GRAPH-REPRESENTATION LEARNING FOR HUMAN-CENTERED ANALYSIS OF BUILDING LAYOUTS

By

VAHID AZIZI

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Mubbasir Kapadia

And approved by

_____

_____

_____

_____

New Brunswick, New Jersey

May 2021

ABSTRACT OF THE DISSERTATION

Graph-Representation Learning for Human-Centered Analysis of Building Layouts

by **Vahid Azizi**

Dissertation Director: Mubbasir Kapadia

Floorplans are a standard representation of building layouts. Computer-Aided Design (CAD) applications and existing Building Information Modeling (BIM) tools rely on simple floorplan representations that are not amenable to automation (e.g., generative design). They do not account for how people inhabit and occupy the space. These are two central challenges that must be addressed for intelligent human-aware building design and this thesis's focus. This thesis addresses these challenges by exploring graph representation learning techniques to implicitly encode the latent state of floorplan configurations, which is more amenable to automation and related applications. Specifically, we use graphs as an intermediate representation of floorplans. Rooms are nodes, and edges indicate a connection between adjacent rooms, either through a door or passageway. The graphs are annotated with various attributes that characterize the semantic, geometric, and dynamic properties of the floorplan with respect to human-centered criteria. To address the variation in graphs' dimensionality, we utilize an intermediate sequential representation (generated by random walks) to encode the graphical structure in a fixed-dimensional representation. We propose the use of RNN-based vanilla/variational autoencoder architectures to embed attributed floorplans. We enhance graph-based representations of floorplans with human occupancy attributes extracted by statically analyzing the floorplan geometry and running simulations on large datasets of real and procedurally generated synthetic floorplans. We

explore the potential of our proposed methods and floorplan representations on various tasks, including finding semantically similar floorplans, floorplan optimization, and generative design. Our approach and techniques are extensively evaluated through a series of quantitative experiments and user studies with expert architects to validate our findings. The qualitative, quantitative, and user-study evaluations show that our embedding framework produces meaningful and accurate vector representations for floorplans. Our models and associated datasets have been made publicly available to encourage adoption and spark future research in the burgeoning research area of intelligent human-aware building design.

ACKNOWLEDGMENTS

First of all, I would like to express my deepest gratitude to my thesis adviser, Professor Mubbasir Kapadia, for his continuous encouragement and creative advice. I had this honor to complete my Ph.D.under his supervision.

Furthermore, I would like to thank all of my thesis committee members: Professor Mridul Aanjaneya, Professor Gerard de Melo, and Professor M. Brandon Haworth, for kindly accepting to be among my committee members and their helpful comments.

Finally, I would like to thank all of my labmates and collaborators, especially Professor Petros Faloutsos at York University, for their outstanding collaboration and helpful comments.

DEDICATION

To

My Parents, for their unconditional love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Problem Statement and Motivation

Floorplans provide a well-established means to represent buildings. As such, they afford a wide range of design activities such as ideation, analysis, evaluation, and communication. Computer-Aided Design (CAD) and Building Information Modeling (BIM) approaches to support the creation of digital building models, from which floorplans can be extracted. Current approaches, however, do not support the systematic comparison of floorplan features derived from geometric and semantic properties as well as more advanced performance metrics, such as space utilization and occupant dynamics features generated via simulation. While prior research has proposed computational strategies to extract floorplan features by processing them as images or graphs, these approaches ignore semantic information that describes each space's function and time-based analytic of human movements and activities.

From design inspiration to the democratization of housing layouts for affordable living, matching the desired features of a space to a floorplan is crucial to these desired interfaces. This goal requires some challenges to be overcome; the ability to retrieve an optimal floorplan and generate new ones as well. While it is trivial to lookup a value of a floorplan, such as the number of bathrooms, it is much more difficult to look up a floorplan with a certain number of bathrooms *and* similar room sizes (and other attributes). As the complexities of design and architecture are well known, the relationship between a single floorplan and the numerous measures one would associate with it makes this a challenging search task (i.e., floorplans' configuration space is extremely high-dimensional, continuous, and non-convex). The use of visual search tools (e.g., shape-grammars[1]) to match similarity is an

active research topic in Architecture and Design. However, in such an ample search space and intrinsically tracking multiple quantitative metrics, the utility of geometric similarity quickly diminishes. Recently, research has shown the use of machine learning techniques for associating environment configurations with metrics [2, 3]. However, these works require an explicit search through the environment configuration space.

This thesis addresses the mentioned challenges by representing floorplans with low dimensional vectors that accurately encode geometric, semantic, and human dynamic features. These features are extracted statically or by running simulations to extract the dynamic features. We formalize floorplans as graphs, where nodes are rooms, and they are connected if there is a door or passageway between them. By representing floorplans with graphs, their geometric structure is captured. Though the geometric structure of floorplans is a necessity, it is not enough. We need to integrate semantic design features and human dynamic features as well. Therefore, we augment the graphs with semantic and dynamic features. To capture global graph structures, we utilize random walk to convert them to sequences as intermediate data. Then, we propose deep learning methods to embed these sequences into low-dimensional vector representations. These vector representations are used to cluster and query floorplans with similar characteristics and attributes. Some of the advantages of representing floorplans as vectors are:

- Compact representation.
- An efficient way to compare designs and fast retrievals.
- Scoring designs and providing feedback/recommendations.
- Categorizing design according to our target features.

Moreover, these representations' potential on a variety of tasks, including floorplan optimization and generative design, are studied. We conduct a series of quantitative, qualitative experiments and user studies with expert architects to evaluate our works. The evaluations

show that our embedding framework produces meaningful and accurate vector representations for floorplans. As part of this thesis, two datasets (the extended housExpo dataset and a newly created synthetic dataset) are released to maximize adoption and spark new research in this burgeoning area.

1.2 Limitations in Prior Work

Prior methods have used different image processing techniques and deep learning methods to extract meaningful information from floorplan images for representation. However, these methods are suitable for object-centric datasets in which floorplans are annotated with furniture or specific visual symbols. These features do not correctly capture the high-level design structures [4, 5, 6, 7, 8, 9]. Moreover, human behavioral features are not considered in these methods, whereas occupants' behaviors (i.e., trajectories) are often highly correlated with environments [10].

In order to improve floorplan representation, some other works aim to use graphs for representing floorplans. Since in these works, floorplans are represented with graphs, all of them capture the floorplans structure, and some of them add attributes to nodes like furniture annotated in rooms. However, they do not include semantic and high-level features as well as human dynamic features. Moreover, all of these methods use graphs as a final representation and do not convert graphs to low dimensional vectors. These methods use graph matching methods for finding similarity, which is directly applied over graphs and are computationally expensive [11, 12, 13, 14].

Symbol-spotting methods are utilized for providing operations over floorplans, like retrieving similar floorplans. The query image (floorplan) is divided into some sub-images, and these sub-images are searched over other images. Based on the availability of queries (sub-images), the similarity is scored. Symbol spotting methods are applied to small object-centric datasets which do not have complex images [15, 16, 17, 18, 19].

As part of automation in floorplan analysis, generative design is an emerging topic that

relies on procedural techniques or deep learning. These methods are usually human-in-the-loop, where experts specify some constraints regarding their target floorplan. These methods generate a floorplan with satisfying those constraints either via procedural or deep learning methods [20, 21]. The ability to optimize environment structures is an emerging trend in design toward assisting in designing more context-aware layouts. In [2, 22], they optimize environment layouts within user-desired parameters like current generation methods. Training a model to learn the correlation between structure and design semantic and dynamic features can be a more robust approach for generating new floorplans and optimizing floorplans.

1.3 Proposed Approach

We propose *floorplan embedding* – "latent" representations of building layouts using an attributed graph as an intermediate representation that encodes not only design and structural features but also human dynamic features. Specifically, a Long Short-Term Memory (LSTM) autoencoder [23, 24] is trained to represent these graphs as vectors in a low dimensional vector space. To integrate features on edges (direction of room connections), we extend our model, and we propose a novel technique for floorplan representation that uses a parallel Long Short-Term Memory (LSTM) Variational Autoencoder (VAE) with attributed graphs as intermediate representations. We integrate the direction of room connections which means our representation is more accurate because the room symmetry is encoded. This model's generative nature also facilitates generative design applications that are studied in this thesis [25, 26].

In order to study the role of latent floorplan representations for design space exploration, we first develop a synthetic dataset of 5000 plausible floorplans of 12 unique styles, semantic and isovisit metrics associated with each. We then develop a new embedding model for this synthetic dataset using a Gated Recurrent Unit Variational Autoencoder (GRU-VAE) to represent floorplans in a latent space. Next, we demonstrate two local search approaches

over the latent space: (1) retrieval-based approach with nearest-neighbor queries, and (2) generative approach exploring the optimal vector sequences . We then run a series of experiments for the two approaches that search the floorplan dataset for 5 different metrics and interesting combinations of them. Besides, we release two annotated floorplan dataset with semantics and human dynamic features generated from simulations.

In this thesis, our main focus is on floorplan representations. However, we also consider the role of graphical representations in other application domains such as robotic manipulation, which is provided as an appendix to this thesis. We present a geometric approach to identify a complete set of object subsurfaces in a cluttered scene, which permits an end-effector to approach and grasp the object. This work proposes the motion constraint graph (MCG) representation for this purpose. It is able to efficiently reason about the space of permissible end-effector poses by placing constraints on the hand's configuration space. These constraints account for the surface area of contact points, end-effector kinematics, and collisions with objects in the scene. For a given end-effector and an arbitrary scene, it is possible to formulate and solve a constraint satisfaction problem to compute the set of reachable subsurfaces, which permit valid grasps. The proposed approach is general and applicable to any kind of scene geometry, such as arbitrarily cluttered scenes and curved surfaces and complex end-effectors with multiple degrees of freedom. In simulation experiments, the approach increases the success rate of grasp planning while also reducing total online planning time at the cost of a small amount of precomputation [27].

1.4 Main Contributions

The main contributions of this thesis are listed below:

1. A novel graph-based representation to efficiently encode attributed floorplans that contain semantic and dynamic behavioral attributes. Specifically, we propose RNN Autoencoder to embed graphs with features on nodes and edges.

2. We are extending the representations to have generative capacity using Variational Autoencoders (VAE's).

3. We are exploring the use of these representations for design space exploration and optimization. We utilize the embedding space by finding neighbors as well as sampling from embedding space.

4. A comprehensive set of quantitative experiments to validate our approach and demonstrate its efficacy on various tasks includes retrieval, generation, and optimization.

5. An expert user study to evaluate the validity of floorplans retrieved from the embedding space with respect to a given input (e.g., design layout).

6. The release of two datasets (the extended housexpo dataset and a newly created synthetic dataset) to maximize adoption and spark new research in this burgeoning area.

7. Appendix: A novel geometric approach to identify a complete set of object subsurfaces in a cluttered scene, which permit an end-effector to approach and grasp the objects in real-time.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. In chapter 2 we review prior works in floorplan analysis, representation, generation, and optimization. In chapter 3 we describe our initial model for representing floorplans. In chapter 4 we present our extended model for integrating edge features and generative capabilities. In chapter 5 we investigate the role of latent floorplan representations for design space exploration. Finally, we provide concluding remarks and discuss avenues of future investigations. Appendix A presents the role of graphical models of environment geometry for robotic manipulation.

CHAPTER 2
RELATED WORK

Section 2.1 summarizes techniques for floorplan analysis using static, geometric methods and using dynamic simulation-based methods. Chapter 2.2 reviews prior work in floorplan representation for computer-aided design applications. Chapter 2.3 reviews prior work in floorplan generation for generative design. Chapter 2.4 summarizes prior work in floorplan optimization and design space exploration.

2.1 Analysis of Floorplans

Analyzing building layouts, whether for virtual or built-environment, is a critical phenomenon in environment design. There has been a tremendous amount of work that analyzes different aspects of the environment such as structure [28], lighting [29, 30], energy [31, 32], construction efficiency [33], and building HVAC [34, 35].

It is also important to estimate how a design layout would impact the movement behavior of potential occupants of the space. Overlooking this can result in producing environment layouts that do not perform as expected in terms of operational productivity and user experience, and safety [36, 37]. Computational approaches are used to evaluate environment layouts for human occupancies. Researchers have used both static and dynamic workflows using geometric and topological environment design information and crowd simulations, respectively, to assess environments for occupants' movement behaviors.

2.1.1 Static Analysis of Floorplans

The Space-Syntax [38] methodology has been widely used among the static approaches to analyze human spatial behaviors in space [39]. It represents the environment as a spatial graph (e.g., visibility graph [40]) and computes spatial relations and connectivity among

the graph nodes to infer user behaviors. Some of the salient measures from Space-Syntax include *accessibility*, *visibility* and *organization* of space, and have been used to analyze occupants' movement behaviors [41, 42]. An interactive user-in-the-loop framework is presented in [43] that utilizes measures from Space-Syntax analysis to compute diverse alternate design layouts. A study is presented in [44] to investigate occupant navigation and way-finding tasks in a single and multi-level building design using Space-Syntax. The work presented in [45] uses Space-Syntax measures and allows designers to interactively analyze their environments for occupant movements from within the environment modeling platform (e.g., Autodesk Revit).

However, the static approach lacks any explicit modeling of occupants and their activities over time and solely relies on static spatial environment configurations.

2.1.2 Dynamic Analysis of Floorplans

Occupant movements and behaviors are contextual and dynamic in nature, and therefore, it is essential to evaluate the impact of the design layout on its potential inhabitants. To this end, the dynamic approach uses an agent-based simulation (e.g., crowd simulation) to analyze the dynamics of human movements in the environment. The crowd simulation methods explicitly model the individual virtual occupants or occupancy groups [46, 47], their behavioral characteristics (e.g., walking speed), and the activities they will engage in, to provide a time-based representation of the occupant–building interactions.

Crowd simulation techniques have been applied to simulate pedestrian movements [48] in numerous contexts, for example, to simulate day-to-day occupant behaviors in universities [49], hospitals [50], and in a retail environment and shopping mall [51]. They have also been used to understand and analyze emergent egress activities [52, 53]. The works presented in [54, 55, 56, 57, 58, 59] evaluate virtual corridors and hallway crossings by simulating crowds to approximate the egress flow as a function of design layout. Researchers have also explored parametric approaches where environment and crowd be-

haviour parameters were jointly explored to find optimal environment-crowd configurations for egress [60]. A multi-paradigm framework is presented for event-based simulations of dynamic crowds in built-environments to account for environmental conditions such as temperature and acoustics [61]. The work in [62, 63] presented a gamification approach to adopt crowdsourcing and community-driven design of environments based on crowd simulations using multiplayer games and networking. More recently, crowd simulation techniques have been used to compute and validate egress plans for the environment layouts by complying with building codes for means of egress [64].

By simulating and analyzing occupant movements in space, designers can generate more human-aware environment layouts.

2.2 Floorplan Representation

Floorplan representation aims to represent floorplans with numerical vectors that their structure and their features are encoded in these vectors. To the best of our knowledge, representing floorplans by numerical vectors is not done to date. There are some prior works for retrieving similar floorplans with representing floorplans as images or graphs. They can be mainly divided into three categories: image-based, graph-based, and symbol-spotting methods.

2.2.1 Image-based Methods

Several approaches based on conventional image processing techniques for comparing floorplans are proposed. In these approaches, floorplans are represented as images and Histogram of Oriented Gradients (HOG) [4], Bag of Features (BOF) [5], Local Binary Pattern (LBP) [6] and Run-Length Histogram [7] have been utilized for extracting features from these images. Then, these extracted features are used for comparison and retrieving floorplans. In [65] a deep CNN is presented for feature extraction to address the limitation of conventional image processing techniques for extracting features. This method

suits object-centric floorplans datasets in which floorplans are annotated with furniture or specific visual symbols. However, these features are not semantics and do not correctly capture the high-level design structures. Moreover, human behavioral features are not considered in these methods, whereas occupants' behaviors (i.e., trajectories) are often highly correlated with environments [10].

2.2.2 Graph-based Methods

In this category, floorplans are represented with graphs, and graph matching methods are utilized for measuring their similarity. Different strategies are used for representing floorplans as a graph. In [11] rooms are nodes, and edges capture the adjacency between the rooms. In addition, nodes are augmented with furniture types annotated in floorplans. In [12], the graphs are augmented with more attributes like room area and furniture style in three different representation layers. Since in these works, floorplans are represented with graphs, all of them capture the floorplans structure, and some of them add attributes to nodes. However, mainly they do not include semantic and high-level features as well as human behavioral features. Moreover, all of these methods use graphs as a final representation and do not provide numerical vectors. These methods use graph matching methods for finding similarity, which is directly applied over graphs.

2.2.3 Symbol Spotting Methods

Symbol spotting is a special case of Content-based Image Retrieval (CBIR) [15, 16] which is used for document analysis. By giving a query, system retrieves zones from the documents which are likely to contain the query. Queries could be a cropped or hand-sketched image. Pattern recognition techniques are used in symbol spotting methods like moment invariants such as Zernike moments in [17]. Reducing search space in symbols spotting methods is proposed based on hashing of shape descriptors of graph paths (Hamiltonian paths) in [18]. SIFT/SURF [19] features being efficient and scale-invariant are commonly

used for spotting symbols in graphical documents. Symbol spotting methods are applied to small datasets which do not have complex images, and they are only applicable for retrieval purpose.

2.3 Floorplan Generation

Floorplan generation aims to generate floorplan designs automatically by satisfying some constraints like room sizes and adjacency between rooms. We can divide them into two groups: Procedural methods and recently deep learning methods.

2.3.1 Procedural Methods

Procedural modeling techniques have been proposed for a wide range of virtual words, including buildings. In [66] an extensive survey is provided. In these methods, the constraints like dimension and adjacency constraints are manually defined, and optimization methods are used for constraint satisfaction to generate new floorplans [67]. In [21], they used the Bayesian network to synthesize floorplans with given high-level requirements. In [68] they proposed an enhanced Evolutionary Strategy (ES) with a Stochastic Hill Climbing (SHC) technique for floorplan generation. In [69] floorplans are generated with a combination of 2D shapes, which are used for extruding building facades. Shapes grammars is proposed in [70] for floorplan generation. They utilized shape grammars for generating a floorplan schema with its basics units(rooms). The rooms' functionality is assigned, and rooms are filled with furniture by an extensive library of individual room layouts. A graph-based method is proposed in [71]. User-defined grammars generate the graphs. The start point is the front door, and then public rooms are added, and their specific function is assigned. After this step, private rooms are created, and finally, stick-on rooms are introduced. With determining a 2D position for each room, these graphs are converted to a spatial layout. In [66] a generic semantic layout solving approach is proposed. By utilizing a semantic library, rooms are mapped to a class, and the constraints are defined based on semantic

adjacency between rooms. Our work is complementary to the wide body of work in procedural content generation [66] and focuses on the use of deep learning, specifically graph representation learning, for encoding floorplans.

2.3.2 Deep Learning Methods

There are a limited number of works with this approach. In [20] a deep network was proposed for converting a given floorplan layout as input to a floorplan with predicting rooms and walls location. In [72] they proposed a method comprising three deep network models to generate floorplans. In the first step, the model generates the layout, room locations, and furniture locations. In this model, users are in the loop, and they can modify the input for the next steps. In [73] they proposed a framework based on deep generative network. Users specify some properties like room count, and their model converts a layout graph, along with a building boundary, into a floorplan. A graph-constrained generative adversarial network is proposed in [74]. They took an architectural constraint as a graph (i.e., the number and types) and produced a set of axis-aligned bounding boxes of rooms.

2.4 Floorplan Optimization

The ability to optimize environment structures is becoming an increasing trend to help designers design more context-aware layouts. The work presented in [75] used current-generation crowd simulators with an adaptive mesh refinement approach to optimize the placement of pillars to increase the crowd flow in egress scenarios. A user-in-the-loop approach is presented [43] to optimize environment layouts for diverse design alternatives within user-set parameter bounds of design alteration using spatial environment configurations (e.g., topological and geometric information). The work in [22] presented a framework for a parameterized procedural representation of virtual environments, which are then altered using Markov Chain Monte Carlo (MCMC) to obtain the desired crowd behaviors during the simulations. However, all of these works explicitly search through the whole

solution space of the set parameters, which are usually high-dimensional and require extensive computation cycles. And for that very reason, these approaches are often prohibitive for complex and large-scale environment configurations.

An alternate approach to avoid the extensive search of the solution space is to use machine learning techniques to learn the relationship between the environment (e.g., a floorplan) and the desired objective (e.g., crowd flow). The work in [2] presented a neural network-based model to learn the relationship between environment layouts and crowd movements, which is then used in an optimization framework to automatically generate environments that yield user-desired crowd behaviors without running extensive simulations.

CHAPTER 3
FLOORPLAN EMBEDDING WITH LATENT SEMANTICS AND HUMAN BEHAVIOR ANNOTATIONS

3.1 Introduction

In this chapter we propose *floorplan embedding* – "latent" representations of building layouts using an attributed graph as intermediate representation that encode not only design and structural features, but also human behavior features. Specifically, a Long Short-Term Memory (LSTM) [23] autoencoder [24] is trained to represent these graphs as vectors in a continuous space. These vector representations are used to cluster and query floorplans with similar characteristics and attributes. Some of the advantages of presenting floorplans as vectors are: (a) compact representation, (b) efficient way to compare designs and fast retrievals, (c) scoring designs and providing feedback/recommendations, and (d) categorizing design according to our target features. In addition, since there is not any dataset augmented with semantics and human behavioral features, we release a novel annotated floorplan dataset with semantics and human behavioral features generated from simulations. The key contributions can be summarized as follows: (i) intermediate representation of floorplans as attributed graphs and augmented with crowd behavioral features (ii) novel unsupervised deep learning model to learn a meaningful vector representation of floorplans (iii) creation of a floorplan dataset augmented with semantic and crowd behavioral attributes generated from simulations.

3.2 Method Overview

The proposed framework is illustrated in Figure 3.1. It consists of two components. The first component is for preprocessing floorplans and convert them into attributed graphs. The

result of this step is a novel floorplan dataset augmented with design semantic and crowd behavioral features. The second component is the embedding for creating latent vectors of these graphs. These latent vectors represent floorplans as numerical vectors in a continues space. Further details on each component are presented in the following sections.

3.3 Dataset

HouseExpo dataset [76] is used in this work. It includes about 35000 2D floorplan layouts that are presented in JavaScript Object Notation (JSON) format. They are mostly floorplan segments belong to big buildings. A sparse labeling for layout components is also given for each floorplan. Figure 3.2 shows an image of a raw floorplan compiled as image. There are about 25 different components types in the whole dataset. Some of them, however, have similar semantics (e.g. toilet and bathroom, terrace and balcony, etc.). We reduce the components types to 11, considering only single type-name for rooms labeled for similar purposes and removing minor components like freight elevator.

3.4 Floorplans to Attributed Graphs

In order to convert floorplans into attributed graphs, first we recognize all potential room boundaries by a series of image processing operations. Then, we assign labels based on provided annotations. Some of the floorplans are not annotated perfectly (e.g. the given coordinates for the labels are not match with room coordinates). In such cases, we calculate overlaps of each room with given label coordinates, and then assign labels to rooms which have the maximum overlap.

The rooms are the nodes in the graph and edges are the potential connections between them. There will be an edge between two rooms if there is an immediate door between them. Edges are formed by image processing techniques. To detect doors, we create a combined image of two rooms and apply blob detection on it. If the number of blobs found is one, it means there is a door between them, otherwise, there is no connection between

the two rooms. Up to now, graphs are representing the structure of floorplans but they are not attributed. Next, we assign to nodes (rooms) their respective semantic features like square footage and room type, which are calculated using image processing techniques. To generate crowd related features, the 2D floorplans are converted to 3D models loadable in a crowd simulator, SteerSuits [77]. The simulator automatically populates virtual agents in each room with the target to exit the floorplan. It then calculates features like maximum, minimum and average evacuation times and traveled distances, as well as overall exit flow for all the agents in room. More details on these crowd aware features is presented in the following section. This part is done in an end to end procedure and does not need human interactions.

As mentioned, as a part of this work we are generating a novel floorplan dataset augmented with semantic and crowd behavioral features which is the result of this section. In this dataset, for each sample we have a JSON file augmented with semantic and crowd behavioral features.

