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:

Rematerializing Graphs
Learning Spatial Configuration

AUTHOR:

Michael Stesney

ACCEPTED BY ADVISORY COMMITTEE:

[redacted]		May 25, 2021
Daniel Cardoso Llach	Principal Advisor	DATE
[redacted]		May 23rd 2021
Daragh Byrne	Advisor	DATE
[redacted]		25 May 2021
Molly Wright Steenson	Advisor	DATE

Abstract

In architecture, recent developments in machine learning technologies have reenergized the established research areas of generative design and design analysis. Proponents argue that data-driven systems can learn the implicit rules of architectural design for the generation of new designs or the incorporation into new methods of architectural research and analysis. This recent research has focused primarily on machine learning model development, in particular the algorithms, but not the data from which the models learn. However, without a concurrent reflection on the data and the representations used to encode the data, a critical understanding of the affordances and limitations of data-driven machine learning tools in architecture is not possible. Therefore, in this research I interrogate the data with a focus on graph-based representations of spatial organizations. As a research method, I present the theoretical concepts tying graphs to computational design theory to selected cutting-edge graph-based and data-driven generative and analytical approaches. I also reconstruct an example of early generative design software and create computational instruments to explore, demonstrate and evaluate the concepts presented in the literature research. Placing recent analytical and generative techniques in the lineage of graph-based representations demonstrates how the motivations of the early theorists, and limitations of graph representations themselves, are carried forward into the machine learning context. Further, investigating the floor plan data sets essential to these technologies reveals the inherently contingent nature of graph-based architectural data, and by extension, the contingency of the results.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivations

Broadly, my research interest lies in generative systems used to support architectural design and research. I was introduced to these systems in the Generative Systems for Design course taught by Prof. Ramesh Krishnamurti and Pedro Veloso. My final project in the course explored procedural form generation using the Rhino CAD application with the Grasshopper plug-in, Embryo, and the Biomorpher plug-in (Harding, 2016; Harding & Brandt-Olsen, 2018). Embryo is a generative system utilizing genetic programming. Inspired by philosopher Manuel DeLanda's argument that evolutionary design algorithms must allow for topological variation of the generated forms, not only parametric variation, its generative search method is modeled on computer scientist John Koza's research into automated evolution of computer programs (DeLanda, 2002; Koza, 1992). Biomorpher assists a designer by clustering large numbers of generated options by similarity. The designer's evaluation of the filtered options informs the successive searches.

During the summer of 2020 I continued exploring generative systems while working for a modular building manufacturer. The manufacturer desired to offer an online building design generator to their developer clients. Leaving aside for a moment the possibility that a complete architectural design process could ever be automated, even for as tightly constrained a problem as these modular buildings, deeply engaging in the work brought me many insights about generative design and exposed another way of considering topology in generative design. Specifically, the incremental development of the tool gradually revealed the need to generate and evaluate solutions constrained by their spatial configurations, not simply formal assemblages. The generator we were building needed to derive both formally and functionally successful buildings, yet one characteristic did not guarantee the other. Allowing influence from both constraints promised to allow a broader formal search while also increasing the likelihood of finding functionally correct solutions.

Researching computational representations of spatial organization took me beyond simple bubble diagrams and into a study of the history of graphs in architectural design. Architectural historians are researching the role mathematical representations of shape and space played in 20th century architectural theory. The historical research coincides with a burgeoning research interest in graphs in other domains such as biology, computational chemistry, social network analysis, and recommendation systems. Machine learning on graph structures within these domains is an active research area and those efforts are beginning to influence computational research and design in the architectural domain. The intersection of machine learning and graph-based architectural representations suggested to me opportunities for a timely research topic.

1.2 Design Generation

Today, within the building and design industry significant investments are being made to develop powerful new tools that purport to generate designs. These investments are occurring in tandem with advancements in visualization, physical modeling, and document production. As more professionals gain the necessary skills, these technologies are gradually becoming part of everyday design practice. In parallel, advanced manufacturing, materials, and construction technology offer designers seemingly ever-expanding opportunities for creativity and higher performing buildings. Increasing computer power, professional knowledge, market pressures and the investments being made by software developers has led many people to suggest that generative tools are poised to take a greater role in architectural practice in the near future.

An example of the dissemination of generative tools is software developer AutoDesk's generative design platform for Revit (*Revit Gen Design*, n.d.). It offers designers a ready-made workbench to automate the search of parametric design spaces. Because Revit is the industry-standard building information modeling (BIM) software, access to the capabilities of this tool will introduce the concept of generative design to many designers who may not have otherwise sought it out. Other efforts go beyond tools that may assist designers at the scope of a single project and instead attempt to redefine industrial and professional roles and relationships. Spacemaker.ai, recently acquired by Autodesk, is an urban planning tool for developers and designers (*Spacemaker AI*, n.d.). It allows developers to evaluate the feasibility of a building site and real estate project using computer-generated building options. Alphabet, the parent company of Google, is developing a similar platform called Delve, under the umbrella of its subsidiary Sidewalk Labs (*Delve*, n.d.). The Sidewalk Labs led development proposal for the (now defunct) smart-city project in Toronto, Canada, suggests the opportunity big technology companies see in generative tools as an entry to the design and construction market.

However, the idea of a generative system that can automate a design process is not new and has been discussed since Aristotle (Mitchell, 1979). Pursuit of these systems has many motivations, including speedier exploration of multiple design possibilities, the capability to address ever larger and more complex design problems or even the organizational symmetry of coupling an automated manufacturing process with an automated design process. For Australian architectural theorist William Mitchell, the application of generative systems required the framing of architectural design as a “special kind of problem-solving process" within a "framework of a general theory of problem solving" (Mitchell, 1979, p. 37). Essential to reframing design as a procedural method is the creation or adaptation of representational systems amenable to the problem-solving process, and by extension, digital computers.

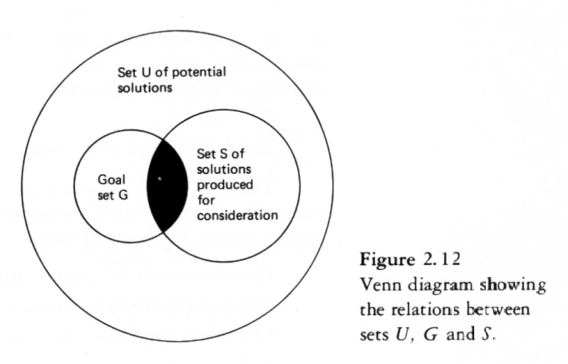

Set U of potential solutions
Goal set G
Set S of solutions produced for consideration
Figure 2.12
Venn diagram showing the relations between sets U, G and S.

Figure 1: Generative design as search within defined solution space (Mitchell, 1979)

To generate suggests mechanical creativity and the ability to bring into existence something that did not exist. However, in the domain of generative systems, the term creativity is misleading. Instead, generation is best defined as an act of finding options, variations or solutions by searching within a defined search space. Figure 1 illustrates this concept. Therefore, generative design in its most general sense has three components: 1) a system representation amenable to computation, 2) an algorithmic search procedure, 3) a defined search space. This neat framing is the basis of Mitchell's argument for the value of digital computer use in architectural design and persists today.

There are many different generative systems, while new and more sophisticated examples are being developed continuously. They can be grouped in numerous ways, but several distinctions are helpful and will be referenced later. All provide output along a spectrum between mathematically deterministic to (pseudo) random. Broadly, this is a function of the level of constraint placed on the search of the design space. Additionally, all have a characteristic level of designer interaction during the generative process. At one extreme is no interaction at all. The system produces results solely from the initial input. At the other extreme is a system that uses feedback from the designer to steer the generative process in real time. During the generation process these systems may also assist the designer by evaluating and sorting options for review by the designer. This process exploits the computer's capability to generate large quantities of options, while also exploiting the subjective expertise of the designer.

As a result of continuous development in machine learning techniques, a new distinction within generative systems has emerged. Data-driven generative processes operate fundamentally differently from conventional rule-based generative processes. The latter rely on explicit rules to define system behavior and by extension the design space. These rules are conceived and crafted by the system designer. Data-driven generative systems use the input data to define the design space. The system then produces results using the implicit patterns and rules learned from the data. Proponents argue that data-driven generative systems offer a powerful tool to explore problems without closed-form solutions, such as most design problems (Hu et al., 2020).

Floor plan creation is an example of a problem without a closed-form solution. At its most basic level, a floor plan is a scaled map of the physical elements of a building. As such it delineates the building's spaces and the physical access between them. Devising generative systems for floor plan creation is a popular area of research. One important lineage within this domain employs mathematical graph abstractions to represent the spatial relationships structured in floor plans. Here, graph nodes represent rooms or spaces, while graph edges represent physical adjacency and passage between spaces.

1.2.1 Hypothesis

The use of graph-structured data for data-driven floor plan generation raises many exciting prospects for new computational methods of architectural design and research. This thesis takes the position that recent scholarship on a specific area of 20$^{\text{th}}$ century architectural research that focused on the mathematical description of spatial organization provides a lens to critically consider these technologies. Importantly, that period coincided with early explorations of the affordances of digital computers in architectural design and analysis. The mathematical abstractions developed are intimately connected to those explorations. Rather than focusing on the algorithms employed by the cutting-edge data-driven generative systems of today, my research interrogates the data and data transformations required to use the algorithms. *The primary hypothesis is that a critical understanding of the history of graph structured representations of space is crucial to understanding the potential and limitation of these new technologies.*

1.2.2 Structure of Document and Methods

The Background chapter is divided into two sections. The Representing Spatial Configuration section covers a particular focus of architectural research spanning between roughly 1965 and 1985 at Cambridge University in England. During this period interdisciplinary scholars at Cambridge University, including Christopher Alexander, Lionel March, and Philip Steadman created rigorous mathematical and geometric descriptions of space and shape. Graph representations played a central role in their theories. They did not initiate the use of graph representations in architectural design, and the intersection of graph representations and architecture was being explored concurrently at other sites, such as the Massachusetts Institute of Technology in Cambridge, Massachusetts. However, the work by Alexander, March and Steadman is integral to the conceptual lineage of graphs in architectural theory and design. The Learning Spatial Configuration section discusses the emerging use of graph-structured data for architectural research and generative design employing machine learning. Machine learning has opened new opportunities to perhaps address some of the practical challenges faced by the 20^{th} century researchers. Unlike the scholars in Cambridge, current researchers of the data-driven generative systems may not be architects or designers; instead, these researchers are computer scientists and mathematicians employing floor plan design and graph representations to explore mathematical packing problems or using design as a site explore ideas about artificial intelligence. Nevertheless, these studies raise many interesting questions about the application of machine learning technology in architectural design and research. In particular they offer an opportunity to ask questions about the spatial representations and abstractions designed specifically for these tools. My analysis has found that while these new research efforts have initiated new avenues of research, they have also re-established some of the inherent limitations of graphical representations of spatial organization.

The Methods chapter is broken into three sections which I have called *computational vignettes*. Vignettes are computational instruments to explore, demonstrate and evaluate the concepts presented in the Background section. Vignette 1 is a reconstruction of the algorithms used by William Mitchell, Philip Steadman and operations researcher Robin Liggett to enumerate a catalog of rectangular dissections. Additional algorithms search the catalog to match dissections to design requirements. Vignette 2 investigates vector representations (embeddings) of graph spatial organizations through machine learning classification experiments. Vignette 3 demonstrates a data-driven floor plan generator. In doing so, it also aims to demonstrate the critical role of data in these systems. Discussions of the results and insights gained from each Vignette are included at the end of each. Finally, the Conclusion will reflect on the overall thesis and propose areas for further research.

Figure 12.4
Graph showing numbers of journeys made daily by workers one to another, for the research institute example.

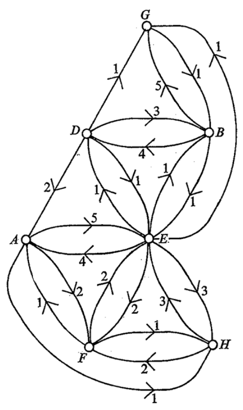
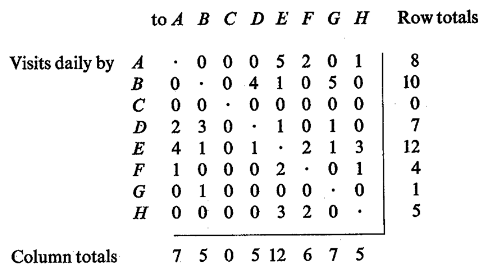


Table C



		to *A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	Row totals
Visits daily by	*A*	·	0	0	0	5	2	0	1	8
	B	0	·	0	4	1	0	5	0	10
	C	0	0	·	0	0	0	0	0	0
	D	2	3	0	·	1	0	1	0	7
	E	4	1	0	1	·	2	1	3	12
	F	1	0	0	0	2	·	0	1	4
	G	0	1	0	0	0	0	·	0	1
	H	0	0	0	0	3	2	0	·	5
Column totals		7	5	0	5	12	6	7	5	

Figure 2: Quantification of building occupant travel patterns (March, 1971)

2 Background

2.1 Representing Spatial Configuration

2.1.1 Scientific Aspirations

During the post-World War II period, architectural researchers at Cambridge University in England pursued scientific theories of architectural design. Motivated by, among other things, the functionalism of the Constructivists and their anti-intuition doctrine, they believed that scientific methodologies were necessary for architecture to meet the design challenges of the built environment of the second half of the twentieth century (Keller, 2017). While attempting to situate architecture as a scientific endeavor, they also placed architecture itself as the subject of scientific inquiry. Four primary themes laced together these inquiries during the "1960s and 1970s: mathematics (particularly graph theory), computer analysis, the quantification of programmatic and environmental requirements, and an overriding reliance on scientific analogies" (Keller, 2017). Figure 2 is an example of quantification of travel patterns within a building collected with the intention of informing future building design efforts.

Having completed his undergraduate degree in mathematics prior to pursuing his architectural studies, architect Christopher Alexander made early interdisciplinary connections between graph theory and the architectural design process. He was also a very early prospector of the use of computers in architectural design. For Alexander, the graph structure provided a formalism to both analyze and structure the design problem as well as describe the structure of a spatial solution (Steenson, 2017). After leaving Cambridge, Alexander completed his PhD at Harvard, the dissertation from which became the seminal *Notes on the Synthesis of Form*, published in 1964. Although Alexander eventually disavowed his mathematically deterministic approach to design, the analysis of space plan layout using graph theory continued to be an important component of the work at Cambridge.

Published by Lionel March and Philip Steadman in 1971, *The Geometry of Environment*, lays out their case for the relevance of "modern geometry" in architectural design (March, 1971). Among its many affordances, mathematics provides a symbolic representation system adaptable to the expanding promise and rapid advancement of digital computer technology into the practice of architecture. However, as Vardouli explains, the introduction of new mathematical abstractions to architectural research was not solely prompted or resultant on computer use. Instead, the refocus was motivated by "historically specific visions about the intellectual and disciplinary benefits" of the new abstractions (Vardouli, 2020). In the Preface to *The Geometry of Environment*, March summarizes the now quantifiable descriptions of spatial relationships available in the adopted mathematical theories:

> "Perhaps the chief difference between the traditional treatment of geometry in architecture and the one presented here, is that, previously, geometry was employed to *measure* properties of space such as area, volume, angle, whereas the new mathematical theories of sets, groups and graphs - to name but a few - enable us to describe *structural relationships* which cannot be expressed in metrical forms, for example, 'adjacent to', 'in the neighbourhood of', and 'contained by.'"

The authors also position their ideas as complementary to *Notes on the Synthesis of Form*, by focusing more on form in contrast to Alexander's emphasis on function. Although function is not completely removed from the mathematical translations proposed, when demonstrating graph representations of spatial relationships nodes are simply rooms, without any indication of function.

By the early 70's enthusiasm for the belief that form could be directly derived from function was waning (Keller, 2017). Research within the Centre for Land Use and Built Form Studies (LUBFS) by Philip Tabor and Tom Willoughby persuasively argued that "quantifiably optimized architectural solutions were largely impossible" (Keller, 2017). Furthermore, architect and historian, Colquhoun argued, in opposition to the burgeoning Design Methods movement, that "the configurations which [a designer] arrives at must be the result of an intention, and not merely the result of a deterministic process" (Colquhoun, 1969). In the Introduction and numerous other locations in *Architectural Morphology*, Steadman cautions against the idea that mathematical descriptions of shape and form are arguments for, or lead inevitably to, either a "functional determinism" or a design method approach to design. He falls back to support the work with the belief that "understand[ing] geometrical limitations and geometrical possibility formed a valuable and ... neglected part of the general education of the designer" (Steadman, 1983). Closing out the book, he proposes that the mathematical study of architectural form, a morphology in the spirit of Geothe and D'Arcy Thompson's studies, is the beginning of a "'natural history' of architecture," organized by geometry and function type (Figure 3). Without abandoning the modernist break from historical type, the graph of spatial configuration becomes solely a classification instrument, rather than a container of generative potential, in the support of a "theory of built form."

Work by the Centre for Configurational Studies to document the morphology of English building stock further clarifies the role of graph representations as a tool for architectural analysis and classification (Steadman et al., 1991). Researchers at the Centre believed that a theory of morphological classification was necessary to support "scientific generalisations ... about the relationships of form to performance." The survey of housing in Cambridge, England, included approximately 400 residences. Each floor plan was stored in a digital format as a typical floor plan and a graph, that indicated both adjacency and access. Documentation and study of actual buildings added to their theory of built form by elaborating on addi-

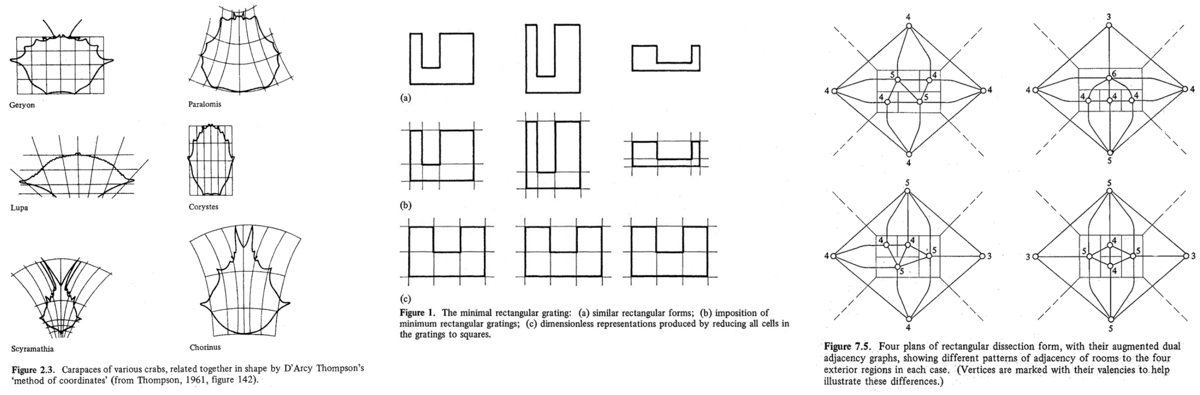

Geryon
Paralomis
Lupa
Corystes
Scyramathia
Chorinus
Figure 2.3. Carapaces of various crabs, related together in shape by D'Arcy Thompson's 'method of coordinates' (from Thompson, 1961, figure 142).
(a)
(b)
(c)
Figure 1. The minimal rectangular grating: (a) similar rectangular forms; (b) imposition of minimum rectangular gratings; (c) dimensionless representations produced by reducing all cells in the gratings to squares.
Figure 7.5. Four plans of rectangular dissection form, with their augmented dual adjacency graphs, showing different patterns of adjacency of rooms to the four exterior regions in each case. (Vertices are marked with their valencies to help illustrate these differences.)

