Reasoning About Interior Building Design, Grounded on Design Rules

by

Christoph Peter Sydora

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Computers have emerged as an invaluable tool in exploring building interior configurations, before committing to a particular layout. Building Information Modeling (BIM) enables designers to create digital representations of alternative interior arrangements and supports computer-automated evaluation of the arrangements under consideration against a variety of construction, accessibility, and stylistic guidelines. This process is known as Automated Code Checking (ACC). Domain Specific Languages (DSLs) can help domain experts code ACC rules in a machine-readable format, focusing only on domain concepts instead of programming knowledge.

This thesis provides a coherent suite of algorithms for reasoning about building designs based on RuleDSL, a user-friendly DSL for describing building spaces, their contents, and the geometric relations among them. The thesis puts forward algorithms for (i) automatically generating alternative interior layouts, (ii) evaluating them against a variety of quality metrics, and (iii) automatically learning RuleDSL rules from example layouts. RuleDSL and the above algorithms have been developed and evaluated in BIM-kit, a state-of-the-art software platform that implements and validates the above algorithms as well as supporting experimentation with a variety of use cases in the broad area of design automation.

Preface

This thesis is an original work by Christoph Sydora. The research project described in Chapter 5 received research ethics approval from the University of Alberta Research Ethics Board, Project Name "Interior Design Layout Data Collection and Evaluation", No. Pro00124859, February 28, 2024.

Segments from this thesis have been published in the following literature:

- Sections 2.1, and 2.2, and 2.6 is research done prior to this thesis that is published in: **Christoph Sydora** and Eleni Stroulia, "Rule-Based Compliance Checking and Generative Design for Building Interiors Using BIM," In Automation in Construction, 2020. [1]

- Section 2.3 and Chapter 3 is the work presented in the conference publication: **Christoph Sydora** and Eleni Stroulia, "Comparative Analysis of Room Generation Methods Using Rule Language-Based Evaluation in BIM," In Proceedings of European Conference on Computing in Construction (EC3) and International CIB W78 Conference, 2023. [2]

- A manuscript on the research presented in Chapters 4 and 5 is being prepared.

- Section 2.5 and Chapter 6 is the work presented in the conference publication: **Christoph Sydora** and Eleni Stroulia, "BIM-kit: An Extendible Toolkit for Reasoning about Building Information Models," In Proceedings of European Conference on Computing in Construction (EC3), 2021. [3]

Some improvements to the language and figures in these publications have been made.

Acknowledgements

This thesis does not exist without the guidance and support of my supervisor, Dr. Eleni Stroulia. From the very beginning, she has and continues to amaze me with her passion and dedication to research and the positive, fun approach she has to it. It has truly been an honour to be a part of her lab.

I express my deepest thanks to my supervisory committee Dr. Omid Ardakanian, Dr. Vicente Gonzalez-Moret, and Dr. Nathan Sturtevant. They have been immensely supportive in helping me shape and improve this work. I would also like to thank my examining committee Dr. Marek Reformat and Dr. Robert Amor for their time reviewing this thesis and serving on my committee; and to Dr. Abram Hindle for much-needed feedback during my candidacy. I thank the Natural Sciences and Engineering Research Council (NSERC) and Mitacs Canada for their financial support of this research.

A Ph.D. is a long endeavour, but a good fellowship keeps it from feeling like an eternity. I have been incredibly fortunate to share the lab with many great labmates, particularly Victor Fernandez-Cervantes, Kalvin Eng, Mashrura Tasnim, and David Turner, who have been there with me for the full duration of my Ph.D.

I thank my friends for sharing their time with me to enjoy a life outside the lab and my family, in particular my siblings Michael, Audrey, Colin, and Brendan, for keeping my spirits up.

Finally, I thank my parents, Dr. Beate Sydora and Dr. Richard Sydora. Having their support and help navigating my graduate endeavours has been immeasurable. Not once did I feel pressured to follow in their steps with a Ph.D. but still they are examples I will forever aspire towards. I will continue to strive to make even a fraction of the impact they have made.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

ACC
Automated Code Checking

API
Application Programmable Interface

AR
Augmented Reality

BERA
Building Environment Rule and Analysis

BIM
Building Information Modeling

BIMQL
BIM Query Language

BIMRL
BIM Rule Language

BIMRLSS
BIMRL Simplified Schema

BOM
Building Object Model

CAD
Computer Aided Design

CNN
Convolutional Neural Network

DDLS
Data-Driven Layout Synthesis

DSL
Domain Specific Language

EDM
Express Data Manager

GA
Genetic Algorithm

GANs
Generative Adversarial Networks

GS
Grid Search

GSA
General Services Administration

GUID
Globally Unique Identifier

ID
Unique Identifier

IFC
Industry Foundation Classes

JS
Jump Search

LLMs
Large Language Models

LOD
Level-of-Detail

MCMC
Markov Chain Monte Carlo

MEP
Mechanical, Electrical, Plumbing

ML
Machine Learning

MVD
Model View Definition

NLP
Natural Language Processing

NN
Neural Network

PSO
Particle Swarm Optimization

QTO
Quantity Take-off

RASE
Requirement, Applicability, Selection, Exception

RDF
Resource Description Framework

SA
Simulated Annealing

SMC
Solibri Model Checker

SQL
Structured Query Language

UI
User Interface

UML
Unified Modeling Language

VAE
Variational Autoencoder

VCCL
Visual Code Checking Language

VPL
Visual Programming Languages

VR
Virtual Reality

Chapter 1

Introduction

Buildings are a vital part of our everyday lives. They serve various functions, from homes for living, offices for work, schools for learning, etc. The design of our buildings significantly influences how well we, their occupants, can carry out our activities and perform our tasks within the available space. Building interiors are one aspect of the design that greatly impacts the ability of occupants to perform tasks efficiently. A well-designed space complies with building codes and standards for accessible and green space and meets the aesthetic styles and individual preferences of its occupants. In such well-designed spaces, occupants can live and work more safely, efficiently and happily.

Over the years, experts have researched different design aspects that have resulted in well-designed interior spaces. Their expertise is often formulated in the form of rules and guidelines. A rule-compliant design can therefore be expected to perform better than when not. Naturally, the goal of any designer is to design a space as best as possible. Toward this goal, a designer should attempt to comply with the rules pertinent to the relevant codes and occupant needs for a particular space.

Planning is a vital stage in which a designer can examine multiple possible layout configurations before making any laborious or financial commitments. With advancements in computer technology, Computer Aided Design (CAD) planning tools have emerged, capable of creating 2D and 3D renderings of building designs. Building Information Modeling (BIM) is the process by which all building information is digitally created and managed within a 3D

CAD model, including specifications about the building components and the relationships among those components [4]. BIM combines 3D models with a conceptual representation of the building, its spaces, their geometry, and interior furnishings. By digitizing the building information, visualization and assessment of proposed layout configurations (and consequently design planning and optimization) becomes easier.

A key benefit to the organization and standardization of building information in BIM is the potential to automate many design tasks. One envisioned and highly sought-after use case for BIM is the automatic review of designs to assess the performance of a design, also known as automatic model checking or Automated Code Checking (ACC). In ACC, a proposed design is assessed automatically against expert rules to gauge its performance [5]. The expert rules must be written in a machine-readable format, i.e. computer code, such that a computer can programmatically check the model. This has been a prevalent challenge as rule programming to date is often written in general programming languages that require years to master.

1.1 Domain Specific Languages for Automatic Code Checking

More recently, efforts have been made to mitigate the challenges of programming expert rules. Domain Specific Language (DSL)s offer one solution. DSLs are programming languages that use vocabularies that are more high-level and relevant to the domain in which they are used. In the case of building design evaluation, several DSLs have been proposed.

Two rule DSL of note are BERA [6], [7] and BIMRL [8], [9]. The Building Environment Rule and Analysis (BERA) language was developed as a simplified Java-based rule language with a domain focus on spatial circulation and paths. BIM Rule Language (BIMRL) is a Structured Query Language (SQL) approach for performing queries on the BIM data. Similar to the work in this thesis, both report the use of a simplified BIM data representation; BERA uses the Building Object Model (BOM) and BIMRL the BIMRL Simplified

Schema (BIMRLSS).

However, these works fall short when compared to this thesis is in the domains they are applied to, as neither has been tested on the interior furnishing arrangement domain, and their use beyond model checking. The RuleDSL used in this thesis, described in detail in Section 2.6, is a precursor to this work explicitly conceived to support reasoning about interior design [1]. It has been tested on rules specifically targeting the geometric relationships among furnishing objects in different living spaces. Additionally, its feasibility as an evaluator in generative layout search has been tested and demonstrated to produce rule-compliant layouts automatically, which is improved through its ability to scale rule results instead of returning boolean pass or fail results.

1.2 Automated Interior Layout Generation

Two general methodologies have been proposed for interior layout generation. First are methods that rely on explicitly encoded rules, dating back to Merrell et al. [10], where rules are hard-coded in general-purpose programming languages within the CAD tool. With these methods, rule coding becomes a non-trivial and error-prone task as expertise in rule interpretation and programming is required to effectively write the rules.

The others are the data-driven layout generation approaches, where plausible layouts are input and probabilistic models are trained to re-create equally plausible layouts such as the work in Fisher et al. [11]. Recently, image generation has also become a useful tool in interior design where layout images can be created to suggest plausible layouts, such as the works of Betker et al. [12] and Zhang et al. [13]. In both cases, generated layouts are made to "look" like the input models but there is no understanding of whether or not these layouts comply with relevant design rules. Therefore, while useful for recreating believable and aesthetic layouts, these methods do not guarantee the creation of rule-compliant designs.

In this thesis, layout generation uses constraints in the form of an easy-to-use DSL, the RuleDSL, with the explicit goal of generating rule-compliant

layouts.

1.3 Thesis Objectives

The goal of this thesis is to develop and evaluate new algorithms relating to model checking and automated layout generation tasks. The algorithms presented in this thesis use or create rules in the form of RuleDSL a user-friendly DSL for describing rules on building components' geometric relationships. In particular, I will be pursuing the following research questions throughout this thesis.

Question 1: Two key criteria for automated layout generation algorithms are (i) the quality of the layouts in terms of the search goal, in our case the rule compliance score and (ii) how fast a search algorithm converges on a high-quality layout configuration i.e. its search efficiency. I have developed and comparatively evaluated one grid-based methodology and two continuous random sampling algorithms, based on these two metrics. **How do output layout quality and layout configuration search efficiency trade-offs differ among the layout generation algorithms?**

Question 2: The next question is regarding the creation of rules in the RuleDSL format. Rules are edited through a special-purpose syntax-aware editor. Three editors have been developed. One is a form-based editor where rules are created through a series of dynamic drop-down selection and data input widgets. The others are prototypes for a Visual Programming Languages (VPL) using BLOCKLY [14] and Natural Language Processing (NLP) text-to-code tokenizer which are still being tested and evaluated. Coding rules is difficult and while the focused RuleDSL vocabulary on interior design eases the coding process, the syntax and semantics behind relation terms can still cause challenges. A potentially simpler rule creation process might be to only provide examples where the rules are adhered to and ideally have the rules extracted automatically. My algorithm approach looks at the object relationship patterns in the examples and the range of their values, rules can be formulated

on the bounds of the relation ranges and converted to the RuleDSL. **How well does the design-rule learning algorithm capture the quality of the layouts of its input examples?**

Question 3: Rule compliance, while arguably the most important, is not the only metric for whether a layout is good or not. Often users will have preferences for their arrangements that they may not be able to express in geometrical terms like the rules. Therefore, in addition to capturing rule information, the ability of the rule learning algorithm to capture non-trivial preferences and aesthetics is also of interest to someone who may use the rules to automatically evaluate or generate layouts similar to a set of examples. **How well is perceived quality captured and recreated using the same rule learning algorithm?**

Question 4: The final question that I explore in this thesis relates to the synthesis of all BIM reasoning applications as web services, in a coherent software platform. **What software architecture can be employed to support the RuleDSL and other BIM reasoning application Application Programmable Interface (API)s?**

1.4 Thesis Contributions and Structure

In Chapter 2, I will outline background information regarding previous research relating to the work I present in this thesis and information regarding the RuleDSL central to this thesis.

The contribution of Chapter 3 is a new continuous space search algorithm, Jump Search (JS). I describe the algorithm, evaluate its effectiveness in delivering high-quality layouts in many different problem scenarios, and compare its search efficiency and output quality against a previous grid-based method, Grid Search (GS), and a non-greedy continuous space search algorithm based on Simulated Annealing (SA).

Chapter 4 describes a novel rule learning method and demonstrates its quality and effectiveness at capturing known rule information through a rule

correlations analysis. Chapter 5 details a user study for curating a layout dataset and evaluating the rule learning algorithm's ability to capture and recreate perceived quality.

Chapter 6 first puts forward a well-defined, extendable BIM model; the model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set cross-referenced objects. Because the model consists of several distinct objects that are manipulated through well-defined APIs, the model can be more easily shared across different tools. Then, it demonstrates the usefulness of the above model through the implementation of a variety of interactive and automated tools that, together, cover a wide range of activities that reason about building models.

Finally, Chapter 7 summarizes my research findings in terms of the objectives of this thesis, states the contributions of this work, and introduces potential future research directions.

1.5 Research Methodology

The research methodology employed in this thesis is fundamentally experimental. Each chapter is organized as a report on a stand-alone research activity and describes the general motivation of the activity and the specific problem that it addresses. Then, state-of-the-art approaches related to this activity are identified, to provide the context against which the contribution of the chapter should be considered. I have assembled all the related works relevant to all the thesis activities in Chapter 2.

Chapters 3 and 4 report on new algorithms while Chapter 6 describes the concise building data model that I have developed to enable the implementation of these algorithms in an integrated software platform.

The automated layout algorithm of Chapter 3 is comparatively evaluated against two earlier algorithms to provide statistically significant evidence of its effectiveness in different room scenarios (See Section A.1 in the Appendix) and efficiency under computational search budgets. Through this experimental design, I investigate the performance quality trade-offs of my new algorithm.

The experimental evaluation of the rule learning method of Chapter 4 is conducted in Chapters 4 and 5. Because there is no direct competitor to our rule learning algorithm, the experimental evaluation design focuses on the effectiveness of the rule learning algorithm in capturing the design quality information embedded in the input examples in the above room scenarios. For rule learning algorithm analysis in Chapter 4, the room scenarios rules represent the target and the correlation analysis gives a metric of the extent to which the method achieves the target. Further analysis of the rule learning algorithm is performed in Chapter 5 via a two-part user study, with user-created layout data collection first followed by user-perceived quality evaluations.

Finally, Chapter 6 validates the concise building information data model and RuleDSL underlying this thesis by detailing its capability of representing and organizing the required information for fulfilling important reasoning tasks.

Chapter 2

Background and Related Research

I will start this chapter with a brief review of automated model checking (which I use interchangeably with Automated Code Checking (ACC) in this thesis) and Domain Specific Language (DSL)s that have been proposed for checking various building rules. Next, automatic layout generation algorithms are reviewed, looking at their evaluation and search methods separately. As some methods for layout generation rely on learning or extracting knowledge from examples, I examine how their methods represent that knowledge. Finally, software environments for model checking, generative design, and other reasoning tasks are reviewed.

Segments of this chapter were taken from the following publications with some modifications and updates:

1. Sections 2.1, and 2.2 are from research prior to this thesis in the publication: **Christoph Sydora** and Eleni Stroulia, "Rule-Based Compliance Checking and Generative Design for Building Interiors Using BIM," Automation in Construction, 2020. [1]

2. Section 2.3 relates to the work in Chapter 3 which is published in: **Christoph Sydora** and Eleni Stroulia, "Comparative Analysis of Room Generation Methods Using Rule Language-Based Evaluation in BIM," In Proceedings of European Conference on Computing in Construction (EC3) and International CIB W78 Conference, 2023. [2]

3. Section 2.4 belong to the work in Chapters 4 and 5 which has not yet been published.

4. Section 2.5 is from the publication: **Christoph Sydora** and Eleni Stroulia, "BIM-kit: An Extendible Toolkit for Reasoning about Building Information Models," In Proceedings of European Conference on Computing in Construction (EC3), 2021. [3]

More in-depth reviews of model-checking solutions can be found in the BIM Handbook [4] and review papers from Eastman et al. [5], Borrmann et al. [15], Dimyadi et al. [16], Greenwood et al. [17], Ismail et al. [18], Hjelseth [19], Pauwels et at. [20], and Solihin et al. [21].

2.1 Model Checking Tools and Workarounds

When researching compliance-checking tools, Solibri Model Checker (SMC) [22] is frequently mentioned as it is one of the few tools specifically built for the purpose of checking BIM models. SMC takes as input a building model in the form of the BIM industry standard of IFC. While the available rulesets, initially from the Norwegian Statsbygg handbook [23], can be modified by the end user by combining rule sets and deleting rules, support for editing individual rules is limited to changing the parameters of the provided rules. Additionally, there are a few rule templates for creating new rules, however, full customization of rules can only be done through the SMC Application Programmable Interface (API), which is not publicly available.

Model-checking tools have been implemented for the purpose of evaluating the requirements of governing bodies, with differing levels of success. Singapore's CORENET ePlanCheck [24] has been noted as the most successful implementation, since, at one point, it was mandatory as part of the government's building requirement legislation [5], [18]. In Australia, DesignCheck [25] was built on the Express Data Manager (EDM) Model Server but, to the best of the authors' knowledge has since lost support. The General Services Administration (GSA) in the United States mandates that their project

models be checked with rules implemented within SMC [5].

While not specifically model-checking tools, BIM editors, such as Autodesk Revit [26] and Graphisoft ArchiCAD [27], provide APIs for add-on development, allowing access to the model's internal structure and object database and therefore, can, in principle, be used for model checking. This requires a high level of programming knowledge, even for the simplest checks. To address this challenge, some tools have been developed to perform the same functionality in a visual environment. These include tools such as Autodesk Dynamo [28], which works on the Revit platform, and Rhino Grasshopper [29]. These two tools are both graph-based visual editors that have some scripting available - Dynamo's Python scripting rather than C# as the Revit API. BIMServer [30], an open-source IFC model repository platform, has a model-checking plugin, however, it requires direct coding in JavaScript. The scripts are then linked to the model for execution. This also requires programmatic coding knowledge and a strong understanding of the IFC vocabulary and syntax.

2.2 Model Search and Rule Languages

Query Languages and Semantic Web Ontologies: As model-checking is, in theory, a query on a BIM model, and given the emergence of semantic-web technologies, there have been a number of methods using semantic web languages as the basis for the checks. Specifically, methods have worked with extendable IFC-based ontologies of the BIM model to query for design flaws. Pauwels et al. [31] for instance performed acoustic regulation compliance checking for BIM models using Resource Description Framework (RDF) graphs of the building model. The process requires a model to be passed through and IFC-to-RDF converter which can then be queried by a rule described in the Notation-3 (N3) syntax. A more general overview of these types of methods can be found in Pauwels et al. [20]. While this technology can be useful in extending the data schema, the query languages require a steeper learning curve and a data converter from IFC to RDF data models which is not a straightforward task.

BIM Query Language (BIMQL) [32] is another query language, built on the BIMServer platform. The language uses a syntax very similar to SQL that reads an IFC model, the goal being an easier transition for users more familiar with query languages as opposed to learning a programming language. BIMQL has the capability to check the existence of model elements and some minor model manipulation. Therefore, while query languages can perform the task of model-checking, they focus on the existing data using complex query languages and are likely better suited for Quantity Take-off (QTO), that is to count the number of item instances in a model, than model-checking.

Rule-Checking Languages: The Building Environment Rule and Analysis (BERA) language [6] was developed as a domain-specific programming language for model checking. The concept is built on providing model-checking capabilities without the need for precise knowledge of general-purpose programming languages [7]. However, the language derives heavily from Java which may be difficult for non-programmers and it is built on SMC as an IFC engine and therefore is still quite opaque. Although it offers the potential for extensibility, to my knowledge, it has not been developed outside a few building circulation rules.