3.5 Floorplan Dataset Features

This section presents the procedure for generating a floorplan dataset augmented with semantic and crowd behavioral features. At this point, we have floorplans which are presented as attributed graphs. The formation of a graph captures the structure of a floorplan, and attributes of nodes present its features. The available set of features can be seen in Table 3.1. These are divided into two groups: design semantic and crowd behavioral features which are generated with simulations. Design semantic features include room types and square footage which are generated by image processing techniques. Since room types are categorical features, they are presented as one-hot vector with 11 dimension and footage square is represented with an one dimensional scalar. The total dimension of semantic features is 12. All crowd behavioral features are represented with a one dimensional scalar value. In total we have 9 crowd features which lead to have a 9 dimensional vector for crowd

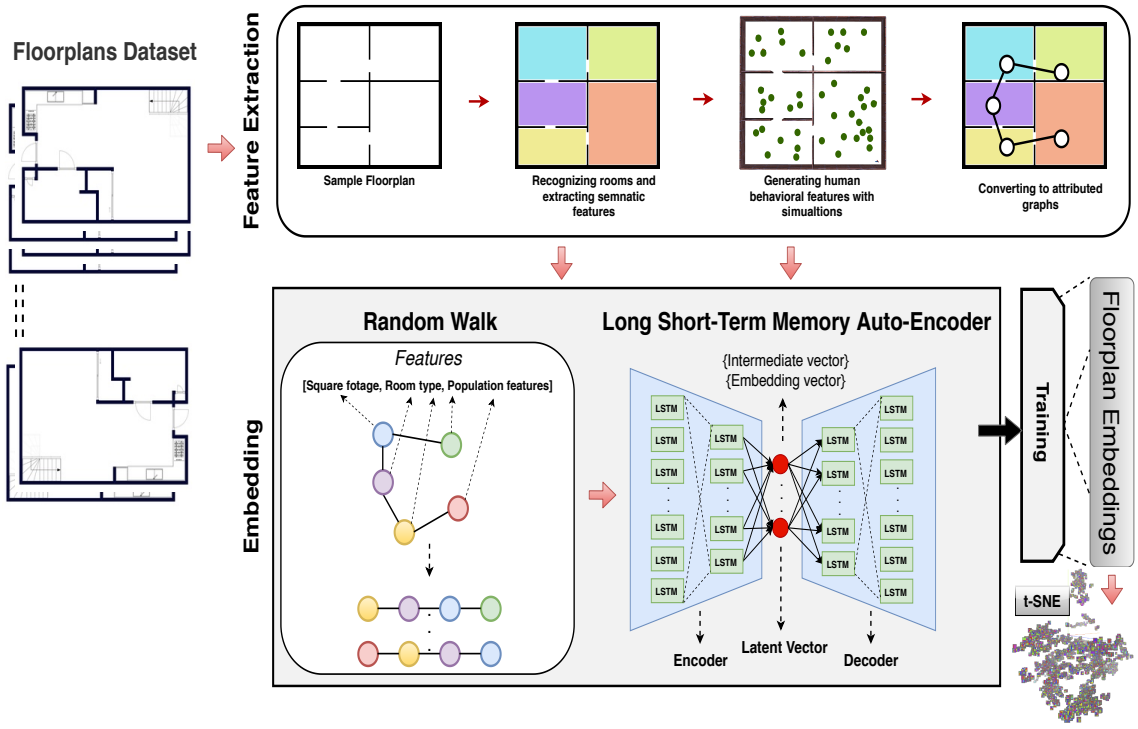
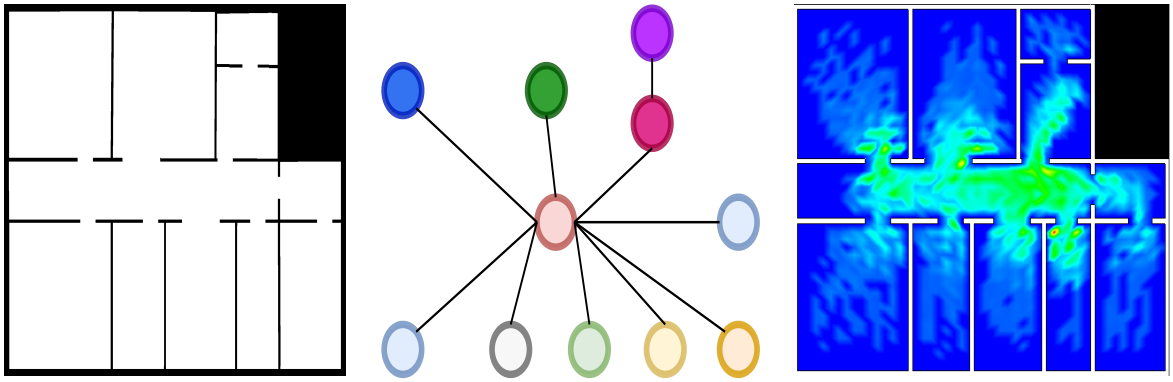


Figure 3.1: An overview of the proposed framework. In pre-processing, rooms and their design semantic properties are identified, then a 3D model is generated as input for crowd simulator to generate dynamic behavioral features, and at the end, the floorplan is converted to an attributed graph. In the embedding part, first a random walk is performed to convert attributed graphs to a set of sequences. Next, a LSTM autoencoder is proposed for training and floorplan predictions. The encoder has a 2 layer LSTM: first has 84 and second one has 64 LSTM units. The decoder has same architect contrariwise.



(a)Raw (b)Semantic Graph (c)Behaviour

Figure 3.2: (a) A sample raw floorplan from the dataset. (a) The black pixels are wall (or outdoors) and white pixels are building components (or indoors). (b) These floorplans are converted to graphs and augmented with design semantic and behavioral features in our preprocessing step. (c) Shows the visualization of behavioral features on floorplan.

features. Sum of all feature dimensions is 21.

However, our approach is not bound to use only the selected features, and more can be adapted into the framework depending on the application. The values of scalar features have different ranges. In order to keep all of them within a same range for robust training, we normalize them between [0, 1] except for features which are presented with one-hot vector. All the outliers are removed before normalization.

3.6 Floorplan Embedding

In this section we discuss the conversion of attributed graphs into continues latent vectors which encode both structure and semantics of the floorplans, as well as their crowd behavioral features. We use graph embedding approach to transform graph nodes, edges, and their features into a vector space (lower dimensional) while preventing any information loss. Graphs, however, are tricky to deal with because they can vary in terms of their scale, specificity, and subject [78].

There exist some approaches to perform graph embedding [79, 80, 81, 82]. However, they are only suitable for unattributed graphs and mostly capture just the graph structure. But in our case, the graphs are attributed. So in order to account for these attributes, we pro-

Feature Class	Features types	Dimension
Design semantic	Square footage	1
	Room type	11
Behavioral	Not completed agents	1
	Max evacuation time	1
	Min evacuation time	1
	Exit flow rate	1
	Completed agents	1
	Max traveled distance	1
	Ave evacuation time	1
	Avg traveled distance	1
	Min traveled distance	1

Table 3.1: First column shows the two feature classes, second column shows features available in each class and third columns is their dimension. Dimension for semantics features is 12 and for behavioral is 9, in total the features dimension is 21, all scaled between [0 1].

posed an Long Short-Term Memory (LSTM) autoencoder. Autoencoders [24] are trained to learn the full properties of the data and reconstruct their inputs. They generally have two parts: an encoder that maps the input to an intermediate representation and a decoder that reconstructs the inputs from intermediate representation. The intermediate representations are latent vectors. LSTM is a recurrent neural network (RNN) for capturing long-distance dependencies in sequential data and also supports varying data lengths.

We propose a LSTM autoencoder to learn embedding space in a way that keeps floorplans of similar structure, design semantic and crowd behavioral features, close to each another in the embedding space. Our proposed model is illustrated in Figure 3.1. We learn the function for mapping graph G to vector R in a d-dimensional space. In a floorplan (graph) $G = (V, E)$, V denotes its vertex set (rooms) and $E \subseteq VxV$ denotes its edge set (potential doors between rooms). We have unlabeled graphs in our dataset. Each node has a constant dimensional feature-vector F_V.

Adjacency matrix is one of the ways to present graphs as input to algorithms. But since the graphs are varying in number of nodes and edges, presenting them as adjacency matrix opens new challenges. This is because the dimension of the adjacent matrix is different for different graphs. To address this, we convert graphs to multiple varying length

sequences. This conversion is done by using Random Walk. In a random walk we start from a given source node and the next node will be selected randomly with probability $1/D(N)$, where $D(N)$ is the degree of node N. Each graph is converted to a set of sequences and these sequences are feed to the model for training. Then the average latent vectors of corresponding sequences to a graph are used as a representation vector for the graphs (Equation 3.1).

$$\Phi(G) = \frac{1}{Nseq} \sum_{n=1}^{Nseq} RS_n \tag{3.1}$$

The $Nseq$ is the number of sequences and RS is corresponding latent vector to each sequence.

3.7 Training

Some of malformed graphs are removed from the dataset. We use about 33000 floorplans' graphs which are converted to a set of sequences. We run random walks on the graphs to generate corresponding sequences. These sequences are feed into our model for training, both as input and output since the method is unsupervised. The loss function in our model is Mean Squared Error (MSE) (Equation 3.2).

$$loss(s) = \frac{1}{|s|} \sum_{i=1}^{|s|} (\bar{Y} - Y)^2 \tag{3.2}$$

Which s is the given sequence and $|s|$ is the number of nodes in the sequence. $\bar{Y}$ is the

reconstructed vector for a node and Y is the true vector for that node. Loss function calculates the difference between reconstructed vector(output of decoder) and input sequences which each node has feature vectors F_v. In other words the encoder is trying to reconstruct the node features in the sequence.

This section validates and showcases the potential of our embedding methodology with the help of 3 different use cases. For these use cases, we trained three models with the architecture described. The difference between these models is the considered features. First model is trained only with design semantic features, second only with behavioral features and the third with all of the features.

3.8 Pairwise Similarity between Floorplans

In this use case we demonstrate a pairwise comparison between input floorplan and its second nearest neighbour from the embedding space. Underline graphs of both floorplans are compared using a popular graph similarity distance metric, Graph Edit Distance (GED) [83]. The edit distance between $G1$ and $G2$, GED(G1, G2), is the count of edit operations that transform $G1$ into $G2$, where the edit operations on a graph G can be an insertion or deletion of an edge or node. All edit operations have the same constant cost which is 1. If two graphs are identical, their GED is 0.

First, we retrieved a semantically similar design (second nearest neighbour) from the embedding space for the given input floorplan such that they have similar structural attributes but significantly different behavioral attributes, Figure 3.3. The GDP value is reported as 0, showcasing that the graphs are similar in their design semantics. Average exit flow values and color-coded density heatmaps are also shown for both floorplans. Querying floorplan yielded comparatively lower exit flow than its nearest neighbour found from the embedding.

Second, we retrieved a behaviorally similar floorplan (second nearest neighbour) from the embedding space for the given floorplan such that they have similar behavioral attributes

but significantly different design semantics. A GDP value of 10 is reported from the graphs comparison, showcasing the two graphs are design semantically different. However, their corresponding floorplans yielded similar exit flow values. A GDP graph transformation as well as density heatmaps are shown, Figure 3.4.

3.9 Behavioral and Geometrically-powered Floorplans Retrieval

In this use case we demonstrate the potential of using our embedding methodology to retrieve geometrically-powered (includes static features), behaviorally-powered (includes crowd-based features) and combined, similar and related floorplans.

Figure 3.5 showcases an example for geometrically-powered floorplans retrieval. This embedding setting contains a 12 dimensional design semantic vector, 11 for room types and 1 for room dimensions. Given an input floorplan, a set of 5 nearest neighbors from the embedding space are retrieved. For an embedding space to be meaningful and valid, a queried graph should have itself as the first nearest neighbor during a retrieval, and therefore, in the figure, the first neighbor is same as the queried floorplan itself. For first query, the second and third nearest neighbours have same structure, the hallway in the middle and two bedrooms, one kitchen and one bathroom around it. Though, they have different design semantic features as annotated in the image. Fourth neighbour has close structure and only with missing the kitchen node. The last neighbour almost has same structure but the middle node is different.

Figure 3.6 showcases an example for behaviorally-powered floorplans retrieval. In this example the query graph is the same as last one and we retrieve the top 5 nearest neighbours. The model trained by behavioral features is used. All the neighbors have one node in the middle and four in the surroundings with same node degree. The second row shows simulation heatmaps for behavioral features. Since semantic features are not considered in this use case, two of the neighbours have totally different room types and more-less similar square footage. Also, the second nearest neighbour is exactly same as second nearest

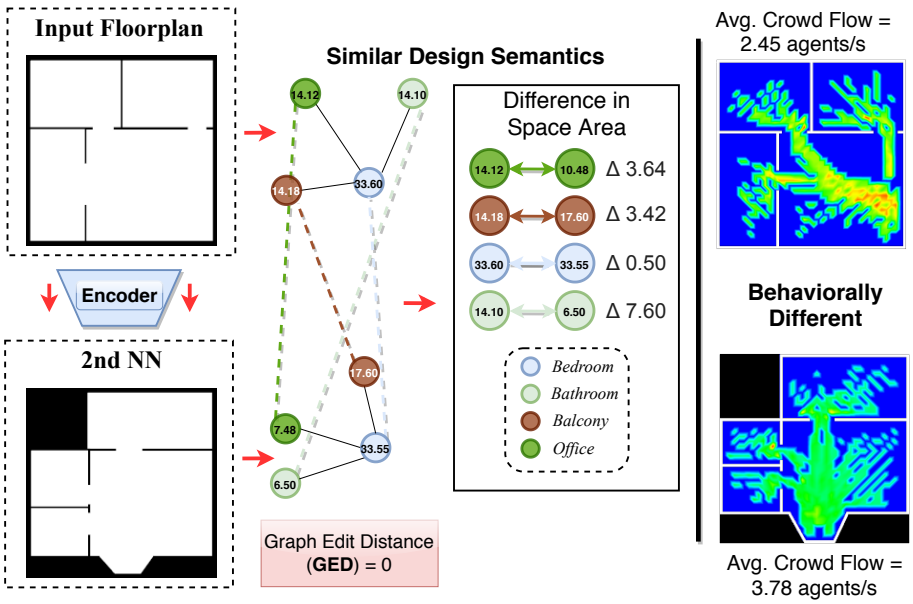


Figure 3.3: Two floorplans which are structurally similar (Graph Edit Distance (GED) = 0) but have different crowd behavioral attributes.

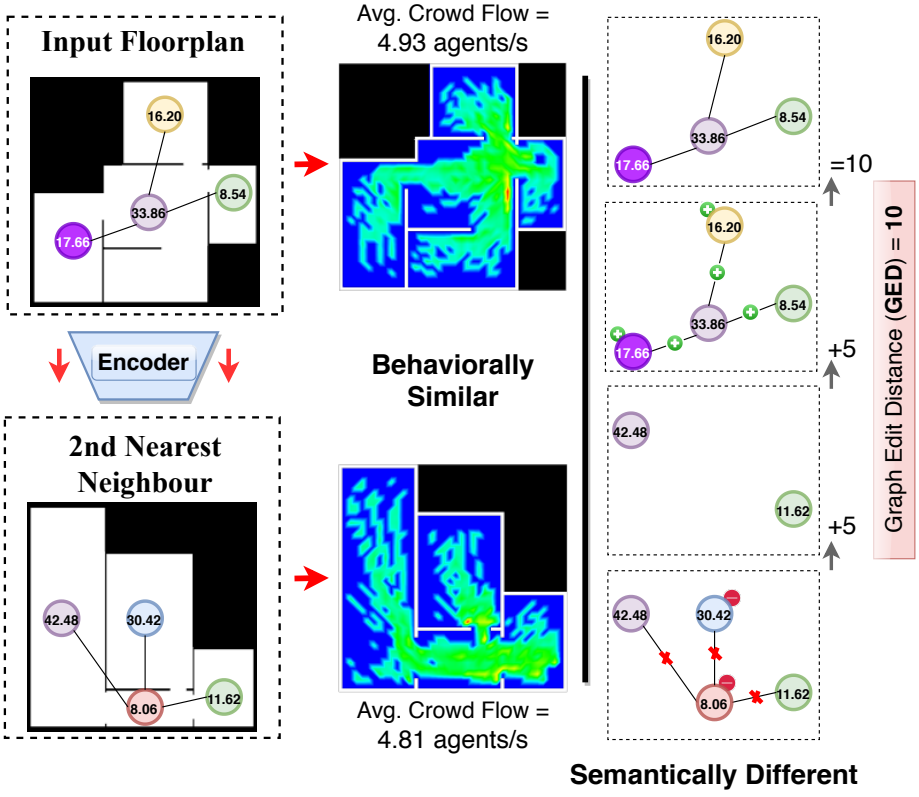


Figure 3.4: Two floorplans which are behaviorally similar (having similar average exit flows) but design semantically different (Graph Edit Distance (GED) = 10).

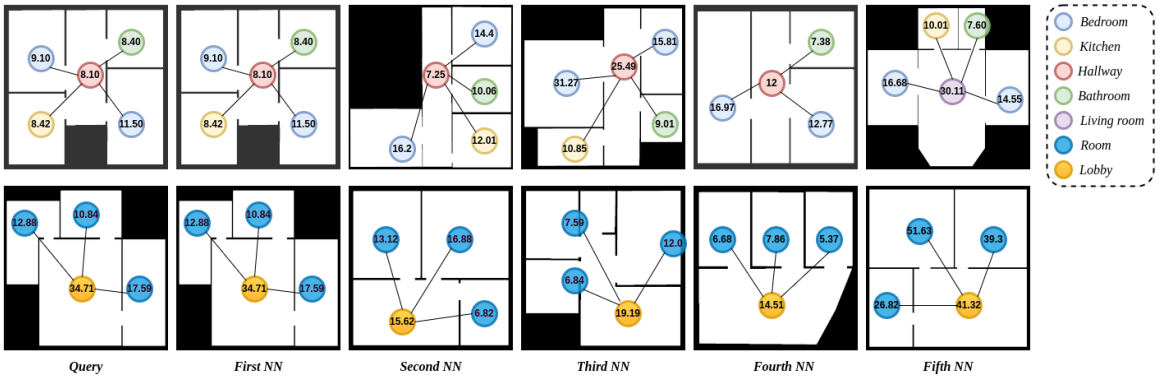


Figure 3.5: Floorplans retrieval from the embedding with respect to design semantic features alone. Top 5 nearest neighbours of two queried floorplans are shown.

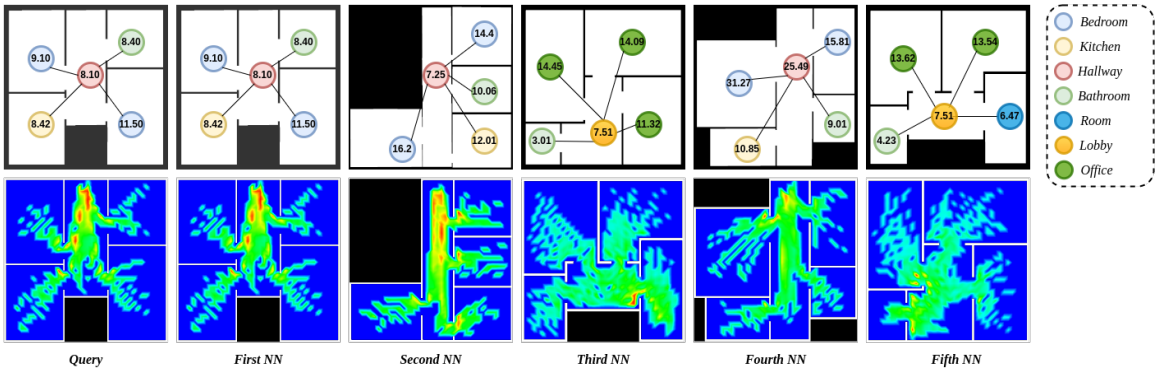


Figure 3.6: Floorplans retrieval from the embedding with respect to behavioral features alone. Top 5 nearest neighbours of for queried floorplan is shown. The ranking of the neighbours is computed based on differences in their behavioral attributes. Time-based behavioral dynamics for human-building interactions are also shown as color-coded heatmaps, where red areas highlight over crowded regions in space.

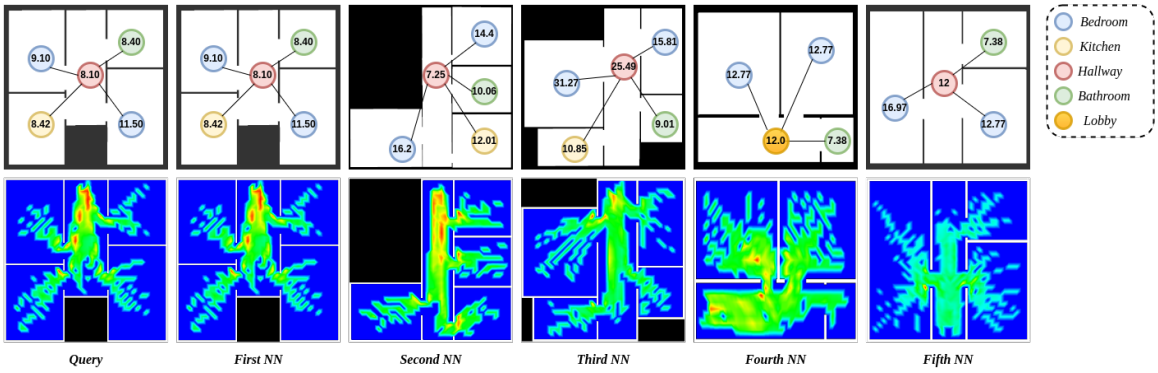


Figure 3.7: Floorplans retrieval from the embedding with respect to combined design semantics and behavioral features. Top 5 nearest neighbours for the queried floorplan is shown. Time-based behavioral dynamics for human-building interactions are also shown as color-coded heatmaps, where red areas highlight over crowded regions in space.

neighbour in case 1. It shows even without design semantic features, the structure and behavioral features can capture floorplans similarity.

Figure 3.7 showcases an example of floorplans retreival from a collectively trained model with both semantic and behavioral features. The features dimension for each node is 21. The top 5 nearest neighbours for the same input floorplan are retrieved from the embedding. Since in this case all of the features are considered, the neighbours must follow similar structure, similar design semantic and also similar behavioral pattern. Regarding structure, they almost have same structure, one node is missing in fourth and fifth neighbours and one node is different in fourth neighbour. But node degrees and structure of the graphs are almost same. The heat-maps visualizing the behavioral features are shown in second row and as it is clear their similarity decreases for later neighbours. Semantic features as annotated in the images are more-less similar. There are some noticeable points in this study. First the floorplan in second rank has the second rank in two later cases. This shows the validity of embedding space which the similarity and dissimilarity between floorplans are captured perfectly in vectors in embedding space. The third neighbours is also seen in last two cases. Semantically it was the third neighbour but behaviorally it was in fourth rank, the accumulation of both class features push it in third rank. This behavior is another justification for embedding space validity and benefit of considering behavioral features. Behavioral features weakly represent the semantic attributes and integrating them helps to fin the best fit nearest neighbours. Third point is the presence of a new floorplan in fourth rank which is not seen in both cases but the accumulation of features classes as discussed later lead it to get fourth rank. Fifth neighbour was the fourth neighbour in first study and was not seen in second study. It means this floorplan has similar structure design semantic but behaviorally different pattern which their accumulation keeps it in fifth rank.

In order to have an evaluation over embedding space, we find the rank of each floorplan by itself between top 5 nearest neighbours. A meaningful embedding space should lead to a case that each graph has its own as first nearest neighbor. We repeat this process for

Ranks/Models	**Model1**	**Model2**	**Model3**
First rank(query graph itself)	100%	100%	100%
Second rank(proxy graph)	86%	83%	85%
Third rank(proxy graph)	13%	13%	14%
Fourth rank(proxy graph)	1%	3%	1%
Fifth rank(proxy graph)	0	2%	0

Table 3.2: This table shows the percentage of the floorplans that have them-self as first nearest neighbours and the rank percentage of their proxy graphs.

all three models, model trained with only design semantic features, model trained with only behavioral features and model trained with all features. As Table 3.2 shows, in all models the query graph by itself is first nearest neighbor. As additional metric, we generate one proxy graph for each floorplan. Proxy graphs are the result of running random walk one more time that makes different sequence for each floorplan. These proxy graphs are not involved in training. We feed them to the trained model and add their representative vectors to embedding space. Then we find the top 10 nearest neighbors for each floorplan. The reason for top 10 nearest neighbours is to make sure we see them between neighbours. Having these proxy graphs in higher ranks shows the validity of embedding space. The result is provided in Table 3.2 for all three models. The reason that all of the proxy graphs are not seen in second rank is because our graphs are unidirectional and we allow having loop in random walk. The randomness in random walk could lead to sequences that is back and fourth between only two nodes or a repeated subsequence because of loop. In both cases the generated sequences are not presenting the graph properly. This situation also has dependency to node degrees in the graphs since next node in random walk is selected based on weighted probability of nodes degree. This can be addressed by running random walk more than one time for each node or avoiding loops, both in training and test time.