Figure 3: Topological lineage (left, right) (Steadman, 1983), (middle) (Mitchell et al., 1976)

tional concepts that constrained the geometrical limits of graphs and rectangular dissections. While crucial, these constraints are contingent and changing and therefore secondary to the immutable rules of topology. The constraints include "functional need, technological factors and legal requirements." The order of these concerns, from dimensionless configuration to specific dimensioned form, is expressed clearly in Figure 4.

Although the Centre for Configurational Studies moved away from graph representations as seeds for an automatic floor plan design process, other researchers have continued the work of the functionalist program. The progressive development of new algorithms aspires for results that are more similar to what a designer might create manually. "A Graph Theoretical Approach for Creating Building Floor Plans," includes a succinct history of the progress and contributions made by each new generative algorithm using graph theory (Shekhawat, Pinki, & Duarte, 2019). For example, some additional functionalities are the inclusion of non-rectangular rooms, acknowledgment of exterior adjacencies, and inclusion of circulation space. The progress is, however, not cumulative as each algorithmic approach is fundamentally different from the others. Even the method proposed in the paper, although clearly able to create dissections of greater variety than rectangular dissection algorithms of the 1970s and 1980s, is not convincingly more useful than any of its predecessors. Figures 5 and 6 are examples of floor plans generated in 1980 and 2021, respectively.

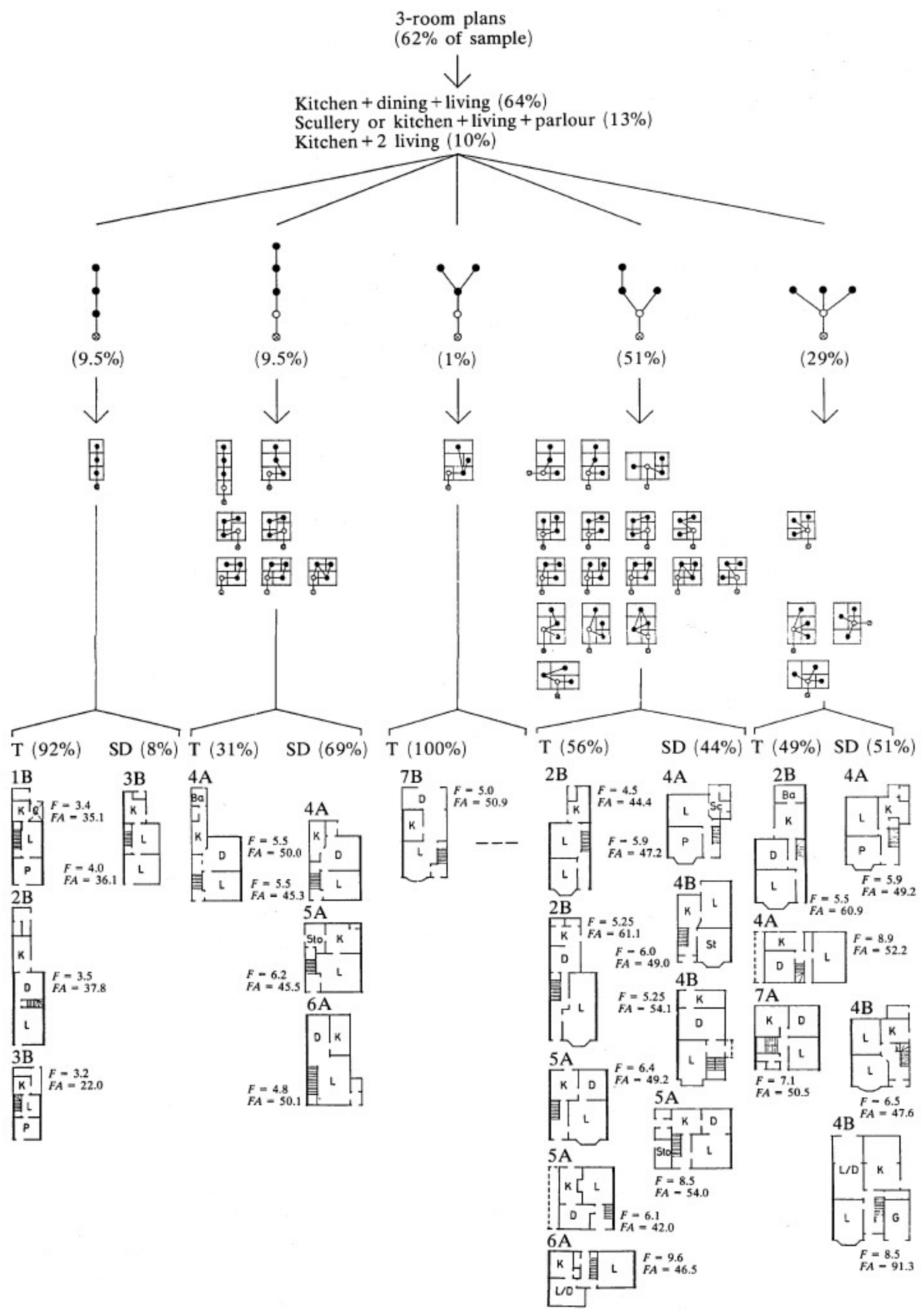


Figure 3. Morphological classification of dwelling plans from the Cambridge sample. The figure relates to terrace (T) and semidetached (SD) houses with three principal rooms on the ground floor. At the bottom is a representative selection of plans (but not including all dimensional variants); at the centre are the plans represented as rectangular dissections, with their access graphs superimposed; at the top are five spanning subgraphs of tree form to which these access graphs correspond. *F* is frontage width and *FA* is gross floor area.

Figure 4: English housing stock classification (Steadman et al., 1991)

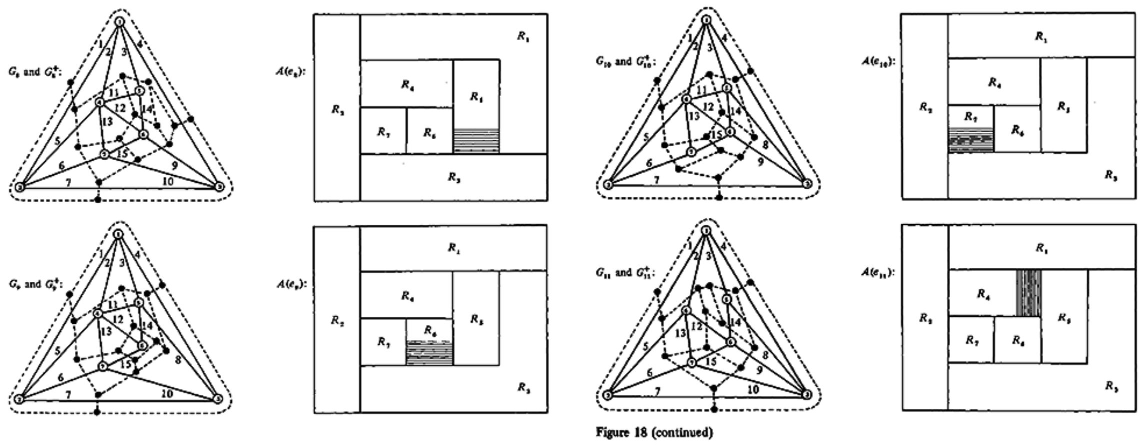


Figure 5: Examples of planar graphs mapped to floor plans with the addition of circulation spaces (Baybars & Eastman, 1980)

2.1.2 Beyond Dissections

As an alternative to layout generation with sophisticated rectangular dissectors using a graph structure as input, other pairings have been proposed. Shape grammars, introduced by computer scientist James Gips and design and computation theorist George Stiny in 1972, are a generative method that works directly with shape and form without reducing them to abstraction (Stiny & Gips, 1971). Vardouli argues that graph abstractions and unmediated shapes operate in two different "representation spaces" (Vardouli, 2020). The following three projects attempt to utilize the strengths of each. Heitor et al. proposed a method to constrain a shape grammar generator, the Malagueira grammar by José Duarte, with space syntax graph methods (Heitor, Duarte, & Pinto, 2004). Duarte created the Malagueira grammar for his PhD thesis at MIT. It is a generative shape grammar derived through study of Portuguese architect Alvaro Siza's realized projects in the Malagueira housing subdivision (Duarte, 2001). Through derivation of a design with the grammar, a correct spatial organization is resultant from the formal generation. In other words, the spatial logic is implicit in the grammar rules. Following the creation of the grammar, Duarte along with Heitor and Pinto, analyzed the Siza designed houses and the grammar-generated houses using space syntax measures. This analysis led them to propose rules to evaluate the developing graph structure

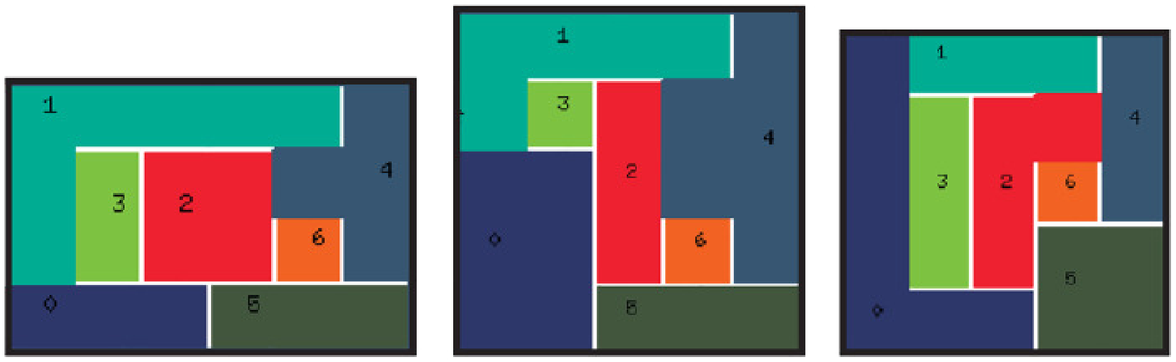


Figure 6: Recent examples of floor plans generated from graph input (Shekhawat et al., 2021)

during the generative process. The additional evaluation, operating parallel to the formal grammar, created a method of maintaining stylistic consistency without over constraining the formal generative process.

A similar attempt at a single framework combining the abstract structure of graphs with the formal generative process of shape grammars is proposed by Al-Jokhadar (Al-Jokhadar & Jabi, 2016). Like the Malagueira grammar, this grammar was derived from existing buildings but focused on a single type, rather than the work of a single designer. The work hypothesizes that the "social, cultural and contextual aspects" of vernacular North African homes can be encoded into a generative grammar at conception. Thus, they are arguing that the spatial organizing structure can be explicitly situated within the rules of the grammar, along with the formal principles and materiality.

Graphs can also be generated algorithmically with a grammar. Grasl and Economou proposed such a generative system by translating the generative rules of a shape grammar to the generative rules of a graph grammar. They successfully applied this approach to the well-known Palladian grammar created by Stiny and Mitchell (Grasl & Economou, 2010; Stiny & Mitchell, 1978). Spatial subdivision is achieved through the translation of edge attributes into walls and openings. Although somewhat awkward in conception, this project demonstrates the potential of entwining the formal system process of a shape grammar with the spatial organizing structure of a graph grammar.

2.1.3 Summary

The mathematical representations of spatial configuration that were the focus during this period supported two interrelated modes of architecture and design research: analytical and generative. The analytical research of morphology conducted at the Centre for Configurational Studies sought a scientific understanding of spatial configuration. Graphs were considered valuable tools for generalization and classification of the built environment because they did not require subjective taxonomies of formal or historical typologies. Digital computers played an important role in the creation and storage of morphological studies, as demonstrated by the British housing stock study. This early database of architectural information is a precursor to the data sets discussed in the following chapters.

Exhaustive enumerations of dissections span both the analytical and generative modes. Analytically, the enumerations served as objective demonstrations of the capabilities and limitations of the new mathematical representations of spatial configurations. The production of enumerations also suggested a potential use for digital computers in architectural practice. Steadman speculated that such a catalog of the enumerations could serve as a reference to support research such as the British housing stock study. For generative design purposes, the exhaustive enumerations served as a neutral data set of geometric relationships in support of the generative process. Generation from enumerations is directly related to the first computational vignette prepared in this thesis. Vignette 1 reconstructs an important example of this approach and offers an opportunity to experience the proposed design workflow.

Graphs of adjacency requirements also served as constraints in generative processes that sought singular or small collections of solutions. Because the size of a dissection catalog grows explosively as the number of rooms increases, catalogs of dissections are impractical for anything but small rectangular floor plans. Instead, explicit algorithms based on graph theoretical concepts create sets of solutions specific to the input adjacency requirements.

Generative approaches that do not use a catalog are the more common approach described in the literature.

Today, graphs are common in architectural practice as bubble diagrams. Designers often use bubble diagrams to quickly explore and communicate programmatic concepts prior to resolving geometrical relationships. The natural analogies of bubbles to spaces and lines to physical access is intuitively understood and make bubble diagrams ready tools for communicating abstract spatial relationships. Planners use adjacency matrices to communicate design requirements, called the program, to the design team. Adjacency matrices serve as a specification to guide and a target against which a realized design can be compared. 20th century British architectural historian John Summerson [1] claimed that "The programme as the source of unity is ... the one new principle involved in modern architecture" (Summerson, 1990). Graphs have proven to be an apt container for this principle.

2.2 Learning Spatial Configuration

Recent advancements in machine learning open new possibilities for spatial analysis and design generation using the mathematical abstractions established by the architects and mathematicians of the Cambridge group. Those two modes of research will structure the areas of inquiry in this section. However, before returning to those ideas, three floor plan data sets and the concept of vector embedding of graphs will be introduced. New research is being conducted in the digital computing world of hardware optimized for machine learning tasks, accessible through cloud computing services that make available vast computing power even to small scale architectural researchers. The distribution of computing power is a notable difference from the research discussed in the previous section. Because this is a nascent area of research, work is only beginning to be published. All of the examples discussed in this section have been published in the past two years and some propose complex machine learning algorithms. While the technical aspects are critical to the evaluation of the project, my intent is not to provide a technical explanation of the projects. Instead, I aim to place the projects

[1] Introduced to me by Molly Steenson

within a larger design context and lineage of the graph representations in architecture.

2.2.1 Creating Data

Large quantities of data are typically required for machine learning applications. This is particularly true for data-driven generative systems. Although the common practice of employing crowd-sourcing marketplaces to sort, label, and clean data floor plan data presents unique challenges. Public collections of floor plans in digital formats are relatively rare and often contain a limited range of spatial and building types. It is reasonable to assume that digital collections only represent a small fraction of the floor plans in archives. When available, digital floor plans are often stored in raster formats that do not allow for convenient extraction of semantic information or geometry. This limitation is noted by nearly all researchers in the field. At least one academic institution is explicitly addressing this challenge. By offering a public database to store floor plans and tools to extract semantic information from rasterized plans the institution hopes to encourage contributions (*Building Database & Analytics System (BuDAS)*, n.d.). The current focus of the database is limited to domestic architecture.

Computer vision (CV) methods capable of converting common digital image formats to vector formats usable by CAD software are an active research area. To accomplish this, CV methods seek to emulate the expert human skill required to manually interpret plans. Importantly, CV is also a machine learning technology that requires human labeled training data which can be costly and time consuming. The enormous range of common graphic styles for floor plans presents a great challenge to automated format translation systems. Extensive pre-processing is often required, limiting the generalizability of CV tools.

Computer aided design software (CAD) is a standard technology for building design and engineering practices and has been for decades. Even so, CAD formatted plans of published projects and built work are typically not made available to the public. Makers of CAD software, especially those hosting clients' models on cloud computing services have a new and unique position to collect contemporary plans. Understanding the value of such data for development of their own products and to the design and research community in general, some are currently exploring avenues to collect plans from the users of their software (Cheng, 2021). Access to this data will need to address significant privacy, intellectual property and safety concerns.

Sensor data, IoT data, and other modes of spatial data suggest another method to build data sets of spatial organization (Abdelrahman et al., 2020). Although there are significant privacy and other important issues to consider, sensor data and data collected from other sources, such as social media, may provide a closer to ground truth model of the built environment. As noted above, data sets of floor plans are built on several levels of representation and abstraction. The original floor plans were crafted for varying audiences, thereby including, excluding, and emphasizing different information. Among other possibilities, sensor data may create new opportunities to continue the functional and programmatic analysis of space use initiated in the 20th century.

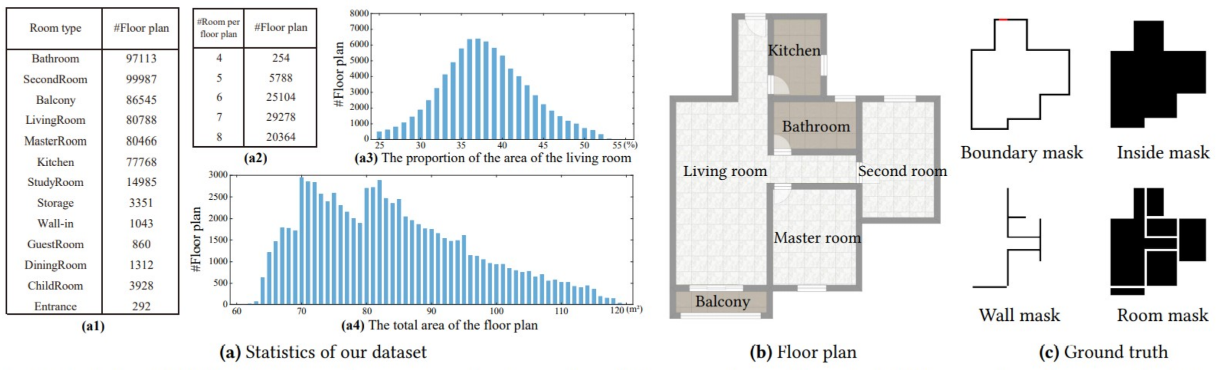
Three particular data sets are important to the projects presented in this background section and to the vignettes. The first is the mosques and monasteries data set (Ferrando et al., 2019). The data set contains the CAD plans, initial research data and tabular form data of 19 monastery and 20 mosque complexes. Multiple online sources supplied the initial information on each complex. Collection, sorting, interpreting and ultimately transforming the raster images into a CAD format required expert skill and was primarily a manual process. The elements in the CAD format were carefully devised to allow for automated extraction of spatial relationships. Although too small to be used for a generative system, this data set is presented for several reasons. First, the research that initially assembled and experimented with the data set is discussed in the next section. Vignette 2 uses the same data set for classification experiments because the spatial structures are complex and varied. Finally, the original paper demonstrates and explicitly calls attention to the human labor required to transform commonly available formats of architectural information into formats applicable to machine learning.

The next two data sets address the challenge of sparse data and a labor-intensive collection process in different ways. Both are composed of Asian residential floor plans and both are large enough to be used for data-driven generative systems. Like the mosques and monasteries data, both of these residential data sets were created for research exploring the intersection of machine learning and spatial arrangement.

