded.csv");

//Header writing.
csvWriter.append("name");
csvWriter.append(",");
csvWriter.append("edge_value");
csvWriter.append(",");
csvWriter.append("area_size");
csvWriter.append(",");
csvWriter.append("isovist_value");
csvWriter.append(",");
csvWriter.append("room_area_to_total");
csvWriter.append(",");
csvWriter.append("num_of_neighbors");
csvWriter.append(",");
csvWriter.append("degree_centrality");
csvWriter.append(",");
csvWriter.append("function");
csvWriter.append(",");
csvWriter.append("privacy_level");
csvWriter.append(",");
csvWriter.append("accessibility");
csvWriter.append(",");
csvWriter.append("sleepability");
csvWriter.append(",");
csvWriter.append("livability");
csvWriter.append(",");
csvWriter.append("\n");

//Writing node info.
for (RoomNode node: nodeList) {
    csvWriter.append(node.serialNumber);
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.directAdjacent.size() * 3 + node.mediumAdjacent.size() * 2 +
        node.doorAdjacent.size()));
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.areaSize));
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.isovistValue));
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.room_area_to_total));
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.num_of_neighbors));
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.degree_centrality));
    csvWriter.append(",");
    csvWriter.append(node.function);
    csvWriter.append(",");
    csvWriter.append(String.valueOf(node.pivacyLevel));
    csvWriter.append(",");
}

```

```

        csvWriter.append(String.valueOf(node.accessibilityLevel));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node.sleepability));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node.liveability));
        csvWriter.append(",");
        csvWriter.append("\n");
    }

    csvWriter.flush();
    csvWriter.close();
}

//Construct the edge_embedded csv
private void edgeEmbedded(List<RoomNode> nodeList, String path) throws IOException {
    FileWriter csvWriter = new FileWriter(path + "/Edge_Embedded.csv");

    //Header writing.
    csvWriter.append("edge_area_weight");
    csvWriter.append(",");
    csvWriter.append("edge_node1");
    csvWriter.append(",");
    csvWriter.append("edge_node2");
    csvWriter.append(",");
    csvWriter.append("edge_isovist_weight");
    csvWriter.append(",");
    csvWriter.append("edge_connection_weight");
    csvWriter.append(",");
    csvWriter.append("edge_area_to_total");
    csvWriter.append(",");
    csvWriter.append("edge_num_of_neighbors");
    csvWriter.append(",");
    csvWriter.append("edge_degree_centrality");
    csvWriter.append(",");
    csvWriter.append("node1_function");
    csvWriter.append(",");
    csvWriter.append("node1_privacy");
    csvWriter.append(",");
    csvWriter.append("node1_accessibility");
    csvWriter.append(",");
    csvWriter.append("node1_sleepability");
    csvWriter.append(",");
    csvWriter.append("node1_livability");
    csvWriter.append(",");
    csvWriter.append("\n");

    //Writing edge info.
    for (RoomNode node1: nodeList) {
        for (RoomNode node2: node1.allAdjacent) {
            if (!node2.serialNumber.equals(node1.serialNumber)) {

```

```

        csvWriter.append(String.valueOf(node1.areaSize + node2.areaSize));
        csvWriter.append(",");
        csvWriter.append(node1.serialNumber);
        csvWriter.append(",");
        csvWriter.append(node2.serialNumber);
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.isovistValue + node2.isovistValue));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.allAjaacent.size() + node2.allAjaacent.size()));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.room_area_to_total + node2.room_area_to_total));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.num_of_neighbors + node2.num_of_neighbors));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.degree_centrality + node2.degree_centrality));
        csvWriter.append(",");
        csvWriter.append(node1.function);
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.pivacyLevel));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.accessbilityLevel));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.sleepability));
        csvWriter.append(",");
        csvWriter.append(String.valueOf(node1.liveability));
        csvWriter.append(",");
        csvWriter.append("\n");
    }
}

csvWriter.flush();
csvWriter.close();
}

//Construct the path_embedded csv
private void pathEmbedded(List<RoomNode> nodeList, String path) throws IOException {
    FileWriter csvWriter = new FileWriter(path + "/Path_Embedded.csv");

    //Header writing.
    csvWriter.append("path_area_weight");
    csvWriter.append(",");
    csvWriter.append("path_isovist_weight");
    csvWriter.append(",");
    csvWriter.append("path_area_ro_total");
    csvWriter.append(",");
    csvWriter.append("path_num_of_neighbors");
    csvWriter.append(",");
    csvWriter.append("path_degree_centrality");
    csvWriter.append(",");

```

```

csvWriter.append("start_function");
csvWriter.append(",");
csvWriter.append("start_privacy");
csvWriter.append(",");
csvWriter.append("start_accessibility");
csvWriter.append(",");
csvWriter.append("start_sleepability");
csvWriter.append(",");
csvWriter.append("start_livability");
csvWriter.append(",");
csvWriter.append("node1");
csvWriter.append(",");
csvWriter.append("node2");
csvWriter.append(",");
csvWriter.append("node3");
csvWriter.append(",");
csvWriter.append("\n");

//Writing edge info.
for (RoomNode node1: nodeList) {
    for (RoomNode node2: node1.allAdjacent) {
        for (RoomNode node3: node2.allAdjacent) {
            if (!node3.serialNumber.equals(node1.serialNumber)) {
                csvWriter.append(String.valueOf(node1.areaSize + node2.areaSize + node3.areaSize));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.isovistValue + node2.isovistValue + node3.isovistValue));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.room_area_to_total + node2.room_area_to_total +
node3.room_area_to_total));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.num_of_neighbors + node2.num_of_neighbors +
node3.num_of_neighbors));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.degree centrality + node2.degree centrality + node3.degree centrality));
                csvWriter.append(",");
                csvWriter.append(node1.function);
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.privacyLevel));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.accessibilityLevel));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.sleepability));
                csvWriter.append(",");
                csvWriter.append(String.valueOf(node1.liveability));
                csvWriter.append(",");
                csvWriter.append(node1.serialNumber);
                csvWriter.append(",");
                csvWriter.append(node2.serialNumber);
                csvWriter.append(",");
                csvWriter.append(node3.serialNumber);

```

```
        csvWriter.append(",");
        csvWriter.append("\n");
    }
}
}

csvWriter.flush();
csvWriter.close();
}
```