3.10 Generation of a Composite Floorplan

Embedding spaces have commonly been used to make predictions and retrieving similar objects given some input criterion or an object. In this use case we demonstrate that our

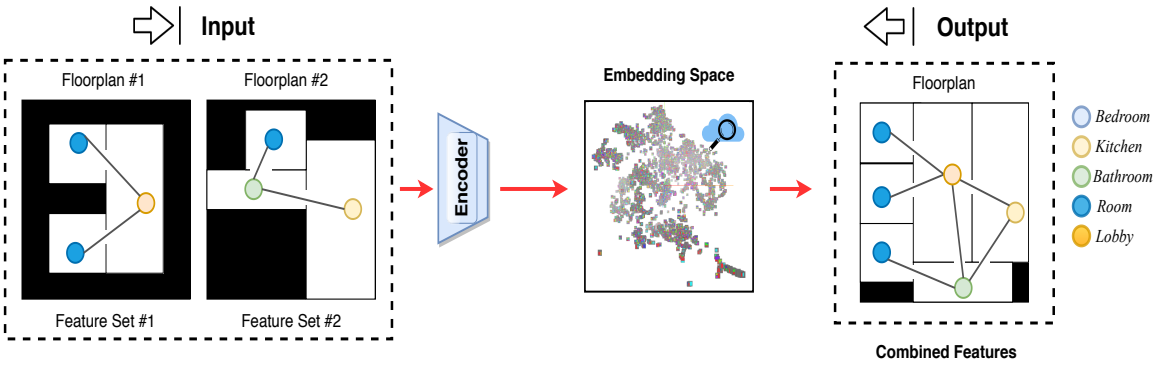


Figure 3.8: Generation of a single floorplan with combined features from an embedding space given two floorplans with different set of features as input. Nodes of underline graph for each floorplan are color-coded based on room types.

embedding methodology can also be used to generate a compound floorplan with collective features given multiple floorplans with a set of completely or partially different features. Figure 3.8 showcases one such example. Two floorplans with 2 and 3 different room types respectively, are given as input to our autoencoder, and the first nearest neighbour is a composite floorplan consist of 5 different room types. It not just contains all the features from input floorplans but also maintains a significant proportion of their geometric symmetries.

CHAPTER 4
GRAPH-BASED GENERATIVE REPRESENTATION LEARNING OF SEMANTICALLY AND BEHAVIORALLY AUGMENTED FLOORPLANS

4.1 Introduction

Floorplan representations support a set of fundamental activities in the architectural design process, such as the ideation and development of new designs, their analysis and evaluation for any selected performance criteria, and the communication among the stakeholders. While Computer-Aided Design (CAD) and Building Information Modeling (BIM) approaches to support the creation of digital building models from which floorplans can be extracted, these methods do not support the systematic representation or comparison of floorplan features, which could be derived from geometric and semantic properties, as well as more advanced performance metrics, such as space utilization and occupant behaviors [21, 54]. Image processing techniques and Convolutional Neural Networks (CNNs) have been utilized to extract features from floorplan images [11, 65]. These features are used for floorplan representation. In another branch, the floorplans are represented with graphs, and graph matching methods are utilized for comparing and retrieving similar floorplans [13, 12]. Other approaches like symbol spotting methods are utilized for retrieving similar floorplans. However, none of these works represent floorplans with numerical vectors. Besides, they overlooked the design semantic and human behavioral features.

We propose a novel technique for floorplan representation that models floorplans with attributed graphs as an intermediate representation to address the limitations in previous works. A novel Long Short-Term Memory (LSTM) Variational Autoencoder (VAE) model is proposed to embed the attributed graphs in a continuous space. This method considers the design semantics and high-level structural characteristics, and crowd behavioral at-

tributes of potential human-building interactions. This approach represents floorplans with numerical vectors in which design semantics and human behavioral features are encoded. These vectors facilitate different applications related to floorplans, such as recommendation systems, real-time evaluation of designs, fast retrieval of similar floorplans, and any application that needs to cluster floorplans. The qualitative and quantitative results show the performance of our model for generating representative embedding vectors such that the considered features are encoded accurately. A user-study is conducted to validate floorplan retrievals from embedding spaces to their similarity with the input floorplans. Floorplan generation is an active area of research in computer graphics. Recently floorplan generation methods based on machine learning have been integrated into design workflows to facilitate and enhance the design process, [73, 72, 20]. Although floorplan generation is not our approach's primary goal, the proposed model is generative. We can automatically generate floor plans with desired characteristics, as demonstrated by our experiments.

Our contributions can be summarized as follow:

- A workflow to represent floorplans as attributed graphs, augmented with design semantic and crowd behavioral features generated by running crowd simulations.
- A novel unsupervised generative model to learn a meaningful vector representation of floorplans using LSTM Variational Autoencoder.
- Generation of new floorplans using the proposed model. A user study to evaluate the qualitative performance of our approach.
- Provision of a publicly released dataset of floorplans of indoor environments, which are augmented with semantic and behavioral features. The semantic design features are extracted by our automated tool, and the human behavioral features are generated by hours of the running simulation.

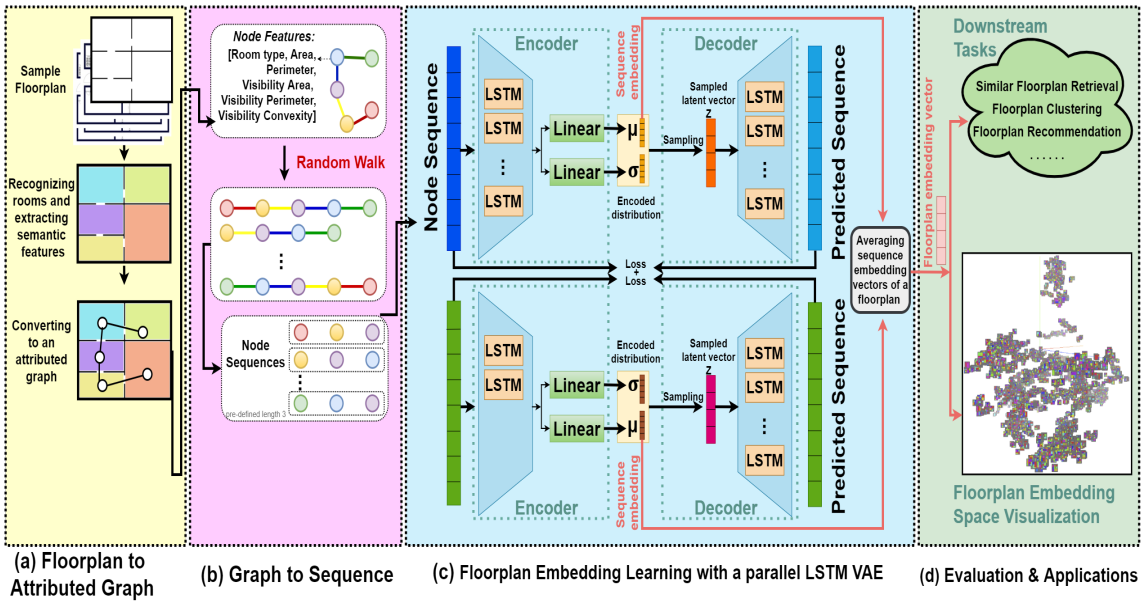


Figure 4.1: An overview of the proposed approach. **(a)** floorplan is firstly converted to an *attributed graph* as immediate representation, with attributes residing on both nodes and edges; **(b)** *random walk* is applied to the attributed graph to generate a set of node sequences and edge sequences; (**c**) floorplan embedding vector is learned with a novel parallel LSTM VAE model; (**d**) the learned floorplan embedding vector can be used for visualization and many other downstream tasks.

4.2 Method Overview

Figure 4.1 illustrates the proposed framework, comprising of two components. The first component models floorplans with attributed graphs that nodes and edges of these graphs are augmented with design semantic and human behavioral features. The second component embeds attributed graphs in a continuous space. The details are provided in the following sections.

4.3 HouseExpo++ Dataset

We used the HouseExpo dataset [76] that includes $35,126$ 2D floorplan layouts in JavaScript Object Notation format. There are 25 room types in this dataset where some of them share similar semantic labels (e.g. toilet and bathroom or terrace and balcony). We reduced the types to 10, including unknown types. This reduction is done by removing less common

types based on reported statistical metrics in the dataset (e.g. freight elevator) and considering a unique type for similar propose components. The final room types are Bedroom, Bathroom, Office, Garage, Dining Room, Living Room, Kitchen, Hall, Hallway and Unknown. Unknown type is considered for room segments with a noisy label. Additionally, we remove from the set the floorplans with inaccurate or missing labels. At the end of this process, we obtain $8,729$ floorplan layouts. This preprocessing is done to make the dataset solid for training. Corrupted data will decrease the training accuracy and, consequently, the test accuracy with undesirable outcomes of misleading the model.

The original HouseExpo dataset includes $35,126$ 2D floorplans. For each floorplan, the number of rooms, bounding box of the whole floorplan, a list of vertices and a dictionary of room categories, as well as their bounding boxes are provided [76]. While we can use the provided bounding boxes of the rooms for segmentation, these bounding boxes are not accurate, so we use them only for labeling. We compile these floorplans (in JSON format) to images. Then, we segment the images to find the rooms, their connections if there are any, the direction of connections and their square footage. The provided bounding boxes in the original dataset are used for assigning labels to room segments by the criterion of maximum overlapping. The described processes are done with our automated tool by image processing techniques. Moreover, we convert these JSON-formated floorplans to files in a readable format with our 3D crowd simulator (SteerSuite), and by running simulations, we record the human behavior features (features are provided in Table 4.1).

4.4 Floorplans to Attributed Graphs

After pruning the dataset, we model each floorplan with an attributed graph. The rooms compose nodes, and the edges are their connectivity if there is an immediate door between the room pairs. For this conversion, we compiled HouseExpo samples as images. Then we utilized a series of image processing techniques for room segmentation and finding their connectivity. Graph structure resembles the structure of floorplans like the number of

	Feature Classes	Feature Types	Dimension
Node Features	Design Semantic	Square Footage	1
		Room types	1
	Behavioral	Not completed agents	1
		Max evacuation time	1
		Min evacuation time	1
		Exit flow rate	1
		Completed agents	1
		Max traveled distance	1
		Avg evacuation time	1
		Avg traveled distance	1
		Min traveled distance	1
Edge Feature	Design Semantic	Direction	4

Table 4.1: Features on nodes and edges.

rooms and their connectivity. Considering floorplan structure is necessary but not enough. To have a better and more meaningful representation, we need to integrate high-level design semantic features. Moreover, humans inhabit these buildings, and their interaction with the environment provides valuable implicit information. Integration of how they interact with environments is necessary for safety or other types of design metrics like visibility and accessibility. Therefore, we augmented the graphs with both high-level design semantic features and human behavioral features (Table 4.1).

The semantic design features include room type, square footage, and the connection direction. The room types represented with a 10-dimensional one-hot vector where if the type is i^{th} type and other entities are zero $roomType_i = 1$. The square footage is represented with a scalar value and direction of connection with a 4-dimensional one-hot vector. We considered four main directions: North, East, South, and West. Thus, $direction_i = 1$ if direction belongs to i^{th} direction and other entities are zero. The room types are provided as a label in the dataset, and image processing techniques extract both square footage and direction of the connection. We are not given the cardinal directions. Therefore, we considered the top left corner of floorplan images as the origin. Hence, the $+y$ axis points to the

north, and other directions are considered relatively. For calculating the direction, we need a reference point (i.e., room). The direction between rooms is the direction from the node (i.e., room) with the highest degree (room with more connections) to the node with a low degree. Usually, the node (i.e., room) with the highest degree is the main room in the floorplans like the living room or hallway. In other words, the node with the highest degree is the reference for setting the directions. Please note in the graph the edges are bidirectional if there is a door between two rooms, and connection direction is the assigned feature to edges. The room type and square footage are specific to each room, and we consider them as node features. However, the connection direction is a shared property between room pairs; we add it to the edge features (edge between room pairs).

The human behavioral features are generated by simulation (Figure 4.3). They include metrics regarding evacuation time, traveled distance, flow rate, and the number of successful/unsuccessful agents to exit from the corresponding building (Table 4.1). To generate these behavioral features, we converted 2D floorplans to 3D models loadable in a crowd simulator, SteerSuite [84]. The simulator automatically populates virtual agents in each room with the target to exit the floorplan. Note that in our simulations, the only obstacles the agents interact with are the walls of the environment (static obstacle) or the other agents (dynamic obstacle). However, our simulation setup does not restrict us from including different kinds of obstacles in the environment (e.g., pillars or other physical objects). The past research has shown that the placement of pillars or other obstacles at proper locations can often facilitate movements (e.g., crowd flow) during the evacuation of the environment [56, 57, 55].

Figure 4.2 shows a snapshot of the simulation in the presence of 4 obstacles in the environment. All human behavioral features are presented with one scalar value with a total dimension of 9. These features are generated for each room; hence we added them to the nodes feature vector.

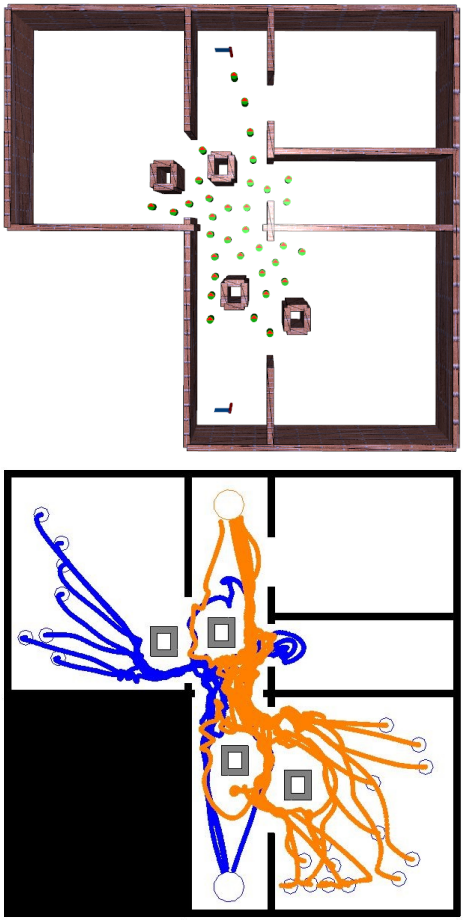


Figure 4.2: A snapshot of the simulation in the presence of the obstacles (e.g., pillars/hurdles). Top: the crowd simulation in 3D. Bottom: Crowd trajectories overlayed on top of the environment layout.

4.5 Floorplan Embedding

We model the floorplans with attributed graphs with features on both nodes and edges as described in section 4.4. These graphs represent floorplan geometry, their design semantics, and behavioral features. They can be used directly for floorplan representation. However, graph analysis is expensive in terms of computation and space cost. This challenge is addressed by proposing efficient graph analysis methods like [85, 86, 87] but are not efficient enough. Besides, These methods do not represent graphs with a compact numerical vector. Another solution for addressing the complexity of graph analysis is graph embedding. Graph embedding maps the graphs to a low dimensional space in which their properties and information are maximally preserved. In this low dimensional space, the graphs with similar properties are close. We have different graphs like the heterogeneous graph, homogeneous graph, attributed graph. It means the input for graph embedding methods are varying, and a single method can not handle all types. Graph embedding can mainly be divided into node embedding, edge embedding, hybrid embedding, and whole-graph embedding [88].

We are dealing with whole attributed graph embedding. We want to represent each attributed graph with a vector. These vectors encode graph (floorplan) structure as well as their design semantics and behavioral features. Besides, these graphs are unlabeled, i.e., we do not have a label for each graph to perform supervised classification or regression. Moreover, the graphs vary (in terms of the number of nodes), and the number of nodes is relatively small. We can use other types of embedding like node embedding and then use the node embedding vectors' average as the whole-graph representation. However, with this strategy, the whole graph structure is not appropriately captured and does not lead to accurate vector representation [89, 80].

There are quite a few works for the whole graph embedding. Some whole-graph embedding methods rely on the efficient calculation of graph similarities in graph classification

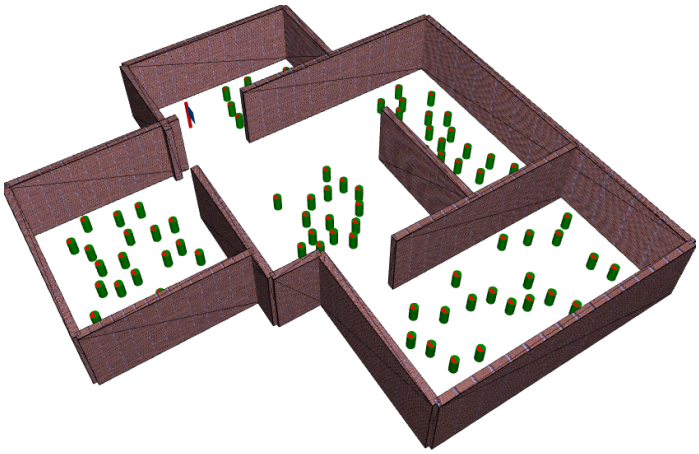


Figure 4.3: Crowd simulations are used to compute behavioral features for the floorplans. Crowd simulations are used to compute behavioral features for the floorplans. Layout walls are shown in Brown, crowds are shown in Green, and the Blue Flag shows the building exit point (for evacuation).

tasks [90, 91, 92, 89]. These methods are supervised and need a labeled dataset. Besides, they are designed for unattributed graphs. On the contrary, our graphs are attributed and unlabeled. Therefore, these methods do not apply to our problem, and we need an unsupervised method. Graph2Vec [82] is an unsupervised method that maximizes the likelihood of graph subtrees given graph embedding and generates vector representations. However, since this model uses subgraphs, the global graph information is not captured correctly [80]. Besides, this method is not applicable for graphs with attributes both on nodes and edges. In [80] for capturing the whole-graph structures, they take advantage of random walk for converting graphs to a set of sequences. Then an LSTM autoencoder is presented to learn graph representations. However, this method suits unattributed graphs.

Sentences are presented with a sequence of words. Each word in sentences is represented with a (embedding vector). In other words, we have a sequence of vectors in sentences. In [93, 94] LSTM Variational Autoencoder is used for text and sentence embedding and generation. The methods proposed in [93, 94, 80] motivate us to convert our graphs to sequences (like sentences which are a sequence of vectors) and propose a novel LSTM Variational Autoencoder model that suits our unlabeled attributed graphs with feature both

on nodes and edges. In particular, we convert each floorplan (graph) into a set of sequences (section 4.7), and we propose a generative model that maps our graphs (sequences) to a d-dimensional space $\theta : G \rightarrow R^d$. The proposed model is detailed in the next section.

4.6 Model

We present a novel LSTM Variational Autoencoder architecture illustrated in Figure 4.1. LSTM is a special kind of Recurrent Neural Network (RNN). It is designed for learning long-term dependencies by introducing state cell [23] to address the vanishing gradient in vanilla RNN with long-term sequences [95]. Autoencoders are a type of unsupervised neural network with two connected networks. The first network is an encoder that converts the inputs to latent vectors in a low dimensional space. The second network is the decoder, which reconstructs the original input vector from latent vectors [96]. However, the vanilla autoencoders map each input to a constant vector. The embedding space with vanilla autoencoder is not continuous, and interpolation is not allowed. Variational Autoencoder (VAE) is a generative model designed to address vanilla autoencoders' limitations by learning the Probability Density Function (PDF) of the training data [97]. The VAE generates a continuous embedding space in which vector operations are allowed.

As mentioned we have attributes both on nodes and edges with different dimensions. To address this difference in dimension, we consider two parallel LSTM VAE, one for node sequences and one for corresponding edge sequences. Let $S_n = s_{n_1}, s_{n_2}, ..., s_{n_i}$ be a node sequence and $S_e = s_{e_1}, s_{e_2}, ..., s_{e_{i-1}}$ be the corresponding edge sequence. We aim to learn an encoder and decoder to map between the space of these two sequences and their continuous embedding $z \in R^d$ where d is a hyperparameter. In each branch, the encoder is defined by a variational posterior $q_\phi(z|S_n)$ and $q_\phi(z|S_e)$ and the decoder by a generative distribution $p_\theta(S_n|z)$ and $p_\theta(S_e|z)$, where θ and ϕ are learned parameters. For each branch, loss function has two terms (Equation 4.1). The first term is reconstruction error that we used Mean Square Error (MSE). This term encourages the decoder to learn

to reconstruct the data $\bar{S}_n, \bar{S}_e$. The second term which is a regularizer is Kullback-Leibler (KL) divergence to penalize loss if the encoder outputs representations that are different than a standard normal distribution $N(0,1)$ [98]. In training, two branch loss functions are summed for back-propagation (Equation 4.1).

$$Loss_{total} = (||S_n - \bar{S}_n||^2 + KL[q_\phi(z|S_n), N(0,1)]) + (||S_e - \bar{S}_e||^2 + KL[q_\phi(z|S_e), N(0,1)]) \tag{4.1}$$

In both branches for the encoder, we have an LSTM layer with 256 units. In the following, we have two fully connected layers with dimension 16 for generating μ and σ and consequently the $d = 16$. Then we have the sampling, and finally, we have the decoder with one LSTM layer with 256 units for reconstructing the input sequences. The number of layers and number of units/neurons are set experimentally for best performance. Adam [99] is used as optimizer. Before training, each graph is converted to a set of sequences (section 4.7), and these sequences are used as input for training. After training, each graph is represented by averaging its sequences' embedding vectors.

4.7 Graphs to Sequences

Graphs can be converted to sequences by methods including but not limited to random walk or Breadth-First Search (BFS). In [80] the random walk, BFS, and shortest path between all pairs of nodes are utilized. The experiments show sequences generated by random walk lead to a better vector representation. The reason is random walk captures more than immediate neighbors of nodes. The random walk is introduced in [100] for converting graphs to sequences. In this version, we pick a node, and then we choose one of its edges randomly to move to the next node. We repeat this procedure until we get a walk of some predefined length (length of a walk is defined by the number of nodes on the walk, and a

shorter walk than the predefined length will be padded into the predefined length). Later, two other versions are proposed.

Random Walk 1. In [79] the random walk is modified to have two parameters Q and P. Parameter Q is the probability of discovering the graph's undiscovered parts, and parameter P is the probability of returning to the previous node. We call the random walk proposed in [79] 'Random Walk 1'.

Random Walk 2. In [80] the random walk is modified by adding probability $1/D(N)$, where $D(N)$ is the degree of node N. We start from a node in this walk, and the next node will be selected by its probability $1/D(N)$. We call this random walk presented in [80] as 'Random Walk 2'.

We used both walks with different walk lengths to find the best performer walk and walk length. Besides node sequences, the edge sequences are captured at the same time. Both nodes and edge sequences are used for training the model.

4.8 Training

As described in section 4.7, we converted graphs to nodes and edge sequences. We utilized both mentioned walks with walk lengths 3, 5, and 7. For random walk 1, we set both Q and P to 0.5. For each graph, we run the random walk 11 times. Therefore, we have 11 sets of node and edge sequences for each graph. Out of 11, one set is considered as a proxy set (proxy graph). These proxy sets do not participate in training and are only used later for evaluation. As mentioned, we have two random walks, and we run each of them with walk lengths 3, 5, and 7. In total, we have 6 different sets of sequences. On average, we have 5 nodes in each graph. However, because of randomness in random walks, all sequences are not valid. For example, it happens to have a sequence which is the repetition of two nodes. These types of irregularities are pruned. On average, we have 306440 node sequences and 306440 corresponding edge sequences in these 6 sets. The sequences with lengths less than the target length are padded with zeros. We trained three models considering a different set

of features.

Model 1. In this model, we only considered the design semantic features on nodes. We removed the edge branch, and the model is trained only with nodes sequences. The dimension of node features is 11.

Model 2. In this model, we considered semantic design features both on nodes and edges. The model is trained with both branches. The features dimension on nodes is 11 and on edges is 4.

Model 3. In this model, we considered all semantic design features and human behavioral features. The model is trained with two branches. The features dimension on nodes is 20 and on edges is 4.

All three models have the same described architecture and loss function. Note that in Model 1, the edge branch is removed, and we have only the node branch's loss function. However, in the other two models, the loss is the summation of two branches' loss. The learning rate was set empirically as 0.001. All three models are trained on a machine with 32 GB RAM, 12 * 3.50 GHz cores CPU, and Quadro K620 GPU with 2 GB Memory. On average, each model takes about 4 hours for training with 50 epochs.

4.9 Quantitative Evaluation

This section presents quantitative results from the experiments.

4.9.1 Nearest Neighbours Ranks

As mentioned, by embedding, the graphs are mapped to a continuous embedding space. The graphs with similar structures and properties should be close to each other in the embedding space. The closeness of two floorplans can be measured by the euclidean distance of the two corresponding embedding vectors (smaller distance denotes higher similarity). If this embedding space is well constructed, similar floorplans in terms of structure, semantic and behavioral properties should be closed. Therefore, for each graph (called query graph),

we compute the euclidean distance from this graph to other graphs (including itself) and rank the other graphs for this graph according to the distance. We hypothesize that each graph should find itself as the first nearest neighbor and its proxy graph (a different set of sequences for query graph) in close ranks. This study is independent of considered features on graphs and only shows the model's effectiveness for generating valid embedding vectors. Therefore, we use all 3 models to obtain the top 5 nearest neighbors for each floorplan in their corresponding learned embedding space. We calculate the average percentage of graphs that have themselves in the first rank and the percentage of proxy graphs in the other four ranks. Table 4.2 shows these average percentages with different walk lengths for both random walks.