RPLAN is a data set created specifically to feed a floor plan generation tool created by the same researchers (Wu et al., 2019). Numerous selection criteria were employed to standardize the corpus of plans. Unusual layouts, those with unidentified rooms, and those with rooms not among the thirteen most common types were removed. The authors also set criteria for the overall area of the plan, the number of rooms, the presence of a living room, the proportion of the living room to the overall area, and average room size. Figure 7 shows the distribution of various floor plan characteristics with in the data set. Initially the plans were in a vector format, but for the purposes of the machine learning architecture, each was converted to a four-channel 256x256 pixel raster image. The channels contain a boundary mask, inside mask, wall mask, and room mask. The room mask encodes the room types. The final set contains over 80,000 plan images.

The data set is described as "manually collected," but how and when the original vector format plans were created or where they were found, how semantic information like room function was determined, and the evaluation criteria for standard plans is not fully documented. The files available from the research group do not contain the vector formatted files. As noted elsewhere, much research on floor plan data sets has focused on automated creation of vector information and extraction of semantic information from raster images. The seeming discounting of the vector format information is intriguing.

LIFULL HOME's is a much larger and more diverse data set (*Informatics Research Data Repository [LIFULL HOME'S Dataset]*, n.d.). Originally made public in 2015, the real estate property data set contains over 5.3 million properties in the Japanese rental market. The publishers, LIFULL, operate LIFULL HOME'S, "Japan's largest ... real estate and housing information site" (LIFULL, n.d.). The website describes its mission as a platform to connect renters to property owners, movers and other property services. According to the



Room type	#Floor plan
Bathroom	97113
SecondRoom	99987
Balcony	86545
LivingRoom	80788
MasterRoom	80466
Kitchen	77768
StudyRoom	14985
Storage	3351
Wall-in	1043
GuestRoom	860
DiningRoom	1312
ChildRoom	3928
Entrance	292

(a1)

#Room per floor plan	#Floor plan
4	254
5	5788
6	25104
7	29278
8	20364

(a2)

Fig. 3. Our dataset *RPLAN.* (a) Statistics on the occurrence of each room type (a1), room number per floor plan (a2), the proportion of the area of the living room (a3), and the number of floor plan according to the total area (a4). (b) One typical floor plan in our collected dataset. (c) For each floor plan, we abstract the necessary information used in our method, including the boundary mask (the entrance is shown in red), inside mask, (interior) wall mask, and room mask.

Figure 7: Composition of RPLAN data set (Wu et al., 2019)

company's blog post announcing the data set publication, LIFULL HOME'S hoped outside groups "...will use the data set for research and use it for innovation that will transform the search for a place to live" (LIFULLCreators, n.d.-b)!

In addition to images of floor plans, samples of which are shown in 8, the data set contains photographs and many categories of property data such as rent and fees, general location, building type and amenities, and neighborhood information. Although not explicitly stated, the posted information on the data set implies that it includes all of the properties in the LIFULL platform available at the time and the only criteria for property inclusion was the presence of adequate information. In 2016, 5.15 million high-resolution floor plans of size 256x256 were added to the data set. The updated plans were a response to researcher requests and justified that "active research on floor plans will lead to improved convenience for users looking for a home in the medium to long term" (LIFULLCreators, n.d.-a). LIFULL HOME'S also offers a data set toolkit in a GitHub repository. Although publicly available, use of the data set is restricted to academic and research institutions.

2.2.2 Embedding Relationships

Structured relational data is common in many research domains such as computational biology and social network analysis. Typical machine learning techniques require data with

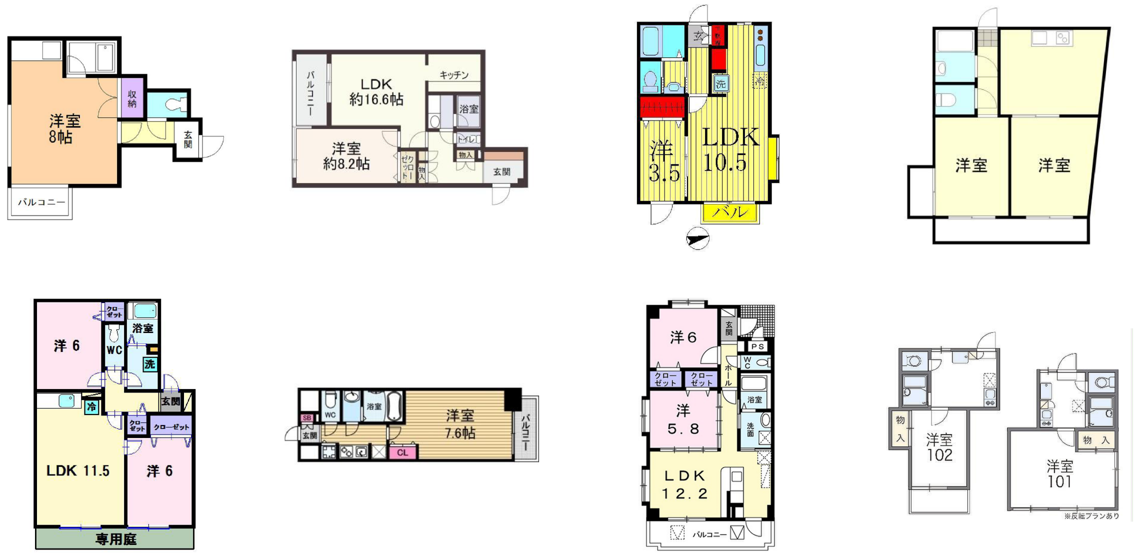


Figure 8: Samples from LIFULL HOMES data set (*Informatics Research Data Repository [LIFULL HOME'S Dataset]*, n.d.)

simple relational structures, such as sequential data or data with fixed dimensions. Therefore, complex relational data structures, like graphs, must be transformed into vector representations before machine learning techniques can be applied. There are many vector-embedding techniques available, but in general, they all attempt to locate nodes or graphs in the vector space by similarity, as diagrammed in 9. Successful techniques minimize the distance between similar nodes or graphs and maximize the distance between dissimilar ones. Techniques that capture the entire graph as a single vector are called graph embedding techniques. Likewise, those that capture nodes are node embedding techniques. Techniques break down into two primary categories: unsupervised and supervised algorithms. Unsupervised algorithms aim to extract the graph structure and the relationship of the objects in the graph. Supervised algorithms also aim to capture the structure of the graph but they also capture node attributes (Chami, Abu-El-Haija, Perozzi, Ré, & Murphy, 2020). The choice of vector representation is crucial to the performance of the machine learning application (Grohe, 2020).

Recent work involving graph representations of spatial organization explores the potential

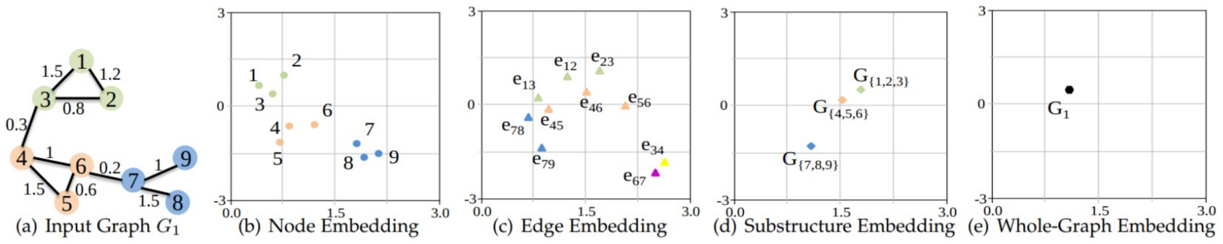


Figure 3. Graph embeddings convert the graph structure (vertex and edges) into vector representation. Image adapted from [4]

Figure 9: Diagrams of graph, edge and node embedding into 2-dimensional vector space (Abdelrahman et al., 2020)

utility of different embedding approaches. In architectural terms, vector embeddings aspire to capture the semantic information extracted from floor plan representations and encoded in the graph representation. As such, they are an additional abstraction level removed from the original floor plan representation. Which spatial characteristics to include as node attributes is a primary consideration when creating graph representations and the following research takes several approaches. Function is a common choice, but any number of other attributes are possible. Equally important is the edge representation. Edges commonly represent adjacency and physical access and therefore make-up the relationship structure of the graph. Edges may also be attributed with information that qualifies or quantifies the type of the relationship represented by the edge. For example, values may be used to define both desired connections and disconnections.

2.2.3 Spatial Analysis with Embeddings

For researchers looking to further the analytical methods of space syntax, vector embedding methods may provide a new lens to study the relationship of spatial configuration and social configuration. Initial steps toward the integration of machine learning into the lineage of space syntax tools is proposed by Ferrando (Ferrando, 2018). By encoding the node information as feature vectors, this research asks whether machine learning models can usefully learn the quantitative measures of space syntax defined by architectural theorists Bill Hillier and Julienne Hanson. From another perspective it is asking whether the feature vectors are able to effectively contain the imprint of the social and cultural forces that they believe are inherent in the spatial layout. Space syntax measures, such as betweenness centrality and closeness centrality, are derived from the spatial organization graph using graph theoretic methods. Metric measures such as room area and room isovist area are derived from the floor plans. Further, the research seeks a method to "characterize abstract architectural qualities in terms of quantifiable spatial features." The two qualities chosen were spatial privacy and spatial intimacy. Labeling of the spatial qualities for a training data set required expert human evaluation which limited the size of the data set. Classification experiments predicted the privacy rating of the input space with modest success. Several problems may have impacted the results, the complexity of the models, the size of the data set, and chosen features of the vector representation.

Similar to the work by Ferrando but without employing space syntax measures, the search for evidence of architectural spatial qualities within graph representations has also been conducted with machine learning node embedding algorithms in lieu of hand-crafted feature vectors (As, Pal, & Basu, 2018). This research scored a small set of speculative house designs for "liveability" and "sleep-ability" qualities. The unsupervised random walk node embedding algorithm, node2vec, which will be discussed in detail in Vignette 2, was used to construct models intended to capture those qualities. The reported results were encouraging, but as demonstrated by Ferrando's research above, the limitations of a small data set are significant. Therefore, the value of the research is the demonstration of possible methods with potential for deployment when larger data sets become available.

The combination of graph embedding and a limited number of space syntax measures has been successfully used to train a model to classify buildings by type. Exploring the relationship between "spatial configurations and typological traits," Ferrando, Dalmosso, Mai and Cardoso Llach trained a machine learning model to recognize the difference between mosque and monastery floor plans translated to graph representations. Samples from the data set are shown in Figure 10 (Ferrando et al., 2019). In contrast to previous work, the process made no explicit attempt to classify by abstract architectural characteristics. Instead, at least in concept, those characteristics were learned by the graph embedding algorithm and present in the vector space representation of each building's spatial organization graph. Classification experiments of the graph embeddings proved to be very successful.

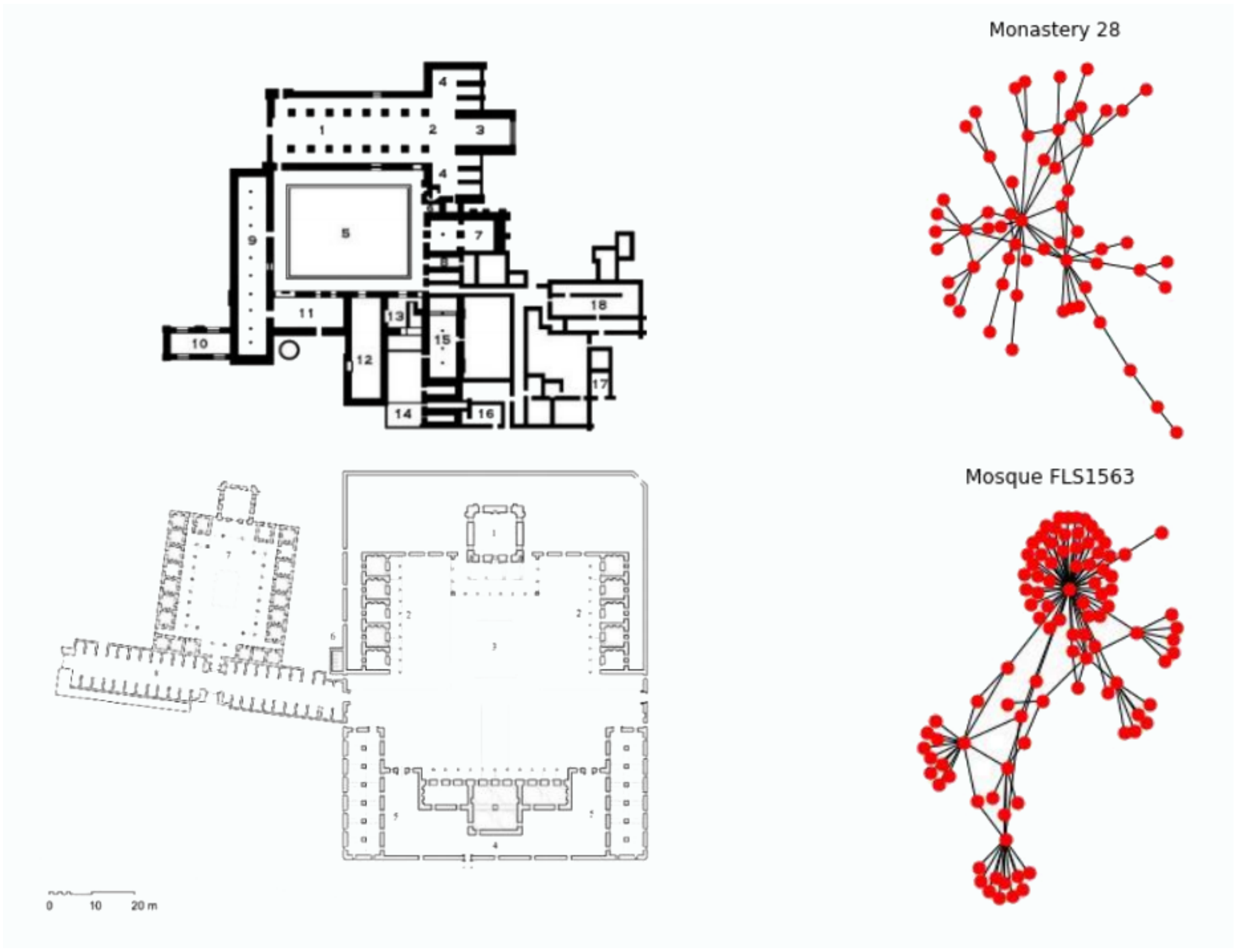


Figure 10: Samples from Ferrando data set (Ferrando et al., 2019)

In addition to classification, machine learning applications using node embeddings have been tasked with discovering clusters of spaces comprising common related functional assemblages. As et al. hypothesized that these relational substructures could operate as functional design building blocks in a generative system (As et al., 2018). Because the function of a node was essential to the vector representation, a supervised embedding algorithm was employed. Although a generative system was not proposed, the concept of identifying and classifying higher level substructures, essentially a functional pattern language, within a data set of architectural plans presents a promising line of research.

2.2.4 Data-driven Generative Systems

Researchers and designers see different opportunities for structured data and graph representations of spatial organization when used to drive generative systems. Data sets of floor plans, graph embedding algorithms and generative machine learning systems offer new and intriguing possibilities. Researchers are motivated by the desire to increase efficiency and democratize the floor plan design process and provide a co-design AI for designers (Hu et al., 2020; Nauata et al., 2020; Cheng, 2021). Researchers also use floor plan design to experiment with particular types of geometric packing problems or use design in general as an AI research problem (Para et al., 2020; Cheng, 2021). The research groups are interdisciplinary with individuals being associated with both computer science and design academic departments. The creators of the House-GAN projects, in particular have a strong computer science and software industry bias. For example, the lead researcher, Nelson Nauata, is a PhD student in the computer science department of Simon Fraser University and is located within the lab of Yasutaka Furukawa. In fact, only one of the six authors of the paper, Chin-Yi Cheng, have any architectural education listed in their online profiles. The lab itself and several of the researchers are also affiliated with Autodesk. Additionally, the computer vision tools integral to the House-GAN projects where both developed within the Yasutaka Furukawa lab.

Underlying this work is the argument that machine learning models have the potential to learn the design rules implicit in the human created design examples used for training (Wu et al., 2019). In other words, by pairing actual floor plans with their adjacency graphs, the models "learn" the skill and design knowledge that the human designers used to create the plans, ostensibly from the adjacency graph. Using the rules a generator can create new, yet unseen floor plans from new adjacency graph input. Through this learning process the data-driven generator may avoid altogether the need to create the explicit rules for spatial subdivision that have been the focus of graph theoretic floor plan generator systems.

The developers of the generators employ several strategies to evaluate the results of their generators. It should be noted that these are evaluations of the generated output, and are not related to, at least not directly, the error that is calculated during model training. Three evaluations methods are used - compatibility, diversity and realism. Compatibility is a measure of the similarity of the output adjacency graph to the input adjacency graph. It is determined by calculating the graph edit distance (GED). Although computationally complex, GED represents the minimum number of edits required to transform a source graph into a target graph. Individual outputs are calculated, but results can be aggregated to gauge the performance of the generator in this category. Diversity is a measure of output image variety in aggregate. It is determined using the Fréchet Inception Distance (FID). FID compares the distribution of generated images with the distribution of the real images that were used to train the generator. FID is applicable only for GAN generators using images. Para argues that "The FID score correlates most strongly with the adjacency statistics, since adjacencies can be captured by only considering small spatial neighborhoods around corners and walls of a floor plan, but does not capture topology or room shape statics accurately that require considering larger-scale features" (Para et al., 2020). Realism is a subjective measure with project dependent criteria. However, the realism measure always engages human evaluators to score the quality of the output as an architectural floor plan. Although entirely subjective, the *floor planness* of the output is arguably the key criteria of data-driven generators. The quality relates directly to the motivating claims that the generators can create plans meaningfully equivalent to human created plans. It can also only be indirectly addressed within the generator since human evaluation in not differentiable.

Generative Adversarial Networks (GANs) are one type of neural network model that has recently attracted a lot of attention, both from researchers as well as in the popular press (Goodfellow et al., 2014). The most well known methods work with raster images. Seemingly real but entirely fictional photographs created with GANs can be unnervingly convincing. In the architectural design domain, GANs using raster images have generated floor plans with suggestive but not entirely satisfactory results (Chaillou, 2019). The fact that only raster images are used is essential to this work. Transforming the image of a floor plan into another completely different floor plan is computationally no different than transforming a cat image into a racoon image. The mechanism gains no understanding of the spatial structure represented in the plan. Instead it operates solely within the Euclidean space of the pixels. Its objective is to generate an arrangement of pixels that is similar enough to learned pixel arrangements to be convinced that the generated pixels are of the same class. This is a fundamental difference with the GANs that generate spatial compositions using graph structures.

Four recent generative projects using graph structures demonstrate the potential of data-driven floor plan generation and each will be placed in the context of the themes outlined above. Sample results of each project are shown in Figure 11. The first project was published in 2019 and is the project that launched the RPLAN data set (Wu et al., 2019). The work takes inspiration from methods of automatic generation of floor plans for virtual environments and games. Floor plan generation is defined simply as "...the process of determining the position and size of several rooms." The generator follows a two-stage process of first locating the room center points within a given boundary, followed by sub-dividing the space within the boundary. This process is claimed to "imitate the human design process." Rooms are located sequentially, starting with the living room, using an iterative model employing three deep neural networks. Walls are located with an autoencoder network and are transformed to vector format with a post-processing step. Only a realism criteria was used to evaluate the results. Several pairwise comparison surveys were conducted using the generator results, the results of other cutting-edge floor plan generators, and samples from the training data. Survey participants were asked to choose the plan they felt was more "plausible."