BIM Rule Language (BIMRL) [9], [33] represents another rule language approach. This method draws influence heavily from SQL, therefore, for a non-programmer, the language can appear complex. This language does contribute some key concepts such as the representation of the data from complex IFC data to simplified shape representations and the use of temporary geometry for spacial-based evaluations [34]. Like the RuleDSL in this thesis research, they have designed their DSL with a central focus around a simplified building data representation.

KBim [35], [36] was built specifically for expressing the Korean Building Act legislation into commutable form. The method is broken down into KBimLogic [35], a tool for assisting users in the natural language parsing and information extraction of the rules based on objects, properties, and high-level methods stored in SQL database tables, and KBimCode, which then further

converts the KBimLogic structured rules into computer executable code [36]. The code-checking system is called KBimAssess-lite [37].

Unlike these previous methods, I will show that my language supports scaling the results of rules which is beneficial for generative design. Should a rule fail, my language is able to additionally determine the severity of the failure. This concept is necessary for the generative design to more intelligently adjust and improve configurations of the model layout so that a locally optimal solution can be found.

Visual Programming Languages (VPL): Some approaches have taken the Rule Languages one step further by adding a visual component to them, in the same sense that Dynamo is a visual language for Revit's API. This is intended to allow for more complex rules to be created without adding the need to code programming, although, to my knowledge, this has not been tested for ease of use.

Check-mate [38] first introduced this as a very simple puzzle-based interface that allowed connecting pieces that together would form a structured rule. However, the expressiveness of this language was limited. The Visual Code Checking Language (VCCL) took a node-based approach, calling it a "white-box" approach with the available nodes to be extendable as the project matures [39], [40]. The language was then refined to support more complex rules by modularizing nodes, thus allowing for nodes to build around other predefined nodes [41]. In similar fashion, KBim has also since implemented a VPL version of KBimCode to improve ease of use [42].

The RuleDSL used in this thesis has a puzzle like VPL built using BLOCKLY [14]. The extent to which this editor improves user interaction and experience is something I will be investigating in future work.

Natural Language Processing (NLP): Attempts have been made to parse natural language rules from design handbooks and regulation texts. While such approaches could potentially simplify the rule-creation process, many of the natural language rules lack the clarity and unambiguity required

to be directly parsed without any human intervention or interpretation. One of the more commonly cited approaches in this vein is that of Hjelseth [43] and Hjelseth and Nisbet [44] which used a four-sentence component classification to parse natural language rules, namely Requirement, Applicability, Selection, Exception (RASE). Another use of NLP has been to identify information from rules that are missing or may need to be added to models [45]. Such methods could potentially be antecedent to my method and will be explored in the future.

2.3 Automated Layout Generation Rules and Search Algorithms

In this section, I will first review some rule constraint formulation methods that result in layout evaluation scores used in related automatic layout generation research. This is because layout evaluations are similar to model checks in that both score a layout and report a result. Then, I will outline layout optimization algorithms, specifically the layout modification and action selection algorithms. A more in-depth survey of automated layout generation (or sometimes Scene Synthesis) can be found in Zhang et al. [46].

Configuration Evaluation Methods: A common approach is to define rules in terms of geometrical formulas as in Merrell et al. [10], Akase and Okada [47], Kan and Haufman [48], Li et al. [49], Liang et al. [50], Yu et al. [51], and Zhang et al. [52]. The most prominent criticism of this rule definition approach is the difficulty in defining and testing new rules, specifically for domain experts lacking the skills to formulate the rules. These approaches are also not able to be data-driven, i.e. cannot be directly derived from examples.

Relational graph approaches have been proposed by Yeh et al. [53], Xu et al. [54], Kermani et al. [55], Fu et al. [56], Liang et al. [57], Wang et al. [58], Zhou et al. [59], Li et al. [60], and Kesharvarzi et al. [61]. Typical graphs will pre-define a set of possible relational occurrences, such as "next to" or "facing", with nodes in the graph representing the objects placed in the

layout and edges as the relations. A similar approach is based on statistical likelihoods of relational occurrences as in Fisher et al. [11], Chang et al. [62], and Zhang et al. [52]. Both graph and probabilistic models are typically example-dataset dependent and not intended to be coded by a user.

To make the rule creation more accessible to domain experts, some methods initially use text-based input (such as Liang et al. [57]), before being converted to graphical relation models as rules. Other approaches have defined the rules in terms of actions or activities, rather than classical geometric rules such as in Fisher et al. [63], Ma et al. [64], Qi et al. [65], and Fu et al. [66]. Thus, the evaluation scores are determined by simulation or path planning.

Finally, Machine Learning (ML) methods using Neural Network (NN) have been proposed that determine location or location probabilistic mapping based on scene feature (such as Wang et al. [58], Zhou et al. [59], Li et al. [60], Wang et al. [67], Yang et al. [68], Ritchie et al. [69], and Yang et al. [70]).

The rule language described in this thesis supports a user-friendly process for specifying design rules, in a representation format that maintains textual descriptions rather than geometrical formulas and supports more complex relations than graph representations.

Configuration Variation and Selection Methods: There are two broad categories of approaches for placing objects. Either all objects are placed and then simultaneously moved around in a single action or a single object is moved per action.

Methods for moving all objects include Genetic Algorithm (GA) (Akase and Okada [47]), Particle Swarm Optimization (PSO) (Li et al. [49]), or Simulated Annealing (SA) with Markov Chain Monte Carlo (MCMC) sampling such as Metropolis-Hastings (Merrell et a. [10], Liang et al. [50], Yu et al. [51], Yeh et al. [53], Kermani et al. [55], Qi et al. [65], and Kan and Kaufman [71]). The benefit of moving all objects simultaneously is that objects inherently have arrangement relations to other objects and moving more objects provides a more drastic alteration that could improve evaluation scores. On the other hand, finding more optimal evaluation scores becomes more challenging due

to the number of moving pieces.

Placing objects incrementally in the scene has the benefit of finding locally optimal locations for a single object at a time. The main drawback, however, is that the earlier placements have no concept of the later object placements and their evaluations. Some methods that place objects incrementally in the scene are using probabilistic sampling (Fisher et al. [11], Zhang et al. [52], Xu et al. [54], Liang et al. [57], Chang et al. [62], Fisher et al. [63], and Ma et al. [64]) or procedural placement (Kan and Kaufman [48], Kershavarzi et al. [61], Germer and Schwarz [72], and Kan et al. [73]). Procedural placement generally relies on prior layout knowledge, while probabilistic sampling methods are paired with data-driven methods requiring example models.

In Chapter 3, the proposed Jump Search algorithm explores possible placements for individual objects one at a time, which results in converging to a valid layout faster. To decide which object to place first, it relies on the dependencies among object types and chooses to place first these objects that depend on objects that are already part of the layout.

2.4 Data Driven Layout Synthesis

With the availability of large interior datasets, such as SUNCG [74] and 3D-FRONT [75], research into Data-Driven Layout Synthesis (DDLS) methods has gained much attention [46]. In DDLS, priors are learnt to account for the non-trivial nature of what makes a scene plausible or not [46].

Common in DDLS is to learn object location probability distributions, relative to existing objects in the scene (Fisher et al. [11], Liang et al. [50], Xu et al. [54], Kermani et al. [55], Kershavarzi et al. [61], Chang et al. [62], Xu et al. [76], and Zhang et al. [77]). Others use graph representations for binary pairwise object relations (Zhang et al. [52], Kermani et al. [55], Fu et al. [56], Wang et al. [58], Kershavarzi et al. [61], Chang et al. [62], Fu et al. [75], Xu et al. [76], Zhang et al. [77], and Ma et al. [78]).

With the advancements of Neural Network (NN), more elaborate methods of learning location probability distributions, such as Convolutional Neural

Network (CNN) in Wang et al. [67] and Ritchie et al. [69] or Variational Autoencoder (VAE) based methods in Li et al. [60], Zhang et al. [79], Para et at. [80], and Wei et al. [81], have been developed, outputting location probability maps. Using these location probability distributions, the furnishing placements search is then done either procedurally or by selecting the maximum likelihood (Kan and Kaufmann [71], Kan et al. [73], Ma et al. [78], Xie et al. [82]), or using a sampling approach (Merrell et al. [10], Fisher et al. [11], Liang et al. [50], Yu et al. [51], Kermani et al. [55], Qi et al. [65], Zhang et al. [77], and Raistrick et al. [83]) to iteratively assemble the layout arraignment.

Most recently, image generators have been able to create highly detailed images of buildings and interiors. These generators are either completely text-driven such as the most well-known generator DALL-E 3 [12] or can have some control using edges or input "skeletons" of the desired image [13]. A review of diffusion models can be found in Croitoru et al. [84] and Yang et al. [85]. In general, while producing impressive visuals of layouts, these image generation methods do not appear to be able to enforce the geometrical relations within the layout.

In both ACC and layout synthesis, digital models go through an evaluation. In this paper, I aim to bridge the research gap between learning priors in DDLS and rules in ACC as, to my knowledge, ACC rule learning from examples has not been explored. Learned probability distributions used in layout synthesis are not used as strict adherence rules, as in ACC, but rather plausibility and aesthetic approximation functions that are aimed to be optimized through layout search. Put another way, the learned probabilities are functions that evaluate how likely a configuration is to appear in the layouts trained from.

The proposed rule learning method is similar to the graph approach, however, my relationship ranges are expanded through examples as opposed to being hard-coded binary relationships. Additionally, the proposed rule learning method supports more expressive rules through the use of existential clauses to determine the number of objects of a type that must pass as opposed to just one-to-one graph relationships. This gives them a structure more closely

aligned with ACC rules in the form of the RuleDSL.

2.5 BIM Reasoning Frameworks

BIMServer [86] is a well known open source IFC model server, actively maintained and regularly updated. It is an IFC model repository that is capable of performing version control and enforcing access-control rules, checking user credentials against their authority to access the models they request. The associated bimviews web app enables users to view the models, and additional functionalities can be created by developing BIMServer plugins, such as the example Model Compare and Query Engine Plugins. There have been attempts at developing model-checking plugins for BIMServer, but a complete solution has yet to be developed.

There are a few additional examples centred around IFC models such as BIMCloud [87] that focus on the social interaction around the model, and CloudServerBIM [88] that extended the BIMServer functionality to include web-based services. By using IFC as the model schema, the above methods face the IFC related development challenges, such as the model inconsistencies due to modification. BIM 360 [89] and BIMCloud [90] are some examples of Cloud-based BIM model servers that do not use IFC but therefore are linked more directly to a single design software.

Our work shares much of the motivation and some implementation decisions with three recent publications that have proposed, or used, MongoDB for implementing repositories of BIM models. Ma and Sacks [91] use MongoDB to store IFC files, motivated by the need for a dynamic data schema: their system uses the IFC schema but also enables dynamic addition of user-defined properties. The authors have demonstrated the use of their system for the task of reconstructing reinforced concrete beam models from earthquake-damaged buildings. Lin et al. [92] focus on the experience of end users querying massive IFC models for data retrieval and develop a tool that translates natural-language queries to map IFC entities. On the back-end, the tool uses MongoDB to store the IFC data, with Objects, Relations, Types, Geometry,

and Relational Objects. This is closely related to my implementation however, I use the platform more broadly, allowing for services and applications to modify the model, as opposed to theirs that only adds information to the model for query and retrieval purposes. Jiao et al. [93] propose a "logically centralised but physically heterogeneous" database where non-geometry data is stored in a relational database and geometric data in a MongoDB database. From a Revit file [26], the geometry elements are classified, grouped and stored in HOOPS files collectively, with the non-geometrical information being linked via a Globally Unique Identifier (GUID) and stored in the relational database. Our solution, on the other hand, stores the information collectively and is more intuitive to parse and thus modify.

The most recent and most closely related to my work project is the BIMRL Simplified Schema (BIMRLSS) from Solihin et al. [9]. They argue for a simple model representation to promote accessibility, and believe that at its core, a model should store the "objects, types, their properties, relationships, and their final geometries". Additionally, they cite model-checking as the key motivator to their approach and use a rule-language method for deriving the rules; my rules are being tested on interior design, while their rules relate closely to building paths and lines of sight. They also consider the benefits of Level-of-Detail (LOD) for the stored model; however, they focus on the accuracy of the geometry representations, where I am more interested in the logical organization of the building model and its constituent elements. BIMRLSS and BIM-kit differ in two important ways. First, BIMRLSS adopts a traditional relational storage model for BIM data, i.e., non-geometry data, as opposed to my document-centric MongoDB. Second, BIMRLSS, similar to BIMServer, proposes that additional services should be treated as plugins rather than independent services, which requires a centralized development model of the overall toolkit, and is less extendible than my API-centric approach.

The majority of the previous research does not examine how each of the various design activities uses and manipulates the data in the building model. In this work, I look specifically at the data required for each activity and I propose APIs as the mechanism for accessing the various building-model

elements in a semantically consistent manner.

2.6 Rule Domain Specific Language (RuleDSL)

Central to this thesis is the previously developed RuleDSL outlined in detail in Sydora and Stroulia [1]. In short, DSLs are higher-level programming languages that encompass terms relevant to specific tasks making them easier for non-programmers to interpret and learn, in contrast to general-purpose languages that can take years to master. This is largely achieved through the use of domain terms, types, and logical operations relevant to building and interior design. The language can be broken up into two components that determine its expressiveness: Vocabulary and Grammar.

2.6.1 Vocabulary

The vocabulary of a language refers to the terms used in the language to convey meaning. The RuleDSL vocabulary can be categorized into three groups: (1) object **Type** classifications, (2) the **Property** functions (single object) and **Relation** functions (pairwise), and (3) the logical **Operation** functions.

Types: Central to BIM is the use of object types to classify objects. Industry Foundation Classes (IFC) [94] is the most commonly used type classification, which RuleDSLs uses as the basis for its types list. Types in the RuleDSL are organized in a tree structure such that each type must have a single parent type, up to the *Root* type. RuleDSL supports new types that can be extended downward. For instance, *table* can be extended down to have child types *coffee table* and *dining table*.

Properties and Relations: A vital part of the RuleDSLs vocabulary is the definition of commonly used object properties and pairwise relations, such as *Width(Object)*, *Distance(Object1, Object2)*, and *Facing(Object1, Object2)*, which can be used directly in the language. This is done by associating a programmed function to each property and relation term. When the rule

is compiled, the property or relation term is replaced with the associated function. When the rule executes, the function is called and the value is returned which can then subsequently be checked in the rule expression.

Properties are functions on a single object, while relations are on object pairs. Relation functions on pairs of objects can also be order-dependent or independent. For instance, *Distance(Object1, Object2)* is order independent as the computed value is the same as *Distance(Object2, Object1)*. Functions such as *Facing(Object1, Object2)* are order dependent as the computed value is not always equal to *Facing(Object2, Object1)*.

Operations: The operations are what make up the basic logic and comparisons of the rule language. Much like any general programming logic, these are functions that take in two (or more) values and return a single value between 0 and 1. The operations include the logical operators (AND/OR), the existence clause (ALL/ANY), and the comparison operators ($<$, $\leq$, $>$, $\geq$, $=$, and $!=$). In the RuleDSL, however, the logic is considered on numerical values from 0 to 1 rather than simple 0 or 1 binary. Therefore, the logical operator AND is a function that takes a list of values between 0 and 1 and returns the mean value of the list, while the OR operator returns the maximum value of the list. The existence clause also uses the same logic; the ALL clause is the same as AND, while the ANY clause is the same as OR.

Derived from Merrel et al.[10], a piece-wise check function is used that scales the comparison operators' results to a value between 0 and 1. The key benefit is that comparison results are scaled from 0 to 1 exclusive is a fail and only a 1 is a pass, rather than strictly binary pass/fail results. This has major implications when doing object placement searches.

2.6.2 Grammar

Given the vocabulary, the grammar or syntax of the language is the structure by which the language terms are ordered. This primarily impacts the readability of the language and how the language is parsed. The RuleDSL grammar is comprised of three main components: (1) the property and relation functions

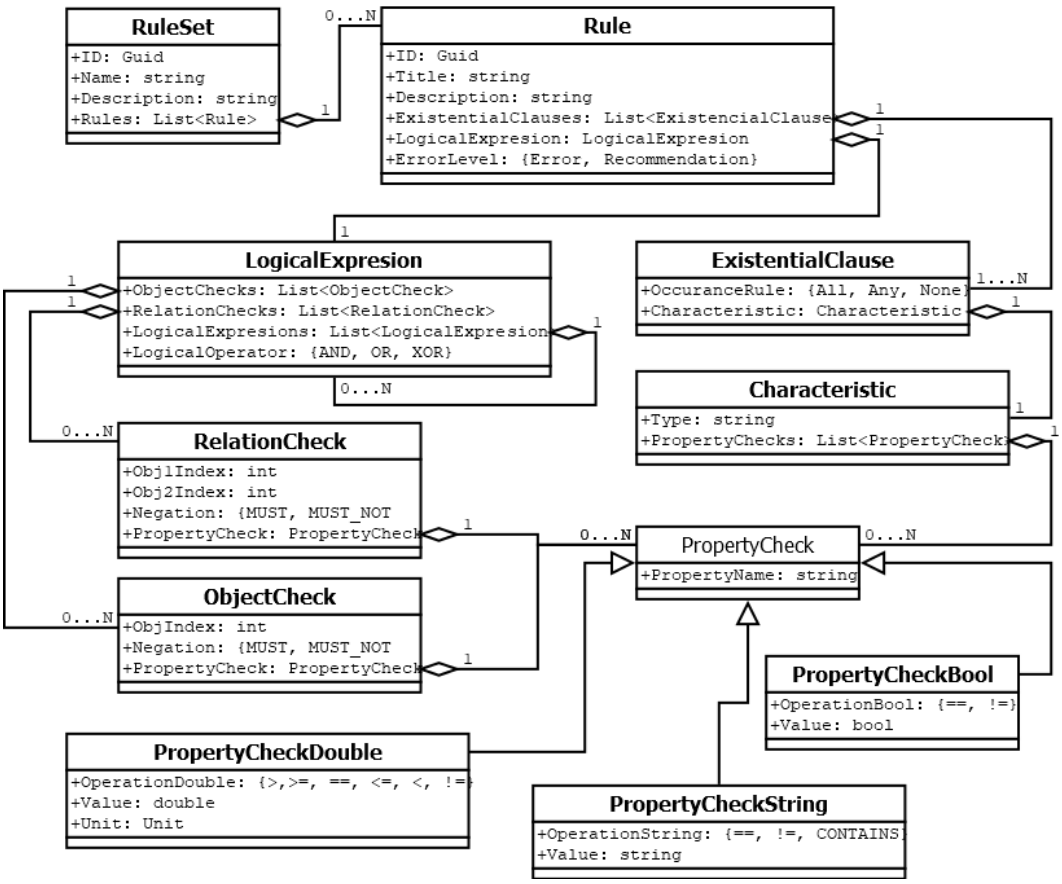


Figure 2.1: Rule structure in the Unified Modeling Language (UML).

or **Property/Relation Checks**, (2) the **Logical Expression**, and (3) the object type filter and pass requirement or together the **Existential Clause**. The RuleDSL structure can be seen in Figure 2.1.

Property/Relation Checks: Relation checks are how the value associated with a relation function is compared against a check value. For example, one might check if the relation *Distance* between objects A and B is greater than a value x. We write this as $Distance(A,B) > x$. In this case, the function for *Distance* is called with objects A and B as input and the value is returned, which is checked against the value x using the *GREATER THAN* function. All relation checks result in a value of 1 if passed or less than 1 but greater than 0 if failed. As described in the vocabulary for comparison operations, the farther the returned function value is from the check value, the closer to 0 the

check value will be.

Logical Expression: The RuleDSL logically joins property and relation checks in the logical expression. For instance, two objects might be required to have a *Distance* greater than a value *AND* a *Facing* value equal to another value. The logical operations are between property and relation checks and nested logical expressions. Collectively, the logic expression of the rule results in an instance of two (or more) objects either passing (score of 1) or failing (greater than 0 but less than 1).