As Table 4.2 shows, in both random walks, walk length 5 leads to better performance. In addition, the random walk 2 is superior. By random walk 2 and walk length 5, each graph by itself is in the first rank and 94% of proxy graphs in second rank. Since proxy graphs are a different set of sequences on the graphs, if the model performs properly, a good percent of the proxy graphs should be present in top ranks. Random walk 2 performs better since it captures our graphs structure better because in graphs (i.e., floorplans) we have always a main node (i.e., room) with high degree. Then moving toward this node gives the sequences that capture our graph structure better. The walk length has dependency to size of the available graphs in dataset. For us walk length 5 is the suitable length since in both random walks, the embedding performance is better in compare to walk length 3 and 5.

As Table 4.2 shows, in both random walks, walk length 5 leads to better performance. Besides, random walk 2 is superior. By random walk 2 and walk length 5, each graph is in the first rank and 92% of proxy graphs in the second rank. Since proxy graphs are a different set of sequences on the graphs, a good percent of the proxy graphs should be present in top ranks if the model performs properly. Random walk 2 performs better since it captures our graph's structure better because, in graphs (i.e., floorplans), we always have the main node (i.e., room) with a high degree. Then moving toward this node gives the

	Walk Length	Rank of query floorplan within 5 NN	Rank of proxy graph within 5 NN
Random Walk 1	3	[100, 0, 0, 0, 0]	[0, 51, 2, 1, 1]
	5	[100, 0, 0, 0, 0]	[0, 76, 4, 2, 2]
	7	[100, 0, 0, 0, 0]	[0, 56, 2, 1, 1]
Random Walk 2	3	[100, 0, 0, 0, 0]	[0 , 85, 3, 2, 1]
	5	[100, 0, 0, 0, 0]	[0, 92, 2 , 1, 1]
	7	[100, 0, 0, 0, 0]	[0, 88, 3, 1, 1]

Table 4.2: Average of nearest neighbor ranks (subsection 4.9.1) with two types of random walks and walk length 3, 5, or 7 for our three models. For each graph, we compute the euclidean distance from this graph to other graphs (including itself and a proxy graph which is a different set of sequences of the query graph) and rank the other graphs for this graph according to the euclidean distance. Each graph should find itself as the first nearest neighbor and its proxy graph in close ranks. We calculate the percentage of graphs that have themselves in the first rank and the percentage of proxy graphs in the other top four ranks (e.g., '[0, 76, 4, 2, 2]' in the table denotes that 76% graphs have their proxy as the top 2 nearest neighbor, 4% as top 3, 2% as top 4, and 2% as top 5). This analysis showcases that walk length 5 can lead to better performance, and random walk 2 is superior.

sequences that capture our graph structure better. The walk length has a dependency on the size of the available graphs in the dataset. For us, walk length 5 is the suitable length since, in both random walks, the embedding performance is better compared to walk length 3 and 5. In [25] with vanilla LSTM Autoencoder, on average 84% of proxy graphs were the second rank. However, this new model improves this percentage 92%.

4.9.2 Clustering

As mentioned in the previous section, floorplans with similar properties are close in the embedding space. This similarity is in terms of floorplans structure, design semantics, and human behavioral features. There are many parametric methods for clustering like KMeans [101] that we need to give the number of clusters as the input parameter. Since we do not want to limit ourselves to a specified number of clusters, we used Density-Based Spatial Clustering of Applications with Noise (DBSCAN). It is a non-parametric clustering method based on density. Each dense region (close-packed points) represents a cluster, and the

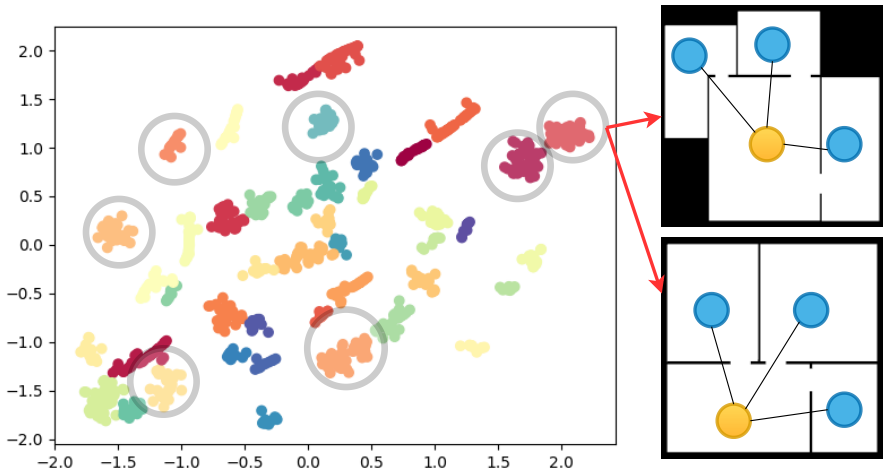


Figure 4.4: The clusters after running DBSCAN over 1000 random samples. Two samples from one of the clusters are shown. They have the same number of nodes, same node degrees, and similar room types (blue nodes are bedrooms, and yellow nodes are living rooms). Besides, the average square footage for the top floorplan is 23.49, and for the bottom floorplan is 25.38. This shows the embedding space indeed captures the design semantics of floorplans.

points in the low-density areas are marked as outliers. It has two parameters, the minimum number of points in each cluster and the maximum distance between two samples in each cluster [102].

We sampled 1000 floorplans, and we run DBSCAN over their embedding vectors generated by our three models to cluster them. We calculated the standard deviation for the number of nodes, average node degrees, node types, edge types, and flow rate for each cluster. The average of mentioned metrics on all clusters is provided in Table 4.3. The number of nodes and average node degrees is almost similar in all three models. The reason is all the corresponding features to these metrics are available in three models. The node type is a common feature in all three models. However, in models 2 and 3, the performance is better. It is because of integrating edge types in these two models. Edge type is not a considered feature in model 1, and we have the worse performance in model 1. Flow rate is only available in model 3, which we have the best performance. However, model 2's performance for the flow rate is satisfactory and shows that integrating edge direction helps find more

	Number of nodes	Average of node degrees	Node types	Edge types	Flow rate
Model #1	0.164	0.092	1.11	2.12	1.71
Model #2	0.168	0.091	1.06	1.34	0.68
Model #3	0.161	0.093	1.07	1.41	0.34

Table 4.3: Average of standard deviation for number of nodes, node degrees, node types, edge type and flow rate in clusters out of 1000 samples in our three models.

similar floorplans with similar flow rate. Figure 4.4 shows the resulting clusters over 1000 samples in model 3. For visualization, the vectors' dimension is reduced to two by TSNE [103], and two sample floorplans from one of the clusters show the graph properties are encoded accurately with our model.

4.10 Qualitative Evaluation

This section presents qualitative results from the experiments.

4.10.1 Nearest Neighbours (NNs)

As described we trained three models with different set of features. Each model makes an embedding space. We selected three random floorplans and found their top 5 nearest neighbours in the corresponding embedding space of each model. The Figure 4.7 shows the query floorplans and their top 5 nearest neighbours. For each sample, first row shows the NNs in the first model's embedding space, second line shows the NNs in the second model's embedding space and third row shows the NNs in the third model's embedding space. As shown in the image, with the first model, the floorplans have the same structure in term of the rooms numbers, room (node) degrees and room types. But the room arrangements are not similar. In the second model, since the edge features are added, the high rank NNs follow the same arrangement and with moving toward low rank NNs the arrangement similarity is dcreased. However, they have similar structure yet. In the third model, the human behavior features are added as well and now floorplans with similar be-

havioral features get close to query floorplans. The last row for each sample shows the visualization of crowd flow rate. The numbers inside floorplans in first and second row shows the square footage of each room. In third rows the numbers depict flow rates. Please note, as mentioned in section 4.4, the north is at the top and other directions are recognized correspondingly.

4.11 Floorplan Generation

Given that variational autoencoders are generative, we study the skill of our model for generating new floorplans. Generating new floorplans can be done with sampling from the posterior distribution of sequences or with homotopies [93, 94].

4.11.1 Sampling from Posterior Distribution

VAE learns the data distribution instead of deterministic mapping. Therefore, we can sample from these posterior distributions for generating new data. As mentioned in section 4.8, for each graph, we run random walk 11 times to generate 11 sets of node and edge sequences. To generate a new floorplan, we select a floorplan and a set from its 11 sets of node and edge sequences. By decoding the samples from posterior distribution of these sequences, we get new sequences. There could be different strategies to produce a new floorplan with these newly generated sequences. We select the node with the highest degree that is repeated in all sequences as the main node. Therefore, the arrangement of other rooms can be fixed relatively. These new sequences give us information about room types and square footage. Our method does not encode the geometry shape of rooms, therefore, we can assume the room shapes are similar to the originally selected floorplan or any arbitrary shapes that satisfy the generated square footages, Figure 4.5. Generating new floorplans with this approach is limited and gives us similar floorplans in terms of the number of rooms and room types, the only changes in new floorplans are the square footage and geometry shape of rooms. The square footage generated in the new sequences do not

have the same value for each room, however, with considering the original sequences as references, the average of generated square footage can be used for a new floorplan.

4.11.2 Homotopies

VAE makes a continues embedding space, and it allows interpolation in this space. We used the concept of homotopy that means the set of points on the line between two embedding vectors. Instead of a set of random points, we limit our experiment to the point in the middle of the line. We can select two random sequences from two random floorplans, and after encoding them, we can calculate the difference of their embedding vectors. Adding half of their difference to the base vector and decoding it gives us a new sequence. Figure 4.6 shows an example. By replacing the newly generated sequence with the old random sequence from the original floorplan, we can generate new floorplans. It can happen between any other random sequences, and in this way, we can generate more derivative samples. Different strategies for interpolation and generating new floorplans could be used here. With homotopy, we do not have the mentioned limitations in sampling from the posterior distribution. Floorplans with varying room types can be generated as a result of interpolation, and the only limitation is the geometry shape of rooms, which is not encoded. We can assume the geometry shapes are the same as reference floorplans or any arbitrary shapes that satisfy the generated square footages to address this limitation.

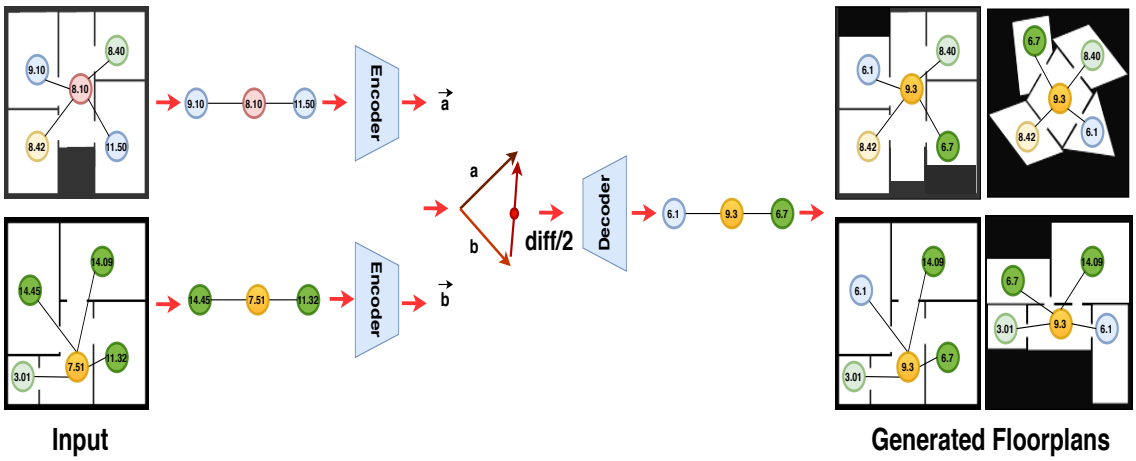


Figure 4.6: Floorplan generation with interpolating in embedding space. We select two random sequences from two random floorplans, and after encoding them, we calculate the difference of their embedding vectors. Adding half of their difference to the base vector and decoding it gives us a new sequence. These new sequences can be used for generating new floorplans. See section 4.11 for details.

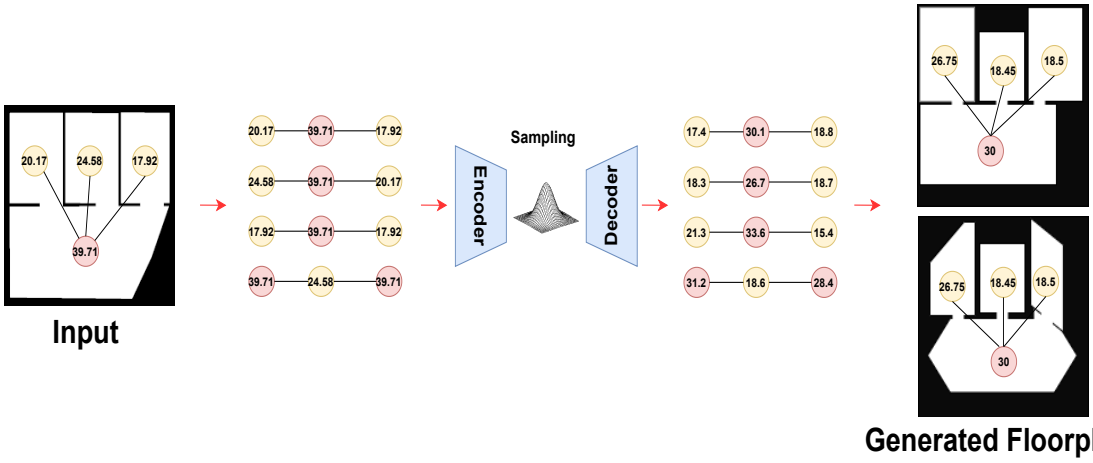


Figure 4.5: Floorplan generation with sampling from posterior distributions. To generate a new floorplan, we select a floorplan from our dataset and a set from this floorplan's 11 sets of node and edge sequences. By decoding the samples from posterior distribution of these sequences, we obtain new sequences and then map them into new floorplans.

Demographic Information

Gender	**Sex**	**Age**	**Country of Residence**
Female: 4 (40%)	Female: 4 (40%)	18 - 24 years old: 4 (40%)	China: 1 (10%)
Male: 6 (60%)	Male: 6 (60%)	25 - 34 years old: 4 (40%)	United States: 4 (40%)
		35 - 44 years old: 2 (20%)	Canada: 5 (50%)

Domain Knowledge

	Poor	**Below Average**	**Average**	**Above Average**	**Excellent**	**Avg. scale**
Ability to interpret architectural or interior designs?	0 (0%)	1 (10%)	1 (10%)	7 (70%)	1 (10%)	3.80
Prior experience with architecture or interior designs?	2 (20%)	0 (0%)	1 (10%)	6 (60%)	1 (10%)	3.40
Prior experience in urban planning and design?	3 (30%)	0 (0%)	2 (20%)	5 (50%)	0 (0%)	2.90
Prior understanding of computational tools for architectural design-space exploration?	2 (20%)	0 (0%)	3 (30%)	5 (50%)	0 (0%)	3.10
Prior understanding of pedestrian movement flow or crowd flow?	0 (0%)	2 (20%)	4 (40%)	3 (30%)	1 (10%)	3.30

Table 4.4: Demographic information and domain knowledge ratings of expert participants (self-reported).

4.12 User Study

In this section, we present a user study to evaluate the quality and efficiency of our models of graph embeddings. Three different embedding models are tested: (1) trained with design semantic features alone, (2) design semantic and edge features, and (3) design semantic, edge and behavioral features. Given a floorplan (input), we query five similar floorplans (nearest neighbours) from each embedding model.

4.12.1 Hypothesis

Our hypothesis is twofold: (a) the user perceived sequence of floorplans as top-five nearest neighbours matches with the sequence captured by our model as nearest neighbours, and (b) users perform better in their perceived sequence of top-five nearest neighbours for models (2) and (3) than model (1) which is only trained with design semantic features.

4.12.2 Apparatus

Floorplans are presented as 2D blueprints (e.g. a top-down skeletal view of an environment layout). The users (e.g. study participants) viewed these blueprints as high-resolution images on their own computer screens via an online survey. For model (1), each room in a floorplan is annotated with room dimension (e.g., square footage area) and color-coded with respect to its room type. The annotation for model (2) is similar to model (1) with an addition of edges between rooms and their respective directions (e.g., North, East, West, South). For model (3), we showed color-coded trajectories of virtual occupants from the rooms they spawned-in to the exit, along with square footage area of each.

4.12.3 Participants

Ten (10) domain experts from the architecture community (4 female and 6 male) voluntarily participated in the user study. Table 4.4 shows the demographic information and domain

knowledge of the experts. On average, all the participants had above-average experience and expertise in interpreting architecture designs and were Knowledgeable of computational tools for design space exploration (self-reported). In addition, every participant was asked for consent before the start of the study.

4.12.4 Procedure and Task

The user study is conducted as an online survey and delivered in four parts. In part (a), users are asked to provide their demographic information and report the domain knowledge and expertise in architecture and urban design. In part (b), users are presented with five different input floorplans. For each input floorplan, a sequence of 5 nearest neighbours are presented in a randomized order, which are retrieved using model (3), and presented to the users "without" any visual annotations. Users are asked to interactively reorder the given sequence of floorplans (e.g., via drag and drop), based on their perceived "similarity" of these floorplans with respect to input floorplan. The ordering sequence is arranged such that, more a floorplan is towards left in the order, the nearest it gets to the input floorplan. In parts (c), (d) and (e), the nearest neighbours are retrieved using models (1), (2) and (3) respectively, for the same five input floorplans which are used in part (b). In parts (c), (d), and (e), the floorplans are presented to the users "with" visual annotations for their respective features. We estimated that the user study will take up to 15 minutes at maximum to complete.

4.12.5 Independent and Dependent Variables

Input floorplans and the retrieved nearest neighbours from the models are the primary independent variables. The rearranged sequences of floorplans by the users are the only dependent variables.

4.12.6 Results

Figure 4.8 shows the user-ordered sequences of the nearest neighbours for the three models from user study. The colored bars for each neighbour of an input floorplan represent the number of users who correctly perceived the order of the neighbour in the given sequence. Overall, about 28.68% of the neighbours are accurately ordered in their sequences based on their perceived similarity with respect to input flooplans for model (1), 59.28% for model (2) and about 77.6% for model (3), collectively by all the users. These results highlight that users least performed when they had to perceive the similarity between floorplans by considering the design semantic features alone, whereas they performed comparatively better when presented with the neighbours annotated with edge and/or behavioural attributes. The users performed the best for the model (3) when presented with the floorplans visually annotated with design semantics (e.g., room types), edge (e.g., movement direction of the agents), and behavioural (e.g., movement flow of the agents) features. The findings from the user study suggest that both of our hypothesis stand valid.

We also wanted to analyze the users' performance in perceiving the ordering sequence of the neighbours when floorplans are not visually annotated with their respective features. To test this, we used the input floorplans and their neighbours from model (3) and presented them as model (0) in the user study. These floorplans were presented to the users without any visual annotations. This was so we could analyze how important is the visual annotation of the features, and its significance to assist users in perceiving the neighbours in their correct order. Interestingly, about 45.68% of the neighbours were accurately ordered in their sequences based on their perceived similarity with respect to input flooplans for model (0). This result revealed that the annotations for design semantic features, alone, are not a good representative to convey the spatial feature information of the floorplans. As well, that the users better perceive the floorplans retrieved from the embedding space that is trained not only with the design semantic feature alone but also with the additional edge or/and dynamic behavioural features.

4.13 Conclusion

This proposed model in this chapter aims to represent floorplans with numerical vectors such that design semantic and human behavioral features are encoded. Specifically, the framework consists of two components. In the first component, an automated tool is designed for converting floorplan images to attributed graphs. The attributes are designed semantic and human behavioral features generated by simulation. In the second component, we proposed a novel LSTM Variational Autoencoder for both embedding and generating floorplans. The qualitative, quantitative, and expert evaluation shows our embedding framework produces meaningful and accurate vector representations for floorplans, and its abilities for generating new floorplans are showcased. In addition, we make our dataset public to facilitate the research in this domain. This dataset includes both the extracted design semantics features and simulation-generated human behavioral features. This contribution holds promise to pave the way for novel developments in automated floorplan clustering, exploration, comparison, and generation. By encoding latent features in the floorplan embedding, designers can store multi-dimensional information of a building design to quickly identify floorplan alterations that share similar or different features. While in this work, we encode features derived from dynamic crowd simulations of building occupancy, the proposed approach can virtually scale to encode any kind of static or dynamic performance metric.

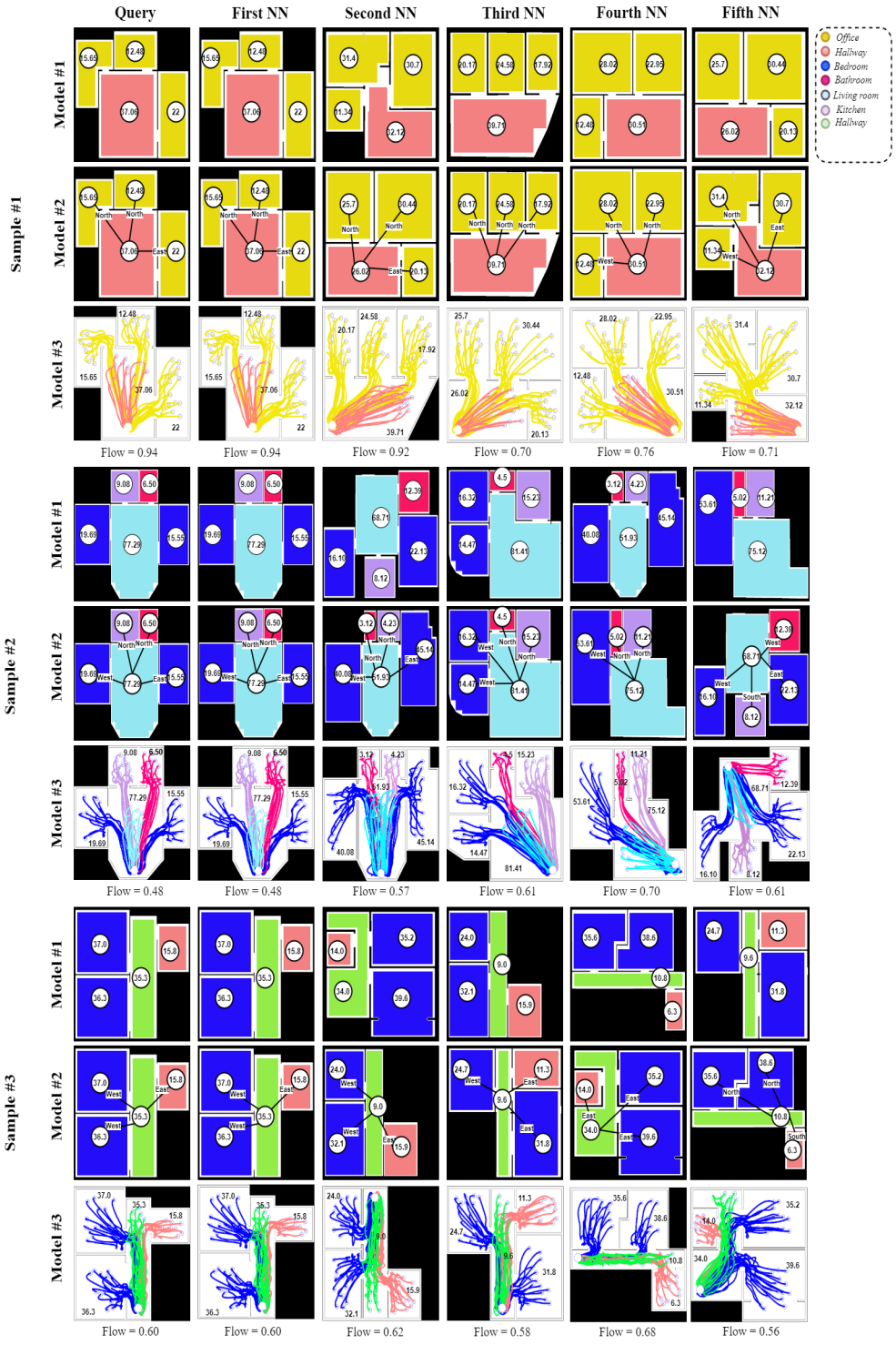


Figure 4.7: The top 5 NNs for three floorplans found with three models. The first row shows the NNs with first model, second row shows the NNs with second model and third row shows the NNs with third model. The color of the room denotes the room type, and the numerical value inside each node denotes the square footage of the room. In the third row of each sample, the crowd flow is visualized and the numbers depicts flow rates. See subsection 4.10.1 for details.