The Graph2Plan project expands on the user interactivity possibilities of the generator developed by Hu et al. and also uses the RPLAN data set (Hu et al., 2020). Specifically, it allows a user to input a floor plan perimeter, room types and adjacency requirements. It then retrieves samples from the data set that satisfy the requirements, allows for adjustment, and generates floor plans compatible with the input information. Here, the graphs are used as "informative templates for adjustment by [the] user." A graph neural network is used to embed the graph into a vector space. The floor plans are generated by a deep neural network, trained on the data set, that primarily operates to fit room bounding boxes into the plan perimeter, following what the authors call "design principles carried in the layout graph." The tool is proposed for game designers and "end users, where they wish to explore early design intents, feasibility analyses, and mock-ups." Computer vision, in particular the Raster-to-Vector tools are used to translate the raster images of RPLAN into a vector format prior to translation into a graph format. Again, a user study is conducted to determine the relative success of the results using a pairwise comparison and "plausible" as the evaluation criteria.

House-GAN (Nauata et al., 2020) and its descendent House-GAN++ (Nauata et al., 2021) were developed by researchers at Autodesk together with others at academic institutions. Both were developed as part of a "co-design with AI" program intending to insert artificial intelligence tools into a design workflow. The stated goals are to avoid interruptions of a designer's flow, decrease repetitive work, ease difficult work and lessen the expense of costly work (Cheng, 2021). These goals are positioned as a response to broad generalizations about a conventional design process.

> "House design is an expensive and time-consuming iterative process. A standard workflow is to 1) sketch a "bubble diagram" illustrating the number of rooms with their types and connections; 2) produce corresponding floor plans and collect clients feedback; 3) revert to the bubble diagram for refinement, and 4) iterate. Given limited budget and time, architects and their clients often need to compromise on the design quality. Therefore, automated floor plan generation techniques are in critical demand with immense potentials in the architecture, construction, and real-estate industries" (Nauata et al., 2020).

House-GAN uses the LIFULL HOME'S data set and the Raster-to-Vector CV tools to convert the raster images in the data set to a vector format. The graph representations extracted from the data set are referred to as bubble diagrams, further emphasizing the purpose of the design tool. A published video demonstrates a computer interface in which a user created bubble diagram in one panel, results in a selection of floor plans in another panel. [2] These bubble diagrams represent the physical adjacencies of rooms and do not

[2] www.youtube.com/watch?v=Wo-8cC_aY4E

imply a physical connection.

Per the name, the model is based on a generative adversarial network in which the generator and discriminator models are convolutional message passing neural networks (Conv-MPN). The graph nodes are not embedded into a 1-dimensional vector space but are instead represented as volumes in the design space. The graph acts as a constraint on the generator. Because the GAN output is a raster image, computer vision (OpenCV) is used to draw bounding boxes around the rooms and transform them to rectangles. The results were evaluated on realism, diversity and compatibility. For the realism evaluation, a group of students and architects selected the "better" plan of a pair. Pairs were composed of House-GAN results, ground truth plans and the results of three other generator.

House-GAN++ modifies several important aspects of House-GAN. First, the input graph is revised from an adjacency graph recording physical adjacency of spaces, to a graph only recording physical access between spaces, i.e. doors. This reduction of connections creates star shaped graphs with few cycles, or looping paths within the graph. The star shape is in contrast to the triangulated tessellation form of the House-GAN graphs. Also impacting the graph representation, an 'exterior node' is added and connected to the living room node with an edge representing the entrance door. Second, the LIFULL HOME'S data set was replaced with the RPLAN data set. As noted above, RPLAN is a more uniform data set created specifically to support machine learning and spatial organization projects. To the House-GAN network a conditional GAN has been added. The conditional GAN uses a previously generated partial model as the next input constraint, enabling iterative layout refinement. Finally, instead of OpenCV, another computer vision application capable of creating non-rectangular rooms, Floor-SP, is used to vectorize the raw raster output.

The last project does not position itself as any sort of design tool, but instead addresses floor plan generation as a class of layout generation within the larger class of content generation. The goal is "topologically and spatially consistent layout generation" through a fully automated process. Additional user constraints such as required room properties and perimeter footprint may be added to the generative process. Both the LIFULL HOME'S and the RPLAN data sets are used. Instead of a generative neural network architecture like a GAN or a variational autoencoder, a transformer architecture that builds graphs sequentially is used. Room size and position are finalized using an optimization process. Unlike the image-based generators discussed above, this method of building the graph and optimizing rectangular shapes is reminiscent of the generator discussed in Vignette 1. Evaluation compares generated results to ground truth with two metrics, FID and layout statistics. Layout statistics are node and edge attributes and graph properties extracted from the database plans. Output is mapped to an image to perform the FID analysis. A formal subjective evaluation of results is not performed.

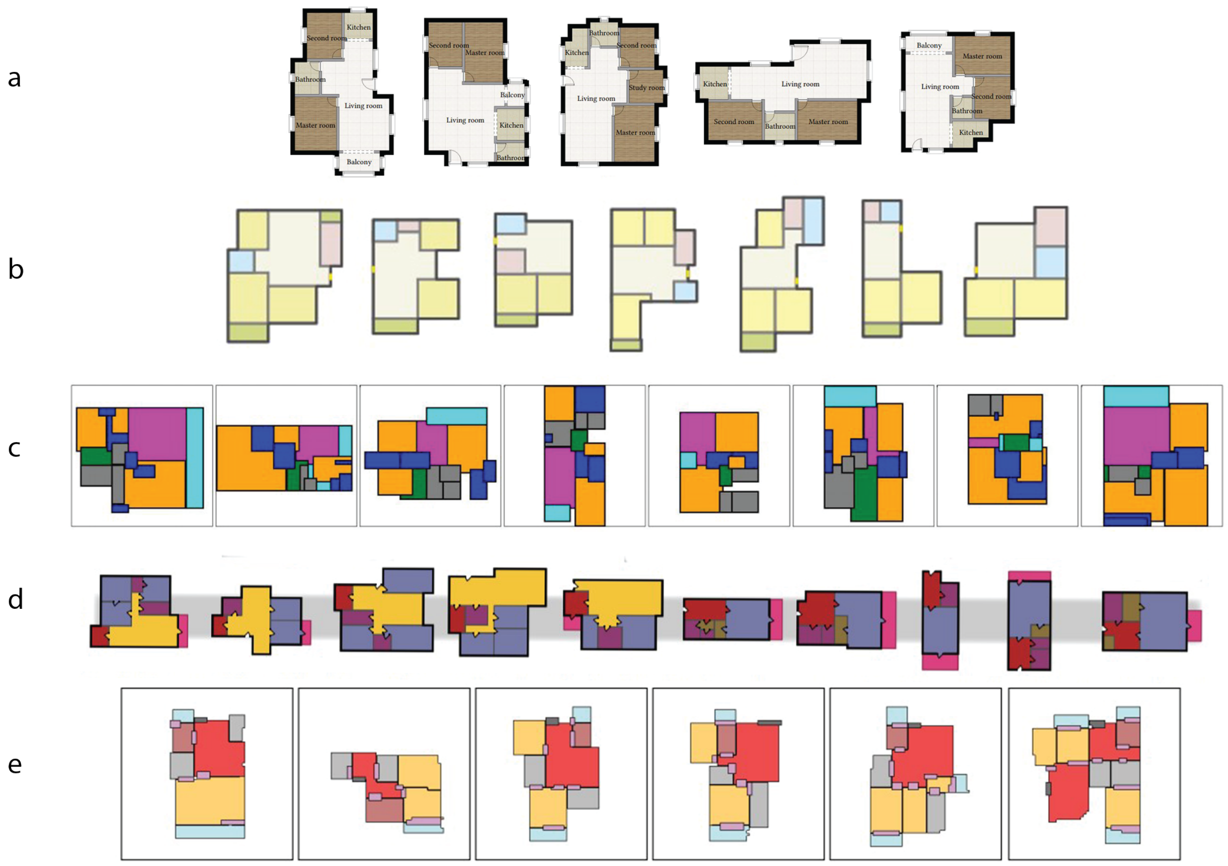


Figure 11: Representative examples of state of the art data-driven floor plan generators. a) (Wu et al., 2019), b) (Hu et al., 2020), c) (Nauata et al., 2020), d) (Para et al., 2020), e) (Nauata et al., 2021)

2.2.5 Summary

Although the world seems awash in data, the lack of floor plan data sets hinders machine learning research into spatial configuration. Small data sets, such as those used by Ferrando et al. limit the strength of results in analytic research. Large, but highly idiosyncratic data sets, such those containing solely Japanese rental apartment, raise questions about the applicability of design solutions generated from them. Translating floor plans from physical and digital formats into digital formats containing semantic information such as walls and room function is a challenging technical problem and has parallels in many other research domains. But it is more than a technical problem. Translation methods like computer vision automate a process that often requires expert human skill and knowledge judgement. Therefore, the decisions made by a tool's designers greatly influences the information contained in the data set made with that tool. Vignettes 2 and 3 will explore two different data sets using analytical and generative modes of research.

Vector encoding of graph and node data offer potential new computational access to structured data unavailable through graph theoretical techniques. Classification research conducted using vector embedding of spatial topology suggests new avenues to pursue the Centre for Configurational Studies' project of a natural history of architecture classified by configuration and shape. Vignette 2 takes up this idea and looks closely at one particular embedding algorithm.

Like graph theoretical floor plan generators, data-driven generative systems use graph structures to contain and input an architectural program into the system. However, instead of generating plans using human crafted explicit rules and procedures, data-driven systems use rules inferred from the rules implicitly contained in the real floor plans. Therefore, the data-driving systems function as a pairing of a particular learning method and particular data set. Although the pairing suggests a similarity to the exhaustive method of generative design reconstructed in Vignette 1, the exhaustive method does not learn anything from the catalog of dissections. Vignette 3 proposes a data-driven floor plan generator to explore the concept of extracting implicit floor plan design rules.

3 Methods

The Methods chapter is broken into three computational vignettes. The vignettes are computational instruments to ground and deepen the investigation of concepts presented in the Background sections. By working directly with some of the computational ideas, a richer and more nuanced understanding of the technology is achieved.

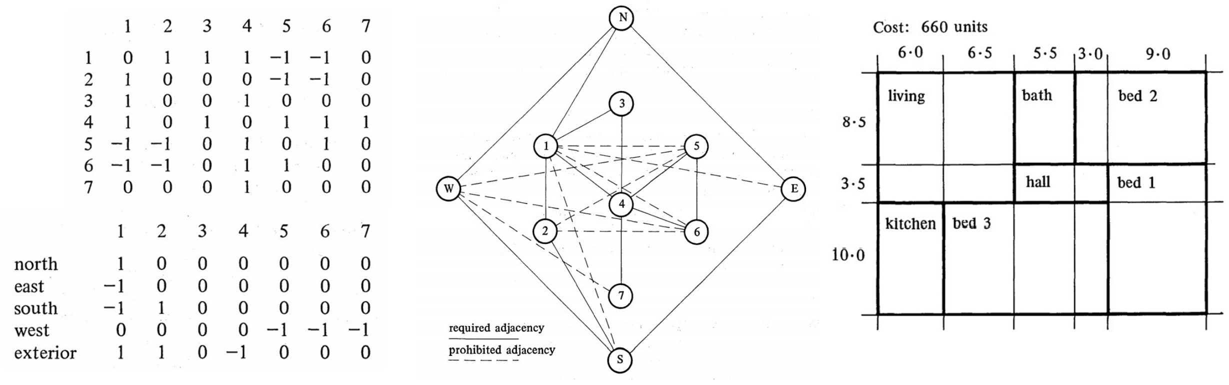
- Vignette 1 - Dissecting and Enumerating, reconstructs software to uncover some of the foundational spatial representation concepts from the 1970's encoded in the algorithms.
- Vignette 2 - Encoding Configuration, investigates graph embeddings of spatial configurations through machine learning classification experiments.
- Vignette 3 - Generating with Data, explores a large-scale floor plan data set using a date-driven generative system.

Vignette 1 - Dissecting and Enumerating, reconstructs software to uncover some of the foundational spatial representation concepts from the 1970's encoded in the algorithms. Vignette 2 - Encoding Vectors and Vignette 3 - Generating with Data use analytic and generative machine learning approaches, respectively, to explore graphs, vector embedding and data sets.

3.1 Vignette 1 - Dissecting and Enumerating

3.1.1 Introduction

Vignette 1 reconstructs selected algorithms from the body of work created by architects exploring mathematical representation and computation of spatial organization during the 1970's. The concept of software reconstruction as a mode of investigation was introduced by Cardoso Llach and Donaldson (Cardoso Llach & Donaldson, 2019). Their research proposes that new understandings of contemporary design practices can be gained by "combining



	1	2	3	4	5	6	7
1	0	1	1	1	−1	−1	0
2	1	0	0	0	−1	−1	0
3	1	0	0	1	0	0	0
4	1	0	1	0	1	1	1
5	−1	−1	0	1	0	1	0
6	−1	−1	0	1	1	0	0
7	0	0	0	1	0	0	0

	1	2	3	4	5	6	7
north	1	0	0	0	0	0	0
east	−1	0	0	0	0	0	0
south	−1	1	0	0	0	0	0
west	0	0	0	0	−1	−1	−1
exterior	1	1	0	−1	0	0	0

Figure 12: Selected images from “Synthesis and Optimization of Small Rectangular Floor Plans.” Adjacency and orientation requirement matrices, adjacency requirement graph, and possible solution.

historical research and creative prototyping." In this vein, the reconstruction aims to reveal how particular concepts of architectural space and building configuration were encoded into the algorithms. In addition to revealing aspects of the algorithms, the reconstruction allows a direct interaction with the particular mode of design practice proposed by the software.

As a demonstrative reconstruction, I selected a graph and plan generation technique described by Mitchell, Steadman and Liggett in their paper “Synthesis and Optimization of Small Rectangular Floor Plans” (Mitchell et al., 1976). The work ties together many important ideas outlined in Steadman’s book, Architectural Morphology (Steadman, 1983). Ideas such as the “‘dimensionless’ representation of rectangular plans,” symmetry and counting are critical to the generative algorithms described in the paper. Equally important are concepts such as adjacency graphs, adjacency requirement matrices, orientation requirement matrices, and their correlation to spatial arrangement. Examples of these concepts are shown in 12. Absent though is any discussion of plan graphs and their dual graph relationship to adjacency graphs featured in Steadman’s earlier “Graph theoretic representation of architectural arrangement” (Steadman, 1973). The enumerations of all possible permutations of a particular class of rectangular floor plans, created by hand in that paper, are generated by a computer program developed in conjunction with his co-authors. The computer allowed all permutations of plans up to and including eight rooms to be enumerated, a cumulative total far beyond what could be created by hand.

In chapter 9 of Architectural Morphology, Steadman makes a distinction between “selective” and “constructive” methods of floor plan generation. “Constructive” methods build a plan to satisfy adjacency requirements. “Selective” methods, such as the one being reconstructed here, find one or more satisfactory plans from an ostensibly complete catalog of existing plans. Plan generation occurs in two stages after the catalog of dimensionless plans has been created and saved. First, all dissections that satisfy the designer’s adjacency requirements are collected. Next, although not demonstrated here, these dimensionless plans are tested against the designer’s room size requirements.

3.1.2 Basis of Reconstruction

The authors provide only a few details of how the algorithms were originally implemented in a computer. For example, the code was written in Fortran and at least some of the research was conducted on an IBM 360/91 computer located at UCLA. It is unclear whether all of the functionality described in the paper existed as a comprehensive computer application, or only existed as a disjoint set of functions and routines. The logic of my own code follows the structure of the paper. Although I could have arrived at similar results with an invented top down structure, I believe that incrementally developing the building blocks presented in the paper offers insight into the thought process and priorities of the authors.

All code was written in Python using Jupyter notebooks. The NetworkX package was employed for graph structures and manipulation. The Numpy package was employed for array structures and array manipulation. Rhino 6 and Grasshopper are used to visualize the dissections and graphs.

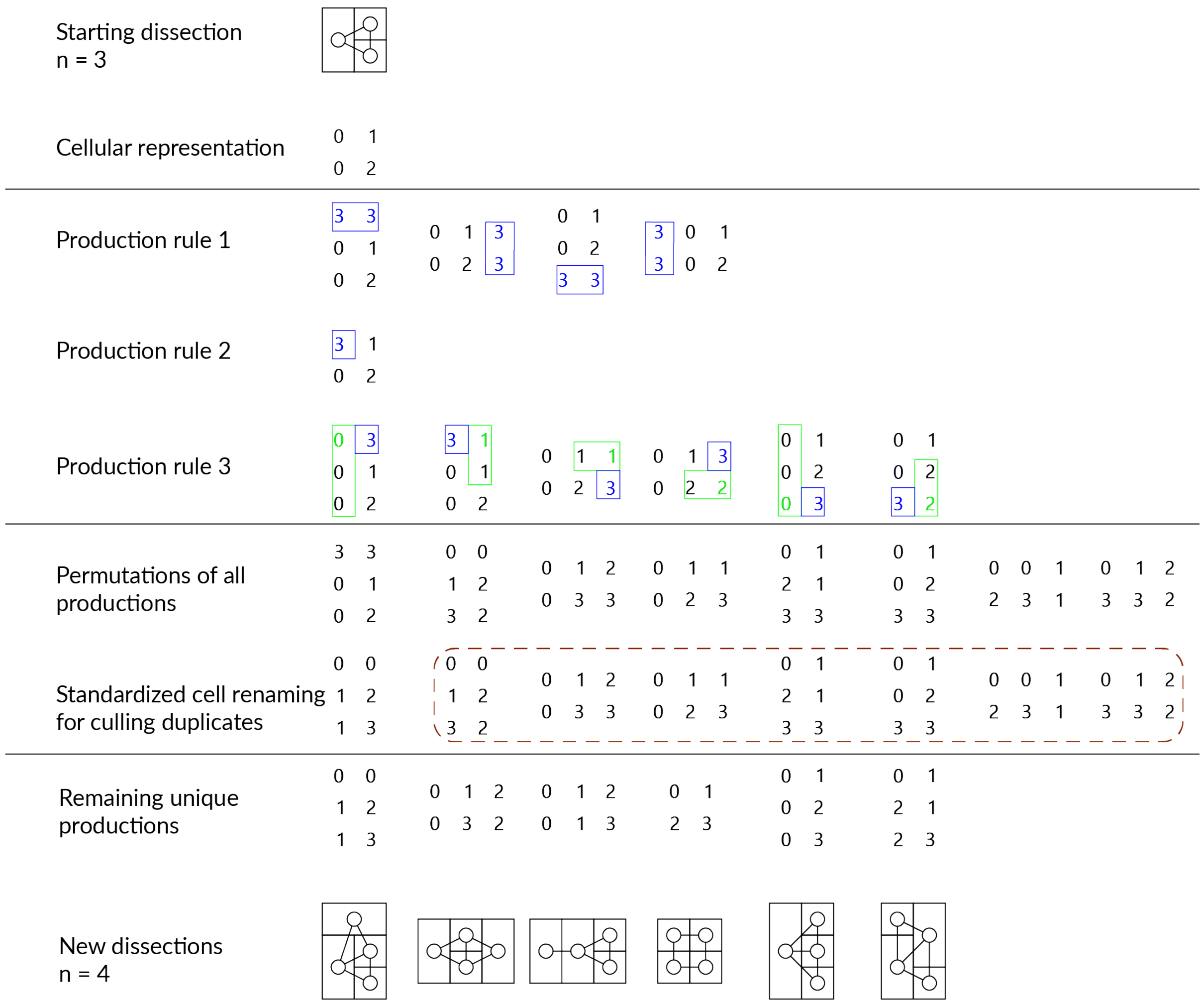

Starting dissection n = 3
Cellular representation
Production rule 1
Production rule 2
Production rule 3
Permutations of all productions
Standardized cell renaming for culling duplicates
Remaining unique productions
New dissections n = 4

Figure 13: Manipulation of cellular representations to generate new rectangular dissections. The permutation, standardization and removal stage are performed on all cellular representations generated by the production rules.