Existential Clause: The RuleDSL uses types to filter for only the object instances relevant to a rule. The filtered objects are then tabulated into rule instances with columns for each existential clause type and a row representing a unique instance of objects from each existential clause type. As an example, if a rule checked the relationships among chairs against tables, each column would be a type (i.e. chairs, tables), and each row would be an instance of the two types, (i.e. chair1 with table1, chair2 with table1, etc.). Each of these instances is passed into the logical expression function, returning a 0 to 1 instance result value.

It is common for rules to require either all instances or only one instance to pass to be satisfied. Therefore, after all instance results are calculated, the *ALL* and *ANY* clause functions check the number of instances for a type that has passed to determine if the rule passed.

2.6.3 Layout Quality Measure

Layout evaluation (or more generally model checking) is the process of evaluating the layout design against the design rules. This process takes the rules in the RuleDSL form, compiles them to executable code, and runs the code with the layout model as input. Rules can have different levels of importance; they can either be principles (hard constraints) or guidelines/preferences (soft constraints). When evaluating, layouts are compared first against principle-level rules. Ties are broken by guideline-level rules, and if still tied, the layouts

are compared against preference-level rules. The final quality of a layout is represented as a percentage of the number of rules that the layout passes over the total number of applicable rules. For this work, I adopt the percentage of rules that the design complies with as a measure of layout quality:

$$Score = \sum_{i=1}^{n} Rule[i].eval/n * 100\% \tag{2.1}$$

where n is the total number of rules and $Rule[i].eval$ is the rule score in the range of 0 and 1. In calculating this score, only soft constraints are considered since a design is not valid if it fails to pass the principle-level rules.

Chapter 3

Automated Layout Generation

In this chapter, I present a novel greedy continuous RuleDSL-based layout generation search algorithm, Jump Search (JS) which is motivated by the need for a continuous non-grid-based search as was the state-of-the-art in terms of RuleDSL-based furniture placement search. The greedy continuous approach of JS is compared against a grid-based search, Grid Search (GS), and a more conservative continuous approach, Simulated Annealing (SA) search algorithm. The algorithms are compared on different room scenarios in terms of their efficiency in finding quality layouts under a specified number of search iterations. The findings of this chapter suggest the JS algorithm is more efficient at finding higher quality layouts in fewer iterations.

This chapter contains the work presented in the conference publication: **Christoph Sydora** and Eleni Stroulia, "Comparative Analysis of Room Generation Methods Using Rule Language-Based Evaluation in BIM," In Proceedings of European Conference on Computing in Construction (EC3) and International CIB W78 Conference, 2023. [2]

Abstract: Automated layout generation can improve interior designs by suggesting constraint-compliant design alternatives. Persistent issues relate to intuitive and explainable constraint formulation and efficiency of layout search that result in highly compliant, diverse alternatives. This chapter proposes a Domain Specific Language for rule representation and evaluation, along with a continuous layout-configuration search, Jump Search. The proposed method is compared against a previous grid-based layout search method and a Sim-

ulated Annealing approach, under rule-based evaluation budgets for different room-type scenarios. Our experimental results demonstrate that Jump Search is able to generate higher-quality layouts more efficiently, while also exploring a larger variety of layouts.

3.1 Introduction

Automation offers the potential for significant improvements in building design and construction, from eliminating repetitive routine tasks, such as stud placement, to automatically generating high-quality building designs. The last stage in design is the arrangement of furnishings in the available interior spaces of the building, a task that arguably has the largest impact on the occupants, as a poor arrangement can lead to inaccessible objects, have negative ergonomic effects on occupants, and generally make a space unpleasant that could otherwise be better utilized. Because of the variety of furnishing types and the complexity of the relevant ergonomic constraints and aesthetic preferences, the potential placement options and arrangement configurations are vast. Therefore, automatic layout design could greatly benefit individuals who might lack the skills and know-how to design practical and functionally efficient layouts.

3D digital representations of layouts can provide visualizations of design options prior to design commitment; in fact, it is a tool often used by vendors and decorators. Building Information Modeling (BIM), an emerging tool for building digitization, integrates specifications of the building architecture with 3D object models annotated with conceptual type, property, and relationship data. In this work, we adopt a BIM-based methodology to ensure that our layout representations fit in the broader building design life cycle and evaluate our algorithms by implementing and evaluating them in the context of BIMkit [3] (Chapter 6), a toolkit that supports a broad range of BIM management use cases, including model storage, model checking against design rules, visual representation and editing, and automated design generation.

Design rules are computational representations of the ergonomic and aes-

thetic constraints and guidelines as well as user preferences applicable to the placement of the objects inside a space. The evaluation of these rules supports the assessment of the quality of an existing layout, and, embedded in a design-space search algorithm, it enables the automatic generation of valid layouts. An important question then arises regarding the language in which these rules should be represented. Earlier work, such as Merrel et al. [10] and Yu et al. [51], proposed rules as geometric formulas, based on shapes, angles, and distance, and many subsequent approaches since have followed suit ([47]–[50], [52]). Such representations are not easy to explain to occupants who tend to think in terms of furnishings, and, for the same reason, they are also difficult for interior design experts to express. This is why, in previous work, a Domain Specific Language (DSL) for rules was designed, that includes furnishing types, properties, and relationships as elements.

Automated design generation is typically formulated as a systematic search through a design space, toward a high-quality design. Numerous such algorithms have been proposed, reporting different metrics on their performance [46]; however, comparing them against each other is quite challenging. Runtime metrics are difficult to standardize since they depend on the underlying hardware, and the design-generation problem is formulated differently across different algorithms, that rely on different representations of layouts and rules. A possible metric that could serve as a standard for comparison is the number of rule evaluations during the search.

In our previous work, we designed a simple generate-and-test design method, where the list of possible locations and orientations for object placement was pre-generated based on a grid defined relative to the walls of the space [1]. At each iteration, one object was placed at the location and orientation producing the current highest evaluation score, then repeated with the remaining objects. Furnishing object placements were evaluated based on the rule-language evaluation score (as a proof of concept of the rule language as a viable evaluation method). From our experiments, we were able to show that the approach was able to re-create input kitchen layouts by removing all objects and placing them back in following the input design rules for kitchens (although the input

kitchens were not implicitly created with the same rules). Then, interpreting the rules from Merrell et al. [10] into their rule language and running living room experiments, we demonstrated that our algorithm could successfully create functional (relative to the design rules) room designs.

In this chapter, we propose a continuous space search method, the Jump Search method. At a high level, the new method places one object at a time, starting with objects subject to more dependency based on the rules. The furnishing object potential next placements are sampled from an incrementally decreasing probability distribution, each evaluated against the rule language evaluator, then greedily selects the highest scoring location and orientation for the movement and repeats until the number of moves is exhausted.

We have evaluated the two methods and a simple Simulated Annealing (SA) method in terms of, first, how efficiently they find layouts of desired quality, and second, how good the layouts they produce are, given the same "budget" of rule checks. Furthermore, we have evaluated their relative performance under different room types, which have different layout rules and furnishing types.

The contribution of this chapter is a new continuous space search algorithm, Jump Search (JS). We have demonstrated the good performance of this novel algorithm, and its effectiveness in delivering high-quality layouts in many different problem scenarios, by comparing it against a previous grid-based method, Grid Search (GS), and a basic continuous space search algorithm based on Simulated Annealing (SA).

The remainder of the chapter is organized as follows. Section 3.2 will describe the previous GS generative design algorithm followed by our new JS generative design algorithm and the SA method. We outline the experimental design for a comparative evaluation of the three methods in Section 3.3 followed by a summary of results in Section 3.4 and discussion points in Section 3.5. Finally, we conclude with a summary of our contributions and our future plans in Section 3.6. Related work for this Chapter can be found in Chapter 2, Section 2.3.

3.2 The Search Algorithms

In this section, we describe the three automatic layout generation search algorithms. Each algorithm uses the same room scenario setup (see Appendix Section A.1). All three algorithms rely on placing a single object at a time which makes the resulting generated layout depend heavily on the order by which the objects are placed. Therefore, all methods will use the same object type placement order which is described at the end of this section in Section 3.2.4.

3.2.1 Grid Search (GS)

Our previous generative design method relies on a grid, defined based on the initial room walls. For the grid creation, lines are created outward parallel to each of the wall lines at regular intervals; the intersections among the grid lines define the possible locations where objects can be placed. This allows control of the spacing between furnishing objects and between the furnishing and wall objects, at the expense of more limited control of the number of points.

In this algorithm, for which the pseudocode is provided in Sydora and Stroulia [1], objects can only be placed on the grid-intersection points, in four (or more) possible orientations. For each object, one at a time, the location and orientation combination with the highest resulting evaluation score is selected as the decision action for that iteration. For subsequent object placements, possible locations are removed if they are under an existing object. The process is repeated with the remaining objects until either all objects have been placed or the highest possible evaluation score is reached.

In this chapter, object placement ordering is introduced based on the rules, which was predetermined in the previous work. The significance of order placement in these methods will be explored in future work.

3.2.2 Jump Search (JS)

The intuition behind the new layout-generation algorithm is to shift one furnishing object at a time into a position that adheres to the relevant layout

design rules. Each object begins near the center of the floor and, through iteratively smaller moves, reaches its final location either when a valid location is found or when a maximum number of moves has occurred. The process repeats until each of the selected objects has been placed in the layout.

Algorithm 1: The Jump Search Algorithm.

```
1  Function GenerateSampleLayout(Layout,Objects,Rules):
       Input:
           Layout - The initial layout
           Objects - The to-be-added objects
           Rules - The rules
           t_MAX - The number of iterations
           N - The the number of moves per iteration
           M - The number of orientations per move
           s_0 - The initial sample radius standard deviation
       Output:
           Layout - The initial layout with the added objects
2      Objects ←ReorderByRuleTypes(Objects, Layout, Rules)
3      Orientations ← GetOrientations(M)
4      foreach obj ∈ Objects do
5          Moves ←GetMoves(1, Layout.Center, 1)
6          xy_Top, o_Top ← Moves[0], Orientations[0]
7          score_Top ←Eval(obj, xy_Top, o_Top, Layout, Rules)
8          t ← 0
9          s ← s_0
10         while t < t_MAX do
11             xy_Pre, o_Pre, score_Pre ← xy_Top, o_Top, score_Top
12             Moves ←GetMoves(N, xy_Top, s)
13             foreach xy ∈ Moves do
14                 foreach o ∈ Orientations do
15                     score ←Eval(obj, xy, o, Layout, Rules)
16                     if score > score_Top then
17                         xy_Top, o_Top, score_Top ← xy, o, score
18                     end
19                 end
20             end
21             if SA == TRUE AND score_Top > score_Pre then
22                 P = 1 − e^((score_Pre − score_Top)/(1−t/t_MAX)) if
                    P < Random.Sample(0, 1) then
23                     xy_Top, o_Top, score_Top ← xy_Pre, o_Pre, score_Pre
24                 end
25             end
26             t ← t + 1
27             s ← s_0 − t ∗ s_0/t_MAX
28             if score_Top == Rules.Count then
29                 break
30             end
31         end
32         Layout ← PutObject(Layout, obj, xy_Top, o_Top)
33     end
```

The JS algorithm is shown in pseudo-code in Algorithm 1. It takes as input the empty room (*Layout*), the applicable ruleset (*Rules*), the list of to-be-placed furnishing objects (*Objects*), and the following parameters: t_{MAX}: the number of iterations exploring different placements before an object is placed in its final position; N: the number of moves in a single iteration; M: the number of orientations attempted for each object placement attempt (The orientations are in the order of π/M Radians); and s_0: the initial standard deviation for sampling movements in the first iteration.

Looking at the rule dependencies, the process determines the order of object types that must be placed in the room, given the existing types already in the room (Wall/Floors/etc.) (Line 2). All possible orientations are calculated based on the input M value (Line 3). This object type order initialization phase is the same for all three algorithms compared in this chapter (See Section 3.2.4).

Starting with the first object in the queue, the object is placed in the center of the room, with a slight offset randomly generated from a normal distribution with a mean value of 0 and standard deviation of $1m$ (Line 5). Then, N move locations are sampled in each iteration, each with an X and Y move value sampled from a normal distribution of mean $= 0$ and iteratively decreasing standard deviation $= s$ (Line 12), initially set to s_0 (Line 9). If the locations generated are outside the room floor, re-sampling occurs.

The potential object placements in each of the N moves and M orientations and compared against each other (Lines 13-20). Because only one object moves at a time, the layout evaluation process is only required to re-evaluate the subset of rules that relate to the moving object, and not the full ruleset, which contributes to the JS algorithm's performance.

The move with the best layout score is selected for the next placement (using the layout evaluation described earlier) (Lines 16-18). t is then increased which reduces the next iteration move amount variation, s (Lines 26-27).

Once the object has been placed at a location with a score equal to the maximum possible score for that object (which is equal to the number of rules relating to that object, i.e., all rules passed) or the maximum number of moves

is completed (t_{MAX} has been reached), the current object is locked into its final location (Line 32). The next object in the queue is selected and the process repeats until the object queue is empty.

3.2.3 Simulated Annealing (SA)

A simpler variant of the above algorithm, based on an implementation of the well-known Simulated Annealing algorithm (Kirkpatrick et al. [95]), was implemented as a competitor. It uses nearly the same sequence as the above, however, rather than testing N moves (Algorithm 1, Line 12), it only attempts one move at each iteration (taking the best of the M orientations). It evaluates the move and if the move has a higher evaluation score, it uses the following formula to determine the move acceptance probability (Algorithm 1, Lines 21-25):

$$P = 1 - e^{(Q_{previous} - Q_{current})/(1 - t/t_{MAX}))} \tag{3.1}$$

As the iteration count increases, $t \to t_{MAX}$, the likelihood of accepting the move increases, while earlier on, exploration is favoured if the move does not drastically improve the current evaluation score.

3.2.4 Placement Order

An important factor in the search methods is the placement order. Early-placed objects greatly impact the downstream placement options, dictating how well the final generation can score. Through the dependencies among the various object types as captured in the design rules (the types in the rules existential clauses), a hierarchy of object types is implied. The least constrained object type (i.e., the object type that is placed without consideration for the placement of other object types) is at the root of the hierarchy. The object types that depend on the root type through rules are children of the root, and so on, until the leaves of the hierarchy that correspond to the object types with the most dependencies to other object types that must have already been placed. This hierarchy guides the order in which the various objects are placed

in the space, similar to Kan et al. [73].

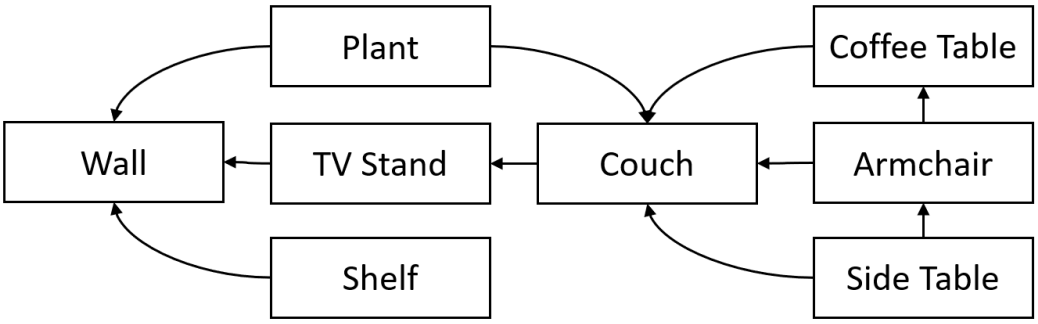


Figure 3.1: Example type dependency graph for a living room. A connection among object types by two objects both are referenced in a shared rule (the types in a rule's existential clauses).

Consider for example the object type hierarchy of Figure 3.1. Its root is the *wall* that is already placed in the model and does not depend on any other furnishings. Children of the *wall* are the *shelf*, the *TV stand*, and the *plant* that are typically placed against a *wall*. The *couch* is also a child of the *wall* since it is typically placed parallel to a*wall*, but also a child of the *TV stand*, since the *couch* must face the *TV stand*. According to this hierarchy, the resulting type order is *Wall*, *TV Stand*, *Couch*, *Plant*, *Side table*, *Coffee table*, and *Armchair*.

3.3 Experimental Design

To comparatively evaluate the three above algorithms, we focus on three questions of interest.

1. How long does it take each algorithm to deliver a layout of a desired quality?

2. How does each algorithm perform in different types of rooms, with different functionalities, different sets of layout rules, and different types of furnishings?

3. How does each algorithm perform at different levels of scenario density?

Performance and Quality Measures: To comparatively evaluate the performance of the three algorithms, we run all three on our experimental room scenarios (described in detail in Section A.1) with the same "budget" of rule checks and we compare the quality of the solutions produced by each of the three algorithms for the same number of checks. For this work, we adopt the percentage of rules that the design complies with as a measure of layout quality (described in Section 2.6.3). The intuition is that the algorithm that produces layouts that meet all the applicable rules with the smallest number of rule checks is best. Thus, embedded in each algorithm implementation is a counter for the number of times a rule-evaluation check is invoked and all algorithms are invoked with a check budget as a parameter.

Because the number of evaluation checks is dependent on the room shape for GS, which is difficult to determine beforehand, the evaluation check budget is calculated for the GS (which is based on the number of grid points) method first. The parameters that result in closely matching evaluation check budget for the JS and SA methods are then calculated. For the JS the t_{MAX} and N are both set to the square root of the number of location points created in the GS grid, while for the SA method, t_{MAX} is set to the number of location points as N is set to 1.

3.4 Results

Figures 3.2 to 3.10 report the performance of the three algorithms on each of the room scenarios. For density reference, Kitchen 2 adds more objects and rules than Kitchen 1. Living Rooms 1 and 2 are L-shaped rooms while Living Rooms 3 and 4 are Rectangular. Living Rooms 2 and 4 have the same rules but additional objects to Living Rooms 1 and 3.

Each experiment was run 30 times for each generation method and each rule check budget. Figure 3.11 shows some of the outputs from the JS method.

For a runtime benchmark: on an Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz processor with 16.0 GB RAM, 10 iterations with 5 moves and 4 orientations for the rectangular living room scenario (14 objects total: 6 initial ob-

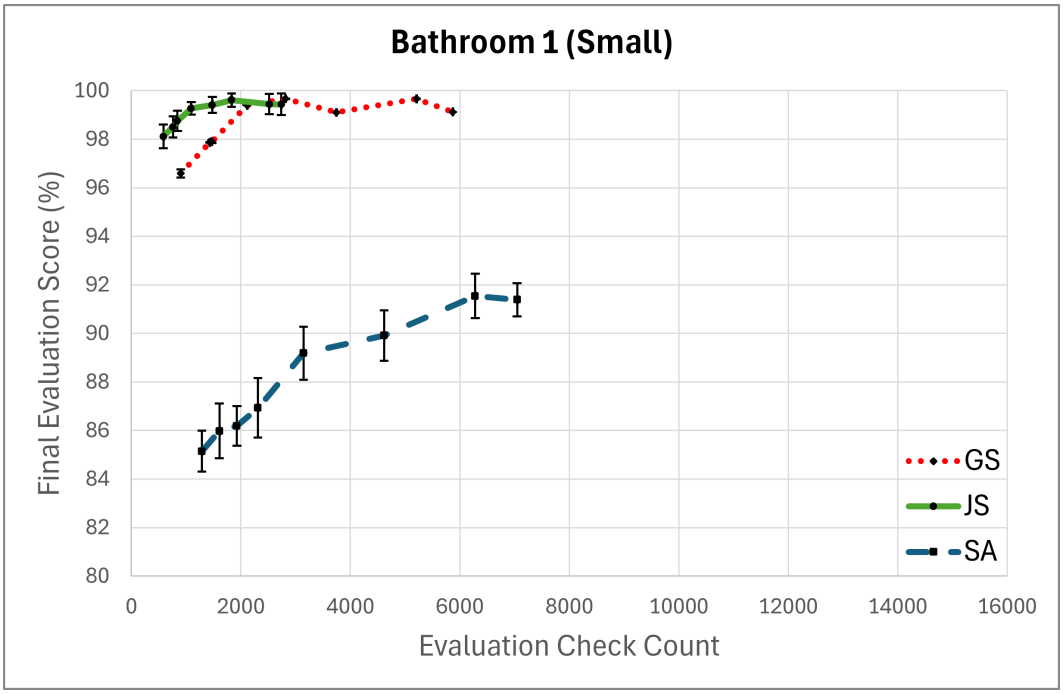


Figure 3.2: Search method comparison for Small Bathroom 1 scenario.