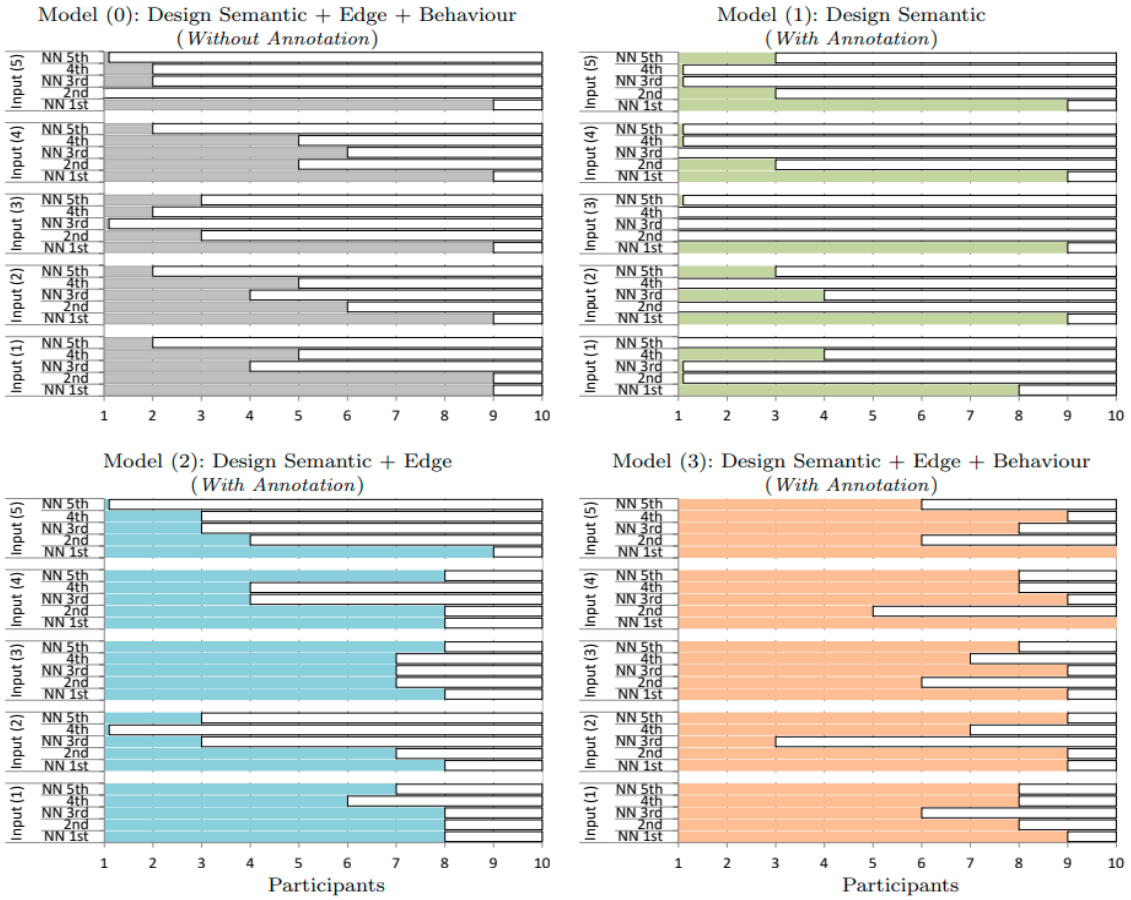


Figure 4.8: The accuracy of user-ordered sequences of the nearest neighbours. Colored bars for each neighbor of an input floorplan represent the number of users who correctly perceived the neighbor's order in the given sequence. Gray bars are for the nearest neighbours which are queried using model (3) but were presented to users without any annotations, Green bars are for nearest neighbours which are queried using model (1) and were presented with annotations, blue bars using model (2) – with annotations, and orange bars using model (3) – with annotations.

CHAPTER 5
THE ROLE OF LATENT REPRESENTATIONS FOR DESIGN SPACE EXPLORATION

5.1 Introduction

From design inspiration to democratization of housing layouts for affordable living, matching the desired features of a space to a floorplan is crucial to these desired interfaces. This goal requires some challenges to be overcome; the ability to not only retrieve an optimal floorplan, but generate new ones as well. While it is trivial to lookup a value of a floorplan, such as the number of bathrooms, it is much more difficult to lookup a floorplan with a certain number of bathrooms *and* similar room sizes (and other attributes). As the complexities of design and architecture are well known, the relationship between a single floorplan and the numerous measures one would associate with it makes this an extremely difficult search task (i.e., the configuration space of floorplans is extremely high-dimensional, continuous, and non-convex).

The use of visual search tools (e.g., shape-grammars[1]) to match similarity in form is an active research topic in Architecture and Design, although in such a large search space and intrinsically tracking multiple quantitative metrics, the utility of geometric similarity quickly diminishes. Recently, research has shown the use of machine learning techniques for associating environment configurations with metrics [43, 2]. However, these works require an explicit search through the environment configuration space. In comparison, our work explores the potential of using a lower dimensional representation that eliminates the need for such explicit searches. While there are numerous ways to represent a floorplan (i.e., different data descriptions), a common method for reducing the complexity of a description is to use *latent representations*, which have been shown to provide promising

results through machine learning and associations of metrics and the environment [25]. In our work, we demonstrate the full potential of using these latent representations as a basis for optimizing the floorplans with respect to the target objective functions and returning the similar floorplans efficiently.

The foundation of our work lays in the relationship between an input floorplan and its location in an abstracted space, relative to other floorplans. This abstraction, which takes a floorplan and represents it as a set sequence of graph nodes, maintains the metrics associated with it. For clarity, we refer to the abstraction as the *latent representation*, the mapping from the floorplan to the latent representation as an *embedding*, and the space that this representation exists within as the *embedded space* or *latent space*.

Our work is centered on the following contributions. First, we build a synthetic dataset of 5000 plausible floorplans of 12 unique styles, generating metrics associated with each. We then develop a new embedding model for this synthetic dataset using a Gated Recurrent Unit Variational Autoencoder (GRU-VAE) to represent floorplans in a latent space. Next, we demonstrate two local search approaches over the latent space: (1) retrieval-based approach with nearest-neighbour queries, and (2) generative approach exploring the optimal vector sequences . We then run a series of experiments for the two approaches that search the floorplan dataset for 5 different metrics, as well as interesting combinations of them.

Our results show the potential for both rapid design iteration tools and consumer searches of real-estate. Given an initial design layout, the designers can minimize or maximize quantitative metrics within the similarity space of the initial design while finding alternative solutions that improve one or more metrics. Likewise, this technique enables online floorplan designers and consumers to explore residencies, not unlike what we do with major e-commerce.

5.2 Overview

The foundation of our work is in the representation, abstraction, and manipulation of floorplans in the latent space. Beginning with a floorplan image, we extract regions based on rooms and corridors that are represented as a graph, with nodes that are assigned attributes relating to space-syntax. These graphs are then embedded in a lower-dimensional space, which we leverage this low dimensional embedding space to optimize floorplans by using nearest neighbours and using generative power of model, Figure 5.1.

Graph Representation. Our framework represents the floorplan as an attributed graph with nodes representing the non-overlapping navigable rectangular spaces in the environment. Two nodes are connected by an edge if they are connected by a navigable path that does not go through any other nodes. Five common attributes considered in space syntax relating to the geometric and visible features of the environment are embedded in each node, as well as a room label that is defined by the geometric characteristics (e.g., m^2 area of room). For the five continuous attributes, geometric features such as Area and Perimeter are used, as well as visibility-based features such as isovists. These measures are discussed in detail in subsection 5.3.2 and subsection 5.3.3.

Embeddings. While floorplans can be simply represented by graphs with associated metrics, performing an optimization or analysis is computationally expensive [85, 86, 87]. A common approach to reducing the complexity, or dimensionality of a dataset, is to map the original data to a lower dimension through an *embedding*. This technique preserves the properties of the original data, with an additional benefit of locating (in the embedded space) similar data near each other, Figure 5.1.

Local Search. Our use of embeddings to represent the graphs means that floorplans with similar properties are closer to each other, and by using a VAE we are able to perform vector operations in the embedded space. Between these two features of our model, we are able to demonstrate two strategies: (1) we can perform fast retrieval-based nearest-neighbour

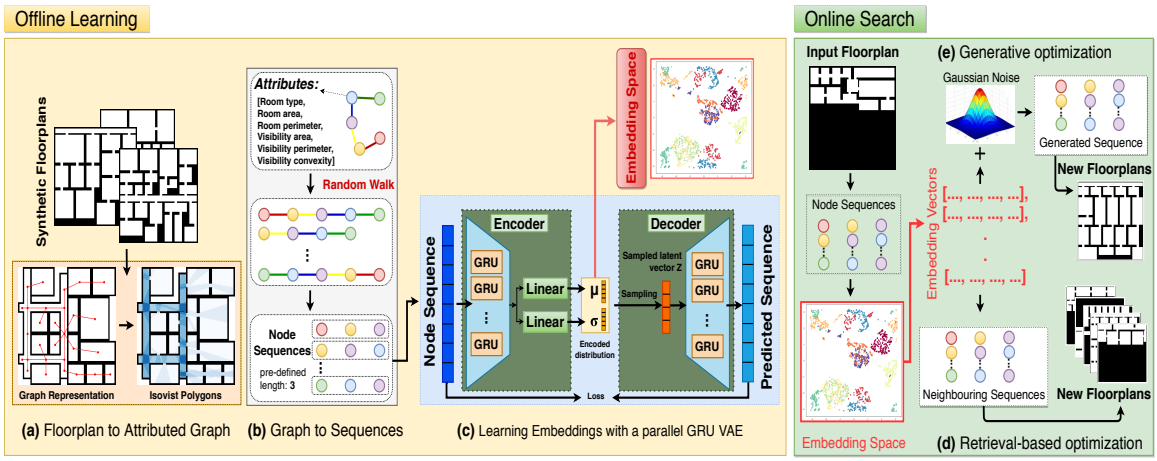


Figure 5.1: Our framework consists of two phases: (1) offline phase – First, the input floorplans are converted into attributed graphs. We then generate sequences of the attributed graphs using random walks. These sequences are then fed into the embedding model to train and learn the embedding space. (2) online phase – Given an input floorplan, we convert it into an attributed graph and generate its sequences using a random walk. These sequences are then fed into our embedding space. We then call *Retrieval* or *Generative* procedures for retrieving similar nearest neighbours or generating a new graph for the input floorplan in the direction of the targeted objective, respectively.

queries in the embedding space, and (2) we can generate new optimized sequences of the vectors representing a floorplan by adding Gaussian noise.

5.3 Synthetic Dataset of Attributed Floorplans

In order to create graphs that represent realistic or real-world floorplans, a well-labeled dataset is required. Although we can extract the graph structure from some existing datasets, such as HouseExpo [76], there are many environments that do not have accurate bounds for rooms or accurate room labels, which the original task of robot navigation does not require but our task demands. Therefore, we rely on procedural generation to create plausible environments that have accurate room labels and node attributes. Furthermore, the ability to synthesize floorplans is vital to searching the embedding space, because as the number of synthesized floorplans increases, the probability that the embedding space has a discontinuity decreases. The following subsections describe the process of generating 5,000 floorplans of 12 unique styles, and converting the images into attributed graphs. The

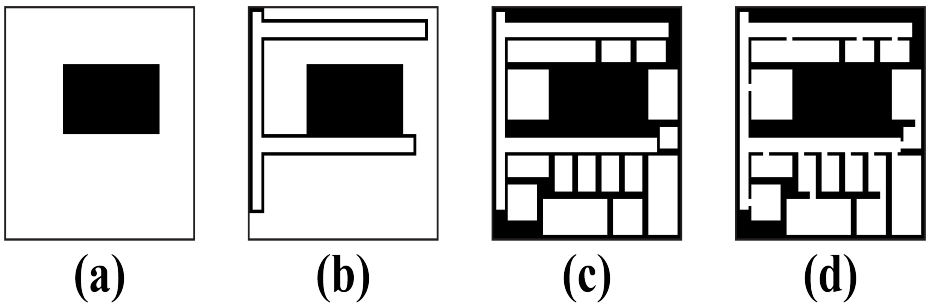


Figure 5.2: The images above show the four phases of floorplan generation: (a) defining the exterior, (b) creating corridors, (c) creating rooms, and (d) connecting rooms and corridors.

dataset of floorplans and graphs will be released after publication.

5.3.1 Procedural Generation of Environments

The procedural generation of floorplans in our synthetic dataset relies on the four-phase methodology from [10], illustrated in Figure 5.2. In general, the arrangement of rooms and corridors in real-world environments can be described by a shape typology and an organization typology [104], where the former categorizes the exterior morphology of an environment and the latter categorizes the corridors within.

Phase 1. The shape typology is determines what is inside and outside of the environment by masking an empty floorplan (Figure 5.2.a). The mask can resemble either a square (Figure 5.3.e), a square with a rectangular hole (Figure 5.3.f), or an elongated rectangle (Figure 5.3.g).

Phase 2. The chosen organization typology populates the environment with corridors (Figure 5.2.b), which can either be single points, segments centered between walls, segments adjacent to walls, or free-form segments (Figure 5.3.a–Figure 5.3.d).

Phase 3. The remaining space is populated with rectangular rooms that have either random dimensions with probability p, or the same dimensions as the previous room with probability $1 - p$ (Figure 5.2.c).

Phase 4. Small openings are then made between rooms such that a room is connected via the opening to its neighbor which is nearest to a corridor (Figure 5.2.d). This ensures

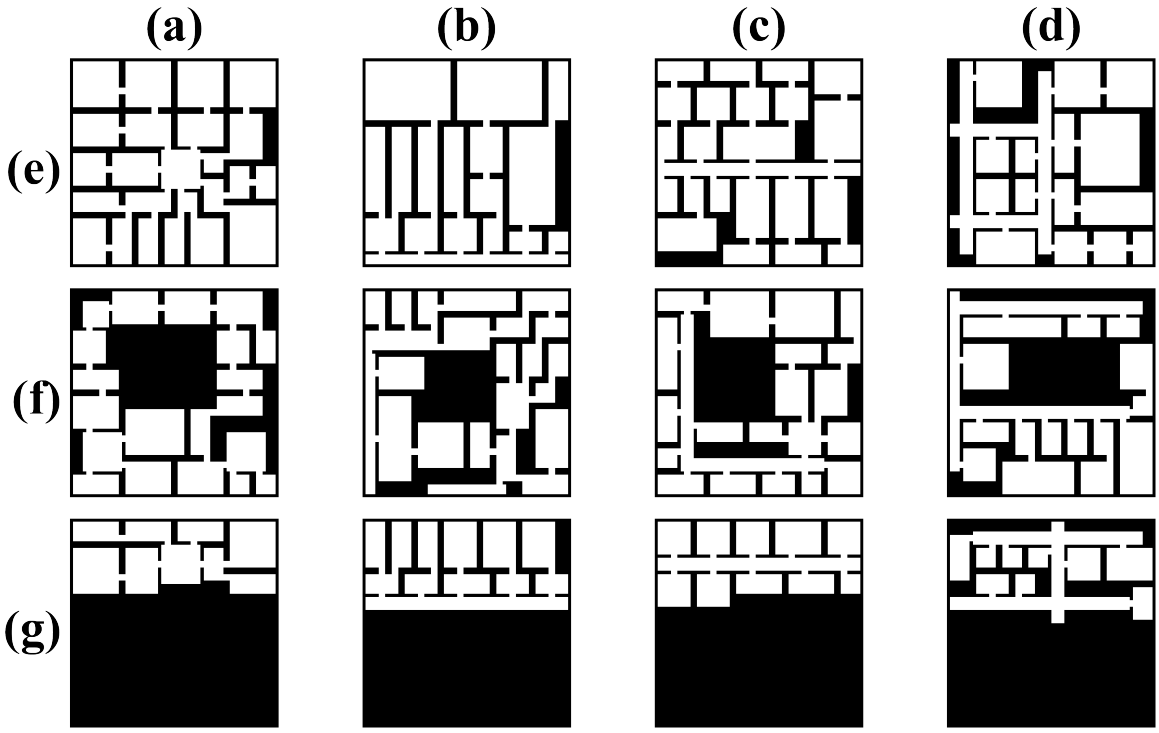


Figure 5.3: The above table shows 12 types of floorplans that differ based on their exterior shape (e–g) and interior arrangement of corridors (a–b).

corridors are areas in the environment that have high centrality, meaning that they facilitate navigation between many rooms [10].

5.3.2 Image to Graph Conversion of Environments

Our dataset is composed of binary images that are parsed into rooms and corridors. Each room is a non-overlapping rectangle, which is associated with a single node (Figure 5.4(bottom)). The corridors, however, can have non-rectangular shapes, and are therefore decomposed into multiple rectangles, which group areas that have similar isovists [10]. Next, edges are defined between two nodes when they are connected by a path that does not go through any other nodes. Edges often go through the small openings created during Phase 4, but these are not considered as nodes since they play a much less significant role than rooms and corridors.

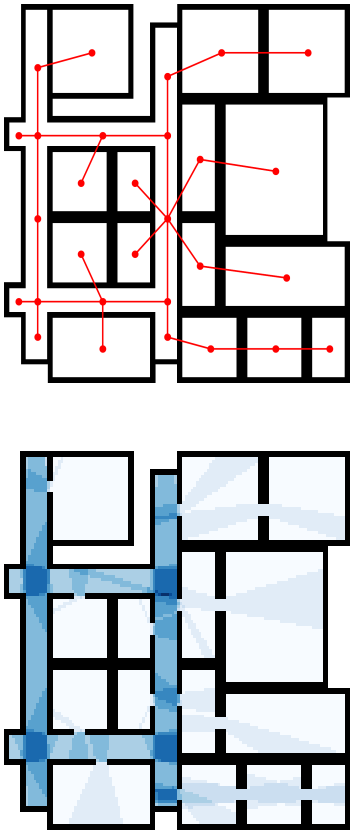


Figure 5.4: The top image shows a graph (red) overlaid on its corresponding floorplan. The bottom image shows the nodes' isovists overlaid on the same floorplan. Pixels that are overlapped by many isovists are dark blue, and pixels that are covered by few isovsts are light blue.

5.3.3 Computation of Node Attributes

Each node in the graph is given 5 continuous attributes that are either physical or visibility-based (Table 5.1), with a 6^{th} attribute that describes a room's function (e.g., as a kitchen or bedroom). The physical measures include the area and perimeter of the node's rectangle. The visibility measures are computed on an isovist–a polygon representing the visible region from a given location–with a viewpoint at the center of the node's rectangle. Figure 5.4(top) shows the isovist of each node in an environment. Finally, we convert each room label into a value between $[0 - 1]$ to provide the vector representation a continuous (rather than discrete) value.

5.4 Latent Representation of Attributed Floorplans

Our framework uses embeddings, a low-dimensional representation of the attributed graphs of floorplans. The benefits of using embeddings are twofold: (1) users can preserve the original data into a low-dimensional space (e.g., layout structure, design semantics, and other behavioral attributes), and (2) data with similar characteristics can be located near each other within the embedding space, making it easier to perform machine learning tasks such as clustering of floorplans with similar attributes.

One challenge in mapping various graphs to embedding space, in particular for floorplans is the lack of consistency between nodes, edges, and connectivity between. Although some techniques such as padded adjacency matrices may normalize the input dimensions, the variety in rooms creates excessive gaps in the data and make training a model difficult. We therefore utilize a random walk technique which is ideal for capturing the global graph structure– similar to constructing sentences by composing multiple elements. More details on random walks can be found in [25].

A set of sequences is extracted from each graph by applying random walks. These sequences are then used as input into the embedding model which is a Gated Recurrent Unit

Node Attribute	Description
Room Area	Area of the rectangular region corresponding to the node.
Room Perimeter	Perimeter of the node's rectangle.
Visibility Area	Area of the isovist from the center of the node's rectangle.
Visibility Perimeter	Perimeter of the node's isovist.
Visibility Convexity	Quotient between the isovist's area and the area of its convex hull.
Room Label	Identifying value associated with a room label in the range [0,1].

Table 5.1: The table above defines each type of node attribute in the graphs.

Variational Autoencoder (GRU-VAE). The GRU-VAE has fewer parameters, and hence, leads to faster training. This is particularly useful when dealing with short length input sequences. After running through the embedding process, embedding vectors are generated for each input sequence. As described in [25] the best aggregator for embedding vectors is average. The resulting vectors from the embedding are then averaged to represent a floorplan graph in the embedding space.

5.5 Local Search Over the Latent Space

This section describes two local search approaches in the trained latent space: (1) a retrieval-based approach and (2) a generative approach. Algorithm 1 shows the steps involved in searching through the latent space for both approaches. First, the input floorplan is converted into an attributed graph (Line 8). The graph is then converted into a set of sequences using random walks (Line 9), and finally the corresponding set of embedding vectors is computed from the model (Line 10). The process of optimizing features takes place in the latent space.

Our search methods consider two types of stopping criteria: (1) when the maximum iteration count is reached (set to 200 for the experiments), and (2) when the change in the objective value is not significant after a specific number of iterations.

5.5.1 Retrieval-based Approach

Sequences with similar attributes are proximate in the embedding space. By leveraging this property, we find the k-nearest neighbours of the each of the sequences of initial floorplan, where $k = 10$. Next, the best combination of the nearest neighbours that satisfy the defined objective is identified and replaces the original sequences. This procedure is repeated iteratively to find the best alternate sequences until the stopping criteria is met.

Implementation-wise, we first precompute the nearest neighbours from the embedding space before running the optimization to reduce the time complexity (Line 3 of Algorithm 1). We then call the $Retrieval$ procedure from Algorithm 1, which computes the objective value for each nearest neighbour and identifies a set of neighbour sequences (representing a floorplan) which improves the objective. This improved set of sequences is then used in the next iteration.

5.5.2 Generative Approach

We use a Gated Recurrent Unit Variational Autoencoder (GRU-VAE) to learn the distribution of sequences, which can sample the distributions to create new sequences since the GRU-VAE is a generative model. In particular, we perturb the embedding vectors of a floorplan's sequences by adding Gaussian Noise (GN). These new embedding vectors are then fed into the GRU-VAE decoder for generating new sequences. We then calculate the objective function with the new set of sequences. If the new sequences improve the objective value, we replace the original sequences and repeat this process until the stopping criteria is met. Algorithm 1 shows the pseudocode for this process within the $Generative$ procedure.

5.5.3 Final Floorplan Candidates for Local Search

Each search approach ultimately produces an optimized set of sequences. However, this set is not interpretable without a corresponding floorplan. Since each floorplan samples

Algorithm 1 The search algorithm for retrieval and generative cases.

Input: Initial graph, $Floorplan$
Input: Length of random walk, n
Input: Precomputed k-nearest neighbours for each sequence, nn_s
Input: Embedding vectors of all floorplans in dataset, E_f
Input: Mean of gaussian noise, μ
Input: Standard deviation of gaussian noise, σ

$G \leftarrow MakeGraph(Floorplan)$
$S \leftarrow RandomWalk(G, n)$
$E \leftarrow Embedding(S)$

procedure RETRIEVAL(S, E)
    while $Terminate() \neq 0$ **do**
        $S' \leftarrow SamplingNN(nn_s, S)$
        if $f(objective : S, S') = True$ **then**
            $S \leftarrow S'$
            $E' \leftarrow Embedding(S')$
    $E_{Flooplan'} \leftarrow Averaging(E')$
    return $Top5NN(E_f, E_{Floorplan'})$

procedure GENERATIVE(S, E, μ, σ)
    while $Terminate() \neq 0$ **do**
        $Noise \leftarrow RandomNoise(\mu, \sigma)$
        $E_N \leftarrow E + Noise$
        $S' \leftarrow Decoder(E_N)$
        if $f(objective : S, S') = True$ **then**
            $S \leftarrow S'$
            $E \leftarrow Embedding(S')$
    $E_{Flooplan'} \leftarrow Averaging(E)$
    return $Top5NN(E_f, E_{Floorplan'})$

its sequences randomly, a floorplan represents a fuzzy area in the latent space instead of a single point. Therefore, we average the optimized set in the latent space as well as all floorplans' sequences in the latent space E_f [25], which produces the centroids of the fuzzy areas. We then find the five nearest floorplans to the optimized set of sequences according to the distances of their average embedding vectors ($Top5NN$).

5.5.4 Multi-objective Search

Since the latent space has been learned using multiple features, it enables multi-objective exploration. Given an initial floorplan and a set of features to each be minimized/maximized, the local search will find a latent representation which improves on the initial floorplan's embedding w.r.t. all desired features. For example, the optimizer can simultaneously maximize the average room area and minimize the average room perimeter to get relatively large and square-shaped rooms and fewer, less-elongated corridors. This process is repeated until a local extremum is reached, which we assume is the most improved floorplan to the original that still remains similar according to the unconsidered features. Through a multi-objective search, we can be more specific about the desired geometries and arrangements of rooms in a floorplan.