3.1.3 Producing the Dissections

Mitchell et al. describe three generative operations used to create topologically distinct rectangular dissections.

- Boundary addition
- Subdivision of component rectangles
- Component expansion and boundary addition

All generative operations manipulate 1-dimensional or 2-dimensional array representations of the dissections. These operations produce large numbers of duplicates, therefore the identification and removal of duplicates is a critical aspect of the generative process. Thus, culling is also the most computationally expensive. After all of the unique dissections have been found, a corresponding adjacency graph is created for each.

Generation is iterative and cumulative and begins with a base dissection of n = 2. Each successive generation of dissections with n + 1 components uses the complete series of dissections with n components as input.

Boundary Addition The first generative operation adds a new component along each perimeter edge of the input dissection. The new component takes on the next largest integer value of the input dissection. Four new dissections with n+1 components are created and returned. Refer to Algorithm 1.

Algorithm 1 Production: Boundary Addition

```
1: procedure ADD($D$)                                ▷ Create and return four arrays
2:     $x \leftarrow$ maximum value in $D + 1$
3:     $D_t \leftarrow D +$ row of $x$ values on top
4:     $D_r \leftarrow D +$ column of $x$ values on right
5:     $D_b \leftarrow D +$ row of $x$ values on bottom
6:     $D_l \leftarrow D +$ column of $x$ values on left
7:     return $D_t, D_r, D_b, D_l$
```

Subdivision of Component Rectangles The subdivision operation splits each input dissection component greater than two cells into two separate components. The new component takes on the next largest integer value of the input dissection. If the dimensions of the component are greater than two in both dimensions, the component is split separately in both directions. Multiple horizontal and vertical splits will occur if a component has greater than two cells in any direction. Importantly, each split is conducted separately, not cumulatively. A new dissection with n+1 components is returned for each split. Refer to Algorithm 2.

Algorithm 2 Production: Subdivision of Component Rectangles

```
procedure SUBDIVIDE($D$)                                  ▷ Subdivide components $>$ 2 cells
    $m \leftarrow$ minimum value in $D$
    $x \leftarrow$ maximum value in $D + 1$
    $list \leftarrow$ empty list
    while $m < x$ do
        $C_{Dm} \leftarrow$ component $m$ in $D$
        if $C_{Dm,no.ofrows} > 1$ then
            for $row \leftarrow 1, C_{Dm,no.ofrows} - 1$ do
                $D' \leftarrow D$
                $C_{D'm}, C_{D'x} \leftarrow C_{D'm}$ subdivided at $row$
                append $D'$ to $list$
        if $C_{Dm,no.ofcolumns} > 1$ then
            for $column \leftarrow 1, C_{Dm,no.ofcolumns} - 1$ do
                $D' \leftarrow D$
                $C_{D'm}, C_{D'x} \leftarrow C_{D'm}$ subdivided at $column$
                append $D'$ to $list$
        $m \leftarrow m + 1$
    return $list$
```

Component Expansion and Boundary Addition The final generative operation both extends input components and adds a new component separately on each perimeter edge bordering more than one component. For example, along an edge with two border components, cells are added along the length of the first component's perimeter edge. The value of these cells matches the input component, thereby expanding the size of that component. A new component is added along the balance of the boundary edge. The new component takes on the next largest integer value of the input dissection. Where more than two components border a perimeter edge, the process is repeated along the perimeter edge. One more component is expanded at each iteration while only one new component is ever added at each iteration. Because this process is directional along each perimeter edge, it is necessarily conducted in both directions, i.e. both left and right, or both up and down. A new dissection of n+1 components is returned for each expansion-addition event. Refer to Algorithm 3.

Algorithm 3 Production: Expansion and Boundary Addition

```
procedure EXPAND AND ADD($D$)                              ▷ Expand 1 component and add 1 component
    $x \leftarrow$ maximum value in $D + 1$
    $list \leftarrow$ empty list
    $n \leftarrow$ number of components in $D_{toprow}$
    if $n > 1$ then
        $comp' \leftarrow$ components in $D'_{toprow}$
        for $c \leftarrow comp'$ do                                        ▷ left to right
            $D' \leftarrow D$
            $row' \leftarrow D'_{toprow}$
            $row'_{leftside} \leftarrow$ components left of and including $c$
            $row'_{rightside} \leftarrow c_x$                              ▷ new component
            $row' \leftarrow row'_{leftside} + row'_{rightside}$
            $D' \leftarrow D'$ with $row'$ inserted above $D'_{toprow}$
            append $D'$ to $list$
        for $c \leftarrow comp'_{reversed}$ do                             ▷ right to left
            $D' \leftarrow D$
            $row' \leftarrow D'_{toprow}$
            $row'_{right} \leftarrow$ components right of and including $c$
            $row'_{left} \leftarrow c_x$                                   ▷ new component
            $row' \leftarrow row'_{left} + row'_{right}$
            $D' \leftarrow D'$ with $row'$ inserted above $D_{toprow}$
            append $D'$ to $list$
    Repeat similar procedure for $D_{rightcolumn}, D_{bottomrow}, D_{leftcolumn}$
```

Remove Duplicates As noted above, removal of duplicates is critical to the goal of an accurate enumeration. Duplicate dissections are those that are topologically indistinct when disregarding the labeling of the components, rotation and reflection. Notably, topologically distinct dissections may have isomorphic graph representations. This is because of the different relationships the rectangular components may have with the perimeter rectangle. Duplicates are removed in two stages during generation. First, the duplicates are removed from the series of dissections produced by the three generative operations applied to each input array. The results are added to an intermediate list of results for that generative pass, e.g. a list of five component dissections derived from all four component dissections. Second, duplicates are removed from the intermediate list. Finally, the now unique dissections are added to the master list of dissections.

Duplicate removal requires three preparatory steps to standardize the arrays and ensure that all can be reliably compared. The first, dimensional minimization, occurs only once for each newly created dissection. For arrays where n $>$ 5, the production rules occasionally create dissections with a redundant row or column. I define a redundant row or column as one that can be removed without modifying the topology of the dissection. Potential redundant rows or columns are those in which every value is part of a multi-cell component. To keep the check simple, I employ a NetworkX implementation of the VF2 algorithm to compare the graph representation of the dissection before and after the potential row or column is deleted from the array. If the size reduced version retains isomorphism with the original, it replaces the original.

The second and third steps are completed each time a new dissection is compared to those in the unique list at both stages of the duplicate removal process. The second step rotates the new dissection 90 degrees four times and then mirrors each rotation resulting in a series of eight arrays topologically identical to the new dissection. The third step standardizes the labels of each permutation. Label standardization is required because the dissection arrays are compared element-wise. Working left to right and top to bottom, the components are relabeled in order starting with zero.

If none of the standardized permutations element-wise match any of the dissections already in the unique dissection list, the standardized version of the new dissection is added to the list of unique arrays. Refer to Figure 13 for a visualization of the steps.

Saving the Dissections The authors suggested that the enumerated dissections should be classified and a storage system devised that utilizes the classifications. The classification system would facilitate retrieval and analysis of individual and groups of dissections. I did not classify the dissections and simply stored them in a list format using Python's pickle serialization library to maintain their Numpy array object structure. Once the dissection arrays are translated to graph representations they are no longer needed. Adding classification attributes to the graph data structure is a more efficient recording method should this information be useful in the future.

3.1.4 Dissections to Adjacency Graphs

Two forms of graph data structures are created and utilized. First, from each dissection a Python dictionary is created. Numpy methods are used to identify collections integers forming the components. Each of these components are represented as nodes in the graph. ID attributes are added to the graph as attributes. The geometric corners of each node are recorded as attributes as well. Edges are created between nodes when the components are adjacent to each other in the dissection array. Edge direction is recorded as an attribute with each edge. Edges directions may be either East-West or North-South. Because the graphs are not oriented at this stage, East-West and North-South could have been any arbitrary designation. The standard dictionary structure is sufficient for most tasks requiring a graph format. However, where sophisticated analysis is required, the dictionary is converted to a NetworkX graph object. Analyses such as isomorphic comparison and planarity checking are conducted with NetworkX graphs. A complete list of adjacency graphs is saved along with the list of dissections.

3.1.5 Dissection Assignment

Searching the catalog for dissections that may satisfy the design specifications requires two inputs from the user - an adjacency requirements matrix and an orientation requirements matrix. A valid input is confirmed with several checks before a search is conducted. The required adjacency matrix contains the "basic functional and circulation considerations" of each space within the design. Within the matrix a '1' indicates a required adjacency, a "-1"

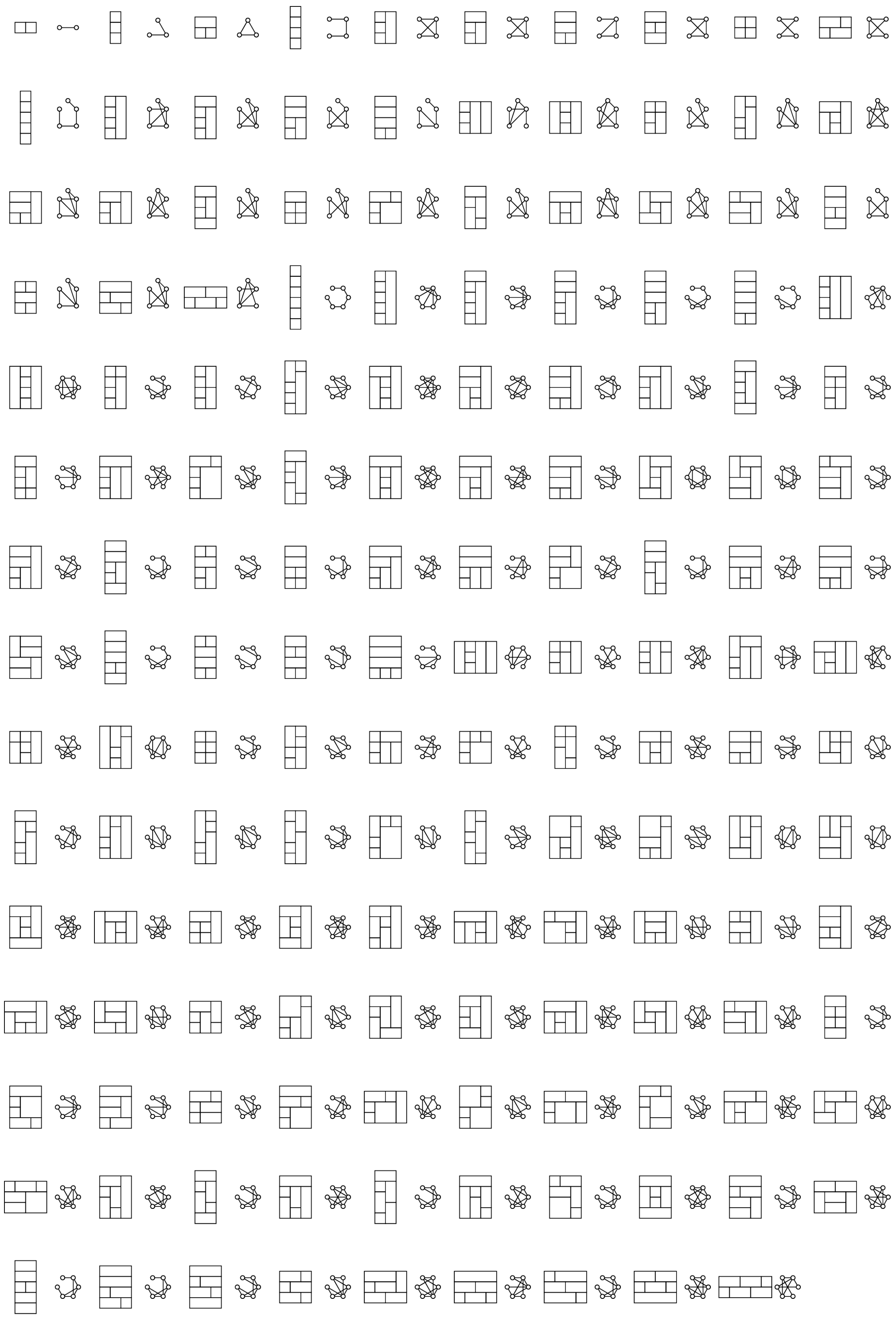


Figure 14: Reconstruction of unique rectangular dissections through n=6

indicates a disallowed adjacency, and a “0” indicates a lack of requirement. Each row and column is associated with a space.

The rows of an orientation requirement matrix represent the cardinal directions with an additional row representing a general exterior adjacency. The columns represent each space. Values within the matrix are defined as in the adjacency requirements matrix.

Validating the Input First, the two input matrices are converted to a single NetworkX graph structure containing all the nodes and edges, including exterior. Each contains a requirement attribute. This graph is then tested to ensure it can be embedded in a plane and can create a rectangular dissection. A NetworkX function checks for planarity. Rectangular dissection potential is checked by confirming that the graph does not contain any cliques larger than three.

Filtering Potential Dissections Immediately, all dissections without the correct number of nodes are removed from consideration. The degree sequence of each remaining dissection is compared against that of the requirement graph. Should the dissection degree sequence not provide the necessary connectivity, that dissection is removed from consideration.

Assignment The assignment algorithm is only lightly covered in the paper but is the key functionality connecting the catalog of dissections to a floor plan design method. Since the cataloged dissections are not oriented, the first assignment step creates eight rotated and mirrored permutations of each filtered dissection. This process adds north, east, south and west nodes and edges to the dissection graph. The goal of the assignment algorithm is to find all successful mappings of the adjacency requirements graph to catalog of dissections. The number of permutations to check becomes intractable even with relatively low numbers of potential dissections. Therefore, heuristics are employed to eliminate search paths that are guaranteed to fail. For each dissection graph, the process starts by assuming that each adjacency requirement node can be mapped to each dissection node. Two simple checks are then made. If the degree of the dissection node is not equal to or greater than the degree

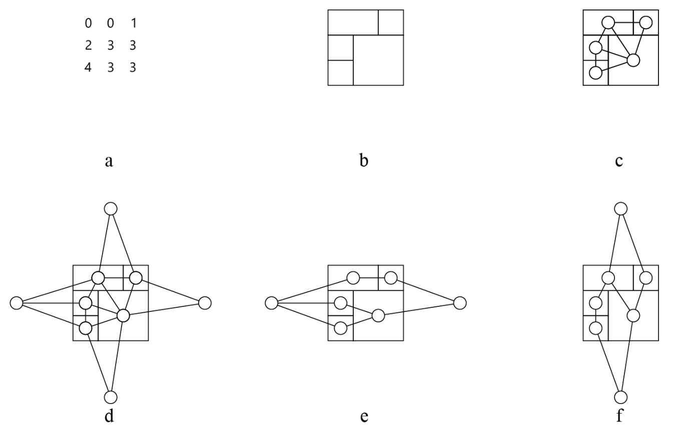


Figure 15: Graphic representations: a) cellular representation; b) rectangular dissection; c) adjacency graph extracted from dissection; d) exterior nodes added to adjacency graph; e) half-graph with east-west adjacencies; f) half-graph with north-south adjacencies

of the adjacency requirement node mapped to it, the adjacency requirement node cannot be mapped to that node. Second, if any dissection node does not have edges to the exterior nodes required by the adjacency requirement node, then the adjacency requirement node cannot be mapped to it.

The last heuristic further reduces unnecessary computation by creating a list of "start nodes." I define a start node as a node-to-node mapping that must be present for the remainder of the mapping to be valid. For instance, if a particular adjacency requirement node has a degree of four, and only one of the dissection nodes has a degree of four, that mapping is a start node. Only permutations that contain these start node mappings are checked. For a reasonably constrained adjacency requirement graph, these heuristics greatly reduce the search time of the assignment process. After the duplicates are culled, the list of successful dissection graphs, the list of mappings, the adjacency requirement graph, the adjacency requirement matrix and the orientation matrix are saved. All graphs are converted to JSON dictionaries prior to being saved.