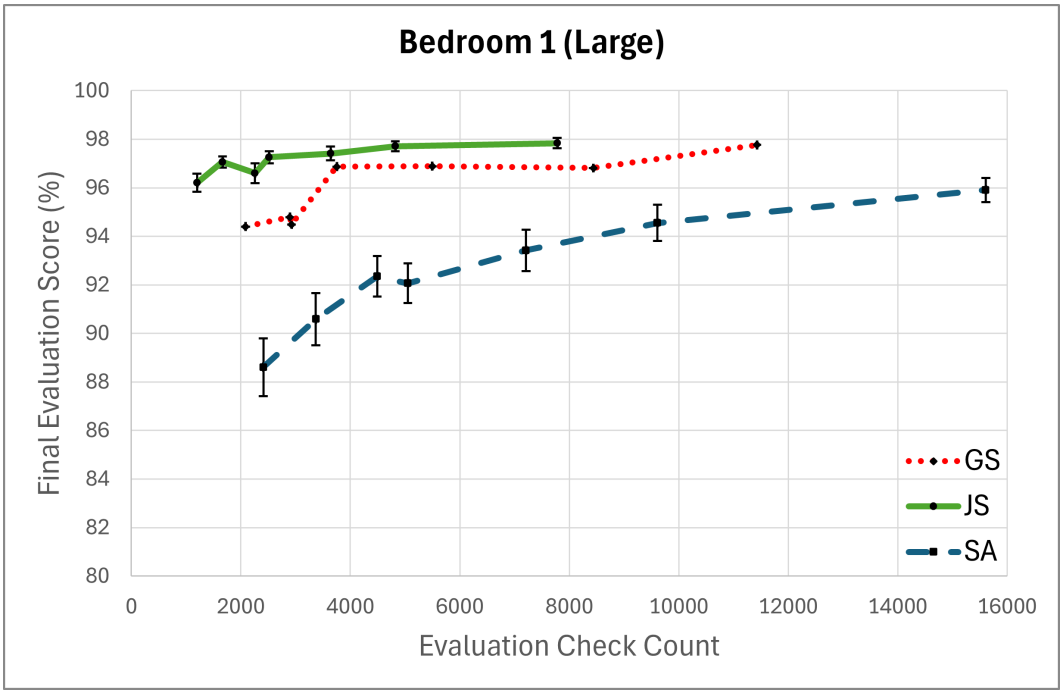


Figure 3.3: Search method comparison for Large Bedroom 1 scenario.

jects and 8 added furnishing objects) resulted in a compliance score of 96.78%, required 1616 evaluation checks (not all rules checked each evaluation), and a

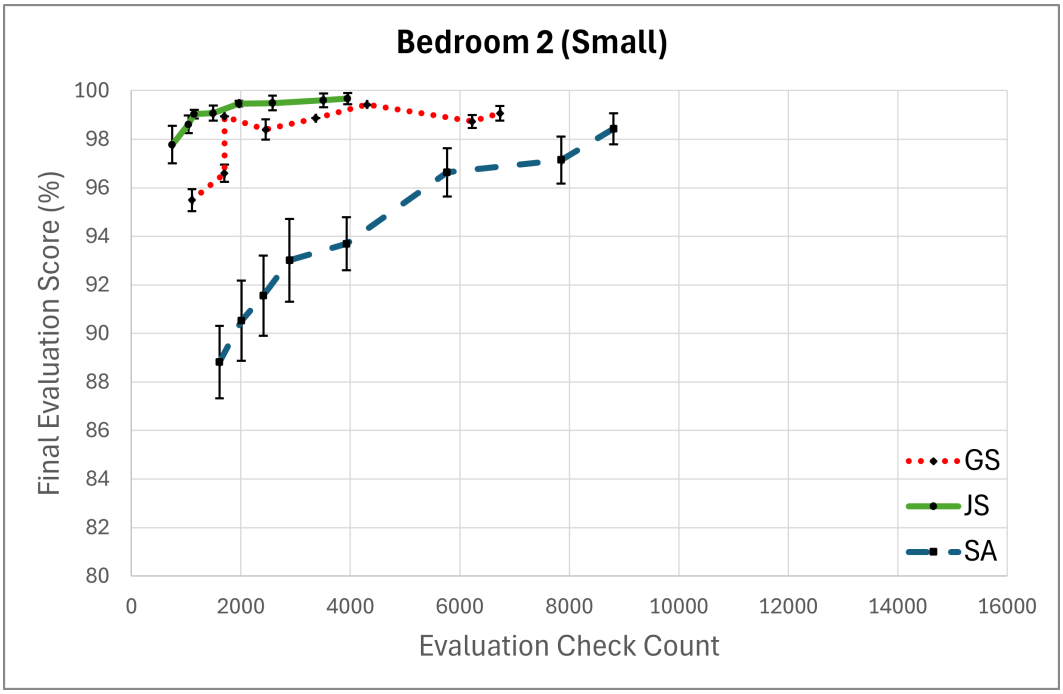


Figure 3.4: Search method comparison for Small Bedroom 2 scenario.

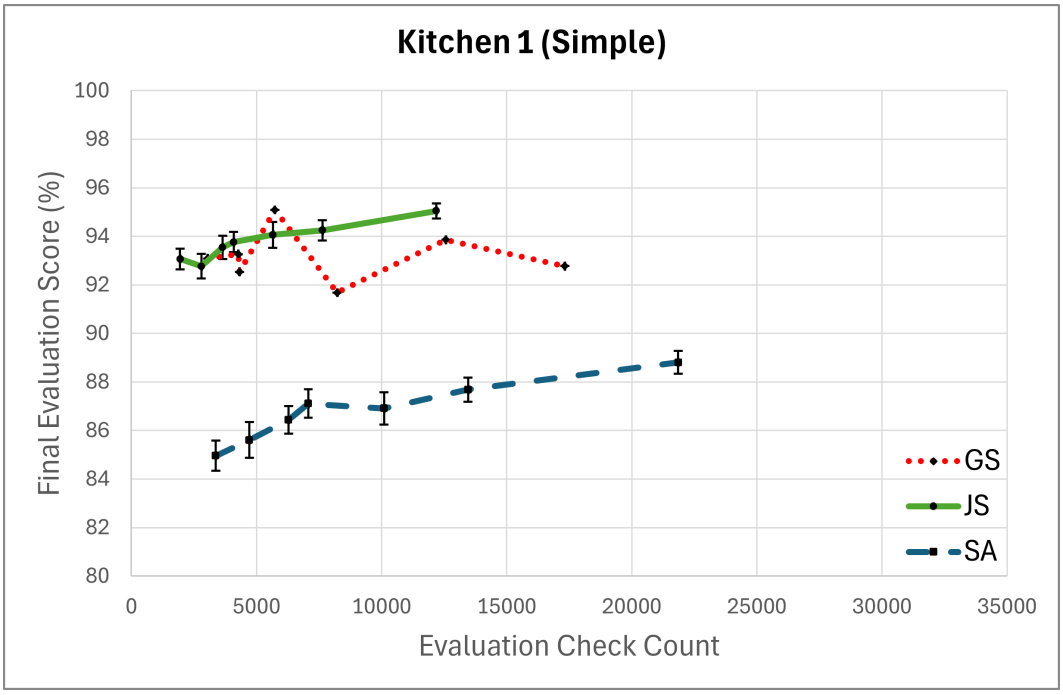


Figure 3.5: Search method comparison for Simple Kitchen 1 scenario.

runtime of 23.3083559 seconds.

Let us now revisit the three evaluation questions above, in light of the

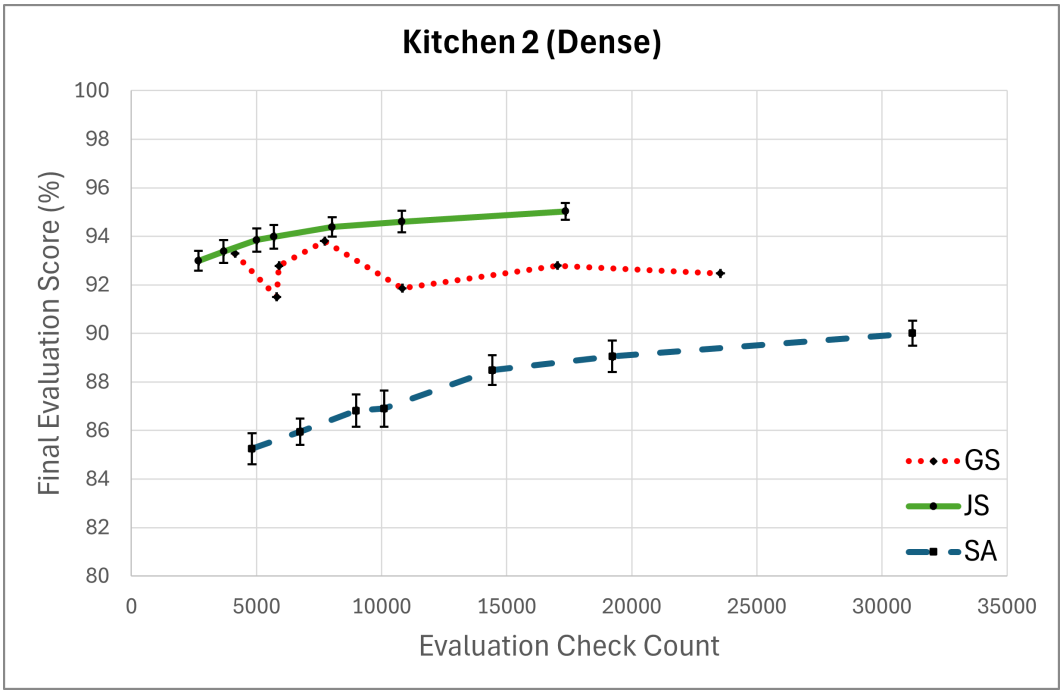


Figure 3.6: Search method comparison for Dense Kitchen 2 scenario.

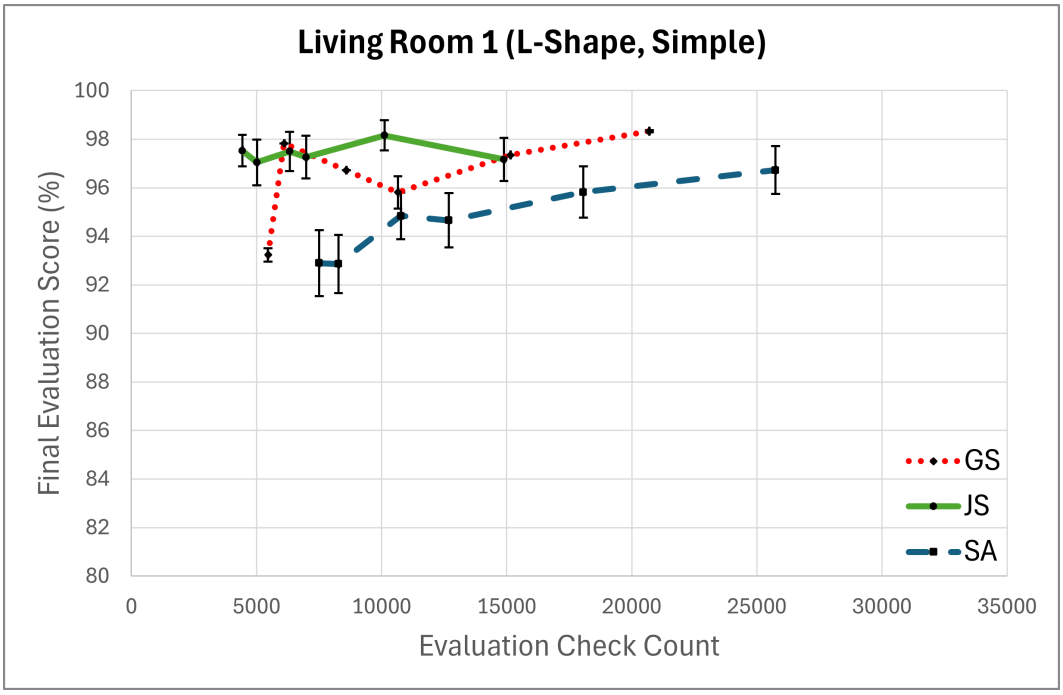


Figure 3.7: Search method comparison for Simple L-shaped Living Room 1 scenario.

experimental results.

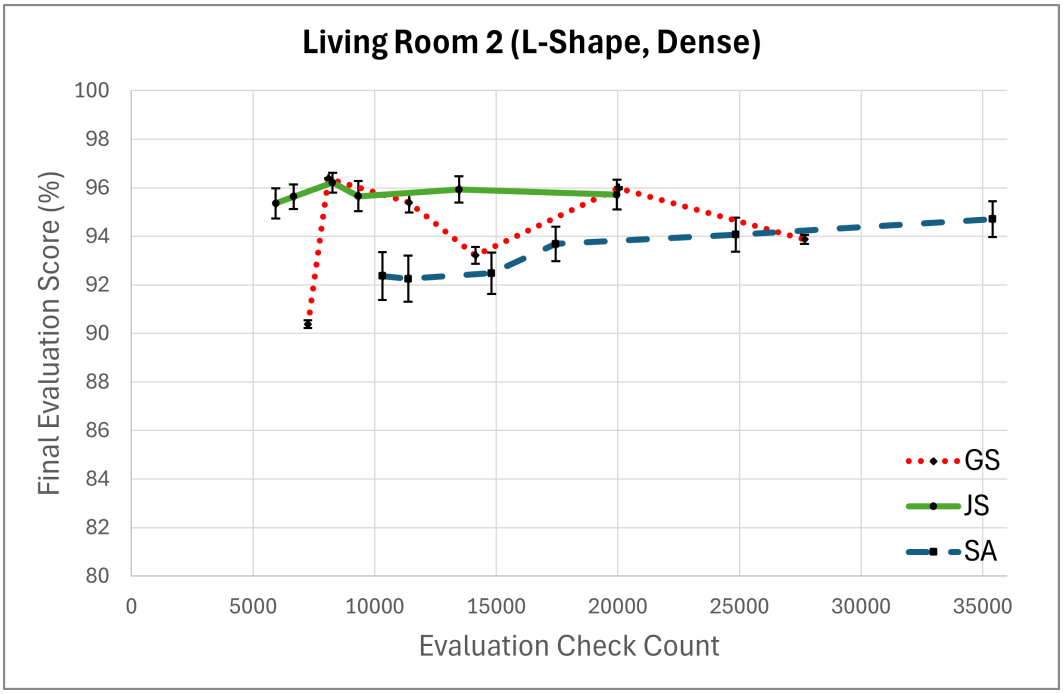


Figure 3.8: Search method comparison for Dense L-shaped Living Room 2 scenario.

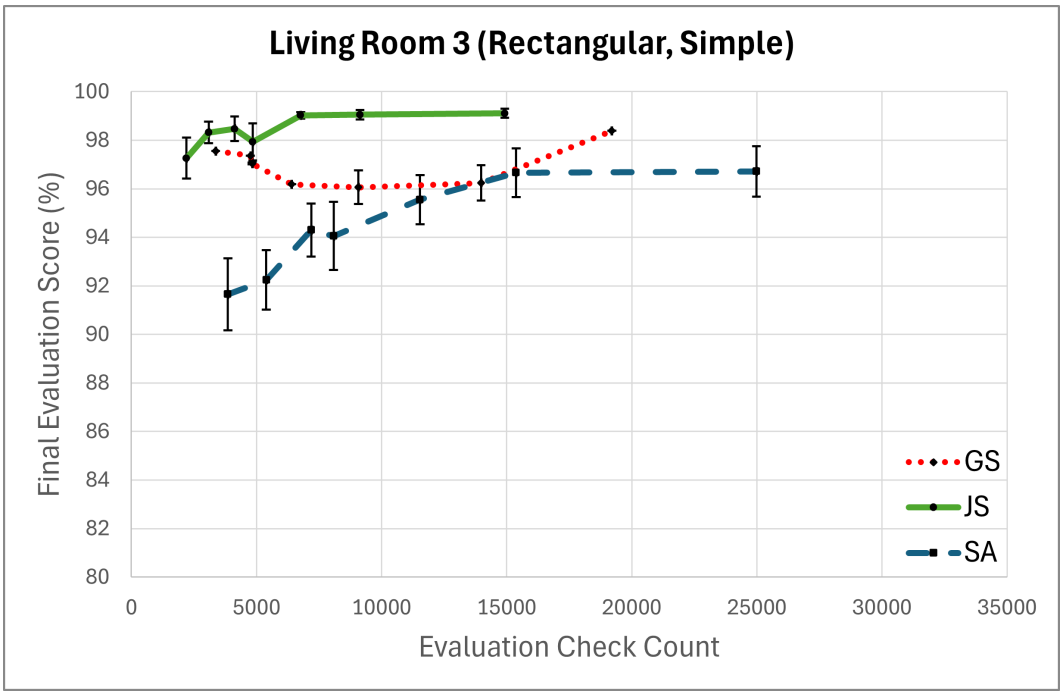


Figure 3.9: Search method comparison for Simple Rectangular Living Room 3 scenario.

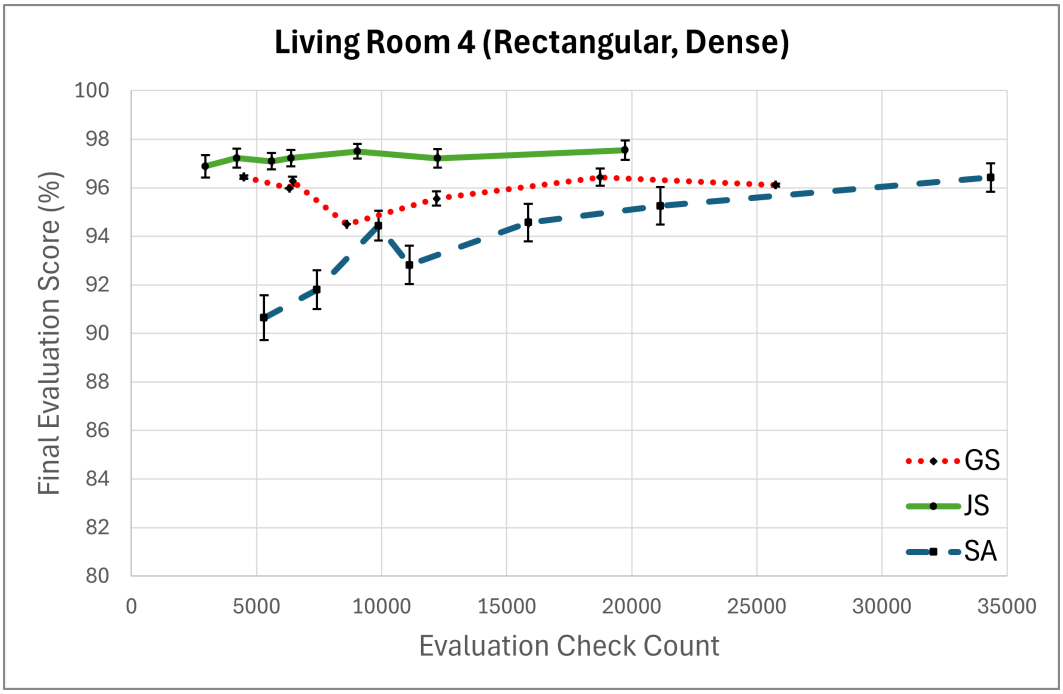


Figure 3.10: Search method comparison for Dense Rectangular Living Room 4 scenario.

Q: ***How long does it take each algorithm to deliver a layout of a desired quality?***

The answer to this is largely dependent on the room size, type, objects, and rules, however, our experiments show some trends for different room scenarios. The most prominent difference between the grid-based search method (GS) and the continuous search methods (JS and SA) is notable consistency in the correlation between the check budget and final evaluation score. As evident by the jagged lines, the GS method suffers from heavy reliance on the grid spacing parameter, thus making the final evaluation less predictable and more dependent on parameter tuning.