5.6 Experiments

5.6.1 Training Details

The method described in section 5.3 was used to produce a total of 5,000 floorplans. As mentioned, each floorplan is converted to a set of sequences. We utilized random walk with walk length 10 for this dataset experimentally and both P and Q are set to 0.5. Parameter Q is the probability of discovering the undiscovered parts of the graph and parameter P is the probability for returning to the previous node. As mentioned we have 5000 floorplans and in total we have 363983 sequences. The sequences are divided to train and test set with rate 80 to 20 percent. The sequences are normalized to make all features in same scale and

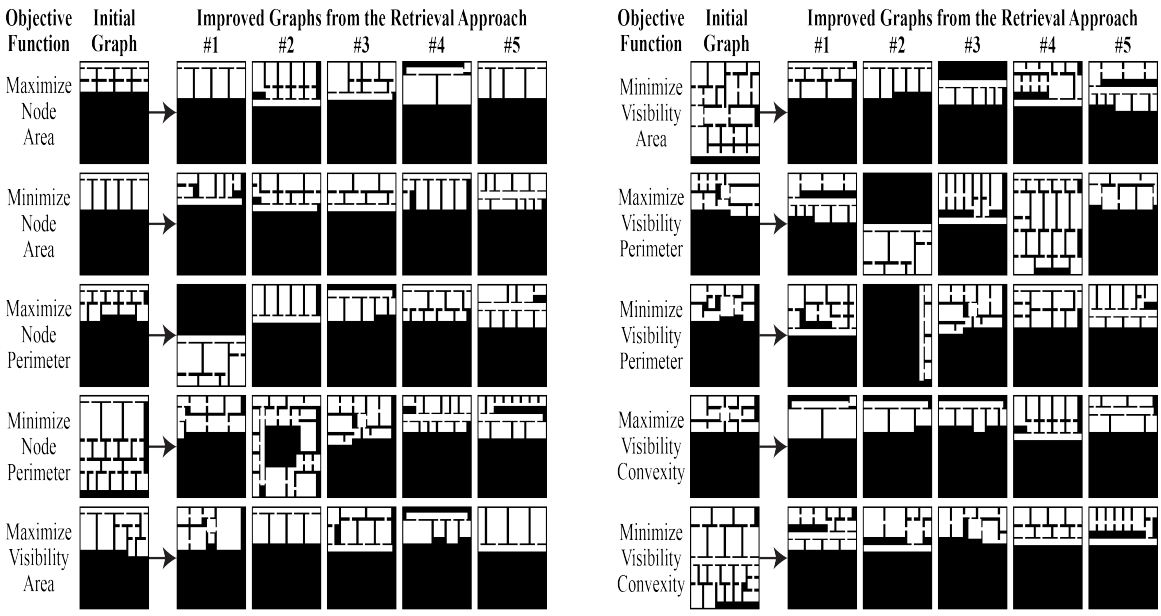


Figure 5.5: Each row in the above table shows an initial graph and 5 graphs that improve upon an objective function using a retrieval-based approach.

fed for training. The learning rate was set empirically as 0.001. The model is trained on a machine with 32 GB RAM, 12 * 3.50 GHz cores CPU and Quadro K620 GPU with 2 GB Memory. In average the model takes about 30 minutes for training with 50 epochs.

5.6.2 Optimizing Individual Features

For each continuous type of node attribute (i.e., room area, perimeter and visibility area, visibility perimeter, visibility convexity) in the floorplan dataset's graphs, the retrieval-based and generative approaches were used to find graphs that increase or decrease the average value of the attribute compared to an initial graph. This yields a total of 20 searches. An initial graph serves as the starting point of each search, which has been varied to showcase the improvement in graphs found by the two search methods. Each method is used to produce 5 improved graphs, which are ordered from closest (#1) to furthest (#5) from the initial graph in the embedding space. In other words, the best candidate has the largest difference in the desired features and the smallest difference in all other features compared to the initial graph.

Graph	Room Area	Room Perimeter	Visibility Area	Visibility Perimeter	Visibility Convexity
Initial	*772.0*	139.3	746.0	154.6	0.84
Improved #1	*486.9*	109.1	509.5	144.2	0.69
Improved #2	*438.4*	95.6	525.9	158.7	0.59
Improved #3	*582.0*	117.0	601.5	162.9	0.65
Improved #4	*586.3*	119.4	597.0	147.5	0.77
Improved #5	*395.6*	90.9	403.0	120.5	0.70

Table 5.2: This table compares graphs produced by the retrieval approach when minimizing room area.

Graph	Room Area	Room Perimeter	Visibility Area	Visibility Perimeter	Visibility Convexity
Initial	*381.3*	100.0	417.9	138.5	0.65
Improved #1	*955.2*	154.4	905.8	163.8	0.91
Improved #2	*491.2*	101.6	635.6	169.3	0.68
Improved #3	*584.0*	111.5	624.4	156.1	0.69
Improved #4	*1324.0*	164.0	1300.5	190.2	0.83
Improved #5	*955.2*	154.4	966.4	180.3	0.80

Table 5.3: This table compares graphs produced by the retrieval approach when maximizing room area.

For single-feature optimization, the average improvement in a desired feature relative to the initial value is 32.96% for the retrieval-based approach and 26.17% for the generative approach. Therefore, we showcase examples of the retrieval-based approach in Figure 5.5. The feature values from all 20 searches can be found in Table 5.4, which shows that both search methods are able to successfully retrieve graphs that have improved upon the initial graph in terms of the desired feature. The Improved Graph columns of images in Figure 5.5 directly correspond to the columns of feature values in Table 5.4. Table 5.2 and Table 5.3 each compare the average feature values of the initial and improved graphs for different single-feature objective functions, which shows that the best improved graph aims to balance a large change in the desired feature with small changes in all other features.

5.6.3 Optimizing Compound Features

By considering compound-feature as an objective function, we can be more specific about the desired geometries and arrangements of rooms. An environment that simultaneously maximizes the average room area and minimizes the average room perimeter is expected to have rooms that are relatively large and square-shaped and have fewer, less-elongated corridors. Using the generative approach (subsection 5.5.2), the optimization of floorplans according to the objective confirms this expectation (Figure 5.6). The inverse objective of minimizing room area and maximizing room perimeter is expected to produce small, elongated rooms that are connected to multiple elongated corridors. Figure 5.6's second row shows that the generative approach is able to identify these types of floorplans. The average room area and perimeter values of the graphs are reported in Table 5.4. In the following compound-feature optimizations, we showcase the visual results for the generative approach, because it improved the desired features by 18% on average, while the retrieval approach improved features by 14% on average.

While the prior compound-feature objectives influence the size and shape of rooms and corridors, optimizing compound visibility-based features can influence the connections between rooms. For example, an objective function maximizing visibility area and minimizing visibility perimeter is optimized with rooms that are large in size and are not visible to many areas outside (relative to the area inside). The inverse of this objective function is optimized with rooms that are small in size and are visible to more areas outside. This leads to fewer, less-elongated corridors that have a high connectivity with rooms, which are arranged in such a way that rooms are visible to each other across other rooms and corridors. The generative approach has evidenced the formation of floorplans that have these characteristics for both visibility-based objectives (Figure 5.6, Table 5.4).

Search Method	Minimized Feature	Maximized Feature	Initial Graph	Improved Graph #1	Improved Graph #2	Improved Graph #3	Improved Graph #4	Improved Graph #5
Retrieval Approach	Gray	Room Area	381.33	955.2	491.2	584	1324	955.2
	Room Area	Gray	772	486	438	582	586.28	395.63
	Gray	Room Perimeter	87.63	129	137.33	119.33	96.4	104.8
	Room Perimeter	Gray	104.22	76.11	72	81.33	84.66	80.92
	Gray	Visibility Area	621.88	894.18	905.8	683.62	737.8	1136.8
	Visibility Area	Gray	741.11	567.37	673.91	487.25	624.20	623.22
	Gray	Visibility Perimeter	165.44	175.71	199.15	172.68	188.65	178.11
	Visibility Perimeter	Gray	185.75	160.86	128.57	180.79	157.13	128.31
	Gray	Visibility Convexity	0.65	0.83	0.74	0.82	0.67	0.72
	Visibility Convexity	Gray	0.62	0.61	0.59	0.60	0.61	0.55
Generative Approach	Gray	Room Area	381.33	476	1512	580.23	476.44	555.55
	Room Area	Gray	772	629.33	645.71	452.8	586.28	468
	Gray	Room Perimeter	87.63	106.85	92.72	102.18	127	101.6
	Room Perimeter	Gray	104.22	76.36	90.33	87.33	101.6	92.33
	Gray	Visibility Area	621.88	848.5	905.8	905.85	672.5	955.57
	Visibility Area	Gray	741.11	653.21	574.71	406.54	420.3	475.4
	Gray	Visibility Perimeter	185.75	179.91	187.04	189.83	216.49	196.84
	Visibility Perimeter	Gray	165.44	165.61	134.50	159.92	144.67	117.08
	Gray	Visibility Convexity	0.65	0.89	0.76	0.72	0.72	0.69
	Visibility Convexity	Gray	0.62	0.61	0.55	0.56	0.60	0.61
Retrieval Approach	Room Area	Room Perimeter	524.36 : 96.36	450.0 : 103.5	491.2 : 101.6	494.22 : 104.0	503.27 : 101.09	476.0 : 106.0
	Room Perimeter	Room Area	182.76 : 554.44	109.6 : 596.8	111.55 : 557.77	119.0 : 602.0	126.28 : 709.71	117.0 : 582.0
	Visibility Area	Visibility Perimeter	587.2 : 146.12	523.6 : 150.81	562.26 : 178.70	505.05 : 162.75	568.68 : 148.90	560.25 : 159.18
	Visibility Perimeter	Visibility Area	178.28 : 537.56	151.45 : 579.43	140.63 : 550.18	158.36 : 760.92	169.28 : 635.65	148.39 : 604.43
Generative Approach	Room Area	Room Perimeter	524.36 : 96.36	480.0 : 100.4	453.81 : 96.77	433.6 : 98.4	452.44 : 103.55	365.33 : 97.33
	Room Perimeter	Room Area	182.76 : 554.44	105.14 : 690.28	113.5 : 556.0	114.0 : 776.0	130.0 : 768.0	125.14 : 628.57
	Visibility Area	Visibility Perimeter	587.2 : 146.12	569.04 : 194.78	564.81 : 154.29	568.54 : 171.08	497.66 : 175.85	510.17 : 162.81
	Visibility Perimeter	Visibility Area	178.28 : 537.26	169.60 : 617.72	165.59 : 824.0	137.06 : 547.5	155.96 : 651.4	163.79 : 905.8

Table 5.4: The above table shows the results of single and compound-feature optimizations. For single features, each right-hand column shows the value of either the initial graph or one of the top 5 improved graphs found by the respective search method. For compound features, each right-hand column shows two values: one for each optimized feature, where the left-hand value corresponds to the minimized feature and the right-hand value to the maximized feature.

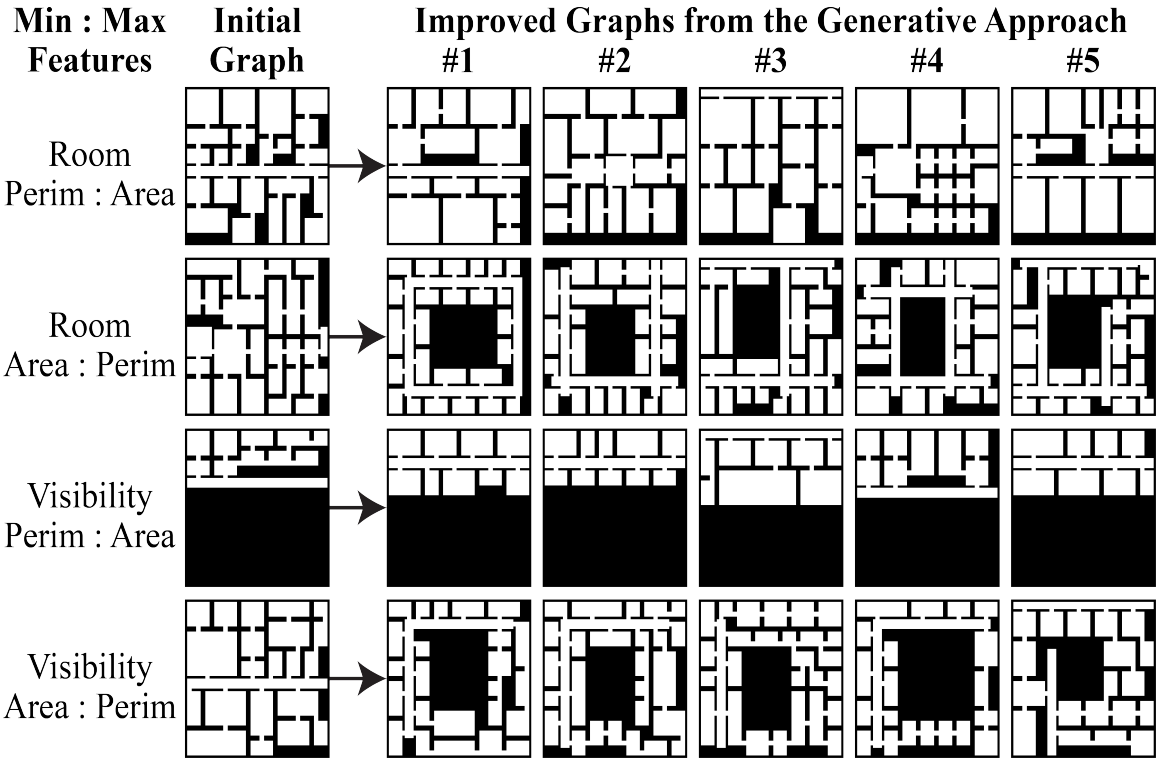


Figure 5.6: Each row in the above table shows an initial graph and 5 graphs that improve upon the minimization of one feature and maximization of another.

CHAPTER 6
CONCLUDING REMARKS AND FUTURE DIRECTIONS

6.1 Conclusion

This thesis aims to represent floorplans with numerical vectors such that design semantic and human dynamic features are encoded. Specifically, we represent floorplans with attributed graphs and augment them with static and dynamic features. These features are extracted either by image processing and geometric techniques (statically) or by running simulations. We propose embedding methods to convert floorplans to low dimensional vectors to address the time complexity of operations on graphs and the limitation of machine learning methods on graphs. To address the variation in graphs' dimensionality, we utilize an intermediate sequential representation (generated by random walks) which allows us to encode the graphical structure in a fixed-dimensional representation. First, we propose an LSTM Autoencoder for encoding floorplans with semantic and dynamic features. Then, the model is extended to integrate edge features with generative capabilities. We propose a parallel LSTM VAE for handling features both on nodes and edges. These vector representations are used to cluster and query floorplans with similar characteristics and attributes. Some of the advantages of presenting floorplans as vectors are: (a) compact representation, (b) efficient way to compare designs and fast retrievals, (c) scoring designs and providing feedback/recommendations, and (d) categorizing design according to target features. The generative capabilities of the model are studied for design space exploration. We use homotopies and sampling from sequences distribution for generating floorplans. Finally, the floorplan optimization is proposed based on the inseparability of embedding space. We explore the potential of latent representations for identifying floorplans that meet specific criteria, which can serve as the basis for interactive design space exploration and other

search-based design applications. Through experiments, we have demonstrated the efficacy of such an approach. In theory, a key advantage of a latent space is that searching for improved floorplans can potentially achieve near-constant computational cost with spatial hashing [105], while existing methods have a linear cost for iterating through all floorplans.

These contributions hold promise to pave the way for novel developments in automated floorplan clustering, exploration, comparison, and generation. By encoding latent features in the floorplan embedding, designers can store multi-dimensional information of a building design to quickly identify floorplan alterations that share similar or different features. While in this work, we encode features derived from dynamic crowd simulations of building occupancy, the proposed approach can virtually scale to encode any kind of static or dynamic performance metric.

6.2 Limitations and Future Work

This study considers a subset of geometric, semantic, and human dynamic features for attributed floorplans. There are a wide variety of other features (e.g., shapes) that can also be encoded into latent representations and is the subject of future exploration. In the current study, rooms are assumed to be of a fixed rectangular shape, thus limiting the model's generative capabilities to axis-aligned rooms and environments. Similarly, additional dynamic and behavioral features can be extracted from simulations, or possibly observations of real people can also be integrated into our models.

Currently, our methods perform a search on a set of available floorplans, which relies on procedural generation to ensure that the set is comprehensive according to 2 coarse features: exterior shape and interior corridor arrangement. The latent space of floorplans can be used with a more exhaustive set of features to create clusters that consider more than just the prior two features. This would increase the fidelity of the representative sample. Furthermore, the embedding of floorplans into the latent space can be reversed to generate a new floorplan image from an arbitrary latent vector, which would enable integration with

automated CAD workflows. This would remove the local search's dependence on existing floorplans, enriching the exploration of the latent space where floorplan samples were sparse during training.

Appendices

APPENDIX A
GEOMETRIC REACHABILITY ANALYSIS FOR GRASP PLANNING IN CLUTTERED SCENES FOR VARYING END-EFFECTORS

A.1 Introduction

As the capability of robots has increased significantly, they can perform more complex tasks. The research in robotics and automation includes but not limited to humanoid walking [106], medical robotics [107, 108, 109, 110, 111, 112], mobile robots [113, 112] and multi-agent systems [114, 115]. Emerging automation applications will necessitate robotic equipment to manipulate a large variety of objects in unseen, cluttered scenes. One key requirement towards meeting this far-reaching goal is to develop efficient, yet complete algorithms for grasp planning in complex, cluttered scenes (Figure A.1). This work seeks to accelerate existing grasp planners by geometrically analyzing scene geometry to automatically identify a complete set of object subsurfaces, which permit an end-effector to approach and grasp the object.

Many existing approaches precompute a *grasping database* [116, 117] for each target object type and end-effector, to generate a discrete set of feasible grasps, which are scored using a variety of robustness measures [118]. During an online planning process, these grasps are sampled to check for reachability and collisions. Nevertheless, some critical challenges arise in this context:

1. The granularity of the database is resolution-dependent, which presents a trade-off between completeness and efficiency.

2. Time and space complexity increases combinatorially with the complexity and clutter of objects.

3. A database is needed for each end-effector/object pair. In cluttered scenarios, as in

Figure A.1: A cluttered tabletop scene.

Figure A.1, the goal object could be heavily occluded by other objects, resulting in a narrow range of viable grasps. Existing discrete grasp databases do not account for collision constraints, and may not even provide a single collision-free grasp.

In this context, we propose a method for precomputing grasp contact semantics, or *graspable surfaces*, in arbitrarily complex, cluttered scenes, which can be effectively used to accelerate online grasp planning. The proposed method abstracts the capabilities of an end-effector through a graphical representation of interval constraints, referred to as a *motion constraint graph* (MCG).

Given the MCG and an arbitrary scene, an interval constraint satisfaction problem is defined to compute a complete set of subsurface combinations that give rise to all possible grasp configurations for a particular end-effector. The following constraints can be satisfied as part of the solver:

- End-effector constraints, which include the distance ranges between contact points

of the end-effector, as well as the surface area of each contact point.

- Collision constraints between objects in the scene and the end-effector.

The proposed approach is applicable to any kind of scene geometry, including arbitrarily cluttered scenes and curved surfaces. It also allows reasoning for a variety of end-effectors with multiple degrees of freedom. Experiments in simulation indicate that the proposed geometric reachability analysis increases the success rate of grasp and motion planning processes, while also reducing online planning time, at the cost of a small amount of precomputation. The proposed approach can be applied in a variety of contexts, including:

- recommending permissible grasps for grasp planning.
- pruning existing grasp databases such as GraspIt [116].
- identifying optimal end-effector designs for a given scene.

A.2 Background

This section reviews a small portion of the significant literature on grasp generation and planning [119, 117].

Model-based Approaches for Grasping Known Objects: Computing grasps that optimize metrics, such as stability [120], task coverage [121], or contact point uncertainty [122], can help to safely grasp and manipulate objects. This can be a time-consuming procedure, which often means precomputation is desirable. If both the objects and type of end-effector are known, then the grasps can be sampled and optimized offline [123]. Such "grasp databases" can also be used for objects that share some similarity in shape [116]. Agglomerative Clustering [124] groups points that share similar normals and position values into a hierarchical data structure so as to adapt precomputed, reachable grasps online using optimization methods. Other methods focus on generating collision-free grasps given

a set of sampled preshapes [125]. The method proposed in this work assumes the object model but can work directly over a geometric representation of a scene to generate "graspable surfaces", which satisfy constraints for:

- the end-effector kinematics
- the contact point geometry and
- the proximity of other objects. The resulting surfaces can then be used for several different applications, with grasp generation being one of them.

Grasping Unknown Objects: Without an object model, grasp generation can focus on "graspable" features given sensor data [126]. A method uses the swept volume of the end-effector and a bounding box decomposition of the sensed object, to generate grasps [127]. Another approach incrementally learns a heuristic reachability function from experimental data [128]. Learning has been used for detecting graspable points over a point cloud [129] and has been integrated with geometric reasoning to detect grasps in cluttered scenes for a parallel-jaw gripper [130]. A recent method needs a single kinesthetic demonstration for learning to recommend grasps for unknown objects, which then have to be tested for collisions and reachability [131]. In automation where the objects are known, the benefit of the proposed approach is that:

- it does not require extensive training
- it can easily generalize to different and new types of end-effectors. Integration of the current work with machine learning methods can assist to operate directly over sensing data.

End-Effector Abstraction: Task Space Regions [132] specify constraints on the end-effector and the target object to guide grasp generation and motion planning. For known end-effectors, it is possible to define the contact points and motion capability of the end-effector, so as to enforce constraints on grasp generation [133, 134]. The current work

similarly aims to abstract end-effector kinematics and takes advantage of fast computational geometry primitives, which are effective in analyzing geometric surfaces to compute affordances for locomotion behaviors [135]. The idea is to quickly operate over the continuous surfaces of the objects directly instead of a sampled set of grasps given descriptions of the end-effector and a complex, cluttered scene.

A.3 Problem

Consider a setup similar to Figure A.1, where a manipulator **R** is equipped with a set of end-effectors **F**, and is tasked with grasping a set of known movable objects **M** amidst a set of static obstacles **O** (e.g. the table). Then, for a movable object $m \in \mathbf{M}$ and an end-effector $f \in \mathbf{F}$, the objective is to compute the continuous graspable regions, or *graspable surfaces*, of m s.t. f is capable of grasping the object m.

Each end-effector $f \in \mathbf{F}$ can achieve a set of grasping modes **G**. A grasping mode defines a spectrum of distinctly different configurations an end-effector can achieve for grasping objects (e.g., open-palm or pinch grasps for a fingered end-effector). Each mode can be abstracted into a series of kinematic, geometric, and spatial constraints, which accordingly forms the basis of a motion constraint graph (MCG) [135].

By defining the primary contact points **C** of the grasping mode (e.g., fingertips for pinch grasps), as well as the pairwise spatial constraints **E** that contact points are subject to, each mode $g \in \mathbf{G}$ can be represented as a motion graph $MCG_f(g) = \{\mathbf{C}, \mathbf{E}\}$. Each contact point $c \in \mathbf{C}$ defines a relative position on the end-effector, a normal pointing outward, and a bounding circle (although other geometric representations could be used). Other contact point constraints, such as friction, could also be incorporated as a constraint. Each spatial constraint $\mathbf{e}(i, j) \in \mathbf{E}$ defines the valid distance interval between the contact points. If a contact point can rotate, the MCG also specifies the rotation axis and a possible range of rotations for that contact point. An example MCG is shown in Figure A.2.

Given the mesh of an object m, a surface s consists of the set of triangles whose normals

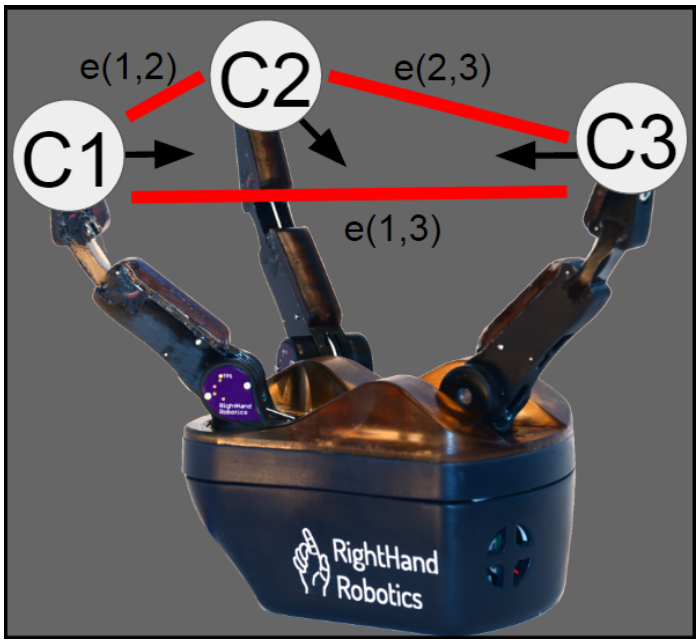


Figure A.2: A motion constraint graph (MCG) defining a grasp mode for the ReFlex. Arrows are contact normal vectors. Edges are distance constraints between pairs of contact points (min & max).

are all within an ϵ threshold, while also sharing at least one common vertex to another triangle in the surface. A *graspable surface* is the subset of a surface (subsurface) in which the constraints of an MCG have been satisfied. An object $m \in \mathbf{M}$ is said to be **graspable** if it has at least one *graspable surface*, and is **reachable** if there is a collision-free trajectory for the arm to the object.