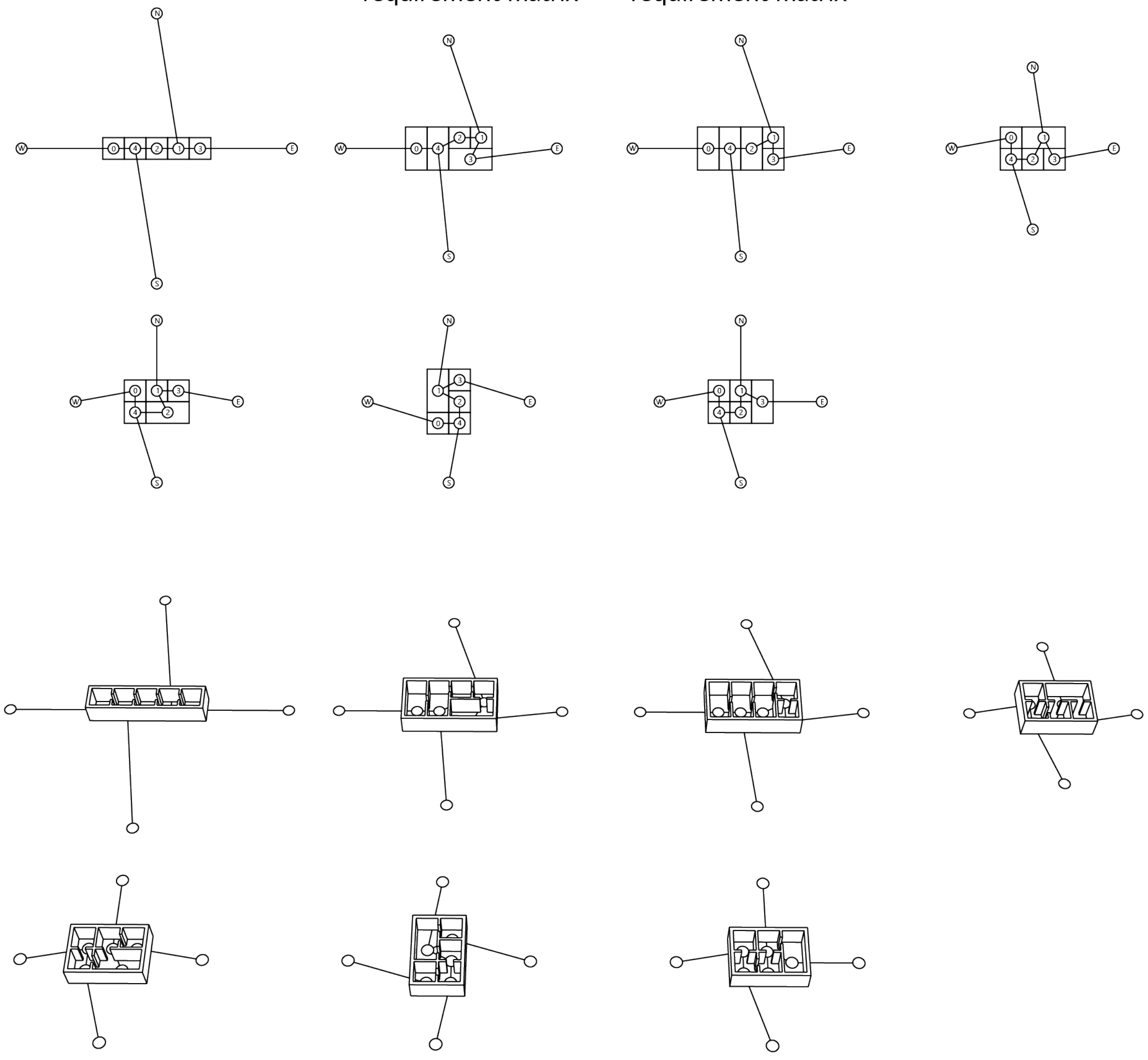
User input

	0	1	2	3	4
0	0	0	-1	-1	1
1	0	0	1	1	-1
2	-1	1	0	0	1
3	-1	1	0	0	0
4	1	-1	1	0	0

Adjacency requirement matrix

	0	1	2	3	4
N	0	1	-1	0	0
E	0	0	-1	1	0
S	0	0	-1	0	1
W	1	0	-1	0	0
Ex	1	1	-1	1	1

Orientation requirement matrix



System output - requirements assigned to cataloged dissections

Figure 16: Assignment of adjacency and orientation requirements to rectangular dissection

3.1.6 Visualizing Dissections and Graphs

The Rhino / Grasshopper platform and its Python scripting interface is used to display the dissections and graphs. GHPython does not support Numpy and NetworkX, therefore, no sophisticated graph or array analysis or manipulation can be executed in Grasshopper. The graphs and adjacency requirements are imported into Grasshopper as JSON files and CSV files respectively. GHPython scripts create geometry from the graphs. The rectangles, nodes and edges are styled after the graphics in the paper and other work published by Steadman. I did not attempt to create geometry similar to Steadman's hand drawn dissections because the graphic construction rules appear loose and idiosyncratic.

In addition to rectangular representations, I created geometry for graph representations with nodes and edges, half-graphs, Steadman's hand drawn stylized graphs and integer arrays. Figure 15 provides examples of various display possibilities. The geometry scripts are used to display both the dissection catalog and assignment graphs and dissections. To reinforce that the assignment graphs and dissections describe floor plans, three-dimensional wall geometry is created for each of the plans.

3.1.7 Results

Enumerations I used the dissection generation algorithms to create dissections with up to eight components. The results were inline with the authors' findings. I manually checked the results for n <= 6 against the chart published in the appendix of the original paper and there were no discrepancies. Refer to Figure 14 for a matrix of all dissections produced by the algorithm for n <= 6. However, for dissections where n=7 and n=8 my code found slightly more unique dissections than those reported in the paper. Specifically, 2 additional n=7 dissections and 33 additional n=8 dissections were found, amounting to differences of 0.3% and 0.7%, respectively. Refer to Table 1 for tabulated results.

The authors do not claim that the algorithm produces exhaustive results, only that the computer-generated results up to n=6 correspond to those created by hand in previous

no. of components	3	4	5	6	7	8
results in paper	2	7	23	116	683	4866
results from reconstruction	2	7	23	116	685	4899

Table 1: Table of dissection enumeration results

research. Even if there were tables of dissections to compare, the sheer number of dissections makes such an effort practically impossible for the scope of this thesis. However, Mitchell et al. state that a rigorous exhaustive enumeration is not required, and that the results are sufficient.

Later work by Bloch, Krishnamurti and Roe arrived at consensus results counting rectangular dissections and that work is summarized in "Architectural Morphology" (Steadman, 1983). Using different algorithms, they found 24 n=5, 126 n=6, 815 n=7 and 6,465 n=8 dissections. Clearly, the algorithm reconstructed above does not enumerate all of the possible dissections, even when the rectangle count is relatively small.

Assignment The assignment of adjacency requirements to suitable rectangular dissections was also successful. Figure 16 demonstrates the results of a search conducted for a five room adjacency requirement matrix. The paper does not provide benchmarks against which to compare results. However, testing with numerous sets of adjacency and orientation requirements produced satisfying results. Rigorously checking that the results are complete becomes intractable for even for relatively small plans. Therefore, satisfactory indicates that the results appear complete based on expected results and manual exploration of options.

3.1.8 Discussion

Reconstructing the code following the structure of the paper was successful in that the process provided insights that I would not have discerned otherwise. This was most notable within the enumeration portion of the exercise. The large differences in quantities between the total possible dissections found by the authors and the total possible dissections found by Krishnamurti's color algorithms several years later, reflect a different concept of the enumeration problem. Although presented as cellular representations, the three production methods all follow the intuitive tactic of adding one room at a time to a set of valid floor plans. This strategy is inherently graph based as it expands a valid planar graph one node at a time without risk of loss of planarity. That is because the cellular representations, and by extension the geometry of rectangular dissections, provide a framework to maintain graph planarity. In a sense, it is an architectural approach to the problem. By this, I mean that the algorithm does not escape from the fact that it is creating spatial relationships. However, it is highly inefficient and ultimately does not achieve its goal of an exhaustive catalog of rectangular dissections. Paradoxically, the coloring algorithms appear to achieve better results far more efficiently by foregoing notions of rooms and room adjacencies at the conception of each dissection. The spatial relationships are resultant from the process rather than providing an originating source. Engaging with the dissection assignment tool as a design tool proved difficult. The paper does not contain an example of how the input was structured. A text-based matrix input method is consistent with the likely method used in the original implementation. Creating adjacency and orientation requirements in a matrix form is an unintuitive and challenging exercise.

For a small and familiar program such as a residence, I found myself first sketching spatial relationships before committing to the effort of translating the relationships to matrix form. The sketches unavoidably contained my thoughts on spatial and architectural qualities. They also reflected my habits and personal tendencies of room shape and proportion. Translation to matrix form excises this information and this suggests two possible designer interaction modes. First, to follow my experience, rather than creating initial plans from a source set of adjacency requirements, the tool can be used to quickly search for alternative configurations of a designer's initial sketch. Using the tool this way, a designer may find alternate configurations outside of the constraints of their subjective tendencies. Second, a designer could manipulate the adjacencies in the matrices to quickly modify the topology of the plans and potentially gain intuition of the impact of the adjacency constraints on the pool of satisfactory rectangular dissections. Such an approach could assist a designer to resolve competing and contradictory programmatic desires. Although addressing a highly limited class of floor plans, interacting with the tool revealed more exploratory potential than simply “selecting” from a set of compliant options.

Two additional steps would add research value to this reconstruction. First, the room size optimization algorithms should be implemented. Performance optimization is one of the downstream goals of the design of the computable representations like graphs. Reconstruction would foreground the interrelationships of the mathematical abstractions and optimization processes. Second, an interactive graphic interface should be created. With an easy-to-use interface, non-expert users could interact with the tool and the underlying spatial organization concepts.

3.2 Vignette 2 - Encoding Configuration

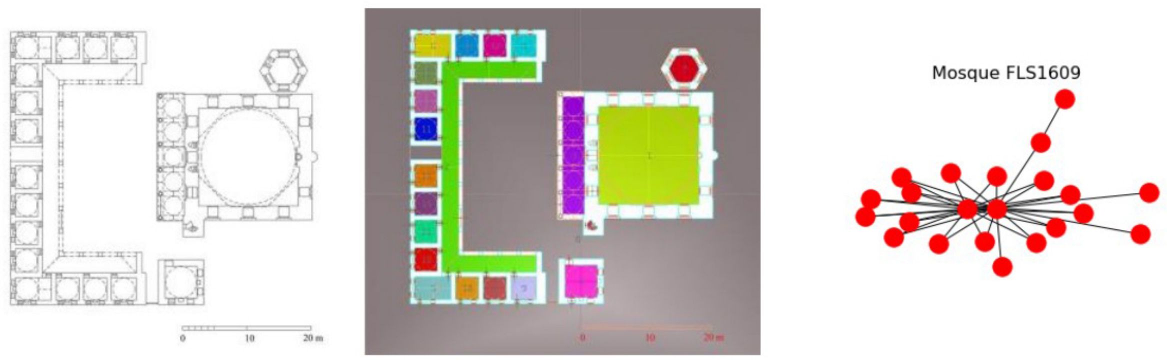


Figure 17: Data set sample (Ferrando et al., 2019)

3.2.1 Introduction

This vignette uses machine learning classification experiments to explore how unsupervised node embedding methods may allow new computational access to spatial structures by way of graph structures. I use the data set specifically created for classification experiments by

Ferrando, Dalmasso, Mai and Cardoso Llach (Ferrando et al., 2019). The data set contains CAD plans and supporting data of 19 monastery and 20 mosque complexes modeled by the researchers. A sample is shown in Figure 17. The buildings in the data set are spatially complex and average over forty rooms per building. I anticipate that the complexity provides greater variation in node neighborhoods leading to clearer differentiation of the embeddings.

3.2.2 Node Attributes

Space syntax analysis theory supports the use of both geometrical attributes like isovist area, and graph theoretical ideas (Ferrando, 2018; Ostwald, 2011). However, without a defined carrier node, metrics such as node depth, mean node depth and others pertaining to the justified plan graph are not measurable. Therefore, those measures are not used in the experiments conducted here. Devising a way to create features with these measures may provide another angle from which to investigate the graph structural qualities that unsupervised embedding techniques can learn.

Social network analysis (SNA) also provides methods for analyzing and quantifying node qualities within a graph and methods for measuring characteristics of subgraphs and entire graphs. Wurzer et al. propose a mapping between SNA measures and architectural programming and space planning concepts (Wurzer, Lorenz, & Wien, 2016). The justification of the mapping is supported by the space syntax theories of Hillier and Hanson, among others (Hillier & Hanson, 1984). However, the paper did not explore SNA analysis on a graph extracted from a building plan. Instead, it investigated the graph measures using a hypothetical spatial adjacency matrix and hypothesized that the technique could be used to evaluate programs, not plans. By evaluating programs, they argue, a designer can understand the functional performance of a class of buildings proposed by the program. Both node and graph level metrics are discussed. The node level metrics they chose involved measures of centrality, in other words, measures of the node's importance or influence within the graph. The space mappings list below shows the authors' mapping of room relative importance to common SNA measures of node relative importance. The building mappings list shows the authors' mapping of room group and building configuration characteristics to common SNA measures of graphs. The algorithms used to calculate the measures are not particularly important to this discussion. Instead, the list is intended to enumerate several of the specific spatial characteristics argued to be uncovered with various SNA measures.

- Space mappings:
  - Betweenness centrality: a measure of relative importance to circulation within the space plan
  - Closeness centrality: a measure of relative position within the space plan
  - Eigenvector centrality: a measure of importance, determined by proximity to other important spaces
  - Degree centrality: a measure of connectedness of the space
  - Local cluster coefficient: a measure of connectedness of the spaces in a neighborhood
- Building mappings:
  - Clique analysis: uncovers highly connected subgraphs of potential functional groupings.
  - Density: a measure of the relative connectedness of all spaces
  - Average local clustering coefficient: a measure of the relative connectedness of all spaces
  - Global clustering coefficient: a measure of the relative connectedness of all spaces

The hypothesis underlying the use of unsupervised node embedding algorithms is that they implicitly learn the node characteristics explicitly calculated by SNA centrality measures (Cohen, 2018; Grover & Leskovec, 2016). Additionally, they also embed within the node embedding vector its relationship to other nodes. The node2vec algorithm is used in this study. It is conceptually based on the skip-gram algorithm for the creation of word embeddings. However, instead of incrementally evaluating each word and its neighbors in a corpus of texts, node2vec explores the graph structure through a series of simulated random walks. Multiple walks from each node sample its neighborhood. A user can control the bias of the walks to explore further from the start node or stay closer to the start node. This is achieved by adjusting the ratio of the "return” parameter, P, and the “in-out” parameter Q. The authors theorize that exploration of larger neighborhoods will embed vectors with structural role characteristics. Conversely, exploration kept close to the start node will embed vectors with a richer representation of their neighborhood.

3.2.3 Data Set

In addition to CAD files the data set contains text files, Python code and Grasshopper code. The CAD files were created with Rhino 3D. Raster images collected from online databases served as the initial data. Room boundary and connection to adjacent rooms were explicitly defined as geometry in the CAD file. A custom Grasshopper script extracted area, isovist area and room access information for each node in each plan. All nodes with access to the exterior were connected to a single exterior node. The data was saved as a TXT file.

I began by creating a JSON file for each building's data. The JSON format works well with the NetworkX python module. The dictionary structure affords easy modification of graph level data, nodes, edges and associated attributes. Although I initially set out to modify the Grasshopper files to export JSON files, I found it more efficient to convert the TXT files to JSON files. In the future, if different data is required from the models, the Grasshopper script modification method will be used.

3.2.4 Feature Engineering

To each initial building data file, I added two new types of node and graph attributes and these are listed in Table 2. The first type includes quantitative measures like the total number of nodes and also graph theoretic measures like node degree. The graph theoretic measures were created using NetworkX.

The second type is the node embedding vectors created with the node2vec algorithm.

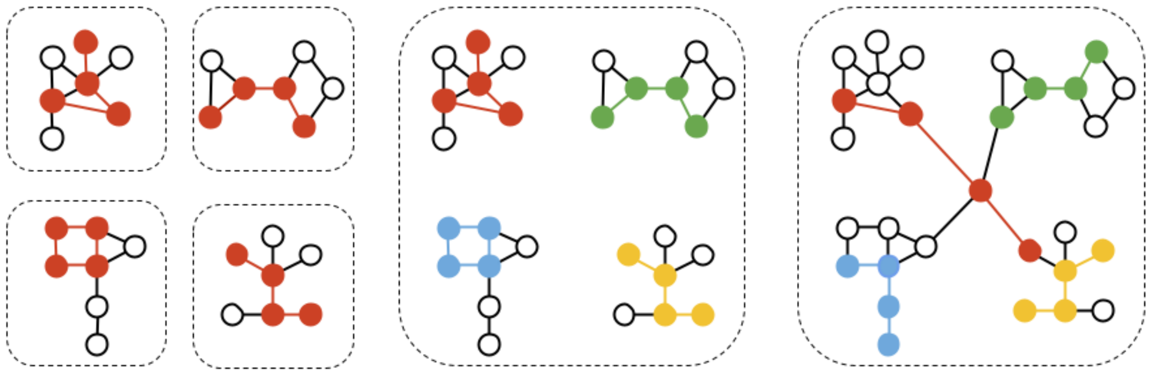


Figure 18: Three scenarios: buildings encoded separately, buildings encoded together, buildings encoded together with a common node

Graph Attributes	Node Attributes
No. of nodes	Betweenness
No. of edges	Degree centrality
Density	Eigenvector centrality
Average cluster coefficient	Closeness centrality
Transitivity	Degree
List of cliques (greater than two nodes)	Clustering coefficient

Table 2: Table of graph and node features

Two different embedding parameter settings, shown in Table 3, were used for each of three scenarios. The vector dimension, walk length, number of walks and worker parameters were chosen arbitrarily as the literature did not provide any guidance for these experiments. The P and Q values of Setting 1 and Setting 2 are taken from original paper. Respectively, they will cause random walks to remain closer to the start nodes to learn neighborhood representations, or stretch further from the start node to learn structural characteristics.

The first scenario created embeddings using each building graph as a separate corpus. This process created 39 separate vector spaces. The second scenario placed the nodes from all of the buildings into a single graph composed of 39 separate components. Although a single graph, the random walks were constrained within each component. However, all node embeddings were created in a single vector space. The third scenario connected the separate

Parameter	Setting 1 (Homophily)	Setting 2 (Structural)
Vector dimension	64	64
Walk length	100	100
Number of walks	200	200
Workers	4	4
P value	1.0	1.0
Q value	2.0	0.5

Table 3: Table of node2vec parameter settings

components in scenario two by replacing the 39 exterior nodes with a single external node. This connection allowed random walks to traverse between buildings. Figure 18 is a diagram of the three strategies.

The node embedding vectors were added to each node in the JSON files. I created a single vector representation for each building by calculating the centroid of the node vectors of each building. This was also added to the JSON files.

3.2.5 Classification Experiments

Like the original research by Ferrand et al., I attempted to classify each node as belonging to a mosque or monastery, and each building as either a mosque or a monastery. I used the Weka machine learning workbench to run the classification experiments. First, I converted the JSON files into CSV files. Separate file sets for building and node classification were created. Each was a set of six files, one for each permutation of the different embedding scenarios and parameter settings. Finally, I chose five algorithms representing different learning strategies: SVM, SVM with a polynomial kernel, J48 decision tree, random forest and naive bayes.

Because there are nearly 1,600 nodes, the node classification experiments were performed with a 10-fold cross-validation. However, the small number of buildings in the data set required a leave one out cross-validation method.

3.2.6 Results

Classification of nodes was more successful than classification of buildings. Node classification achieved above 95% accuracy in several cases, while building classification averaged below 70% accuracy when embeddings were included in the feature set. Tables 4 and 5 summarize the results.

Node Classification SVM with a polynomial kernel and decision tree classifiers achieved consistently high accuracy when classifying nodes in all six combinations of embedding space and node2vec parameter settings as shown in Table 4. In addition, accuracy was similar across the three feature set groups in all six combinations.

Notably, classification using embeddings alone achieved better results than classification with graph theory features alone in all but one category of vector spaces and node2vec parameter settings. Combining the features and embeddings provided similar results on average. The overall highest accuracies of 97.5% and 97.4% occurred with embeddings alone and combined embeddings and features, respectively. The node2vec parameter settings of p=1, q=2 created embeddings that performed slightly better than the p=1, q=0.5 settings when buildings were encoded separately and together. However, the parameter settings of p=1, q=0.5 performed slightly better when the buildings were encoded with a common node. Embeddings created with all nodes in the same feature space were more successful than embedding individual buildings separately. Connecting the buildings with a common node caused a slight decrease in accuracy in comparison to buildings without the common node connection.

Both the graph theoretic features and the embedding vectors achieved high accuracy. This is encouraging and supports the hypothesis that the embedding vectors capture similar information as the selection of SNA measure. However, I am surprised at the similar results across the different combinations of embedding and random walk parameters. In particular, I question the impact of the walk parameters and their purported ability to find and encode the node qualities of homophily and structural equivalence.