All three methods have a general trend of increasing the number of evaluations increases final layout quality. The JS method's final layout quality is equal to or higher than both GS and SA for nearly all evaluation budgets. The continuous search methods follow a more predictable curve, while Grid-based methods are less predictable.

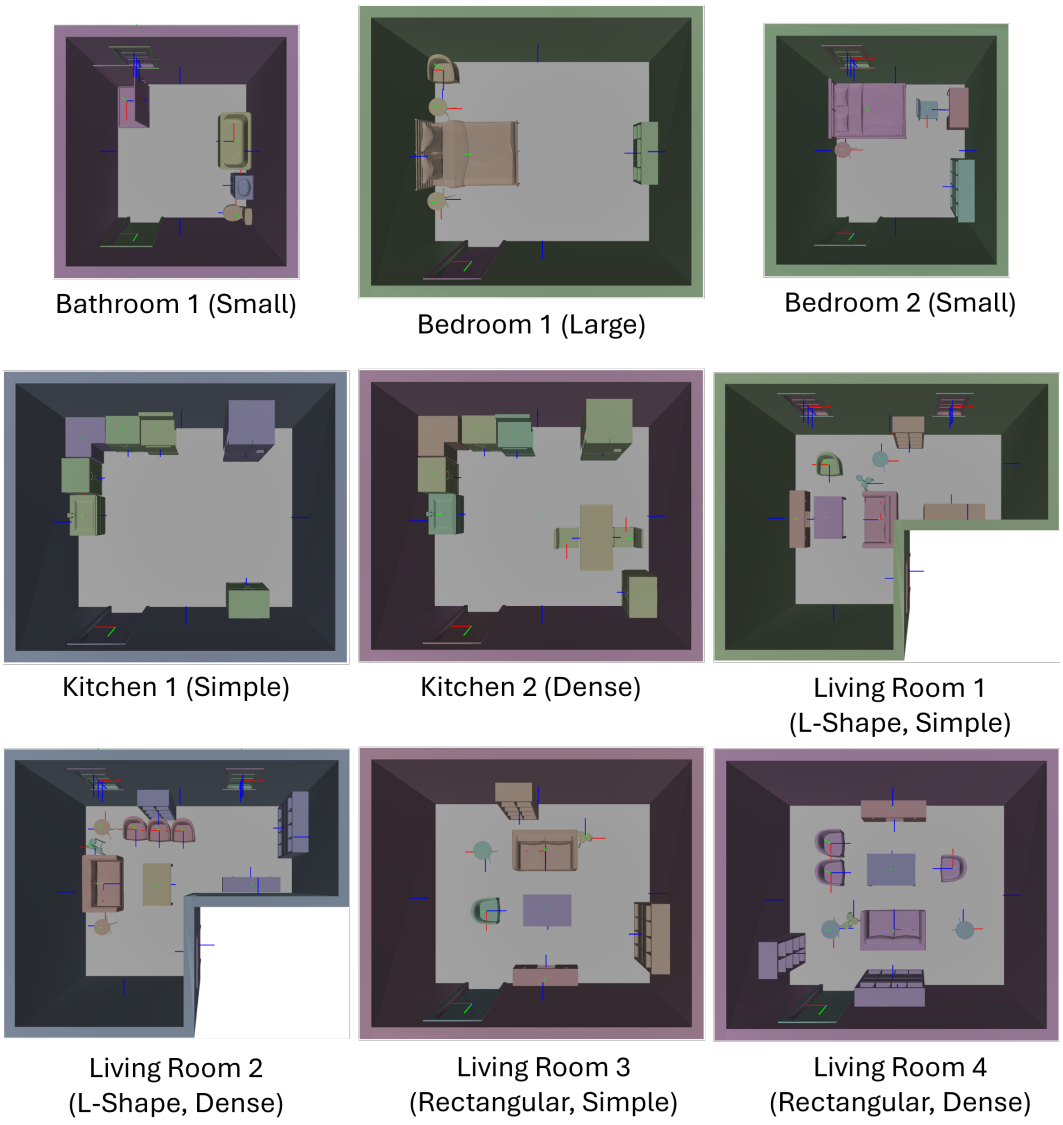


Figure 3.11: Rooms generated from new JS method.

Q: ***How does each algorithm perform in different types of rooms, with different functionalities, different sets of layout rules, and different types of furnishings?***

The strength of the GS method is the ability to place objects directly against a wall, as typically found in many rooms. Therefore, GS performed comparably well in kitchens and bathrooms where nearly all objects are against the wall but performed poorly in living rooms where objects were less wall dependent. Conversely, SA suffers due to the lack of motion of the objects because of the probability of rejecting potential better movements, making

moving towards the walls of the rooms less likely.

The grid-based method performs better for room scenarios where objects are generally placed on the wall than when objects are spread out and closer to the center. The opposite is true for the continuous methods, more prominently for SA.

Q: ***How does each algorithm perform at different levels of scenario density?***

On the comparison of layout density, when more of the same objects are added to the layout, as in the living room case, all methods perform consistently poorer than their less dense counterpart, meaning density inherently makes the search for high-quality layouts more difficult. Evidently, adding additional objects and rules that do not overlap in the kitchen density scenario does not have an effect on final quality, only that more check iterations are required due to the addition of objects.

When density is increased, efficiency drops for all methods but only when presented with more of the same objects. More objects of different variety and different rules may not affect efficiency.

3.5 Discussion

When looking at result variance, the GS method can randomly select over tie breaks, however, these are less frequent given the rule score quantification. Thus, the same result (usually scoring moderately high) is often reproduced. JS and SA, on the other hand, produce more variety in results, although some early selections result in lower local optima. When looking at strictly the maximum of all produced layouts from each of the 10 runs for each experiment, one variant of the JS always produced the highest result.

One area where the JS could have been improved, and more prominently in the SA, would have been a more intelligent sampling method. SA for instance is often paired with sampling methods such as Metropolis–Hastings (Metropolis et al. [96]), which samples from unknown sampling distributions. In this case, sampling could be prioritized closer to higher evaluation scoring

locations, making the location sampling more efficient. However, the JS does this to a minor extent by sampling nearer to the current location, which is also the highest-scoring location. Finally, adjusting the SA to more greedily accept the next location may have improved its results.

Overall, the JS method should be considered the stronger of the compared algorithms as it:

1. Produces equal or better layout quality, given a reasonable check budget
2. Has minimal reliance on the input room shape and size as the only impact is on the initial move amount, thus making the quality more predictable based on budgetary check requirements
3. Is more flexible when more challenging layout experiments are presented
4. Gives more variation in the final layouts, as the locations explored are more random

Some of the limitations of the JS algorithm include the fact that objects start near the center of the room, making movement closer to walls more challenging than the GS. Making no assumptions about any object relationships makes the solution general and fully customizable to the rules. However, intuitively some objects are usually placed relative to each other, for example, couches relative to walls or nightstands relative to beds. In its current form, JS does not take advantage of this domain knowledge. Thus, while JS supports generality, having some procedural placement, like "snapping" would improve runtime in practice if applicable.

3.6 Conclusion

Automation offers the potential for significant improvements in building design and construction. More specifically, the configuration of the layout of furnishings and appliances in the available interior spaces of the building can have a significant impact on the comfort of the occupants, as a poor arrangement can lead to inaccessible objects, have negative ergonomic effects on occupants,

and generally make a space unpleasant when it could have been better utilized otherwise. Automated interior layout generation has been a highly researched problem over the last 10 years. However, previous algorithms suffer from two important shortcomings. First, they tend to rely on non-intuitive and difficult-to-explain formulation of the rules that drive the generation of the candidate layouts, and second, they are difficult to compare against each other to analyze their relative merits and shortcomings.

In this chapter of the thesis, we propose a novel layout-configuration algorithm, Jump Search (JS), that searches through a continuous design space. JS relies on a Domain Specific Language for rule representation and a corresponding method for rule evaluation, which was first embedded in a simple grid-based search algorithm. Relying on the "number of rule evaluations" as a metric, we have conducted a comparative evaluation of JS against a Grid Search (GS) algorithm and a simple Simulated Annealing (SA) search algorithm, in nine different design scenarios of varied difficulty. Our results demonstrate that JS outperforms both other algorithms in terms of layout quality relative to computational density, while also exploring a bigger variety of layouts.

The contributions of this chapter are the presentation of a new continuous search-based algorithm, JS, and the comparison of the search efficiencies of one grid-based search (GS) and two continuous search methods, one more greedy (JS) and the other more conservative (SA).

Potential improvements to the algorithm can be made by improving the location sampling. The significance of object ordering and determining the optimal placement order that yields the highest layout quality is still to be investigated. A limitation of this study is the limited number of room scenarios and the lack of Mechanical, Electrical, Plumbing (MEP) detail of the rooms, as other features such as electrical and plumbing fixtures have not been included. The creation of higher detailed room scenarios will be tested in future work.

Chapter 4

Rule Learning from a Synthetic Layout Dataset

In this chapter, I describe an approach to extracting expert knowledge, in the form of rules, from interior layout examples. In particular, the method uses key relations among objects in the layout and learns the range of values of those relations observations gleaned from the input examples, which I assume represent the set of all possible rule-compliant layouts. I evaluate the quality of the rule learning algorithm through a correlation analysis among the quality scores assigned to layouts by the learnt rules and the expert rules used to generate these synthetic layout examples. After testing the rule learning on datasets from nine room scenarios (Appendix Section A.1), the findings of this chapter indicate that the rule learning algorithm works, as there is a moderate to high rule score correlation depending on the room scenario and the quality of the layout rules are learnt from.

The research in this chapter has not been published, however, a manuscript is being prepared together with the next chapter, Chapter 5.

Abstract: Building Information Modeling (BIM) has become an invaluable tool in the planning and design of buildings, including interior spaces. Thanks to the digitization, organization, and standardization of building components and their information, the ability to automate building evaluations has gained substantial interest due to the potential benefit of building designers wishing to verify and improve their designs. Automated Code Checking (ACC) involves

formulating machine-readable rule code for checking software to run and report rule results. Coding rules, however, is still a challenge for non-experts and while Domain Specific Language (DSL)s can help, one step further would be automatically generating rule code.

This chapter describes a novel method that learns rules from interior layout examples. The rule learning method leverages a rule DSL, a programmable code for ACC. We constructed a dataset of highly compliant layouts through the automated layout generation method (described in Chapter 3) based on a set of expert design rules. This dataset was used as input to our rule-learning algorithm which outputs newly learnt rules. An indication of the rule learner's ability to create rules with similar coverage can be achieved by comparing the expert rule scores and rule scores from the learnt rules on variable expert rule-compliant layouts. The findings suggest that the expert and learnt rule correlation is highest when many high expert rule quality layouts are provided. Aspects of the learner that influence the quality of the rule learner in terms of rule correlation scores and rule counts were further analyzed with certain parameters found to both reduce rule scores and increase rule score correlations.

4.1 Introduction

Buildings and in particular interior spaces are constantly evolving over the years. New room shapes, more furnishing object variety, and a generally better awareness of accessibility and sustainability are some of the factors why designs are ever-changing. Rules for arranging furniture are often used to communicate best practices that result in functionally efficient spaces. However, too often the rules are vaguely defined or difficult to transfer from one space to the next. And due to the many objects, and thus many interdependence relationships, it remains challenging to rigorously explore configuration alternatives. The result is the increasingly laborious task for home designers, both professional and amateur, of configuring novel spaces that maximize layout functionality. Furnishing objects in particular can be large and expensive, thus, fixing design

mistakes can be strenuous, time-consuming, and costly.

In recent years, computers have helped in mitigating many of the challenges associated with design. Computer Aided Design (CAD) tools represent layouts digitally to design and more accurately visualize potential layouts. Building Information Modeling (BIM) [4] is an extension of CAD that includes specifications of building components and their relationships. With the advancements of these technologies, CAD and BIM have become increasingly accessible and numerous design operations can be performed automatically. One such operation of significant value to designers is automated layout rule checking and evaluation, which is the process of scoring a digitized layout 3D model against a set of coded design rules [5]. A persistent challenge in automated rule checking is formulating machine-readable design rules accurately and more easily.

In this chapter, we aim to automate the process of creating rules that can be used in automated layout checking. We utilize rule-compliant example layout datasets to learn relationship constraints to which all furnishing objects in the examples adhere. The constraints are then converted to machine-readable and executable rule code for checking a layout configuration.

The machine-readable rule code we utilize is the Rule Domain Specific Language (RuleDSL) described in Chapter 2, Section 2.6 and in detail in Sydora and Stroulia [1]. This allows for employing the RuleDSL supporting workflow, including its rule editors, automated model-checking evaluation, and automated layout generation.

The proposed rule learner approach offers (1) an increased understanding of layout designs and identifying object-wise relationship patterns within groups of layouts through the RuleDSL, (2) a method to evaluate other layouts' compliance with the learnt rules via the model-checking, and (3) the ability to automatically generate new rule compliant layouts.

The contributions of this chapter are the description of a novel rule learning method and the demonstration of the rule learning methods' quality and effectiveness at capturing known rule information through a rule correlations analysis. The method is further analyzed by investigating the impact of input

training layout count, the impact of rule limiting, and verifying the importance of the selected relations. Therefore, four questions are proposed in the validation and analysis of the rule learning:

1. **What is the rule learning quality, measured by the correlation scores among expert and learnt rules, for each of our room scenarios?**

2. **How does the amount of training layouts impact rule learning quality?**

3. **How does limiting the number of rules to be learnt affect the rule learning quality?**

4. **How does the target rule template impact rule learning quality?**

The remainder of this chapter is structured as follows: The proposed method for learning the rules in the RuleDSL format is outlined in Section 4.2. Section 4.3 contains the experimental setup including the synthetic layout generation and rule learning experiment. The validation and analysis of the rule learning method are described in Section 4.4. Findings are discussed in Section 4.5 followed by conclusions, contributions, and future directions in Section 4.6. Chapter 2, Section 2.4 provides an overview of previous work in the realm of learning interior design knowledge from layout examples.

4.2 Rule Learning Method

The general approach of the proposed rule learning algorithm is inspired by Concept Learning [97], a field of Machine Learning (ML) used to determine which features from a feature set are relevant to a concept. For example, given a concept such as *rainy day* and recorded instances with boolean features such as *clouds*, *high temperature*, and *high humidity*, the idea is to learn which of these features are relevant and irrelevant using many observed concept-feature instances. Learning which features, either by their presence or absence, can predict the concept or which have no apparent correlation to the concept.

Similar to Concept Learning, our approach focuses on features (in our case pairwise relations) that are consistently present throughout the observed layout examples. However, rather than using categorical features, our approach takes advantage of the numerical nature of the relation values. Therefore, our approach expands the relation ranges each time a new layout example is observed. The output of the proposed algorithm (which is in tabular format) is converted to the RuleDSL to utilize the RuleDSL tools including the RuleDSL editor, automated layout checker, and automated layout generator.

The rule learning algorithm is broken up into five key steps:

1. Deciding on the target rule relations, i.e. the rule template.
2. Extracting object-pair instances for each training layout example.
3. Combining object-pair instances into type-pair rules within a training layout.
4. Combining type-pair rules from multiple training layouts.
5. Reconstructing rules in RuleDSL format from the type-pair rules.

4.2.1 Target Rule Template

Central to rule learning are the geometric relationships among the objects in the layout arrangement. Relation functions calculate pairwise relationship values between two objects based on the two objects' mesh representations, locations, and orientations. The most basic and common relationship is the mesh distance between two 3D objects placed in the layout.

The relations we selected for the rule learning in this chapter are based on commonly occurring relationships based on literature and rules. The selection of these relations dictates the expressiveness of the output learnt rules as information that cannot be encompassed by these functions is unlikely to be covered by the expert rules in the examples. The goal is that the selected relations cover much of the expert rules' information and thus will increase the likelihood that our learnt rules correlate to the expert rules. The selection of these relations significantly impacts the rule learning ability.

To provide a rough analysis of how well we selected the relations and how important each relation is for a particular room scenario, a relation ablation experiment is performed which we describe in Section 4.4.4.

The following are the relations we believe are most common in rules and can broadly cover a majority of the rules:

- *Distance*: using the length of the shortest line between the two meshes.
- *Facing*: the angle between the forward direction vector of the first object and the vector pointing to the center of the second object.
- *Alignment*: the angle between the forward direction vectors of two objects.

Distance and *Alignment* are independent of the order of the input meshes, however, *Facing* is order-dependent. Therefore, we use *Facing(A, B)* and *Facing(B, A)* as separate relations.

4.2.2 The Observations Table

Table 4.1: Corresponding observation table from the example bedroom in Figure 4.1.

A	B	D(A,B)	F(A,B)	F(B,A)	A(A,B)
Bed1	Chair1	1m	90deg	90deg	180deg
Bed1	Desk1	0.7m	90deg	90deg	0deg
Bed1	Door1	1.75m	45deg	135deg	0deg
Bed1	Wall1	1.5m	20deg	20deg	180deg
Bed1	Wall2	0m	45deg	45deg	90deg
Bed1	Wall3	0m	135deg	45deg	0deg
Bed1	Wall4	1.5m	80deg	10deg	90deg
Bed1	Window1	1m	110deg	60deg	0deg
Chair1	Desk1	0.5m	0deg	0deg	180deg
...	...	...	...	...	...

* D(A,B): A distance to B, F(A,B): A facing angle to B, F(B,A): B facing angle to A, A(A,B): A alignment with B.

Once we have determined the relations of interest, an observation table is extracted for each example layout. For a particular rule creation regarding two object types, all relation values among all pairs of objects of the two types are calculated. An example of an observation table used for learning rules extracted from the example in Figure 4.1 can be seen in Table 4.1.

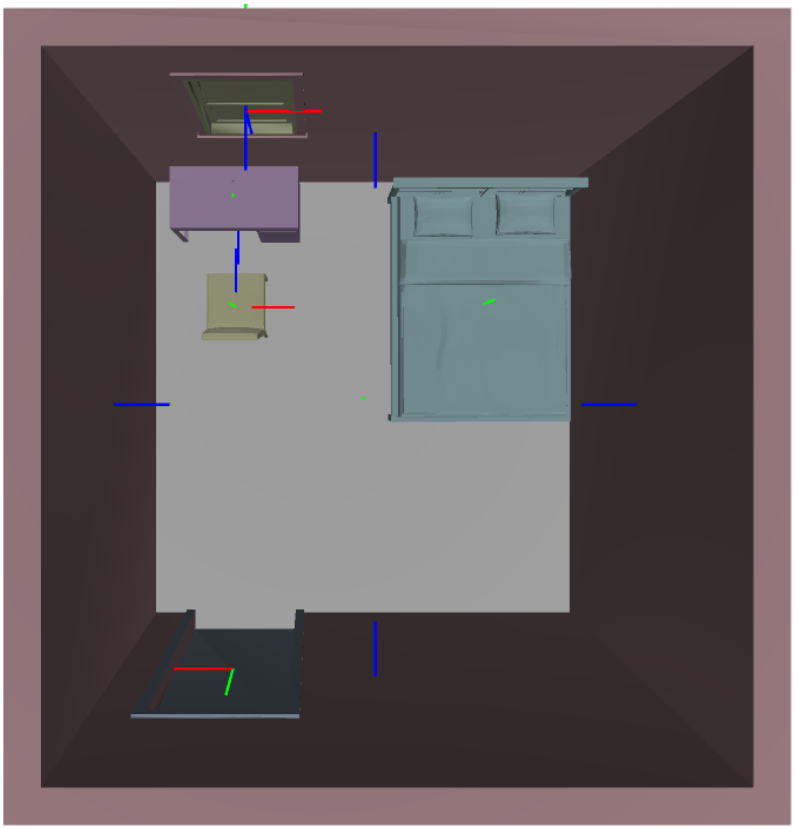


Figure 4.1: Example bedroom.