An assumption of this work is that both the type object, and its mesh, are known. Although the underlying techniques of this work could be extended to work with sensing data and unknown objects, this is not the focus of the current work. Accordingly, given an observation of the scene, the framework loads the meshes of all detected objects and obstacles at estimated poses assuming access to a vision-based solution for pose estimation.

A.4 Geometrics Reachability Analysis

The proposed Geometric Reachability Analysis (GRA) consists of three major components which are **Adaptive Surface Clustering**, **Surface Pruning** and **Constraint Satisfaction**

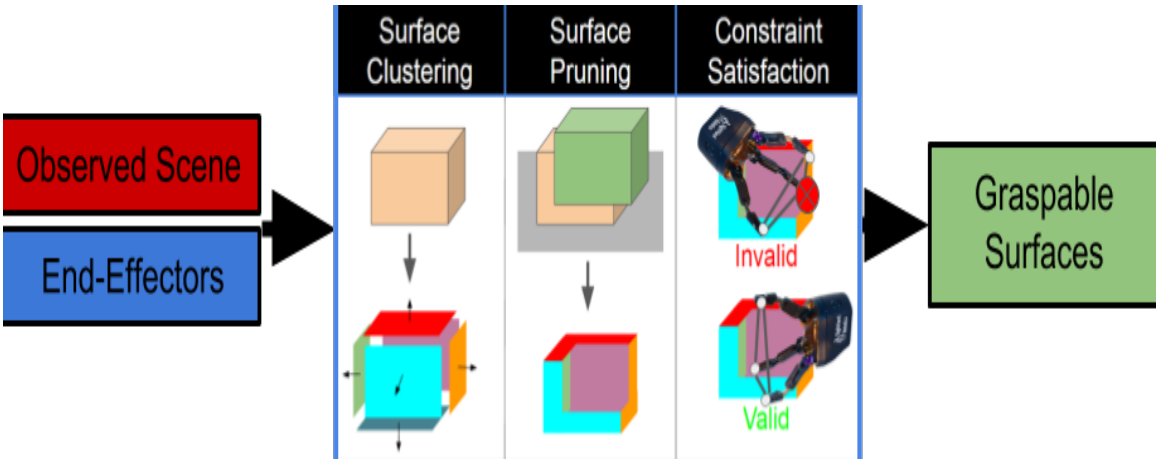


Figure A.3: The meshes of all objects are clustered into surfaces as part of **Adaptive Surface Clustering**. Surfaces proximal to surfaces of target object pruned in **Surface Pruning**. Given an end-effector, **Constraint Satisfaction** computes the *graspable surfaces* of the target object for each given motion constraint graph. An *invalid* grasp which causes the end-effector to collide, or violate kinematic constraints, cannot be produced from these *graspable surfaces*.

as shown in Figure A.3 Details of these components are explained in the following subsections.

A.4.1 Adaptive Surface Clustering

The purpose of Surface Clustering is to produce a set of *surfaces* given the meshes of all movable objects **M** and static obstacles **O** in the scene. This allows later portions of the framework, primarily Constraint Satisfaction, to operate over a compact representation of the entire scene. A *surface* consists of triangles which share normals within an ϵ threshold, with the additional constraint that a contact point from an MCG must have sufficient area to be placed on the *surface*. In this version of the work, contact points are modeled as circles, however this was an implementation detail and not a requirement of the framework. Any arbitrary polygon could be used to represent the contact point.

Instead of tuning ϵ based on which motion constraint graphs $\mathbf{MCG}$ and objects are present, we instead employ Adaptive Surface Clustering, which incrementally constructs the surface set **S**. First, over all available $\mathbf{MCG}$, we find the smallest contact point geom-

etry - this is the minimum area constraint that is used to validate a surface. Then, ϵ is set to zero and surface clustering is applied. Any surface which meets the contact point constraint (i.e. minimum area constraint) is added to the set. If there are remaining triangles which have not yet been added to an existing surface, then ϵ is incresed by δ and the process continues. The terminating condition is that all triangles belong to at least one surface, or some maximum threshold ϵ_{max} has been reached.

A.4.2 Surface Pruning

This framework also considers the geometric constraints imposed by the scene - in particular, due to clutter from other objects. The objective of surface pruning is to remove any portion of a surface in which it is impossible to place a contact point in a collision-free manner. This condition happens when two surfaces are too close. This can occur, for example, if one object is resting on top of another object, since the resting surfaces of the objects will be unreachable. The resulting "pruned" surface is referred to as a *subsurface*, which are subsets of their original surfaces.

The algorithm for surface pruning is shown in Alg. 1. The inputs are the surface set $\mathbf{S}$ and the contact point width υ of the target end-effector. If the distance between each pair of surfaces s, s' is less or equal to contact point width υ, then the two surfaces are proximal and must be pruned. To accomplish this pruning, the proximal surface s' is swept in the reverse normal direction $\hat{s}$ of $\mathbf{s}$, with the magnitude υ, producing a swept surface s''. Then, the intersection between the swept surface s'' and the target surface s is computed, and intersecting portions from s and s'' are subsequently removed. This in turn creates a *subsurface* for s and s', as they have had their original surfaces altered by this procedure. If there are no intersections between the surfaces, then the resulting subsurface after intersection is the same as s. If they have complete overlap, then the final subsurface is empty.

The subsurfaces must satisfy contact point geometry and area constraints, otherwise

Algorithm 1: Surface Pruning

```
1 Function Surface_Pruning(S, v)
2     foreach s ∈ S do
3         foreach s' ∈ (S − s) do
4             if (DISTANCE(s, s') < v) then
5                 s'' ← SWEEP(s', −ŝ, v)
6                 s, s' ← INTERSECT(s'', s)
```

they will be ignored for further consideration. After this step, for any point on the subsurfaces, the end-effector will not collide with any neighboring objects when attempting to place contact points over these subsurfaces.

A.4.3 Constraint Satisfaction

After all triangles in the scene have been clustered, with any proximal subsurfaces removed, the algorithm proceeds to apply constraint satisfaction over the remaining subsurfaces using the motion constraint graphs $\mathbf{MCG}$ defined for each end-effector. This results in a mapping from end-effector grasping modes to valid *graspable surfaces*. The algorithm is shown in Alg. 2, where the input is the surface set after surface pruning S', the defined $\mathbf{MCG}$ for the target end-effector, and the normal threshold δ.

The algorithm iterates over the defined contact points $\mathbf{C}$ of the current $\mathbf{MCG}$ and over each surface. Then, the current surface is set to be the corresponding surface $\mathbf{CS}$ of the current contact point, and the rotation r between the normal of $\mathbf{CS}$ and the normal of the current contact point is calculated. Consequently, we need to rotate all other fingers with r and find the corresponding surfaces for them if there is any(by `Match_Surface` function). Finding a corresponding surface for the other fingers follows a similar procedure, with the only difference being that we consider their permissible rotations only if it is specified. This allows us to account for all the degrees of freedom of the end-effector while searching for valid surfaces.

Once the corresponding surfaces for each contact point are found, and if all contact points have corresponding surfaces, then constraint satisfaction can be applied to find the *graspable surfaces*. The result of the first step is a vector of IDS (intermediate data struc-

Algorithm 2: Constraint Satisfaction

```
1  Function Constraint_Satisfaction(S', MCG, δ, υ)
2      foreach m ∈ MCG do
3          foreach c ∈ C(m) do
4              foreach s ∈ S' do
5                  CS ← CS ∪ s
6                  r ← COMPUTE_ROTATION(ĉ, ŝ)
7                  if (MATCH_SURFACE(S', MCG, δ, r)) then
8                      IDS ← IDS ∪ ids
9              foreach ids ∈ IDS do
10                 foreach ci ∈ C(ids) do
11                     foreach cj ∈ (C(ids) − ci) do
12                         sd ← p_ci − p_cj
13                         s' ← SWEEP(CS_cj, ŝd, υ)
14                         CS_ci ← INTERSECT(s', CS_ci)
```

tures) which stores the contact points with their corresponding surfaces. Then, all IDSs are iterated over in order to find valid surfaces. The `SWEEP` operation and then intersection is applied to find any intersection between each pair of corresponding surfaces **CS** with each pair of contact points. For sweeping we need a sweep direction $\hat{\mathbf{sd}}$, which is calculated using the normalized direction between position **p** of two contact points.

If there is any intersection, the resulting subsurface after intersection is a valid area for placing corresponding contact points. Finally, the result of the this step is a set of "graspable surfaces" which satisfy the geometric and kinematic constraints of the end-effector subject to the constraints of the scene, which can then be used by other processes (e.g. a grasp planner).

A.5 Use Case of Our Approach

This section describes some of the potential use cases of the *graspable surfaces* produced by this framework.

Grasp Generation: The procedure behaves in the following way: first, a random graspable surface $s_{\texttt{rand}} \in \mathbf{S}(m)$ is selected. Then, a contact point from the corresponding *MCG* is randomly selected and then placed randomly onto $s_{\texttt{rand}}$. This automatically constrains the placements of all other remaining contact points, and defines a grasping configuration g. The end-effector is placed at g and rotated around the normal of the fixed contact point

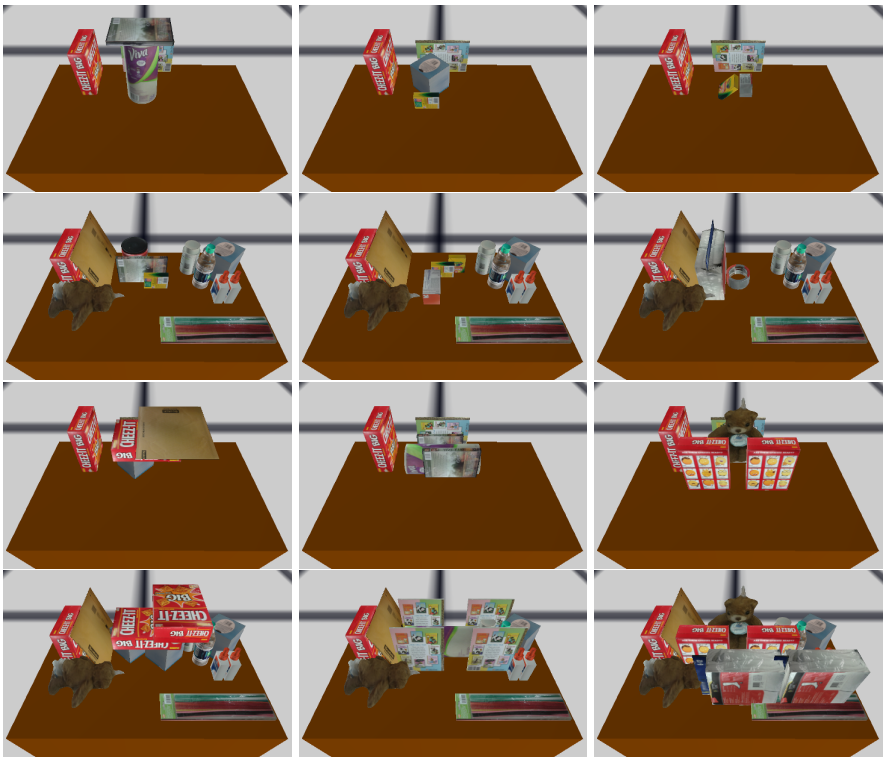


Figure A.4: The benchmarks used in all experiments. (Top) Parallel Gripper Benchmarks: DVD, Crayola, and Duct Tape. (Bottom) ReFlex Hand Benchmarks: Cheezit, Kleenex Paper Towels, and Bear. Left 3 Images (S)parse Clutter, Right 3 Images (D)ense Clutter.

in increments of Θ. Each rotation generates a new grasping configuration g_Θ. If g_Θ is collision-free for the end-effector alone (disregarding the rest of the robot), it is then added to the set of grasping configuration $\mathbf{G}_{\texttt{RAND}}$.

This sampling procedure repeats until N grasps have been sampled and added to $\mathbf{G}_{\texttt{RAND}}$ Each grasp $g \in \mathbf{G}_{\texttt{RAND}}$ is given as input to an IK solver[136] and the resulting state of the manipulator is collision checked. Collision-free grasps are added to the final set of grasps $\mathbf{G}$. After grasp generation, the planning framework receives a set of valid grasps, $\mathbf{G}$, along with their corresponding IK solutions. The reachability of each grasp $g \in \mathbf{G}$ is evaluated through a standard manipulation planning framework, such as the Grasp-RRT [137]. The first collision-free trajectory to any of the grasps in $\mathbf{G}$ is returned to the robot for execution.

Database Pruning: The advantage of using a grasp database [123, 116], over an online grasp generation method, is that additional time can be spent optimizing various criteria (e.g. stability[120], task coverage[121], or contact point uncertainty [122]), which can greatly increase the grasp success rate of the robot. If the robot must deal with clutter in its workspace, these databases must be sufficiently large enough so that the probability of finding a successful grasp is high. Rather than blindly searching through the database, the *graspable surfaces* could be leveraged to search a focused subsection of the database. This could be accomplished by using the aforementioned grasp generation method on the *graspable surfaces*, and performing a nearest-neighbor query on the database to extract similar grasps.

End-Effector Selection: For multiple different end-effectors, the *graspable surfaces* provide an evaluation criteria that a task planner could use to select which end-effector of the robot to grasp with. By computing *graspable surfaces* for each available end-effector, a planning framework could avoid spending extra time collision checking grasps, or even motion planning, for end-effectors that do not have any viable surfaces. Being able to skip planning on an end-effector becomes critical as the number and complexity of available end-effectors increases for the robot.

ID	G	Object	C	GOC	EC	CT(s)
S0	P	DVD	S	6/12	58/172	1.95798
D0	P	DVD	D	6/12	1049/2276	2.78367
S1	P	Crayola	S	6/12	24/48	0.565691
D1	P	Crayola	D	6/12	1021/2164	1.7192
S2	P	Tape	S	66/256	24/48	4.6155
D2	P	Tape	D	66/256	1028/2179	11.4858
S0	R	Cheezit	S	6/12	30/60	1.5875
D0	R	Cheezit	D	6/12	1027/2176	4.63645
S1	R	Towel	S	34/124	30/60	3.2792
D1	R	Towel	D	34/124	1033/2188	11.734
S2	R	Bear	S	77/604	36/72	18.939
D2	R	Bear	D	77/604	1065/2254	30.4325

Table A.1: Computation time for generating *graspable surfaces*, relative to the geometric complexity of the scene. **G**: Gripper: Parallel (P) or ReFlex (R). **C**: Clutter: Sparse (S) or Dense (D). **GOC**: Goal Object Complexity in # of surfaces and triangles triangles. **EC**: Total Scene Complexity in # of surfaces and triangles. **CT**: Computation Time.

A.6 Experiments

This section evaluates the applicability of the proposed method relative to the use cases described in section A.5 (grasp generation, database pruning, and end-effector selection) through evaluation in simulation.

Setup: Each experiment was conducted on a single computer with an Intel(R) Xeon(R) E5-1650 3.50GHz CPU using a motion and task planning simulation framework [138]. The robot used in the experiments was a Yaskawa SDA10F equipped with both a parallel gripper and a 3-finger ReFlex hand. Each benchmark had a designated "goal object" for the manipulator to grasp, and two variations of clutter in the scene: sparse (3+ objects) and dense (11+ objects). The benchmarks used in all experiments are shown in Figure A.4.

Motion Constraint Graphs: Constructing the **MCG** set consisted of evaluating the geometric and kinematic properties of both end-effectors. The relative position, approximate radius, and distance constraints of each contact point was found through documentation of the end-effectors, as well as physical experimentation. For example, the Parallel-Jaw MCG consists of 2 contact points, one each placed at the center of the parallel-gripper with normal

vectors pointing inwards. A single edge between the contact points was added, representing the closing and opening distances of the jaw. For the 3-finger ReFlex hand, a single MCG was used, which consisted of one vertex per finger, positioned at the fingertips, with rotation and distance constraints imposed between each pair of fingers (Figure A.2). This MCG was sufficient in representing the kinematics of the ReFlex.

Computational Overhead: Table A.1 reports the amount of time spent by the proposed method for computing *graspable surfaces* for each benchmark. The computation time increases as a function of the geometric complexity in the scene, expressed by the number of surfaces generated by `MCG` and the total number of triangles in the scene's meshes. It should be noted that the parameters used to generate the surfaces in all benchmarks were kept static and not changed - with some tuning, it would be possible to achieve faster times on more complex objects (such as the Bear).

Grasp Generation: This experiment evaluated the applicability of `MCG` for generating grasps using the approach described in section A.5 (benchmarks shown in Figure A.4). We compare against a baseline method, `RAND`, which used the same grasp generation method, but did not compute *graspable surfaces*, and instead sampled over the entire object mesh. The purpose of such a comparison was to establish a potential use case of `MCG`. Both methods used the same parameters (N of 500 and Θ set to 15).

The evaluation metrics were: *number of valid grasps*, *success rate*, *grasp generation time* and *motion planning time*. The number of valid grasps measures how many collision-free grasps were found by the grasp generator. The success rate corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds. Grasp generation time measures how long it took to compute collision-free IK solutions for each valid grasp. If grasp generation fails to produce a collision-free solution, it continues to sample and collision-check new grasps run up to 60 seconds, after which point the run is reported as a failure. Motion planning time measures how long it took the motion planner to compute a solution trajectory.

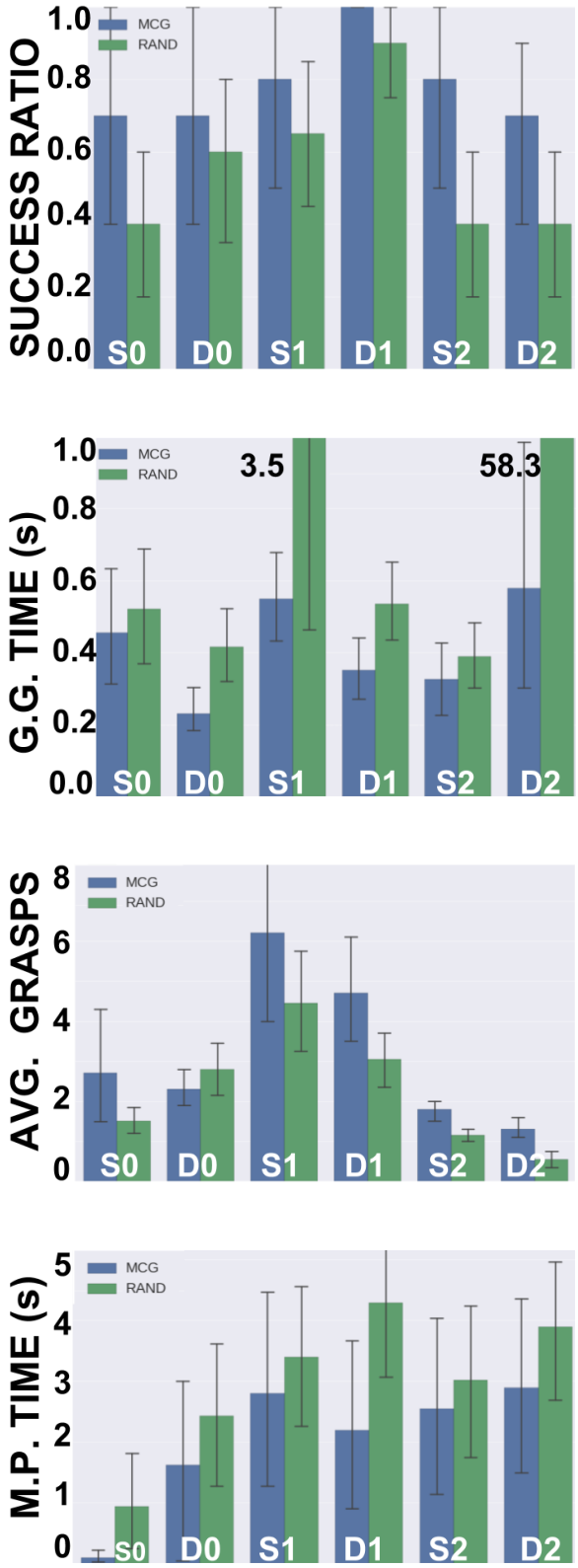


Figure A.5: Experimental Results for Grasp Generation. **Success Ratio** corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds. **Avg. Grasps** is the average number of collision-free grasps that were computed, the **Grasp Generation (G.G.) Time** is the average time the grasp planner spent to compute IK-solutions and approach motion plans, and the **Motion Planning (M.P.) Time** is the average time the motion planner took to compute a collision-free trajectory.

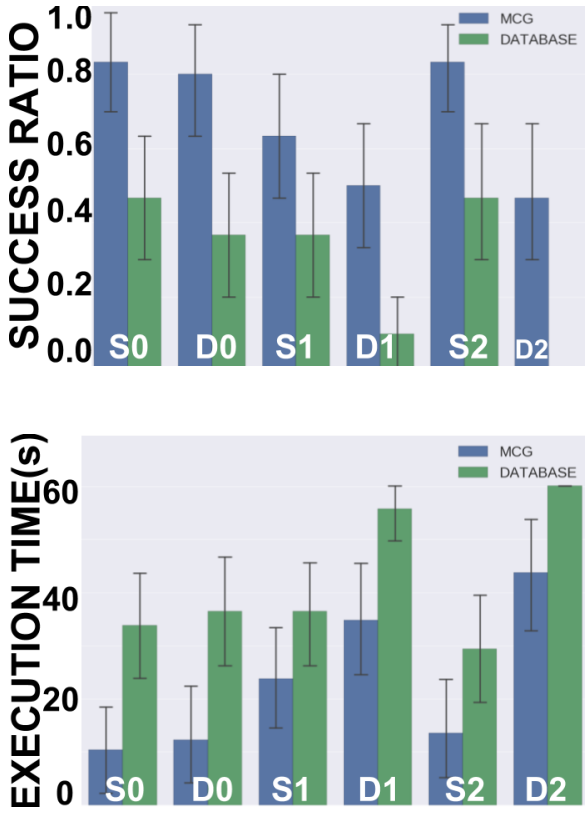


Figure A.6: Experimental Results For Database Pruning. **Success Ratio** corresponds to whether a collision-free trajectory for grasping the goal object was found within a time limit of 60 seconds.**Execution Time** represents the average total accumulated time by all components (grasp database validation, motion planning, and when applicable, **MCG** surface construction.)

Grasp Generation Analysis: The results are shown in Figure A.5 (the four leftmost graphs). In all of the benchmarks, MCG provided a larger number of valid grasps, spent less time during grasp generation, and improved the overall success rate of the manipulation planning framework. Although RAND did not spend any time computing *graspable surfaces*, MCG was able to produce a viable grasp for the planner faster. For some of the easier benchmarks, the differences between the methods were not as significant. This was to be expected, as the benefit of using MCG arises when the goal object is severely occluded by other objects in the scene.

Database Pruning: This experiment examined how effective MCG can be for decreasing the amount of time it takes for finding a valid grasp in a precomputed grasp database. As before, the environments used are shown in Figure A.4. As described in section A.5, the *graspable surfaces* can be used to reduce the number of grasps evaluated from the database. The comparison method, DATABASE, did not compute *graspable surfaces*, and instead evaluated grasps from an existing database incrementally until a collision-free grasp was found or the amount of allotted time ran out.

The evaluation metrics were: *success rate* and *execution time*. Success rate corresponds to the same metric as the previous experiment. Execution time represents the total time taken by the method, but only for the instances where the method was successful in solving the problem. Of important note here is that since grasp databases were being used, no time was spent by either method in grasp generation. To account for the randomness inherit in the motion planning framework, each benchmark was executed 30 times and the resulting averages and standard deviation are reported.

Database Pruning Analysis: The results shown in Figure A.6 (the two rightmost graphs) indicate that MCG was able to significantly reduce the execution time in the benchmarks, as well as improve upon the overall success rate of the motion planner. In terms of where the time was spent, the bottleneck in the experiments was during grasp validation, which evaluated IK-solutions for the grasps, as well as grasp approach plans for the

manipulator.

End-Effector Selection: By examining the time efficiency in each benchmark as shown in Table A.1, there is an indication that `MCG` could be used as a selection criteria for end-effectors. The computation time spent by `MCG` scaled with the complexity of the scenes; in all cases, it was much faster to query `MCG` than to wait for a failure criterion (i.e. evaluating grasps for 60 seconds). The method could therefore be used by a task planning framework to first evaluate whether or not an end-effector can grasp the target object, without resorting to more expensive processes.

A.7 Conclusion

We presented a method for annotating an unstructured scene with graspable surfaces given constraints for end-effectors. It utilizes geometric reasoning and constraint satisfaction to quickly extract a continuous representation of viable grasp surfaces. The approach can be used to generate candidate grasps, prune existing grasp databases or alternatively select end-effectors suitable for a scene. In contrast to methods for grasp generation, this work sets the foundations for reasoning in a continuous manner about a scene and the affordances it provides to an end-effector.