Node Classification											
		Setting 1					Setting 2				
		A1	A2	A3	A4	A5	A1	A2	A3	A4	A5
	Features	56.9	54.6	89.2	**90.4**	53.0					
VS1	Encoding	53.1	88.8	75.3	**92.0**	59.7	52.0	85.3	76.5	88.8	61.8
	Combined	54.1	89.9	84.7	90.2	62.46	52.3	86.8	84.0	89.7	62.5
VS2	Encoding	86.6	**97.5**	81.1	96.0	67.2	81.9	96.6	82.3	95.8	65.5
	Combined	86.6	97.2	86.7	95.8	68.7	84.2	96.5	84.8	96.4	66.4
VS3	Encoding	79.7	94.6	77.3	94.5	72.5	82.0	95.8	77.8	95.8	77.3
	Combined	80.4	95.8	85.2	96.3	63.3	83.3	95.6	87.4	**97.4**	68.5
A1 = SVM A2 = SVM with polynomial kernel A3 = Decision tree (J48) A4 = Random forest A5 = Naive Bayes VS1 = Buildings encoded separately VS2 = Buildings encoded together VS3 = Buildings encoded with a common node											

Table 4: Node classification results (Percent correct)

Building Classification Unfortunately, consistently separable nodes did not provide much benefit when classifying the buildings represented by those nodes. In most cases, building features alone were classified with the most accuracy. Vectors created when buildings were encoded together and using the p=1, q=0.5 parameters created a building vector that classified with highest accuracy. However, in all but that one case, adding the building vector to the other features reduced the classification accuracy. Potential strategies to address this problem are presented in the Discussion section.

3.2.7 Discussion

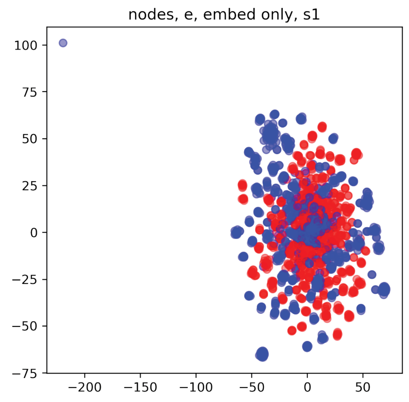
There are numerous additional steps that can be taken with this data set and these experiments. First, an in-depth error analysis should be performed to gain some understanding about which nodes and buildings were misclassified and why. This will hopefully bring

Building Classification											
		Setting 1					Setting 2				
		A1	A2	A3	A4	A5	A1	A2	A3	A4	A5
	Features	12.8	64.1	71.8	64.1	**76.9**					
VS1	Encoding	41.0	43.6	59.0	38.5	35.9	56.4	59.0	53.8	56.4	43.6
	Combined	38.5	41.0	**74.4**	51.3	69.2	56.4	61.5	66.7	64.1	71.8
VS2	Encoding	46.2	46.2	53.8	53.8	38.5	48.7	41.0	**69.2**	66.7	43.6
	Combined	46.2	53.8	48.7	64.1	61.5	59.0	53.8	**69.2**	**69.2**	56.4
VS3	Encoding	48.7	56.4	56.4	59.0	56.4	48.7	56.4	56.4	59.0	56.4
	Combined	48.7	53.8	48.7	59.0	61.5	48.7	48.7	53.8	59.0	**64.1**
A1 = SVM A2 = SVM with polynomial kernel A3 = Decision tree (J48) A4 = Random forest A5 = Naive Bayes VS1 = Buildings encoded separately VS2 = Buildings encoded together VS3 = Buildings encoded with a common node											

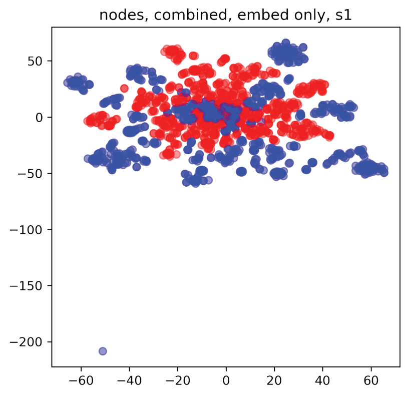
Table 5: Building classification results (Percent correct)

insight into the relationship between features and suggest improvements to feature space design. When conducting early work in this vignette, I attempted a clustering process to explore the data. The experiments provided limited insight, as shown in Figure 19. However, both node2vec papers referenced in this study used clustering to demonstrate the algorithm's abilities to capture relevant structural information of a graph. Future work to understand this divergence may provide another avenue to clarify the nature of the information encoded by the algorithm, specifically with respect to the configuration of spaces within floor plans. Also, I believe it is prudent to confirm the node classification results. The impressive results may be too good to be accurate.

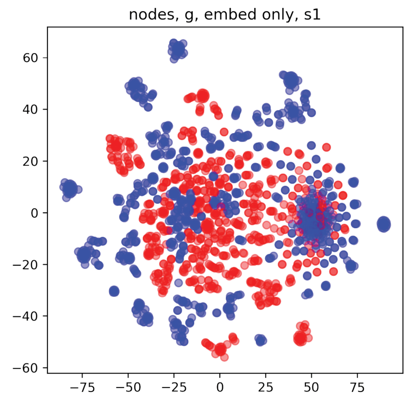
Next, alternate ways of encoding buildings should be explored. As seen in the results, using the centroid of a building's node vectors provided minimal traction for the classifiers. Because node2vec is based on the word2vec algorithm, I will start by researching the methods used in document classification. These methods are thoroughly developed and may be



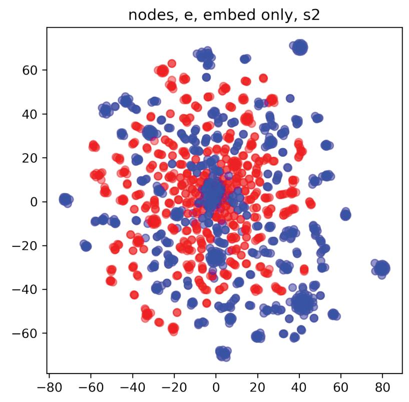
a



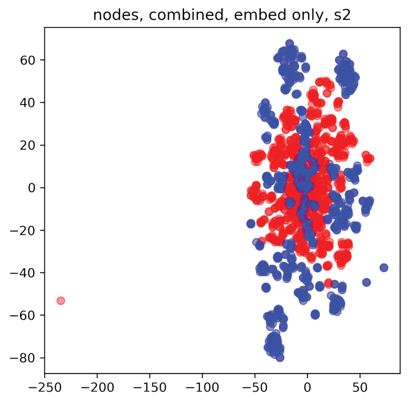
b



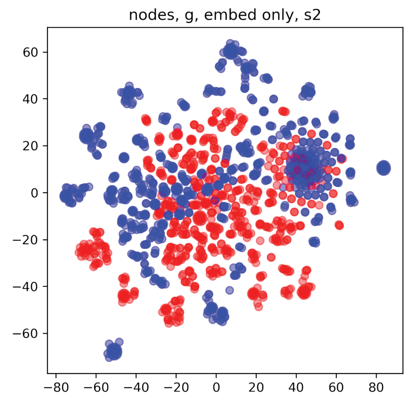
c



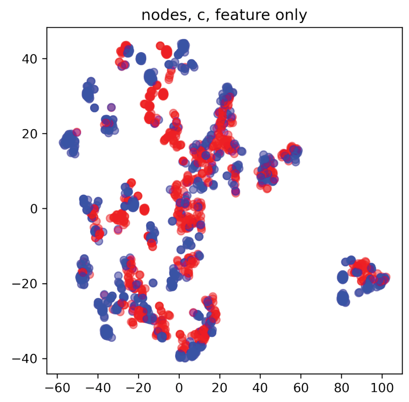
d



e



f



g

Figure 19: t-SNE plots of node vectors: a) embeddings only,buildings encoded separately, setting 1; b) embeddings only, buildings encoded together, setting 1; c) embeddings only, buildings encoded with common node, setting 1; d, e, f) same as a, b, c with setting 2; g) feature vectors

adaptable. Or, perhaps equally important, the research may illuminate the limitations of the space-as-word, building-as-document transformation underlying the use of these algorithms on spatial structure.

One important difference between the workings of node2vec and word2vec is the building of the vocabulary. Spatial nodes are unique and therefore their vectors embed only what can be learned from the context of that singular instance of the node. However, the same word will occur many times within a corpus of documents. Because of this, the learned word vector is influenced by many different neighborhoods. A common node labeling system across all buildings in the corpus could be analogous to a word vocabulary. As et al. appear to take this approach in their research (As et al., 2018). While they use space function as the label, other label classification systems could offer opportunities as well.

Intuitively, a building's exterior node and its relationship to the interior nodes and potentially other buildings' exterior nodes seems like an important consideration when creating embeddings. However, the variations tested in these experiments did not suggest a strong relationship between the exterior node configuration and classification accuracy. Because of the importance of the exterior node to early graph research by Steadman and to the justified plan graphs of space syntax, I believe experiments with other exterior node configurations are warranted.

Finally, supervised embedding algorithms should be investigated. These preserve both the structural and semantic information contained in the graph (Chami et al., 2020). Comparison between classification experiments performed with both supervised and unsupervised may provide further insight into the nature of the spatial organization information actually captured by vector embeddings.

3.3 Vignette 3 - Generating With Data

3.3.1 Introduction

Vignette 3 continues exploring floor plan data sets and the representation of that data as graphs and vector embeddings. Instead of a machine learning model for classification, a generative machine learning model that creates new floor plans from an input graph is the instrument of the exploration. A selection of the LIFULL HOME'S data set (*Informatics Research Data Repository [LIFULL HOME'S Dataset]*, n.d.) is used for floor plan data and the model will be based on an implementation of a conditional GAN architecture created by Jinmo Rhee and Pedro Veloso (Rhee & Veloso, 2021). The model is simpler than the state of art models designed specifically for floor plan generation discussed in the background section. However, it employs the basic concept of training a generative model on pairs of floor plans and their associated graphs. In this case, the graph is represented by a vector embedding. The process of preparing the data and employing a generative model is intended to provide insights into the affordances of vector embeddings that classification models may not.

3.3.2 Investigating the Data Set

Generative machine learning models require large numbers of samples for their learning. The projects discussed in the Background section used data sets between approximately 60,000 and 120,000 plans. The LIFULL HOME'S data set, described in the Background section, will be used for this study. I do not have the full data set of 5.3 million plans; therefore, I use vector format translations that other researchers have made available from their work with the data set. The developers of the Raster-to-Vector algorithm used the LIFULL HOME'S data set to test their floor plan reading computer vision algorithms (Liu, Wu, Kohli, & Furukawa, 2017). From the data set, they selected 870 plans. They then manually annotated and vectorized the plans as the ground truth training data for their neural network. The Raster-to-Vector GitHub page makes available the ground truth data, the neural network model and text data for over 100,000 vectorized floor plan images. I have downloaded both sets of data. House-GAN, discussed in the Background section, also uses the LIFULL HOME'S data set and uses the Raster-to-Vector model to vectorize the data. From the total data set, 117,587 vectorized plans were created and this collection is available on the House-GAN GitHub page. The format of this data is optimized for House-GAN and is slightly different from the Raster-to-Vector data. The differences will be explained below. The Raster-to-Vector data is available as a collection of text files, while the House-GAN data is available as a list of numpy arrays.

As a first step in understanding the data, set I visualized the data with geometry in Rhino / Grasshopper using GHPython and the ground truth data from the Raster-to-Vector project. I chose this data because it was manually annotated during creation, and therefore I expected it to more closely reflect the information in the raster version of the floor plan. Additionally, I was able to find elsewhere and then correlate the vector representation data with the original floor plan raster image. This allowed me to spot check the concurrence between the raster images and the vector representations.

I created custom scripts to translate the coordinate data to wall objects, perimeter polylines, bounding boxes and geometry that represented doors and windows. Many of the coordinate points left gaps between walls and resulted in walls overlapping. I was able to craft fixes for several of the problems, but also created rules to reject plans that did significant geometry problems. Nevertheless, the annotated ground-truth data was in fact congruous. I noted the following observations about both file types. The raster images are highly stylized diagrams and while the graphic style is similar, it is not uniform throughout the data set. Rooms and spaces are often labeled, but the use of many spaces is signified only by symbols for toilets, bathtubs, stoves, etc. The plans are not the same scale and contain no regular indication of cardinal orientation. They also do not contain any indication of adjacent building mass or any information on the space outside the entry door, i.e. is it an interior space like a hallway or the exterior. In the vector representations each space in the plan is assigned one of eleven labels by the annotator. However, the vector representations do not contain any explicit indication where to separate two or more uses that are not physically separated by a wall, e.g., a kitchen and living area. Exterior doors and windows are conflated, although the entry door is often indirectly indicated with an entry mat icon.

After examining the data available from the Raster-to-Vector research, I investigated the House-GAN subset. As noted above, this data set was created using the LIFULL HOME'S data set and the Raster-to-Vector computer vision algorithm. That algorithm was said to be 90% accurate by the original researchers (Liu et al., 2017). Although each vector floor plan instance included the unique identifier of the original floor plan, those images are not available and therefore direct comparisons are not possible. Therefore, I made judgements on whether the geometry made sense as a floor plan in general and as a floor plan in the family of floors plans that I had seen in the data set. Because of slight differences in format, the Grasshopper / Rhino geometry reconstruction script had to be modified. The first important item to note is that the list of possible room types was modified. The three different bathroom types were consolidated into a single type, while a dining room type and laundry room type were added. Also importantly, it appears that the problem of different functional spaces not having physical separations was resolved by combining all such spaces into a single space or by filtering out plans with such conditions. Indications of entry doors were also not included in the House-GAN versions.

Despite the divergence from the original raster images by error or deliberate modification, I have chosen to use the House-GAN vector representations for several reasons. First and most importantly, the corners and connections of the geometry are better coordinated, leaving no gaps and therefore reducing errors in the Grasshopper / Rhino geometry. Second, by examining the raster images, I successfully created heuristics to infer which door was the entry door. (A more thorough statistical analysis could be conducted.) Finally, the large number of samples is sufficient for training a generative machine learning model.

Custom scripts extract the nodes and edges from the vector geometry of the floor plan. Although many room and configuration characteristics are available from the data, I chose to limit the scope and complexity of the exploration by only recording room function and spatial adjacency in the graph.

3.3.3 Floor Plan Encoding

The method of vector embedding the graphs is critical to this vignette. To date, I do not have a working solution. The node2vec embedding method used in Vignette 2 is transductive. All of the graphs and nodes must be present at the time of embedding. Additional nodes or graphs cannot be added to the vector space without re-embedding the entire set of information. This works well for the classification experiments where no new information was introduced after the initial embedding process. However, the generative process proposed in this vignette requires an inductive embedding method. In other words, the method must be able to create new embeddings in the same vector space for graphs not seen during the initial embedding process. New embeddings of new input graphs become the input for the generator.

My research is focused on two potential methods, GraphSAGE and a Graph Attention Network (GAT) method (Hamilton, Ying, & Leskovec, 2018; Velickovic et al., 2018). Both have code available that I am currently trying to adapt for this project. The researchers demonstrated the capabilities of each with inductive node-classification experiments on several large data sets and graph-classification experiments of unseen graphs using a data set of protein-protein interactions. The relevant experiment for my research is the classification of protein graphs because the creation of embeddings for whole unseen graphs is a necessary function of my proposed floor plan generator. There are significant differences between the data sets. The protein-protein interaction data set is a small set of 24 graphs with an average of 2372 nodes and is therefore much smaller than the floor plan data set and the graphs are much larger and more complex. However, it is the best example that I have been able to locate.

3.3.4 Data Representation

The Rhino / Grasshopper platform is inefficient for processing large numbers of floor plans, and the necessary embedding libraries are not directly accessible. Therefore, after the meth-

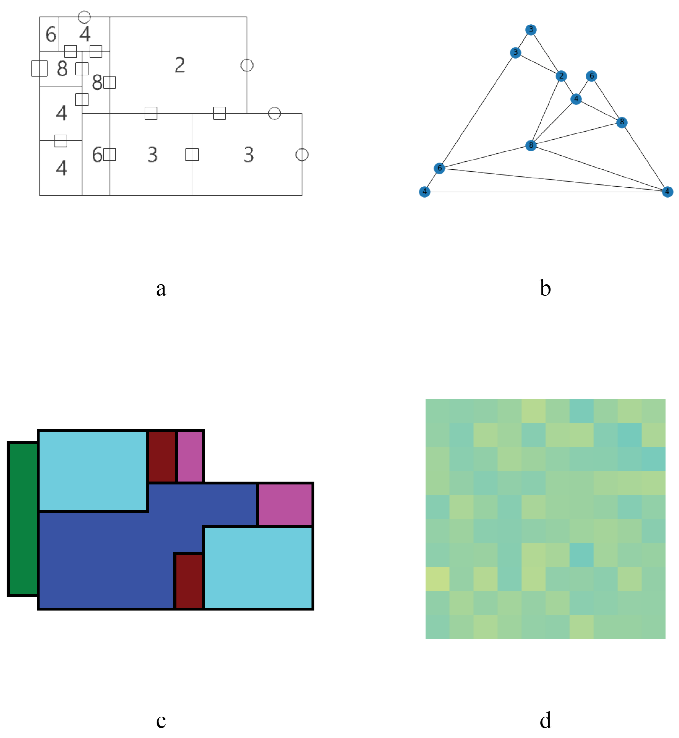


Figure 20: a) Reconstructed vector format floor plan; b) Graphic representation of adjacency graph; c) Raster image of floor plan; d) Vector encoding raster image

ods for creating the plan geometry and extracting the graphs were functional, I exported the GHPython code to a Jupyter notebook and used the Rhino3dm library for geometry objects. From there the raster images are created for the GAN model. The Matplotlib library is used to create 256x256 pixel images of the plans as color blocks, coded by function. Prior to image creation each plan is translated to a center position and scaled to fit within a white border. Examples of these three representations are shown in Figure 20 items a, b and c

The vector embedding procedure will be located within the notebook when it is functional. The method to generate a 256x256 pixel image of an embedding vector to pair with the floor plan image is complete. An example of a vector represented as a colored block is shown in Figure 20 item d.

3.3.5 Generative Model

The conditional GAN architecture is based on the image-to-image translation model pix2pix and was adapted by Rhee and Veloso (Isola, Zhu, Zhou, & Efros, 2018). Their model is trained with raster image pairs of colored floor plans and corresponding colored bubble diagrams. New raster floor plans are generated from new colored bubble diagrams input by a model user. I modify the model to train on pairs composed of a colored floor plan and an image of the corresponding vector encoding of the adjacency graph extracted from the floor plan. Because the embedding method is not functional, I have not been able to achieve any preliminary results to evaluate the applicability of this particular GAN architecture for the experiment.

3.3.6 Discussion

Although generated results have not been achieved, the effort to create the generator brought important insights. Hands on manipulation of the chosen data set revealed its ambiguities and the significant level of reduction and abstraction from the ground truth of real apartments. Automated translation systems unavoidably make mistakes. As noted above, the Raster-to-Vector algorithm is claimed to be 90% accurate. How the 10% error rate is manifested in the resultant vector floor plans is not clear. Without the original raster images it is impossible for me to determine, but the geometry reconstructions of the House-GAN data set revealed no obvious errors or misconfigurations. Although not precisely errors, the algorithm is forced to ignore or make simplifications of input that it cannot parse. An important example of this limitation is the assignment of a single room type to represent two or more rooms not separated by a door or small opening. Equally important, the original raster images are not to scale, therefore the collection of resultant vectorized plans does not have a consistent scale.