4.2.3 Forming Rules for an Example Layout

It is possible that, within one example layout, multiple objects of the same type are present. For instance, in Figure 4.1 there are multiple walls in a room that have different relationship values with each object. Often, rules will indicate that only one object-pair instance relationship satisfying a constraint is sufficient for the encompassing rule to be satisfied. A simple example is a desk needing only one chair in front of it, as opposed to every chair. In other rule cases though, every object of a type must satisfy the constraints, such as every shelf must be against a wall.

The presented rule learning method addressed this by supporting the distinction between requiring all objects of the same type to satisfy the rule or if it is sufficient for just one object of a particular type to meet the rule requirement, i.e. the ALL and ANY clauses. For rules with two object types, as with our template, there are four possible combinations of existential clauses:

ALL/ALL, ALL/ANY, ANY/ALL, and ANY/ANY. Importantly, an existential clause pair can result in several possible rule outcomes for one example layout.

Algorithm 2: Algorithm for combining relations from object-pair instances.

```
1 Function CombineInstances(inst1, inst2, RuleTemplate):
      Input:
          inst1, inst2 - Two object-pair instances that contain the relation (upper
          and lower) values
          RuleTemplate - The template the contains the list of relations
      Output:
          combinedInstance - The output combined instance with new (upper and
          lower) relation values
2     combinedInstance ← {}
3     foreach relation ∈ RuleTemplate.Relations do
4         relLower = Min(inst1[relation].Lower, inst2[relation].Lower)
5         relUpper = Max(inst1[relation].Upper, inst2[relation].Upper)
6         combinedInstance[relation] ← {relLower,relUpper}
7     end
8     return combinedInstance
9 end
```

Algorithm 2 shows the process of combining relation values of two object-pair instances (*inst1* and *inst2*), i.e. rows in the example layout observation table. For each of the relations in the input rule template (*RuleTemplate*), it takes the maximum of the upper range values (Line 4) and the minimum of the lower range values (Line 5) and sets them as the new combined instance relation range values (Line 6). Thus, the range's bounds are expanded as additional object-pair instances are seen.

Algorithm 3 shows the steps for the four existential clause combinations for two set object types. The function takes in the existential clause of the two types (*ec1* and *ec2*) and the observation table (*observationTable*). The general rule of thumb is for ALL cases, the relation instances for the same object types are combined using the *CombineInstances* function in Algorithm 2, while ANY cases result in each instance producing an additional rule. The ALL/ANY case (Lines 3-6) uses the *InstanceCombineByID* function described in Algorithm 4 to determine the new rules. The *InstanceCombineByID* function recursively goes through all combinations of instances where the first object IDs are differ-

Algorithm 3: Algorithm for learning type-pair rules from an example layout.

```
1  Function LearnExampleRules(ec1, ec2, observationTable):
       Input:
           ec1, ec2 - The two existencial clauses of the two types
           observationTable - The list of object-pair instances and relation values
       Output:
           Rules - The learnt rules (in tabular form)
2      Rules ← {}
3      if ec1 == "ALL" AND ec2 == "ANY" then
4          idList ← GetUniqueFirstIds(observationTable)
5          InstanceCombineByID(0, idList, observationTable, {}, Rules)
6      end
7      if ec1 == "ALL" AND ec2 == "ALL" then
8          combinedInst ← {}
9          foreach inst ∈ observationTable do
10             combinedInst ← CombineInstances(combinedInst,inst,RuleTemplate)
11         end
12         Rules.Add(combinedInstance)
13     end
14     if ec1 == "ANY" AND ec2 == "ANY" then
15         foreach inst ∈ observationTable do
16             Rules.Add(inst)
17         end
18     end
19     if ec1 == "ANY" AND ec2 == "ALL" then
20         idList ← GetUniqueFirstIds(observationTable)
21         foreach id ∈ idList do
22             combinedInst ← {}
23             idInstances ← GetInstancesWithFirstId(id,observationTable)
24             foreach inst ∈ idInstances do
25                 combinedInst ←
                     CombineInstances(combinedInst,inst,RuleTemplate)
26             end
27             Rules.Add(combinedInst)
28         end
29     end
30 end
```

ent and creates a rule for each. The ALL/ALL case returns a single rule which is the combining of all instances (Lines 7-13). The ANY/ANY case returns each table instance as a rule (Lines 14-18). And finally, the ANY/ALL case combines all instances where the first IDs are the same (Lines 19-29).

The outputs of the *LearnExampleRules* function are the type-pair rules from a single example layout that all hold for that layout. Table 4.3 shows an example of the rules formed from the observation table in Table 4.2.

Algorithm 4: Algorithm for combining instances by ID.

```
1  Function InstanceCombineByID(i, idList, observationTable, combinedInst,
     Rules):
       Input:
           i - The index of the ID combining by
           idList - The list of unique IDs
           observationTable - The list of object-pair instances and relation values
           combinedInst - Reference to the combined instance thus far
           Rules - Reference to the learnt rules list
2      if i > idList.Length then
3          Rules.Add(combinedInst)
4      else
5          currentId ← idList[i]
6          currentIdInstances ←
             GetInstancesWithFirstId(currentId,observationTable)
7          foreach inst ∈ currentIdInstances do
8              nextCombinedInst ←
                 CombineInstances(combinedInst,inst,RuleTemplate)
9              InstanceCombineByID(i+1, idList, observationTable,
                 nextCombinedInst, Rules)
10         end
11     end
12 end
```

Table 4.2: Observation table filtered for *Wall* and *Chair* object types for an example layout (Distance relation only for simplicity).

A	B	Distance(A,B)
Wall1	Chair1	0.1m
Wall1	Chair2	0.3m
Wall2	Chair1	0.2m
Wall2	Chair2	0.5m

Table 4.3: Rules from Table 4.2.

				Distance(A,B)	
EC1	Type1	EC2	Type2	Lower	Upper
ALL	Wall	ALL	Chair	0.1m	0.5m
ANY	Wall	ALL	Chair	0.1m	0.3m
ANY	Wall	ALL	Chair	0.2m	0.5m
ALL	Wall	ANY	Chair	0.1m	0.2m
ALL	Wall	ANY	Chair	0.1m	0.5m
ALL	Wall	ANY	Chair	0.2m	0.3m
ALL	Wall	ANY	Chair	0.3m	0.5m
ANY	Wall	ANY	Chair	0.1m	0.1m
ANY	Wall	ANY	Chair	0.3m	0.3m
ANY	Wall	ANY	Chair	0.2m	0.2m
ANY	Wall	ANY	Chair	0.5m	0.5m

Algorithm 5: Algorithm for merging rules from a new example layout with the rules learnt and combined over the previous set of example layouts.

```
1  Function CombineRules(NewRules, OldRules, N):
       Input:
           NewRules - The newly learnt rules from the new example
           OldRules - The rules learnt from the previous examples
           N - The limit on the number of rules maintained in the learning
       Output:
           CombinedRules - The learnt rules combined from the old and new rules
2      CombinedRules ← {}
3      foreach old ∈ OldRules do
4          foreach new ∈ NewRules do
5              combined ← CombineInstances(old,new,RuleTemplate)
6              CombinedRules.Add(combined)
7          end
8      end
9      CombinedRules ← RemoveDuplicatesAndBroadRules(CombinedRules)
10     CombinedRules ← RuleReduceOptions(CombinedRules, N)
```

4.2.4 Combining Rules from Multiple Example Layouts

Each example layout will result in type-pair rules that are satisfied for that layout. Combining rules from all observed training layouts generalizes the rules over a wider range of examples. Because each layout can result in multiple rules, each existential clause (ALL/ANY combination type-pair) rule from an additional example layout must be combined with each rule of the same existential clause found from the previously observed layouts. This means that if N rules of a specific existential clause combination existed previously from the training set and M rules of the same existential clause combination from an additional example layout are learnt, NxM new rules will be formed for that existential clause combination. Algorithm 5 shows the rule combining (Lines 3-8).

Noteworthy is that all learnt rules are valid and do not conflict with each other. Put another way, if you ran these rules over all the examples they are trained on, all rules would return true for each layout which is a property that will always hold for every new example shown as well. Some rules however are arguably more interesting or meaningful than others. The more interesting rules are those that have narrow relation ranges implying more strict object

type relationships. The less interesting rules are those that contain wide relation ranges and are easily satisfied as they restrict placement to a lesser effect.

A few operations can be performed to reduce the total number of rules. This benefits when manually parsing through all output rules and reducing computation when performing the automated model checks. The more basic rule reduction operations include checking if all ranges of one rule are equal to or fit within the ranges of another rule. In this case, the broader rule can be discarded (Algorithm 5, Line 9). This is because if the first, more restricting narrow rule is satisfied, the broader rule will always be satisfied.

Additional, more elaborate rule reduction operations, were created to further reduce the total number of rules created that can be used (Algorithm 5, Line 10). The description and effects of these options will be explored in Section 4.4.3.

4.2.5 RuleDSL Conversion

To utilize the RuleDSL and supporting tools, the last step is to take all the rules learnt from the examples, which up to this point are in tabular format, and convert them to the executable RuleDSL format. The final rule for an object-type pair will take the following RuleDSL structure by creating relation checks from the relationship upper and lower bounds:

```
{ALL/ANY} x ∈ Type1
{ALL/ANY} y ∈ Type2
(Distance(x,y) ≥ ? AND Distance(x,y) ≤ ? AND
Facing(x,y) ≥ ? AND Facing(x,y) ≤ ? AND
Facing(y,x) ≥ ? AND Facing(y,x) ≤ ? AND
Alignment(x,y) ≥ ? AND Alignment(x,y) ≤ ?)
```

4.3 Experimental Setup

The experimental design consists of two steps. First is the creation of synthetic layouts used to train and evaluate the learnt rules. The other is the process by which all possible rules are learnt from an input set of examples, as the

rule learning algorithm as described in Section 4.2 is for learning rules for two specified types and existential clauses.

4.3.1 Synthetic Layout Creation

Synthetic layouts are layouts that are furnished using the automatic layout generation which utilizes the Jump Search (JS) method described in Chapter 3 and the room scenarios described in Section A.1. JS uses expert rules in the RuleDSL format to evaluate placement options. As a result, generated layouts have an associated expert rule score which describes their quality in terms of rule compliance.

The synthetic dataset can be further split into two datasets, a training set and a testing set. Having high-quality training layouts that follow the rules is important; if the layouts do not follow the expert rules, then attempting to learn the expert rules is not realistic. Therefore, the training dataset of layouts is generated automatically with very high search and evaluation budgets, such that they score as high as possible on the expert rules. Full compliance, however, may not be possible if the rules conflict or if there are layout space limitations.

The testing set is generated using the same method and room scenario inputs, however, unlike the training set, the testing set layouts' generation budget is variable, resulting in layouts that have a variety of expert rule scores. Because using either the expert rules or learnt rules as constraints might bias these results, we opted for one-third of the testing layout set to use the expert rules, one-third using the learnt rules, and one-third using no rules and thus random placements. This means layout scores can be compared when layouts are on a spectrum of expert rule quality. For each room scenario, a total of 20 training layouts and 150 testing layouts were used.

4.3.2 Learning All Possible Rules

The existential clauses (ALL/ANY and Types) for a rule are manually determined ahead of time for learning a single rule. However, if learning all possible

Algorithm 6: Algorithm for learning all possible rules from training layout set.

```
1  Function LearnAllRules(Layouts, RuleTemplate, N):
       Input:
           Layouts - The input example layouts
           RuleTemplate - The template the contains the list of relations
           N - The limit on the number of rules maintained in the learning
       Output:
           Rules - The learnt rules in the RuleDSL format
2      types ← GetUniqueTypes(Layouts)
3      ECTypes ← {"ALL","ANY"}
4      Rules ← {}
5      foreach t1,t2 ∈ types do
6          foreach ec1,ec2 ∈ ECTypes do
7              typeRules ← {}
8              foreach layout ∈ Layouts do
9                  observationTable ← ExtractRelations(layout, t1, t2,
                     RuleTemplate)
10                 layoutRules ← LearnExampleRules(ec1, ec2, observationTable)
11                 typeRules ← CombineRules(layoutRules, typeRules, N)
12             end
13             convertedRules ← ConvertToRuleDSL(typeRules)
14             Rules.AddMany(convertedRules)
15         end
16     end
17 end
```

rules, the existential clauses are iterated over each possible clause/type combination. For the experiments in the evaluation methodology in the following sections, all possible rules are learnt.

The full rule learning process, where all rules relating to all object type pairs, can be seen in Algorithm 6. First, all object types in the layouts are collected (Line 2). Going through each pair of types and each pair of existential clauses, rules are created from the layouts (Lines 5-6). Going over each layout, an observation table is extracted in the format of Table 4.2 which is specific to only the types $t1$ and $t2$ and the relations from the template (Line 9). Next, the rules within the layout are learnt based on the two existential clauses (Line 10, Algorithm 3). The rules are then combined with the rules learnt thus far from the previously observed layouts, which include the rule reduction steps (Line 11, Algorithm 5). Once all layouts have been covered, the rules' relations ranges are converted into the final RuleDSL template structure and added to

the list of all rules to be returned by the rule learner (Line 13).

4.4 Rule Learner Evaluation

This section describes the evaluation of rule learning in terms of its quality in producing rules that capture the information of the expert rules in the training layouts. Then, an in-depth analysis is done on various parameters affecting the rule learning quality: The effect of input training layout count on rule score correlations is investigated. Then the options for rule limiting are analyzed. Finally, the relations of interest are tested to roughly inform us how vital each relation is for capturing rule information.

4.4.1 Rule Learning Quality

Q: ***What is the rule learning quality, measured by the correlation scores among expert and learnt rules, for each of our room scenarios?***

When trained on layouts that strictly follow expert rules, i.e. they score (near) perfect on expert rules, a high-quality rule learner should create learnt rules that provide scores similar to those expert rules. If considering a layout that scored well on the expert rules but did not on the learnt rules, or visa versa, there must have been some aspect of that layout configuration that was not accounted for equally in both rules, which is considered a flaw or shortcoming of the rule learner. This motivated us to look at the correlation among expert and learnt rule scores over various layout configurations as an approximation for the frequency and severity of the rule learning quality flaws. Because the output learnt rules will result in many more rules, some of which are less meaningful (i.e. always pass due to large check ranges), a correlation analysis is done as opposed to a direct score percentage comparison.

The challenge of evaluating the rule learner is that, given a collection of layouts, the precise rules that the layouts follow are often implicit. Even when explicit, there is no guarantee they accurately follow those rules if no rule evaluation is provided. Using our synthetic layouts mitigates these challenges. Generated layouts have evaluation criteria that guide the generation toward

the final designs. In the case of our synthetic training layouts, they are precisely the expert rule scores. The learnt rules share the same property; they provide a single evaluation score for the layout configuration relative to its rules. The formula for a layout rule score as a percentage of rules passed is shown in Equation 2.1.

After learning the rule from the training layout set, the testing set of synthetic layouts is used for the evaluation score comparison. The testing set layouts are generated using varying expert rule quality and evaluated against both the expert and learnt rules. Each testing layout is plotted with the X axis representing the expert rule scores, and the Y axis the learnt rule scores. A line of best fit is then created:

$$a = \frac{\sum_{i=1}^{n}(x_i - x_{mean}) * (y_i - y_{mean})}{\sum_{i=1}^{n}(x_i - x_{mean})^2}$$

$$b = y_{mean} - a * x_{mean}$$

$$f(x) = a * x + b$$

A standard correlation metric is the R^2 score:

$$R^2 = \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - y_{mean})^2}$$

The R^2 is a metric for measuring how well a model (in this case the linear model) fits the data. Or more generally, the R^2 value describes how well the data, in our case the two rule scores, are linearly correlated. In our case, rule learning quality is approximated by this value; a score closer to 1 indicates a high correlation and thus high rule learning quality while a 0 score indicates no correlation and low rule learning quality.

Table 4.4 shows the baseline correlations for each layout scenario. The baseline evaluation used all 20 training layouts as input and used the overlap reduction with a rule limit of 10 after each new training example. (Details of this will be explained in Section 4.4.3). Each room scenario presents a different level of layout arrangement and rule compliance which in turn affects the rule learning ability. For instance, kitchens are very challenging to learn the rules

Table 4.4: Baseline Correlations for All Room Scenarios

Room Scenario	Mean Training Expert Score	Learnt Rules	Rule Correlation Score Total
Bathroom	99.405 ± 0.194	118.000 ± 0.000	0.867 ± 0.021
Bedroom 1	97.398 ± 0.175	109.800 ± 1.796	0.753 ± 0.022
Bedroom 2	99.744 ± 0.075	138.400 ± 0.784	0.749 ± 0.028
Kitchen 1	94.418 ± 0.171	139.000 ± 1.640	0.492 ± 0.053
Kitchen 2	94.612 ± 0.217	209.000 ± 1.640	0.410 ± 0.040
Living Room 1	97.382 ± 0.379	254.200 ± 2.867	0.487 ± 0.031
Living Room 2	95.820 ± 0.226	314.400 ± 0.999	0.614 ± 0.035
Living Room 3	98.965 ± 0.142	169.000 ± 4.204	0.812 ± 0.011
Living Room 4	97.513 ± 0.261	228.600 ± 3.539	0.867 ± 0.018

with low correlation values, however, they also have the lowest mean training layout expert rule scores making them the least compliant with the rules. There is a general trend that the more rule-compliant the training samples, the higher the rule-learning quality; a small dip in expert rule compliance results in a significant dip in rule-learning quality.

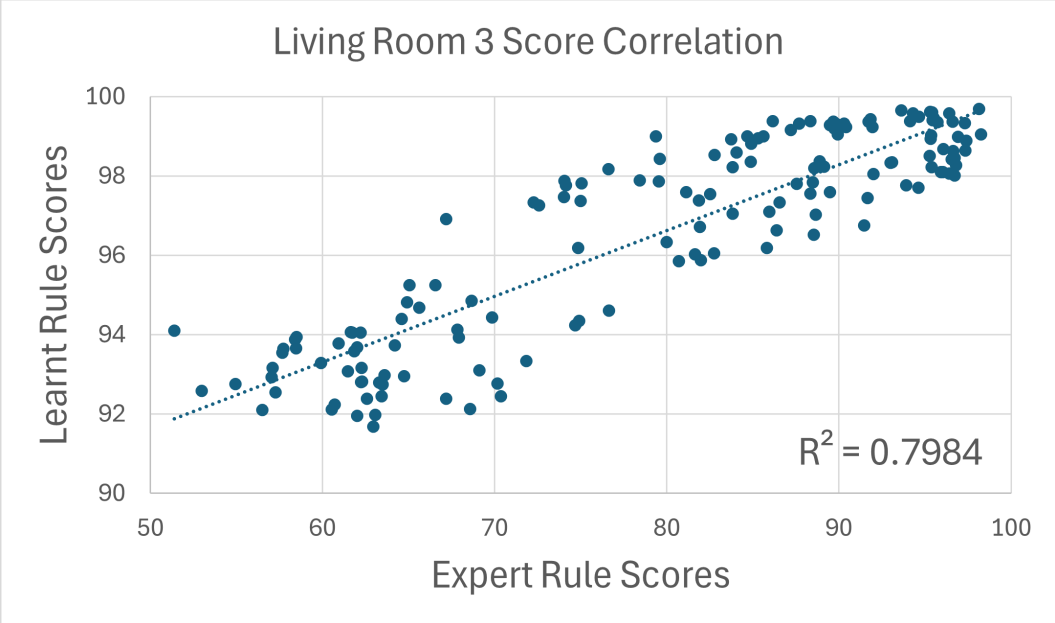


Figure 4.2: Correlations of layout scores from expert and learn rules on a testing set of layouts. Each point represents a testing layout.

To give a depiction of one baseline correlation evaluation, the correlation plot of Living Room 3 can be seen in Figure 4.2. The training layouts scored an expert rule compliance score of 98.856, meaning they followed the rules

well. The correlation is moderate to high with an R^2 of 0.7984, indicating a moderately strong correlation exists. The positive trend shows higher input scores result in higher learnt scores, as expected. The learnt rules scores are very high in general (from 91.685% to 99.689%) because many rules are easy to follow and thus they could potentially be discarded or ignored. These could be rules with large range values or are likely related to pairs of static layout objects (Walls, Doors, and Floors).