The underlying methodology can be extended to operate even when models of the objects and their poses are not known. Instead, mesh-approximation methods can be used to operate over point cloud data. Furthermore, there is work in learning effective grasping shapes for specific end-effectors [129, 133]. Developing a method for creating `MCG` representations of these shapes, would allow the proposed method to not require user input.

REFERENCES

[1] A. Economou, T.-C. (Hong, H. Ligler, and J. Park, "Shape machine: A primer for visual computation," in *A New Perspective of Cultural DNA*, J.-H. Lee, Ed. Singapore: Springer Singapore, 2021, pp. 65–92, ISBN: 978-981-15-7707-9.

[2] K. Hu, S. Yoon, V. Pavlovic, P. Faloutsos, and M. Kapadia, "Predicting crowd egress and environment relationships to support building design optimization," *Computers & Graphics*, 2020.

[3] M. Usman, D. Schaumann, B. Haworth, M. Kapadia, and P. Faloutsos, "Joint exploration and analysis of high-dimensional design–occupancy templates," in *Motion, Interaction and Games*, 2019, pp. 1–5.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005.

[5] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *CVPR*, IEEE, vol. 2, 2006, pp. 2169–2178.

[6] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," *TPAMI*, no. 12, pp. 2037–2041, 2006.

[7] L.-P. de las Heras, D. Fernández, A. Fornés, E. Valveny, G. Sánchez, and J. Lladós, "Runlength histogram image signature for perceptual retrieval of architectural floor plans," in *Workshop on Graphics Recognition*, Springer, 2013, pp. 135–146.

[8] S. Macé, H. Locteau, E. Valveny, and S. Tabbone, "A system to detect rooms in architectural floor plan images," in *Workshop on DAS*, ACM, 2010, pp. 167–174.

[9] P. Dosch and G. Masini, "Reconstruction of the 3d structure of a building from the 2d drawings of its floors," in *Document Analysis and Recognition*, IEEE, 1999, pp. 487–490.

[10] S. S. Sohn, H. Zhou, S. Moon, S. Yoon, V. Pavlovic, and M. Kapadia, "Laying the foundations of deep long-term crowd flow prediction," in *European Conference on Computer Vision*, Springer, 2020, pp. 711–728.

[11] D. Sharma, C. Chattopadhyay, and G. Harit, "A unified framework for semantic matching of architectural floorplans," in *Pattern Recognition*, IEEE, 2016, pp. 2422–2427.

[12] D. Sharma and C. Chattopadhyay, "High-level feature aggregation for fine-grained architectural floor plan retrieval," *IET Computer Vision*, vol. 12, no. 5, pp. 702–709, 2018.

[13] Q. U. Sabri, J. Bayer, V. Ayzenshtadt, S. S. Bukhari, K.-D. Althoff, and A. Dengel, "Semantic pattern-based retrieval of architectural floor plans with case-based and graph-based searching techniques and their evaluation and visualization.," in *ICPRAM*, 2017, pp. 50–60.

[14] A. Karambakhsh, V. Azizi, M. Hoseini, K. Heiran, M. Y. A. Khanian, and M. R. Meybodi, "A novel graph-based matching method to merge the extracted maps from mobile robots," in *2012 IEEE International Conference on Information and Automation*, IEEE, 2012, pp. 317–321.

[15] A. Heylighen and H. Neuckermans, "A case base of case-based design tools for architecture," *Computer-Aided Design*, vol. 33, no. 14, pp. 1111–1122, 2001.

[16] K. Richter, A. Heylighen, and D. Donath, "Looking back to the future. an updated case base of case-based design tools for architecture," Jan. 2007.

[17] G. Lambert and H. Gao, "Line moments and invariants for real time processing of vectorized contour data," in *International Conference on Image Analysis and Processing*, Springer, 1995, pp. 347–352.

[18] A. Dutta, J. Llados, and U. Pal, "Symbol spotting in line drawings through graph paths hashing," in *DAR*, IEEE, 2011, pp. 982–986.

[19] M. Weber, M. Liwicki, and A. Dengel, "A. scatch-a sketch-based retrieval for architectural floor plans," in *Frontiers in Handwriting Recognition*, IEEE, 2010, pp. 289–294.

[20] W. Wu, X.-M. Fu, R. Tang, Y. Wang, Y.-H. Qi, and L. Liu, "Data-driven interior plan generation for residential buildings," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–12, 2019.

[21] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," in *ACM SIGGRAPH Asia 2010 papers*, 2010, pp. 1–12.

[22] C. T. Mathew, P. R. Knob, S. R. Musse, and D. G. Aliaga, "Urban walkability design using virtual population simulation," in *Computer Graphics Forum*, Wiley Online Library, vol. 38, 2019, pp. 455–469.

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] G. E. Hinton and R. S. Zemel, "Autoencoders, minimum description length and helmholtz free energy," in *Advances in neural information processing systems*, 1994, pp. 3–10.

[25] V. Azizi, M. Usman, S. Patel, D. Schaumann, H. Zhou, P. Faloutsos24, and M. Kapadia, "Floorplan embedding with latent semantics and human behavior annotations," in *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 2020, pp. 337–344.

[26] V. Azizi, M. Usman, H. Zhou, P. Faloutsos, and M. Kapadia, *Graph-based generative representation learning of semantically and behaviorally augmented floorplans*, 2020. arXiv: 2012.04735 [cs.LG].

[27] V. Azizi, A. Kimmel, K. Bekris, and M. Kapadia, "Geometric reachability analysis for grasp planning in cluttered scenes for varying end-effectors," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, IEEE, 2017, pp. 764–769.

[28] M. Weizmann, O. Amir, and Y. J. Grobman, "Topological interlocking in architecture: A new design method and computational tool for designing building floors, Topological interlocking in architecture: A new design method and computational tool for designing building floors," *International Journal of Architectural Computing*, vol. 15, no. 2, pp. 107–118, Jun. 2017.

[29] S. Rockcastle and M. Andersen, "Measuring the dynamics of contrast & daylight variability in architecture: A proof-of-concept methodology," *Building and Environment*, vol. 81, pp. 320–333, Nov. 2014.

[30] A. Kaminska and A. Ożadowicz, "Lighting control including daylight and energy efficiency improvements analysis," *Energies*, vol. 11, no. 8, p. 2166, 2018.

[31] J. Clarke, *Energy simulation in building design*. Routledge, 2007.

[32] N. Delgarm, B. Sajadi, K. Azarbad, and S. Delgarm, "Sensitivity analysis of building energy performance: A simulation-based approach using ofat and variance-based sensitivity analysis methods," *Journal of Building Engineering*, vol. 15, pp. 181–193, 2018.

[33] L. Ben-Alon and R. Sacks, "Simulating the behavior of trade crews in construction using agents and building information modeling," *Automation in Construction*, vol. 74, pp. 12–27, Feb. 2017.

[34] M. Shin and J. S. Haberl, "Thermal zoning for building hvac design and energy simulation: A literature review," *Energy and Buildings*, vol. 203, p. 109 429, 2019.

[35] Z. Du, X. Jin, X. Fang, and B. Fan, "A dual-benchmark based energy analysis method to evaluate control strategies for building hvac systems," *Applied Energy*, vol. 183, pp. 700–714, 2016.

[36] B. Lawson, *How designers think: The design process demystified*. Routledge, 2006.

[37] B. Lawson, *What Designers Know*. Architectural Press, 2004.

[38] B. Hillier and J. Hanson, "The social logic of space, 1984," *Press syndicate of the University of Cambridge*, 1984.

[39] S. Bafna, "Space syntax: A brief introduction to its logic and analytical techniques," *Environment and Behavior*, vol. 35, no. 1, pp. 17–29, 2003.

[40] J. Desyllas and E. Duxbury, "Axial maps and visibility graph analysis," in *Proceedings, 3rd International Space Syntax Symposium*, Georgia Institute of Technology Atlanta, vol. 27, 2001, pp. 21–13.

[41] J. Gil, E. Tobari, M. Lemlij, A. Rose, and A. Penn, "The differentiating behaviour of shoppers: Clustering of individual movement traces in a supermarket," 2009.

[42] S. Sharmin and M. Kamruzzaman, "Meta-analysis of the relationships between space syntax measures and pedestrian movement," *Transport Reviews*, vol. 38, no. 4, pp. 524–550, 2018.

[43] G. Berseth, B. Haworth, M. Usman, D. Schaumann, M. Khayatkhoei, M. T. Kapadia, and P. Faloutsos, "Interactive architectural design with diverse solution exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 1, pp. 111–124, 2019.

[44] C. Hölscher, S. J. Büchner, T. Meilinger, and G. Strube, "Adaptivity of wayfinding strategies in a multi-building ensemble: The effects of spatial structure, task requirements, and metric information," *Environmental Psychology*, vol. 29, no. 2, pp. 208–219, 2009.

[45] M. Usman, D. Schaumann, B. Haworth, G. Berseth, M. Kapadia, and P. Faloutsos, "Interactive spatial analytics for human-aware building design," in *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, 2018, pp. 1–12.

[46] M. Kapadia, N. Pelechano, J. Allbeck, and N. Badler, "Virtual crowds: Steps toward behavioral realism," *Synthesis lectures on visual computing*, vol. 7, no. 4, pp. 1–270, 2015.

[47] D. Thalmann, "Populating virtual environments with crowds," in *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, 2006, pp. 11–11.

[48] W. Yan and Y. E. Kalay, "Simulating the behavior of users in built environments," *Journal of Architectural and Planning Research*, pp. 371–384, 2004.

[49] W. Shen, Q. Shen, and Q. Sun, "Building Information Modeling-based user activity simulation and evaluation method for improving designer–user communications," *Automation in Construction*, vol. 21, pp. 148–160, 2012.

[50] D. Schaumann, S. Breslav, R. Goldstein, A. Khan, and Y. E. Kalay, "Simulating use scenarios in hospitals using multi-agent narratives," *Journal of Building Performance Simulation*, vol. 10, no. 5-6, pp. 636–652, Nov. 2017.

[51] M. Usman, T.-C. Lee, R. Moghe, X. Zhang, P. Faloutsos, and M. Kapadia, "A social distancing index: Evaluating navigational policies on human proximity using crowd simulations," in *Motion, Interaction and Games*, 2020, pp. 1–6.

[52] X. Pan, C. S. Han, K. Dauber, and K. H. Law, "A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations," *Ai & Society*, vol. 22, no. 2, pp. 113–132, 2007.

[53] M. L. Chu, P. Parigi, K. Law, and J.-C. Latombe, "Modeling social behaviors in an evacuation simulator," *Computer Animation and Virtual Worlds*, vol. 25, no. 3-4, pp. 373–382, 2014.

[54] T. Feng, L.-F. Yu, S.-K. Yeung, K. Yin, and K. Zhou, "Crowd-driven mid-scale layout design.," *ACM Trans. Graph.*, vol. 35, no. 4, pp. 132–1, 2016.

[55] G. Berseth, M. Usman, B. Haworth, M. Kapadia, and P. Faloutsos, "Environment optimization for crowd evacuation," *CAVW*, vol. 26, no. 3-4, pp. 377–386, 2015.

[56] B. Haworth, M. Usman, G. Berseth, M. Kapadia, and P. Faloutsos, "Evaluating and optimizing level of service for crowd evacuations," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, 2015, pp. 91–96.

[57] B. Haworth, M. Usman, G. Berseth, M. Kapadia, and P. Faloutsos, "On density–flow relationships during crowd evacuation," *Computer Animation and Virtual Worlds*, vol. 28, no. 3-4, e1783, 2017.

[58] B. Haworth, M. Usman, G. Berseth, M. Khayatkhoei, M. Kapadia, and P. Faloutsos, "Towards computer assisted crowd aware architectural design," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 2016, pp. 2119–2125.

[59] B. Haworth, M. Usman, G. Berseth, M. Khayatkhoei, M. Kapadia, and P. Faloutsos, "Using synthetic crowds to inform building pillar placements," in *2016 IEEE Virtual Humans and Crowds for Immersive Environments (VHCIE)*, IEEE, 2016, pp. 7–11.

[60] M. Usman, D. Schaumann, B. Haworth, M. Kapadia, and P. Faloutsos, "Joint parametric modeling of buildings and crowds for human-centric simulation and analysis," in *International Conference on Computer-Aided Architectural Design Futures*, Springer, 2019, pp. 279–294.

[61] D. Schaumann, S. Moon, M. Usman, R. Goldstein, S. Breslav, A. Khan, P. Faloutsos, and M. Kapadia, "Join: An integrated platform for joint simulation of occupant-building interactions," *Architectural Science Review*, pp. 1–12, 2019.

[62] N. Chakraborty, B. Haworth, M. Usman, G. Berseth, P. Faloutsos, and M. Kapadia, "Crowd sourced co-design of floor plans using simulation guided games," in *Proceedings of the Tenth International Conference on Motion in Games*, 2017, pp. 1–5.

[63] M. B. Haworth, M. Usman, D. Schaumann, N. Chakraborty, G. Berseth, P. Faloutsos, and M. Kapadia, "Gamification of crowd-driven environment design," *IEEE Computer Graphics and Applications*, 2020.

[64] M. Usman, Y. Mao, D. Schaumann, P. Faloutsos, and M. Kapadia, "From semantic-based rule checking to simulation-powered emergency egress analytic," in *Proceedings of the Symposium on Simulation for Architecture and Urban Design*, 2020, pp. 43–50.

[65] D. Sharma, N. Gupta, C. Chattopadhyay, and S. Mehta, "Daniel: A deep architecture for automatic analysis and retrieval of building floor plans," in *DAR*, IEEE, vol. 1, 2017, pp. 420–425.

[66] R. M. Smelik, K. J. De Kraker, T. Tutenel, R. Bidarra, and S. A. Groenewegen, "A survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, vol. 2009, 2009, pp. 25–34.

[67] P. Charman, "Solving space planning problems using constraint technology," *Institute of Cybernetics-Estonian Academy of Sciences*, 1993.

[68] E. Rodrigues, A. R. Gaspar, and Á. Gomes, "An evolutionary strategy enhanced with a local search technique for the space allocation problem in architecture, part 2: Validation and performance tests," *Computer-Aided Design*, vol. 45, no. 5, pp. 898–910, 2013.

[69] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation ofpseudo infinite'cities," in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 2003, 87–ff.

[70] A. Rau-Chaplin, B. MacKay-Lyons, and P. Spierenburg, "The lahave house project: Towards an automated architectural design service," *Cadex*, vol. 96, pp. 24–31, 1996.

[71] J. Martin, "Procedural house generation: A method for dynamically generating floor plans," in *Proceedings of the Symposium on Interactive 3D Graphics and Games*, 2006, pp. 1–2.

[72] S. Chaillou, "Ai+ architecture: Towards a new approach," *Harvard University*, 2019.

[73] R. Hu, Z. Huang, Y. Tang, O. van Kaick, H. Zhang, and H. Huang, "Graph2plan: Learning floorplan generation from layout graphs," *arXiv preprint arXiv:2004.13204*, 2020.

[74] N. Nauata, K.-H. Chang, C.-Y. Cheng, G. Mori, and Y. Furukawa, *House-gan: Relational generative adversarial networks for graph-constrained house layout generation*, 2020. arXiv: 2003.06988 [cs.CV].

[75] B. Haworth, M. Usman, G. Berseth, M. Khayatkhoei, M. Kapadia, and P. Faloutsos, "Code: Crowd-optimized design of environments," *Computer Animation and Virtual Worlds*, vol. 28, no. 6, e1749, 2017.

[76] T. Li, D. Ho, C. Li, D. Zhu, C. Wang, and M. Q. .-H. Meng, *Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots*, 2019. arXiv: 1903.09845 [cs.RO].

[77] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, "An open framework for developing, evaluating, and sharing steering algorithms," in *Motion in Games*, Springer, Springer-Verlag, 2009, pp. 158–169.

[78] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[79] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, ACM, 2016, pp. 855–864.

[80] A. Taheri, K. Gimpel, and T. Berger-Wolf, "Learning graph representations with recurrent neural network autoencoders," *KDD*, 2018.

[81] Q. Ai, V. Azizi, X. Chen, and Y. Zhang, "Learning heterogeneous knowledge base embeddings for explainable recommendation," *Algorithms*, vol. 11, no. 9, p. 137, 2018.

[82] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "Graph2vec: Learning distributed representations of graphs," *arXiv:1707.05005*, 2017.

[83] H. Bunke, "What is the distance between graphs," *Bulletin of the EATCS*, vol. 20, pp. 35–39, 1983.

[84] S. Singh, M. Kapadia, P. Faloutsos, and G. Reinman, "An open framework for developing, evaluating, and sharing steering algorithms," in *International Workshop on Motion in Games*, Springer, 2009, pp. 158–169.

[85] P. Kumar and H. H. Huang, "G-store: High-performance graph store for trillion-edge processing," in *SC'16*, IEEE, 2016, pp. 830–841.

[86] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *arXiv preprint arXiv:1204.6078*, 2012.

[87] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 599–613.

[88] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques and applications," *CoRR*, vol. abs/1709.07604, 2017. arXiv: 1709.07604.

[89] Y. Bai, H. Ding, Y. Qiao, A. Marinovic, K. Gu, T. Chen, Y. Sun, and W. Wang, "Unsupervised inductive graph-level representation learning via graph-graph proximity," *arXiv preprint arXiv:1904.01098*, 2019.

[90] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[91] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[92] S. F. Mousavi, M. Safayani, A. Mirzaei, and H. Bahonar, "Hierarchical graph embedding in vector space by graph pyramid," *Pattern Recognition*, vol. 61, pp. 245–254, 2017.

[93] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Józefowicz, and S. Bengio, "Generating sentences from a continuous space," *CoRR*, vol. abs/1511.06349, 2015. arXiv: 1511.06349.

[94] W. Wang, Z. Gan, H. Xu, R. Zhang, G. Wang, D. Shen, C. Chen, and L. Carin, "Topic-guided variational autoencoders for text generation," *arXiv:1903.07137*, 2019.

[95] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[96] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[97] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[98] M. Simonovsky and N. Komodakis, "Graphvae: Towards generation of small graphs using variational autoencoders," in *International Conference on Artificial Neural Networks*, Springer, 2018, pp. 412–422.

[99] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: 1412.6980 [cs.LG].

[100] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*, 2014.

[101] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, *et al.*, "Constrained k-means clustering with background knowledge," in *Icml*, vol. 1, 2001, pp. 577–584.

[102] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, 1996, pp. 226–231.

[103] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[104] T. Dogan, E. Saratsis, and C. Reinhart, "The optimization potential of floor-plan typologies in early design energy modeling," 2015.

[105] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross, "Optimized spatial hashing for collision detection of deformable objects.," in *Vmv*, vol. 3, 2003, pp. 47–54.

[106] V. Azizi and A. T. Haghighat, "Fast and robust biped walking involving arm swing and control of inertia based on neural network with harmony search optimizer," in *2011 16th International Conference on Methods & Models in Automation & Robotics*, IEEE, 2011, pp. 375–380.

[107] F. Koochaki, I. Sharifi, H. A. Talebi, and A. D. Mohammadi, "Nonlinear control of a non-passive bilateral teleoperation in presence of unsymmetric time varying delay," in *2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*, IEEE, 2014, pp. 019–023.

[108] F. Koochaki, I. Sharifi, A. Doostmohammadi, and H. A. Talebi, "A cooperative remote rehabilitation system," in *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, IEEE, 2015, pp. 085–090.

[109] F. Koochaki and H. A. Talebi, "Position coordination of nonlinear teleoperation for unrestricted non-passive environment and operator in the presence of time delay," in *2016 4th International Conference on Control, Instrumentation, and Automation (ICCIA)*, IEEE, 2016, pp. 407–412.

[110] F. Koochaki, I. Sharifi, and H. A. Talebi, "A novel architecture for cooperative remote rehabilitation system," *Computers & Electrical Engineering*, vol. 56, pp. 715–731, 2016.

[111] F. Khosrosereshki, F. Rasouli, H. A. Talebi, F. Koochaki, and I. Sharifi, "Path planning for insertion of a bevel tip steerable needle into a soft tissue in the presence of obstacles," in *2014 22nd Iranian Conference on Electrical Engineering (ICEE)*, 2014, pp. 1348–1353.

[112] A. Karambakhsh, V. Azizi, M. Hoseini, K. Heiran, M. Y. A. Khanian, and M. R. Meybodi, "A novel graph-based matching method to merge the extracted maps from mobile robots," in *2012 IEEE International Conference on Information and Automation*, 2012, pp. 317–321.

[113] S. Habibian, M. Dadvar, B. Peykari, A. Hosseini, M. H. Salehzadeh, A. H. Hosseini, and F. Najafi, "Design and implementation of a maxi-sized mobile robot (karo) for rescue missions," *ROBOMECH Journal*, vol. 8, no. 1, pp. 1–33, 2021.

[114] M. Dadvar, S. Moazami, H. R. Myler, and H. Zargarzadeh, "Multiagent task allocation in complementary teams: A hunter-and-gatherer approach," *Complexity*, vol. 2020, 2020.

[115] M. Islam, M. Dadvar, and H. Zargarzadeh, "A dynamic territorializing approach for multiagent task allocation," *Complexity*, vol. 2020, 2020.

[116] C. Goldfeder, M. Ciocarlie, H. Dang, and P. K. Allen, "The columbia grasp database," in *ICRA*, IEEE, 2009, pp. 1710–1716.

[117] J. Bohg, A. Morales, T. Asfour, and D. Kragic, "Data-driven grasp synthesis—a survey," *IEEE TRO*, vol. 30, no. 2, pp. 289–309, 2014.

[118] M. A. Roa and R. Suárez, "Grasp quality measures: Review and performance," *Autonomous Robots*, vol. 38, no. 1, pp. 65–88, 2015.

[119] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3d object grasp synthesis algorithms," *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 326–336, 2012.

[120] S. Liu and S. Carpin, "A fast algorithm for grasp quality evaluation using the object wrench space," in *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*, IEEE, 2015, pp. 558–563.

[121] Y. Lin and Y. Sun, "Grasp planning to maximize task coverage," *The International Journal of Robotics Research*, vol. 34, no. 9, pp. 1195–1210, 2015.

[122] S. Liu and S. Carpin, "Kinematic noise propagation and grasp quality evaluation," in *Automation Science and Engineering (CASE), 2016 IEEE International Conference on*, IEEE, 2016, pp. 1177–1183.

[123] A. T. Miller and P. K. Allen, "Graspit! a versatile simulator for robotic grasping," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 110–122, 2004.

[124] K. Hang, J. A. Stork, and D. Kragic, "Hierarchical fingertip space for multi-fingered precision grasping," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 1641–1648.

[125] F. Zacharias, C. Borst, and G. Hirzinger, "Online generation of reachable grasps for dexterous manipulation using a representation of the reachable workspace," in *Advanced Robotics, 2009. ICAR 2009. International Conference on*, IEEE, 2009, pp. 1–8.

[126] M. Popović, G. Kootstra, J. A. Jørgensen, D. Kragic, and N. Krüger, "Grasping unknown objects using an early cognitive vision system for general scene understanding," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, IEEE, 2011, pp. 987–994.

[127] Z. Xue and R. Dillmann, "Efficient grasp planning with reachability analysis," *International Journal of Humanoid Robotics*, vol. 8, no. 04, pp. 761–775, 2011.

[128] D. Berenson and S. S. Srinivasa, "Grasp synthesis in cluttered environments for dexterous hands," in *Humanoids*, IEEE, 2008, pp. 189–196.

[129] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[130] A. ten Pas and R. Platt, "Using geometry to detect grasp poses in 3d point clouds," in *Int'l Symp. on Robotics Research*, 2015.

[131] M. Kopicki, R. Detry, M. Adjigble, R. Stolkin, A. Leonardis, and J. L. Wyatt, "One-shot learning and generation of dexterous grasps for novel objects," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 959–976, 2016.

[132] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[133] A. T. Miller, S. Knoop, H. I. Christensen, and P. K. Allen, "Automatic grasp planning using shape primitives," in *Proceedings. ICRA'03. IEEE International Conference on*, IEEE, vol. 2, 2003, pp. 1824–1829.

[134] M. Ciocarlie, C. Goldfeder, and P. Allen, "Dimensionality reduction for hand-independent dexterous robotic grasping," in *IROS*, IEEE, 2007, pp. 3270–3275.

[135] M. Kapadia, X. Xianghao, M. Nitti, M. Kallmann, S. Coros, R. W. Sumner, and M. Gross, "Precision: Precomputing environment semantics for contact-rich character animation," in *ACM Symposium on Interactive 3D Graphics and Games*, ACM, 2016, pp. 29–37.

[136] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *Humanoids*, IEEE, 2015, pp. 928–935.

[137] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous grasp and motion planning: Humanoid robot armar-iii," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 43–57, 2012.

[138] Z. Littlefield, A. Krontiris, A. Kimmel, A. Dobson, R. Shome, and K. E. Bekris, "An extensible software architecture for composing motion and task planners," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, Springer, 2014, pp. 327–339.