Further, the House-GAN creators also made design decisions about the data set. The decision to modify the list of possible room types changed the semantic content of the original floor plan images. The decisions to conflate doors and windows and provide no indication of the front door directly reflect the design priorities of the House-GAN creators. After all of these data set design decisions by the House-GAN creators and my own decisions to adapt the contents of the data set, it is clear that the final content of the data set is inextricably influenced by the intentions of the users. This suggests to me that there should be at least as much if not more effort directed toward the open and deliberate design of data sets as there is to the development of new algorithms. The results of the generator by Para shown in 11 demonstrate the influence of the data set on the results.

4 Conclusion

4.1 Contribution

The research strategy of using historical research and software reconstruction to probe and critique state-of-the-art computational design methods can be used to draw several conclusions. Two important lineages of graph-based representation methods emerged from the foundational developments in the 20th century. The first employs graphs-based representations as a tool for scientific generalization of the built environment. Braced with mathematical rigor, graph-based representations offer the promise of objective classification systems and generalized theories of building performance rooted in spatial configuration and morphology. In this lineage, graph-based representations become a modern instrument to reconfigure architectural history away from the traditional classification of formal and stylistic typologies and toward one based on the immutable laws of geometry. In doing so, the reconfiguration also is a proposal to reconnect to architectural history with techniques supported by digital computers and new mathematical concepts. The second lineage solidifies graph-representations as carriers of architectural programmatic information. In this lineage graph-based representations serve as a functionalist tool for form determination. They are the conduit through which rationally determined functional requirements guide rationally designed form-defining generative systems for optimal solutions.

State-of-the-art applications of machine learning to graph-based spatial representations suggest new possibilities for the two lineages. The successful classification experiments of Ferrando et al. and those I conducted in Vignette 2 demonstrate the potential of vector spaces as locations for comparison and analysis of architectural space. Encoding of graph-based representations sidesteps the role of abstract geometry and graph theory in floor plan analysis without losing the primacy of plan configuration as the principal organizing concept. Although not demonstrated here, extrapolating from work in other domains shows that graphs, nodes and edges can be attributed with data that further enrich the vector encodings creating new locations to pose questions of greater complexity and nuance.

Data-driven floor plan generation uses graph-based representation of floor plans to exploit the modern concept of program in two ways. Like the generators reconstructed in Vignette 1, the generators proposed in Vignette 3 and discussed in section on data-driven generative systems all employ graph representations to carry the program as input to the generator. But program is not attributed to the data set of the rectangular dissections. The dissections are neutral, dimensionless geometry. Their cells are presumptively receptive to any program of matching configuration. In contrast, data-driven systems use graphs to extract the implicit design program from each floor plan in the data set. The generative model is then trained on pairs of programs, represented as graphs, and realized floor plans. Through the association of both functional arrangements and geometric configuration, data-driven models demonstrate more varied results. Admittedly, the work demonstrated so far is small in scale and the results are decidedly normative. The motivations claimed by the creators of the state-of-the-art generators tends toward the desire for automatic production of floor plans much like the creators of the graph theoretic based generators. However, as noted in the Discussion section of Vignette 3, creative and thoughtful data set design may offer alternative manners of creative exploration with data-driven generators.

Lastly, the challenges of creating the vector representations in Vignette 3 raised important and unexpected limitations to vector embeddings. The random walk methods used in Vignette 2 are not applicable to generative or analytical systems that must encode unseen graphs or nodes. Although graph embeddings appear to offer new computation access to spatial configuration, the technology has limitations that will surely be tested as more research employs the techniques.

4.2 Next Steps

By linking state-of-the-art machine learning on graph-based representations to an established lineage of graph-based design and architectural research, a trajectory of future work is suggested. The successful extraction of semantic information from floor plans and the successful vector embedding experiments demonstrate that the combination of graph-based representations and vector embeddings could be developed as a digital classification system for architectural information.

The system would be in the direct lineage of the English building stock research conducted by the Centre for Configurational Studies (Steadman et al., 1991). Of course, the background research and vignettes also demonstrated that the challenge of collecting floor plan information has improved surprisingly little since the Centre performed its studies. A simple uniform format would encourage the academic and professional communities to contribute to databases either their own design work or their data set creation efforts. An attributed configuration-based representation format would be simpler and therefore more likely to be successful and adopted than a system based on standardized representations of conventional two-dimensional representations or three-dimensional models. In addition to supporting research communities, a uniform format is critical to designing data sets for machine learning generative systems.

References

Abdelrahman, M., Chong, A., & Miller, C. (2020, July). Build2Vec: Building Representation in Vector Space. *arXiv:2007.00740 [cs]*. Retrieved 2020-11-10, from http://simaud.org/2020/proceedings/102.pdf (arXiv: 2007.00740)

Al-Jokhadar, A., & Jabi, W. (2016). Humanising the Computational Design Process - Integrating Parametric Models with Qualitative Dimensions. In *Parametricism Vs. Materialism: Evolution of Digital Technologies for Development [8th ASCAAD Conference Proceedings ISBN 978-0-9955691-0-2] London (United Kingdom) 7-8 November 2016, pp. 9-18.* CUMINCAD. Retrieved 2020-11-06, from http://papers.cumincad.org/cgi-bin/works/paper/ascaad2016_003

As, I., Pal, S., & Basu, P. (2018, December). Artificial intelligence in architecture: Generating conceptual design via deep learning. *International Journal of Architectural Computing*, *16*(4), 306–327. Retrieved 2020-08-07, from https://doi.org/10.1177/1478077118800982 (Publisher: SAGE Publications) doi: 10.1177/1478077118800982

Baybars, I., & Eastman, C. M. (1980, September). Enumerating Architectural Arrangements by Generating Their Underlying Graphs. *Environment and Planning B: Planning and Design*, *7*(3), 289–310. Retrieved 2020-11-01, from https://journals.sagepub.com/doi/abs/10.1068/b070289 (Publisher: SAGE Publications Ltd STM) doi: 10.1068/b070289

Building Database & Analytics System (BuDAS). (n.d.). Retrieved 2021-03-28, from https://sites.baylor.edu/budas/

Cardoso Llach, D., & Donaldson, S. (2019). An Experimental Archaeology of CAD. In J.-H. Lee (Ed.), *Computer-Aided Architectural Design. "Hello, Culture"* (pp. 105–119). Singapore: Springer. doi: 10.1007/978-981-13-8410-3_8

Chaillou, S. (2019, July). *ArchiGAN: a Generative Stack for Apartment Building Design.* Retrieved 2020-12-16, from https://developer.nvidia.com/blog/archigan-generative-stack-apartment-building-design/

Chami, I., Abu-El-Haija, S., Perozzi, B., Ré, C., & Murphy, K. (2020, May). Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *arXiv:2005.03675 [cs, stat]*. Retrieved 2020-12-16, from http://arxiv.org/abs/2005.03675 (arXiv: 2005.03675)

Cheng, C.-Y. (2021, March). *Co-design with AI.* Morphing Matter Lab, Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University.

Cohen, E. (2018, April). *node2vec: Embeddings for Graph Data | Towards Data Science.* Retrieved 2020-12-13, from https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef

Colquhoun, A. (1969). Typology and Design Method. *Perspecta*, *12*, 71–74. Retrieved 2020-09-21, from http://www.jstor.org/stable/1566960 (Publisher: The MIT Press) doi: 10.2307/1566960

DeLanda, M. (2002). Deleuze and the Use of the Genetic Algorithm in Architecture. *Architectural Design*, *72*(1), 9–12.

Delve. (n.d.). Retrieved 2020-12-17, from https://hello.delve.sidewalklabs.com/

Duarte, J. P. (2001). *Customizing mass housing : a discursive grammar for Siza's Malagueira houses* (Thesis, Massachusetts Institute of Technology). Retrieved 2020-11-18, from https://dspace.mit.edu/handle/1721.1/8189 (Accepted: 2005-08-23T18:09:22Z)

Ferrando, C. (2018). *Towards a Machine Learning Framework in Spatial Analysis* (Master of Science, Carnegie Mellon University, Pittsburgh, PA). Retrieved from https://kilthub.cmu.edu/articles/Towards_a_Machine_Learning_Framework_in_Space_Syntax/7178417/files/13205726.pdf

Ferrando, C., Dalmasso, N., Mai, J., & Cardoso Llach, D. (2019). Architectural Distant Reading Using Machine Learning to Identify Typological Traits Across Multiple Buildings. In *Ji-Hyun Lee (Eds.) "Hello, Culture!" [18th International Conference, CAAD Futures 2019, Proceedings / ISBN 978-89-89453-05-5] Daejeon, Korea, pp. 114-127.* CUMINCAD. Retrieved 2020-09-13, from http://cumincad.scix.net/cgi-bin/works/Show?cf2019_014

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014, June). Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*. Retrieved 2021-05-22, from http://arxiv.org/abs/1406.2661 (arXiv: 1406.2661)

Grasl, T., & Economou, A. (2010). Palladian Graphs : Using a graph grammar to automate the Palladian grammar. In *FUTURE CITIES [28th eCAADe Conference Proceedings / ISBN 978-0-9541183-9-6] ETH Zurich (Switzerland) 15-18 September 2010, pp.275-283.* CUMINCAD. Retrieved 2020-09-09, from http://papers.cumincad.org/cgi-bin/works/Show?ecaade2010_005

Grohe, M. (2020, June). word2vec, node2vec, graph2vec, X2vec: Towards a Theory of

Vector Embeddings of Structured Data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (pp. 1–16). New York, NY, USA: Association for Computing Machinery. Retrieved 2021-03-26, from http://doi.org/10.1145/3375395.3387641 doi: 10.1145/3375395.3387641

Grover, A., & Leskovec, J. (2016, July). node2vec: Scalable Feature Learning for Networks. *arXiv:1607.00653 [cs, stat]*. Retrieved 2020-08-28, from http://arxiv.org/abs/1607.00653 (arXiv: 1607.00653)

Hamilton, W. L., Ying, R., & Leskovec, J. (2018, September). Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]*. Retrieved 2021-04-16, from http://arxiv.org/abs/1706.02216 (arXiv: 1706.02216)

Harding, J. (2016). Evolving Parametric Models using Genetic Programming with Artificial Selection. In *Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 1* (pp. 423–432). University of Oulu, Oulu, Finland: CUMINCAD. Retrieved 2020-05-11, from http://papers.cumincad.org/cgi-bin/works/paper/ecaade2016_163

Harding, J., & Brandt-Olsen, C. (2018, June). Biomorpher: Interactive evolution for parametric design. *International Journal of Architectural Computing*, *16*(2), 144–163. Retrieved 2020-05-11, from https://doi.org/10.1177/1478077118778579 (Publisher: SAGE Publications) doi: 10.1177/1478077118778579

Heitor, T. V., Duarte, J. P., & Pinto, R. M. (2004, December). Combining Grammars and Space Syntax: Formulating, Generating and Evaluating Designs. *International Journal of Architectural Computing*, *2*(4), 491–515. Retrieved 2020-11-11, from https://doi.org/10.1260/1478077042906221 (Publisher: SAGE Publications) doi: 10.1260/1478077042906221

Hillier, B., & Hanson, J. (1984). *The Social Logic of Space*. Cambridge [Cambridgeshire]: Cambridge University Press. Retrieved 2020-10-22, from http://search.ebscohost.com/login.aspx?direct=true&db=e000xna&AN=711679&site=ehost-live&scope=site

Hu, R., Huang, Z., Tang, Y., van Kaick, O., Zhang, H., & Huang, H. (2020, April). Graph2Plan: Learning Floorplan Generation from Layout Graphs. *arXiv:2004.13204* *[cs]*. Retrieved 2020-08-09, from http://arxiv.org/abs/2004.13204 (arXiv: 2004.13204) doi: 10.1145/3386569.3392391

Informatics Research Data Repository [LIFULL HOME'S Dataset]. (n.d.). Retrieved 2021-03-15, from https://www.nii.ac.jp/dsc/idr/en/lifull/

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2018, November). Image-to-Image Translation with Conditional Adversarial Networks. *arXiv:1611.07004 [cs]*. Retrieved 2021-05-05, from http://arxiv.org/abs/1611.07004 (arXiv: 1611.07004)

Keller, S. (2017). *Automatic Architecture: Motivating Form after Modernism*. Chicago: The University of Chicago Press.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Mass: MIT Press.

LIFULL. (n.d.). *LIFULL HOME'S.* Retrieved 2021-04-03, from https://www.homes.co.jp/

LIFULLCreators. (n.d.-a). *High-definition floor plan image data has been added to "HOME'S Dataset".* Retrieved 2021-04-03, from https://www.lifull.blog/entry/2016/02/01/144650

LIFULLCreators. (n.d.-b). *We will start providing "HOME'S" property and image data sets to researchers!* Retrieved 2021-04-03, from https://www.lifull.blog/entry/2015/11/17/164717

Liu, C., Wu, J., Kohli, P., & Furukawa, Y. (2017). Raster-To-Vector: Revisiting Floorplan Transformation. In (pp. 2195–2203). Retrieved 2021-03-01, from https://openaccess.thecvf.com/content_iccv_2017/html/Liu_Raster-To-Vector_Revisiting_Floorplan_ICCV_2017_paper.html

March, L. (1971). *The geometry of environment: an introduction to spatial organization in design*. London: RIBA Publications.

Mitchell, W. J. (1979). *Computer-aided architectural design*. New York: Van Nostrand Reinhold.

Mitchell, W. J., Steadman, J. P., & Liggett, R. S. (1976, June). Synthesis and Optimization of Small Rectangular Floor Plans. *Environment and Planning B: Planning and Design*, *3*(1), 37–70. Retrieved 2021-01-14, from https://journals.sagepub.com/doi/abs/

10.1068/b030037 (Publisher: SAGE Publications Ltd STM) doi: 10.1068/b030037

Nauata, N., Chang, K.-H., Cheng, C.-Y., Mori, G., & Furukawa, Y. (2020, March). House-GAN: Relational Generative Adversarial Networks for Graph-constrained House Layout Generation. *arXiv:2003.06988 [cs]*. Retrieved 2020-09-10, from http://arxiv.org/abs/2003.06988 (arXiv: 2003.06988)

Nauata, N., Hosseini, S., Chang, K.-H., Chu, H., Cheng, C.-Y., & Furukawa, Y. (2021, March). House-GAN++: Generative Adversarial Layout Refinement Networks. *arXiv:2103.02574 [cs]*. Retrieved 2021-03-10, from http://arxiv.org/abs/2103.02574 (arXiv: 2103.02574)

Ostwald, M. J. (2011, July). The Mathematics of Spatial Configuration: Revisiting, Revising and Critiquing Justified Plan Graph Theory. *Nexus Network Journal*, *13*(2), 445–470. Retrieved 2020-11-06, from https://doi.org/10.1007/s00004-011-0075-3 doi: 10.1007/s00004-011-0075-3

Para, W., Guerrero, P., Kelly, T., Guibas, L., & Wonka, P. (2020, November). Generative Layout Modeling using Constraint Graphs. *arXiv:2011.13417 [cs]*. Retrieved 2021-02-23, from http://arxiv.org/abs/2011.13417 (arXiv: 2011.13417)

Revit Gen Design. (n.d.). Retrieved 2020-12-17, from https://www.autodesk.com/solutions/generative-design/architecture-engineering-construction

Rhee, J., & Veloso, P. (2021, March). *Bubble2Floor.* Carnegie Mellon University, School of Architecture.

Shekhawat, K., Pinki, & Duarte, J. P. (2019). A Graph Theoretical Approach for Creating Building Floor Plans. In J.-H. Lee (Ed.), *Computer-Aided Architectural Design. "Hello, Culture"* (pp. 3–14). Singapore: Springer. doi: 10.1007/978-981-13-8410-3_1

Shekhawat, K., Upasani, N., Bisht, S., & Jain, R. N. (2021, July). A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction*, *127*, 103718. Retrieved 2021-05-13, from https://www.sciencedirect.com/science/article/pii/S0926580521001692 doi: 10.1016/j.autcon.2021.103718

Spacemaker AI. (n.d.). Retrieved 2020-12-17, from https://www.spacemakerai.com/

Steadman, P. (1973). Graph theoretic representation of architectural arrangement. *Architectural Research and Teaching*, *2*(3), 161–172. Retrieved 2020-08-28, from http://

www.jstor.org/stable/24654905 (Publisher: Locke Science Publishing Company, Inc.)

Steadman, P. (1983). *Architectural morphology: an introduction to the geometry of building plans*. London: Pion.

Steadman, P., Brown, F., & Rickaby, P. (1991, March). Studies in the Morphology of the English Building Stock. *Environment and Planning B: Planning and Design*, *18*(1), 85–98. Retrieved 2021-03-19, from https://journals.sagepub.com/doi/abs/10.1068/b180085 (Publisher: SAGE Publications Ltd STM) doi: 10.1068/b180085

Steenson, M. W. (2017). *Architectural Intelligence: How Designers and Architects Created the Digital Landscape*. Cambridge, UNITED STATES: MIT Press. Retrieved 2020-05-28, from http://ebookcentral.proquest.com/lib/cm/detail.action?docID=5205402

Stiny, G., & Gips, J. (1971). Shape Grammars and the Generative Specification of Painting and Sculpture. In *Segmentation of Buildings for 3DGeneralisation. In: Proceedings of the Workshop on generalisation and multiple representation , Leicester.*

Stiny, G., & Mitchell, W. J. (1978, June). The Palladian Grammar. *Environment and Planning B: Planning and Design*, *5*(1), 5–18. Retrieved 2021-04-04, from https://journals.sagepub.com/doi/abs/10.1068/b050005 (Publisher: SAGE Publications Ltd STM) doi: 10.1068/b050005

Summerson, J. N. (1990). The Case for a Theory of 'Modern' Architecture. In *The unromantic castle, and other essays* (pp. 257– 266). London: Thames and Hudson.

Vardouli, T. (2020, June). Skeletons, Shapes, and the Shift from Surface to Structure in Architectural Geometry. *Nexus Network Journal*, *22*(2), 487–505. Retrieved 2020-08-28, from https://doi.org/10.1007/s00004-020-00478-0 doi: 10.1007/s00004-020-00478-0

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018, February). Graph Attention Networks. *arXiv:1710.10903 [cs, stat]*. Retrieved 2021-05-02, from http://arxiv.org/abs/1710.10903 (arXiv: 1710.10903)

Wu, W., Fu, X.-M., Tang, R., Wang, Y., Qi, Y.-H., & Liu, L. (2019, November). Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics*, *38*(6), 234:1–234:12. Retrieved 2020-11-06, from http://doi.org/10.1145/3355089.3356556 doi: 10.1145/3355089.3356556

Wurzer, G., Lorenz, W. E., & Wien, T. (2016). SpaceBook - A Case Study of Social Network Analysis in Adjacency Graphs. In *Herneoja, Aulikki; Toni Österlund and Piia Markkanen (eds.), Complexity & Simplicity - Proceedings of the 34th eCAADe Conference - Volume 2, University of Oulu, Oulu, Finland, 22-26 August 2016, pp. 229-238.* CUMINCAD. Retrieved 2020-11-19, from http://papers.cumincad.org/cgi-bin/works/paper/ecaade2016_018