The evaluation suggests that the proposed rule learning is capable of learning highly correlated rules, however, the extent to which the learnt rules are correlated depends on the scenario and to a considerable extent the training layout quality.

4.4.2 Input Training Layout Count

Q: ***How does the amount of training layouts impact rule learning quality?***

This question looks at how many layouts are required to reach a certain level of rule learning quality. In theory, when more training layouts are continued to be observed, the rule learner will plateau in the information it has seen as presumably all possible variations of the rules will be observed. Our question here is to roughly identify how many training layouts to provide before each new layout no longer adds value. Naturally, this is dependent on how variable a layout generator you are using.

Table 4.5: Adjusting Training Input Count For Living Room 3

Training Layout Count	Mean Training Expert Score	Learnt Rules	Rule Correlation Score Total
1	99.706 ± 0.046	225.600 ± 0.701	0.420 ± 0.019
3	99.621 ± 0.049	214.800 ± 2.902	0.589 ± 0.026
5	99.544 ± 0.053	197.600 ± 7.654	0.686 ± 0.029
10	99.358 ± 0.087	177.200 ± 3.735	0.768 ± 0.021
20	98.856 ± 0.306	171.400 ± 3.661	0.818 ± 0.010

Table 4.5 shows the results of increasing the number of training layouts. When selecting the training layouts, the layouts were first ordered and then

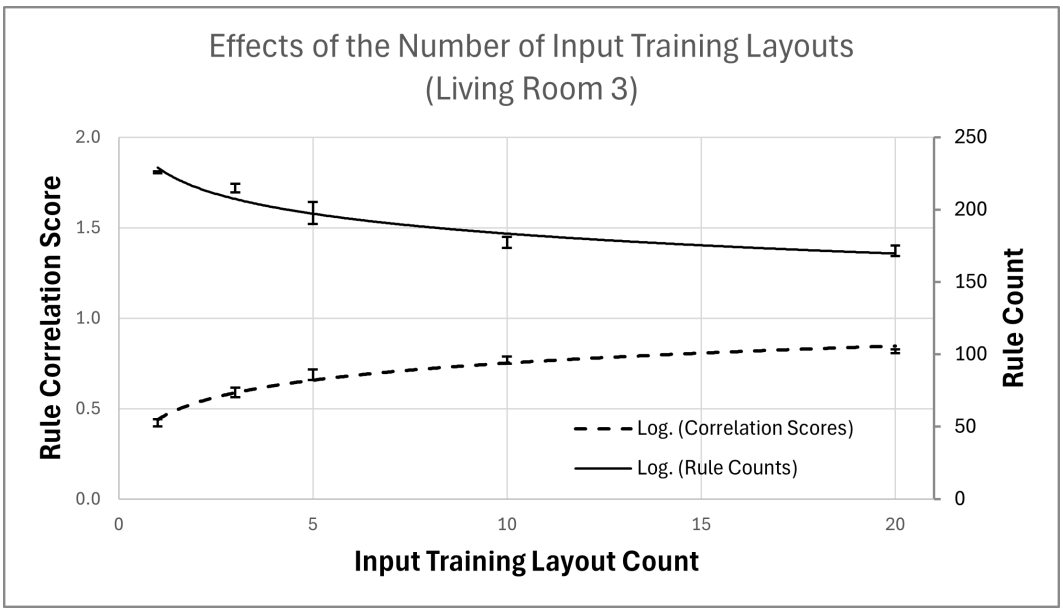


Figure 4.3: Trend of correlation scores and rule counts as the number of input layouts increases for Living Room 3.

the top N layouts were used. Therefore, additional training samples will always bring down the average expert rule score, which is recorded. As the number of training layouts increases two effects happen which can be seen in Figure 4.3. First, the number of rules significantly decreases. And second, the rule correlation does show a steady increase, even as average training scores decrease slightly.

The results verify that as the number of training layouts increases, the rule learning quality increases and the rule counts decrease but with diminishing returns.

4.4.3 Rule Reduction

Q: ***How does limiting the number of rules to be learnt affect the rule learning quality?***

As outlined in Section 4.2, when the next example layout is presented, rules are first created for that new layout, and then combined with all learnt rules from the previous training layouts. Because of this, there is the possibility that the number of rules grow exponentially if no new rules are duplicates or broad rules of the existing rules. While it appears inefficient to combine all

training layout rules exhaustively, in practice the relations are specific enough that the rules remain tightly bound and many broad rules are discarded early. However, even then, as with the case of lots of visually diverse layouts, many rules can potentially still be created. While this is not necessarily a negative effect in terms of rule quality, it does add more computation when computing the learnt rule scores. It also results in much more rule parsing when trying to understand the layout relationships and can make modification difficult. Rule-limiting options can be introduced to mitigate this; the trade-off being that they can potentially affect rule learning quality in terms of correlation scores.

The first rule-limiting option is a check that can be performed if the range bounds of the two observed rules overlap. If they do, the overlapping bounds can be saved as a new rule and the two overlapping rules can be discarded, as they are now broader than the new overlapping rule. This is optional as now two separate rules that could be satisfied by separate objects are forced to be combined which could potentially affect the rule learn quality.

The other rule-limiting option is to keep only the tightest bounding rule for each type and clause (ALL/ANY) pair. In general, a rule with tighter bounds implies that the object type-pair has more strict relationships than if rules had more broad ranges. In this option, all relation ranges (upper minus lower) are summed together to create a rule ranking for each type and clause pair. A note here is that all distance units are in meters and degrees are in radians, giving both 1 meter and 1 radian equal weight, making it only an approximate ranking. This ranking followed by capping the rules can either be performed during the learning process, only at the end once all rules have been learnt, or a combination of the two.

The rule-limiting options can affect the rule learning quality because rules are removed in favour of smaller rules. In rare cases, the larger ones could downstream lead to smaller ranges than an early small range that is forced to expand later. Therefore, the results with the settings both on and off are compared to highlight their impact on both learning quality and output rule count.

Table 4.6: Rule Reduce Method Comparison for Living Room 3

Reduce With Overlap	Keep Only Best Rule at End	Rule Cap	Learnt Rules	Rule Correlation Score Total
FALSE	FALSE	100	1868.200 ± 28.166	0.606 ± 0.017
TRUE	FALSE	100	230.200 ± 04.163	0.720 ± 0.012
FALSE	FALSE	100	1868.200 ± 28.166	0.606 ± 0.017
FALSE	TRUE	100	134.800 ± 00.351	0.831 ± 0.017
FALSE	FALSE	1	134.800 ± 00.351	0.838 ± 0.009
FALSE	FALSE	10	435.800 ± 13.996	0.725 ± 0.013
FALSE	FALSE	100	1868.200 ± 28.166	0.606 ± 0.017

Table 4.6 shows the results of the rule-limiting options. In addition to reducing the number of rules substantially, each option appears to improve the rule correlation. The most likely reason for this is because the broad rules that are kept will always pass making the correlation worse than if they are removed.

From the tests, we see that the rule reduction methods can significantly reduce the final rule count and improve the correlation scores.

4.4.4 Initial Template Relation Ablation

Q: *How does the target rule template impact rule learning quality?*

The relation checks in the template for the rule learner were selected based on the relation functions deemed likely to be relevant based on frequency in research and literature. To individually test the relevance of the four key relations (*Distance(A,B)*, *Facing(A,B)*, *Facing(B,A)*, and *Alignment(A,B)*), the rule learning was performed with each of the relations functions removed. An expected result is the rule quality might differ when certain relations are removed; a decrease in the correlation would indicate the removed relation was relevant. An unaffected correlation would indicate that the relation check could be removed to save computation.

Table 4.7 shows the results of removing one of the relations for Living Room 3. The correlation value decreases most significantly when facing is

Table 4.7: Rule Template Relation Comparison

Relations	Learnt Rules	Rule Correlation Score Total
D, F_{AB}, F_{BA}, A	171.400 ± 3.661	0.818 ± 0.010
F_{AB}, F_{BA}, A	189.000 ± 2.147	0.718 ± 0.018
D, A	176.000 ± 2.479	0.638 ± 0.026
D, F_{AB}, F_{BA}	175.400 ± 2.965	0.789 ± 0.008

* D: Distance, F_{AB}: A Facing B, F_{BA}: B Facing A, A: Alignment

removed, then distance, and alignment only slightly. This implies that in order of relation importance of living room arrangement are facing, distance, and alignment. Therefore, to save computation, alignment could be removed for the living room scenarios.

The effect of removing relations on rule learning quality gives insight into which relation functions are more important for that particular room scenario.

4.5 Discussion

Overall the results of the correlation analysis show that the proposed rule learning algorithm can capture a significant amount of information from the expert rules. Put another way, the algorithm can learn a collection of rules that collectively share the same rule intention as the expert rules that guided the training layouts. From the results, it is evident that the ability of the rule learner, and thus the quality of the learnt rules, is highly dependent on the quality of the training layouts. This intuitively makes sense since it is easier to learn rules from layouts that follow the rules, as opposed to trying to learn rules from layouts that only vaguely follow the rules.

We hypothesized that rule learning quality can be affected by the number of input training layouts. To test this, we increased the number of training layouts which increases the likelihood of diverse layouts. In our experiments, we found that providing more training layouts (of high expert-rule compliance), results in increased learning quality. The interpretation being more diverse positive examples of the rules improves rule learning.

An issue with the rule learning algorithm is that a potentially large number of rules, all being valid, are output. This can make checking against the rules more computationally expensive and more time-consuming to parse through. We analyzed the effects of potential rule-reducing methods. For all rule-reducing options, we found the final rule counts are reduced significantly. Perhaps somewhat surprising is that the rule correlations significantly improve as well. The most likely explanation is that when the broad rules are discarded, the more meaningful rules that better capture the interesting relations are isolated. This suggests that not only does the rule reduction reduce rules but also it should be employed to improve the rule correlations.

Lastly, we experimented with the effect of removing individual relations from the target rule template. This was done by removing one relation and calculating the score without that relation in the template. The results show that different room scenarios are dependent on different relations being present or not. It is naturally safest to include all relations, however, removing a relation that is likely to be less influential for capturing the rules in that room scenario can reduce rule computation when checking the rule score.

4.6 Conclusion

This research aims to make positive strides toward an understandable and user-friendly rule-code creation process. The rule learning algorithm is outlined and its quality is demonstrated through in-depth analysis and experimentation. Using the proposed algorithm, rules can be learnt from relatively few compliant examples which in turn can be used for design evaluations that can be performed more quickly and thus more frequently to approximately verify compliance. This will in turn increase automated code compliance checking adoption leading to more well-designed layouts.

Through the work presented in this chapter, our contributions are as follows:

1. A novel algorithm for automated rule learning by expanding type-pair relation ranges. The algorithm learns rules that are interpreted into the

RuleDSL, supporting the use of its automated checking and layout generation. The rule learning algorithm has been evaluated via a correlation analysis among expert and learnt rule layout scores. The algorithm is general having been tested and evaluated on a variety of room scenarios. Rule learning parameters have been analyzed that affect the rule learning quality, can reduce the learnt rule count, and inform specific relation importance.

2. A new synthetic layout dataset of rule-compliant training layouts with rules and associated compliance scores.

4.6.1 Limitations

We acknowledge a bias for the learnt rules matching the expert rules closely, given that both are in the RuleDSL format and have similar relation functions and structure. The evaluation method could, however, be used to correlate the learnt RuleDSL rule scores with any expert layout evaluation format, provided the other expert evaluation code also returns a single evaluation score.

Second, kitchens performed very poorly in all our experiments. This is mainly due to the design technique used, where cabinets are individual objects moved around as furniture. In practice, kitchen appliances are placed based on electrical and plumbing fixtures with cabinets constructed around the appliance locations. Thus, we believe the cabinet placement should be automated using a more procedural placement after appliance placement, as opposed to using a placement search as we did for these experiments.

4.6.2 Future work

An interesting direction would be exploring ways to iteratively build this template (semi-)automatically, including the logical expression and existential clauses. This would address the fact that the output rules all follow the same template and that that template is not as expressive as the rule language.

Fine-tuning the rule learning method was also left to future work as this would be user-dependent on what object types and rules they are interested in

learning. For this paper, all possible rules were found, regardless of whether the objects were static in the training layouts or not. Simply removing rules where all rule objects type that have no location change from layout to layout would reduce the total number of rules significantly. Further rule reduction could also be achieved by removing ALL/ANY rule combinations that are less meaningful than other combinations for the same object type rules, especially in the common case where a type pair only contains a single instance.

Chapter 5

Rule Learning from User-Generated Example Layouts

In the previous chapter, synthetic layouts were used to evaluate the quality of the rule learning. In particular, rule learning was evaluated on its ability to capture rule knowledge from layouts that were known to follow a particular set of rules. This study examines the case when input layouts do not explicitly follow rules but rather have a level of perceived quality. Because expert knowledge is not known, the comparison is made between the perceived quality of the layouts that went into the rule learning and the perceived quality of layouts generated using the learnt rules. The contribution of this chapter is a case study on the effectiveness of the rule learning algorithm described in Chapter 4, Section 4.2 assessing its ability to capture and recreate user-perceived quality in layouts.

5.1 Introduction

Non-experts design their layouts based on perceived quality, or the perceived ability of the layout configuration to fulfil tasks. Plausibility, or perceived quality, refers to the apparent functional validity of a layout, and by association its object relationships. Perceived quality is difficult to learn because it is subjective to the user and how they envision the space being used even if the configuration is less efficient at performing the envisioned room functions or

tasks.

In the majority of related Data-Driven Layout Synthesis (DDLS) research, perceived quality, or the perceived ability of the space to fulfil occupant needs, is the benchmark to which their methods are assessed. The reason for this is that the layouts in the most common datasets are created by human designers and there is no enforcement on the layouts created, meaning no rules are strictly followed or measured for that matter. Simply put, DDLS often assumes that the layouts learnt from are perceived as good and the goal of these methods is to learn new layouts that are also perceived as good.

This chapter looks at how well our rule-learning algorithm, described in Chapter 4, performs when the input layouts do not follow any particular rules but have some degree of perceived quality because they were created by people. The learnt rules from the user-created training layouts are used to guide the generation of new layouts, or learnt-rule generated layouts. The perceived quality of the input user-created training layouts and learnt-rule layouts is determined by way of a layout perception scoring survey. This chapter assumes that the generative designer can generate layouts that score high on the learnt rules which is backed by the results of the automated layout comparison finding in Chapter 3.

The chapter is organized as follows: First, the user layout collection process is described in Section 5.2 with a brief analysis of the layout creation data. Section 5.3 describes the survey for collecting the perceived quality scores of all layouts created. The results of the survey are presented in Section 5.4 followed by the conclusion in Section 5.5.

5.2 User-Created Layout Collection

To test and evaluate our proposed rule learner's ability to learn and recreate perceived quality, a dataset of layouts was collected[1]. For the collection of perceived quality layouts, participants were recruited using the university-

[1]This research received research ethics approval from the University of Alberta Research Ethics Board, Project Name "Interior Design Layout Data Collection and Evaluation", No. Pro00124859, February 28, 2024.

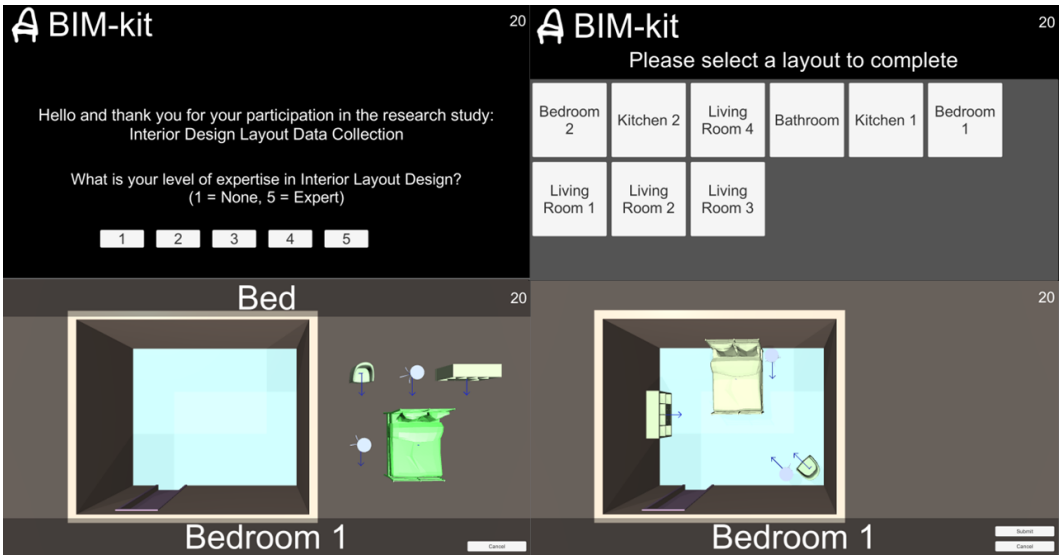


Figure 5.1: Layout collection User Interface (UI)

wide weekly email postings. The only exclusion criterion was age-related; participants were required to be over the age of 18. Participants were invited for one 1 hour in-person session and received a $15 gift card for participating.

In the layout editor, they were shown an initial empty room and various furnishing objects on the right side of the room in random order as seen in Figure 5.1. Their task was to place all furnishing objects in the available space to complete the layout; the layout could not be submitted until all objects were placed. Each move action was recorded, including the action time beginning from when they opened the scenario, the Unique Identifier (ID) of the object moved, its object type, its new location, and its new orientation.

Participants were given all nine room scenarios (described in Section A.1) to complete in random order. They were encouraged to do one as a test to understand the controls, i.e. open one scenario, and test out the controls without submitting this test scenario. During the session, they were asked to create as many layouts as they could in the session time.

Each layout in the dataset contained two files: (1) The 3D layout file containing the geometry of each object as well as the object specifications such as unique ID and type. (2) A metadata file with a designer score which was a self-evaluation of the expertise level of the layout creator, an action sequence

list, and a compiled list of object relations from the verbal descriptions.

20 participants were recruited for the layout collection. Data from two participants were excluded after taking over 10 minutes to complete at least one layout, with one of the two participants unable to complete all 9 layouts in the allotted session time. The remaining 18 participants created one of each of the 9 room scenarios for a total of 162 layouts.

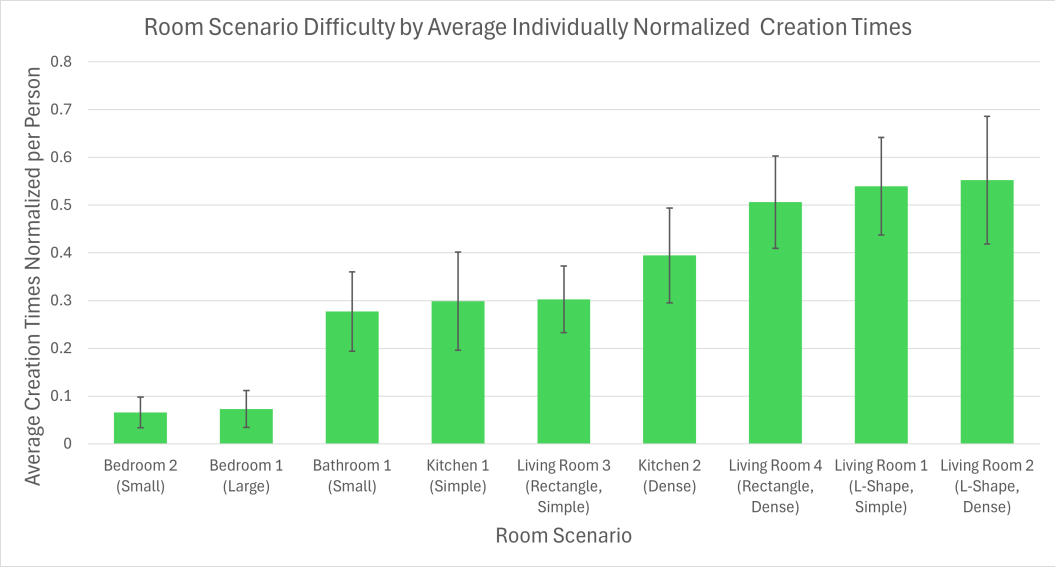


Figure 5.2: Room design difficulty approximation based on average comparison of design durations normalized by each participant.

An interesting piece of information that our dataset provided that is not available in other layout datasets is the sequence of actions taken. This provided insight into the degree of difficulty a room scenario was relative to the other scenarios for an individual participant. One observation was that some participants did many moves in short periods, i.e. lots of minor edits, while others did few but longer, thought-out moves. Therefore, action times were considered more meaningful than the number of moves. To compare scenario difficulty, each participant's layout data was considered individually and the total creation time for each layout was normalized among their created layouts. This meant that a time of 0 was given to the layout that took that participant the fastest to create, and a time of 1 was given to the layout that took them the longest, with all other layout times being a time relative to the two. Then

all normalized creation times were averaged over all participants, the results of which can be seen in Figure 5.2.

Evidently, the bedroom scenarios were the fastest to create, followed by the bathroom, kitchens and the "easy" living room, followed by the "challenging" living rooms. The challenging living rooms were living rooms that had either more objects (higher density), were L-shaped, or both (Living Room 2). As anticipated, difficulty was very closely correlated to the number of objects. A note here is that it is likely that some objects were not as easy to identify or understand how they operated. This was particularly evident with the shower in the bathroom scenario as some participants were unclear about where the access side of the shower was.

5.2.1 Learnt Layout Generation

The process of creating the learnt layouts consits of two steps. First, rules are learnt by passing the user-created layouts into the rule learning method. In our experiments, learnt rules could have either 2, 5, or all of the 18 user-created layouts used as input. This enabled investigation of the effect of the input layout count on the ability of rule learning to capture perceived quality.

Next, layouts were generated using the Jump Search (JS) method described in Chapter 3. Layouts could either use learnt rules as input, no rules (random), or expert rules from the room scenario. For the learnt rule layouts, rather than using the placement order described in Section 3.2.4, the placement order is created based on the average observed object type placements from the training layouts. The reason for this is twofold. For one, the placement order is recorded along with each layout. Second, the number of learnt rules is inconsistent making the dependency order less meaningful. All generated layouts were given a high search iteration budget such that layouts in terms of their input rule quality were maximal.

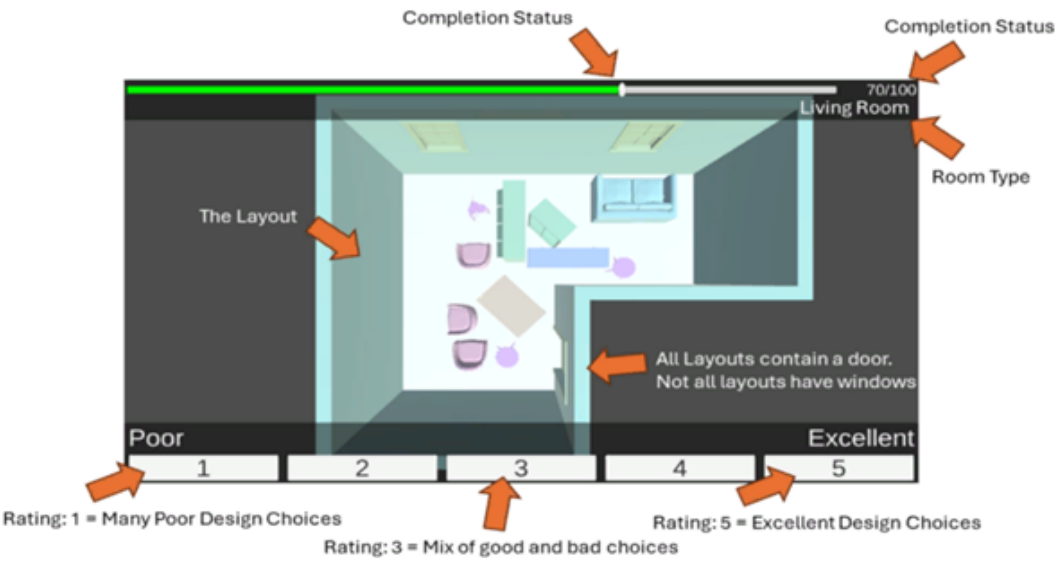


Figure 5.3: Survey website screenshot.

5.3 Perceived Quality Evaluation Survey

Perceived quality can vary from person to person. Someone with extensive knowledge of interior design might be able to identify a layout that functions better than another while others may not be able to explain exactly why a particular layout is good or bad. It is also possible that a design error can be deemed more significant to one person than to another who may consider it only a minor flaw. To get an approximate numerical representation of how good a layout looks i.e. to evaluate the perceived quality of the layout, a survey was created where participants evaluate a layout's quality on a scale from 1 (a poor design) to 5 (an excellent design)[2].

Survey participants were recruited using the University of Alberta university-wide weekly email postings and the Department of Computing Science Graduate Student Slack channel. Potential survey participants self-identified and registered in the survey through the survey website, linked in the recruitment post. A complete survey was expected to take no longer than 30 minutes, assuming they spend 10 to 15 seconds per layout, and participants were able

[2]This research received research ethics approval from the University of Alberta Research Ethics Board, Project Name "Interior Design Layout Data Collection and Evaluation", No. Pro00124859, February 28, 2024.

to exit and return to the survey at any time. Participants who have created layouts as part of the layout collection portion of the study were excluded from the layout evaluation survey portion of the study, although we could not enforce this given the anonymous nature of the survey. As an incentive to improve participant recruitment, we offered the option to enter a draw for one of three $50 gift cards.

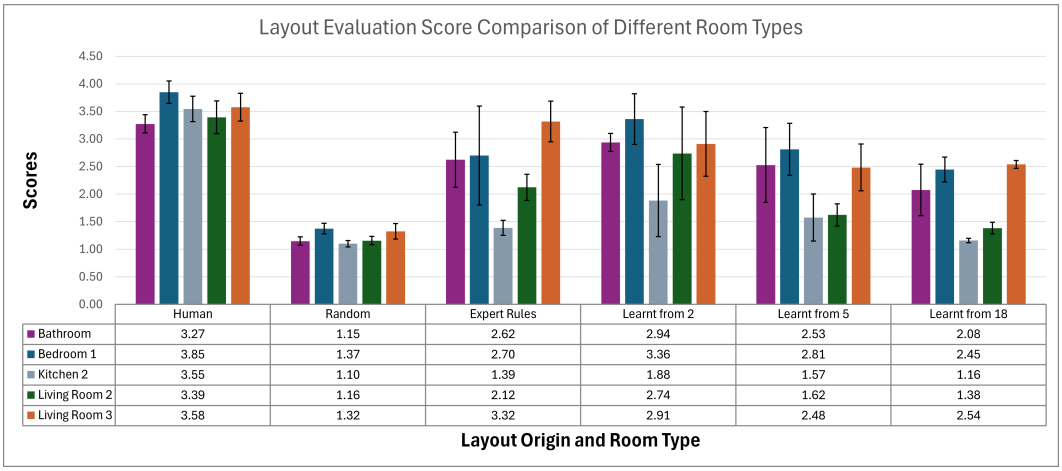
To manage the number of models shown to the users, the survey contained only five room scenarios of varying difficulty: Bedroom 1, Bathroom, Kitchen 2, Living Room 2 (Hard), and Living Room 3 (Easy). The total number of layouts compared was 215: 5 room scenarios each with 18 user-created, 5 random, 5 expert-rule, and 5 learnt of each 2-, 5- and 18-learnt-rule layouts.

The survey required participants to score 100 layouts of the possible 215 layouts to fully participate. Survey participants were each asked to score layouts one at a time based on perceived quality on a scale from 1 (a poor design) to 5 (an excellent design); the survey UI can be seen in Figure 5.3.

Participants would not see the same layout twice and duplicates were not shown but rather given the same score (although there was only 1 duplicate layout). To evenly distribute the layouts, such that all layouts received a roughly equal number of evaluations, the number of times a layout was evaluated was recorded. When the next layout to evaluate was "fetched", the layouts that that user had not evaluated yet were given a probability to be selected based on their previous evaluation frequency from all participants. This meant a layout that the user had not seen and had few total evaluations was given a very high probability of being shown next. The probabilities were given a soft minimum or:

$$p(Layout) = \frac{e^{-l}}{\sum_{i=1}^{k} e^{-l_i}}$$

Where l is the number of times the layout has been scored by all participants and k is the number of layouts not evaluated by the participant.



	Human	Random	Expert Rules	Learnt from 2	Learnt from 5	Learnt from 18
Bathroom	3.27	1.15	2.62	2.94	2.53	2.08
Bedroom 1	3.85	1.37	2.70	3.36	2.81	2.45
Kitchen 2	3.55	1.10	1.39	1.88	1.57	1.16
Living Room 2	3.39	1.16	2.12	2.74	1.62	1.38
Living Room 3	3.58	1.32	3.32	2.91	2.48	2.54

Figure 5.4: Results of the layout evaluation survey.

5.4 Survey Results

A total of 153 participants registered for the survey. 120 of those fully completed the survey and all participants had score average variance values greater than 0.1 meaning they did not give the same score to all layouts. The minimum total number of participant scores received by a layout was 56.

Figure 5.4 shows the average scores for each of the room scenario layouts grouped by the different creation methods. User layouts scored the highest, consistently above 3.0 on average, while unsurprisingly, random layouts scored below 1.5. Expert-rule-created layouts scored moderately well for bathrooms, bedrooms, and easy living rooms, while hard living rooms and kitchens scored low, the latter significantly low. Layouts that were created learning from two layouts performed well for all scenarios except the kitchen scenario. Learning from 5 and 18 layouts had moderate performance for bathrooms, bedrooms, and easy living rooms, while hard living rooms and kitchens again scored low.

5.5 Conclusion

This study provided insight into the ability of the rule learning algorithm to transfer perceived quality from the input user-created layouts to the output learnt-rule generated layouts. From these results, we can deduce that in terms of perceived quality, the rule learning performance is stronger for the easier

room scenarios and when fewer training layouts are provided.

While we showed in Chapter 4 that the rule learner is capable of learning rules that correlate with the expert rules that guided the creation of its training layouts, the rule learning ability with non-rule-guided perceived-quality training layouts was unclear. To test this, we trained with layouts that had a high level of perceived quality, because they were created by participants, and ran them through the rule learner. Then generated layouts based on the output learnt rules. This was an interesting case as unlike before, the rules of the layouts are unknown or more importantly do not explicitly follow a guiding layout ruleset because participants were given free creative range. The test was to see if the rule learner and automated layout generator could create layouts of equally high perceived quality as the input training layouts.

Through a survey of layout perceived quality, our findings suggest that for certain scenarios, layouts of high perceived quality can be created when a few training layouts are provided. More user-created training layouts reduce the learnt-rule layouts' perceived quality. At first glance, this may appear counter to the correlation experiment which shows that more training layouts improve performance, however, the user-created layouts did not concretely follow any rulesets. The difference between the two experiments is that there are fewer (or possibly no) patterns in the user-created training layouts, which is more pronounced as the training count increases. Conversely, the expert-rule training layouts all follow the same expert rules and thus contain some intrinsic design patterns which are better captured as the training count increases.

This case study not only evaluates the rule learning algorithm's ability to capture perceived quality but also gives some insight into the variability of user-created layouts and how the perceived quality of a layout differs from person to person.

In addition to the algorithm limitations in Chapter 4, a limitation of this study is that the participants who created the layouts did not self-identify as experts, i.e. they all rated their expertise in interior design at a score of 3 or lower out of 5. Therefore, the quality of the user-generated layouts may have been higher had we recruited interior designers. It is possible that

experts would have followed more similar guidelines and thus we could have potentially gotten higher quality learnt layouts as well. The quality of the layouts was also not known at the time when the layouts were input into the rule learner, as the quality survey was only once all layouts were collected, both user and computer-generated. A better approach may have been to first collect the layouts, then score the layouts through the survey, then use the top-scored layouts only and eliminate layouts with lower perceived quality. Having higher perceived quality layouts as input may have also resulted in higher perceived quality layouts being learnt. Finally, we will implement and compare DDLS algorithms to see how our algorithm compares to state-of-the-art methods on different datasets.

Chapter 6

BIM-kit: The BIM Reasoning Toolkit

In this chapter, I describe BIM-kit, our in-house toolkit for developing reasoning software applications. This project was built out of the necessity for a more easy-to-use and interoperable set of tools and libraries for building and testing the research in this thesis. The core of BIM-kit is the lightweight BIM data model. It contains few components yet is expressive enough to perform the reasoning tasks I have developed and tested it on. In this chapter, I describe the BIM-kit data model and the reasoning use cases for which it has been developed. I anticipate BIM-kit will continue to be an invaluable toolkit for continued development and testing of future research and application.

This chapter contains the work presented in the conference publication: **Christoph Sydora** and Eleni Stroulia, "BIM-kit: An Extendible Toolkit for Reasoning about Building Information Models," In Proceedings of European Conference on Computing in Construction (EC3), 2021. [3]

Abstract: Since 2010, research on cloud-based Building Information Modeling (BIM) has been receiving increased attention, motivated by the need to increase productivity through interoperability in the construction industry. To date, Industry Foundation Classes (IFC) is the de-facto standard for data exchange among tools in this domain. However, IFC is too large and not sufficiently specific to effectively support the management of a single building model by multiple tools. This paper proposes BIM-kit, a collaborative BIM

platform based on a simple, modular data schema. In this research, we demonstrate how multiple task-specific tools can be used collectively and efficiently in a shared cloud-based BIM environment to support a variety of use cases, across the overall building-design activity.

6.1 Introduction

The construction industry is increasingly being digitized and a variety of tools are being developed to support design-and-construction activities, from building design, materials procurement, construction project management, interior design, and building usage simulation. Collectively, these tools and the process of constructing digital building models are known today as Building Information Modeling (BIM). Towards enabling the interoperability of these BIM tools, the industry developed the Industry Foundation Classes (IFC) [94] interchange format.

IFC is a structured textual format that supports the representation of a BIM model, i.e., most information about a building, including its structural elements, i.e., walls, floors, doors etc., their relations, such as adjacency for example, and their geometries. An essential first step towards interoperability, IFC suffers from a number of shortcomings. First, the complexity of the information required by the potentially many building-construction tools makes the IFC schema quite large. In IFC2x3 (2006), there are 653 entities and 327 types (117 defined types, 164 enumeration types, and 46 select types). By 2015, when IFC4 [98] was released, the numbers had grown to 766 entities and 391 types [8]. In effect, the IFC representation of a building is a quite verbose textual representation of everything that a tool knows about the building at a specific point in time. Because of the size and complexity of the IFC schema, Model View Definition (MVD)s were created to filter the schema into subsets of types and entities relevant to different activities. As a result, even the tools that import and export IFC models are unlikely to understand and use all their elements; instead, they only work with the partial subset of IFC data that is covered by their domain-specific MVD, possibly causing ambiguities

and inconsistencies in the overall model. Finally, large as the IFC may be, it cannot include all the information required by all the tools that may consume it, since the level of detail required by each tool for each IFC element depends on the tool functionality. As a result, each tool tends to add to it additional proprietary information that is not meaningful to, and gets ignored by, other tools.

As long as one uses IFC as an archival representation of the work-product of a tool, these shortcomings are not particularly problematic. They become much more pronounced, however, when one considers the opportunity of a number of users working on different aspects of a building on their own preferred tools, as envisioned, for example, by cloud-based BIM [99]. Current solutions offer shared repositories of IFC building models and a set of Application Programmable Interface (API)s through which third-party tools can download the most recent state of the model or upload their modifications to it. As we discussed above, the inconsistencies and ambiguities that each tool introduces as they modify their own MVD-specific model subsets make this visionary workflow practically impossible.

A number of recent publications, which we discuss in detail in the next section, recognize the shortcomings of IFC's one-(big)-size-fits-all approach. In this Chapter, we describe BIM-kit[1], our research collaborative platform that relies on a different type of model sharing: instead of requiring that each tool exports and consumes a complete IFC model, the shared BIM-kit repository is organized around small cohesive elements, cross-referenced with each other. At the core of this organization are a set of well-defined basic geometry properties and relations, applicable to an extendible hierarchy of object types.

As we will demonstrate, this approach supports commonly sought after use cases of cloud-based BIM, such as model-checking applications, building simulation, viewing and editing applications, and automated design, that are at best challenging to achieve using the standard practice of IFC-centric cloud repositories. In fact, there is no demonstration of IFC seamlessly supporting

[1]BIM-kit code can be found at: https://github.com/csydora/BIMkit

all these tasks. Therefore, BIM-kit is targeted towards both designers that wish to utilize multiple automated cloud-based BIM services, and the developers of BIM services such that standardized and comprehensive models are maintained.

The BIM-kit repository is implemented in MongoDB, a document-centric NoSQL database that naturally supports this kind of representation. To ensure the consistent manipulation of the repository objects, BIM-kit implements a number of APIs to ensure that the objects are correctly updated by the tools integrated with it.

The key elements of the BIM-kit repository are the building models, a catalogue of objects which are referenced in the building models, a list of properties and relations, and a taxonomy of object types. To date, BIM-kit integrates six different tools, all of which exchange data with the central repository. Some of them support users to visually experience and manipulate the models and others implement completely automated model-manipulation services. At a high level, the currently available BIM-kit tools are the following:

1. The *Building Model Editor*, implemented in the Unity Game Engine, enables users to select items from a catalogue of available and pre-designed objects and place them in the building model.

2. The *Rules Management Service* manages a design-rules repository, which stores building codes in a structured format, such that they can be interpreted and executed on a building model.

3. Three *Rule Editors*, each of which enables users to create and modify design rules and to store them in the Rules Management Service. Each editor offers a different user experience allowing end users to use the editor that best suits their abilities or preferences.

4. A *Rule Learning Service* which learns and produces rules from an example layout dataset (see Chapter 4).

5. The *Model Checking Service* takes as input a set of building code rules from the Rules Management Service and the building model from the

BIM-kit repository and returns as output a set of references to model elements that relate to the code rules.

6. The *Generative Design Service* takes the functionality of the Model Checking Service one step further: using a set of rules from the Rules Management Service and an initially empty building design from the central BIM-kit repository, it returns a model that includes a set of desired objects, placed in a manner that respects all relevant rules. The service offers three algorithms, Jump Search (JS), Grid Search (GS), and Simulated Annealing (SA), which have varying degrees of layout search to rule quality efficiency trade-offs (see Chapter 3).

7. Finally, the *Model Occupancy Simulator* takes as input a building model, and given a configuration of occupants and their access to the building spaces, it simulates the occupants' activities in the building and returns a set of interesting usage indicators.

The objective of this research is to describe a cloud-based BIM solution that supports a number of use cases, which include automated design and its supporting model evaluations. We argue that this work makes the following two contributions to the state-of-the-art. First, it puts forward a well-defined, extendible BIM model; the BIM-kit model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set cross-referenced objects. At the same time, because the model consists of a number of distinct objects that are manipulated through well-defined APIs, the model can be more easily shared across different tools. The repository design enables the extension of the model with additional objects, all of which are supported by a core set of geometrical definitions. Second, it demonstrates the usefulness of the above model through the implementation of a variety of interactive and automated tools that, together, cover a wide range of activities that reason about building models. We envision BIM-kit will be a valuable research testbed for building design applications with reduced training overhead.

The remainder of this chapter is organized as follows. Section 6.2 outlines the BIM-kit data model, storage, and access APIs. Following this description,

the tools currently available on BIM-kit, their functionalities and their implementation status are reviewed in Section 6.3. In Section 6.4 we elaborate on our experiments to date with these tools, and Section 6.5 concludes with a summary of the lessons learned through the BIM-kit development process and our plans for future development.

Chapter 2 Section 2.5 reviews the related work and highlights the advantages of BIM-kit relative to similar efforts.

6.2 BIM-kit Data Model & Repository

6.2.1 Data Model

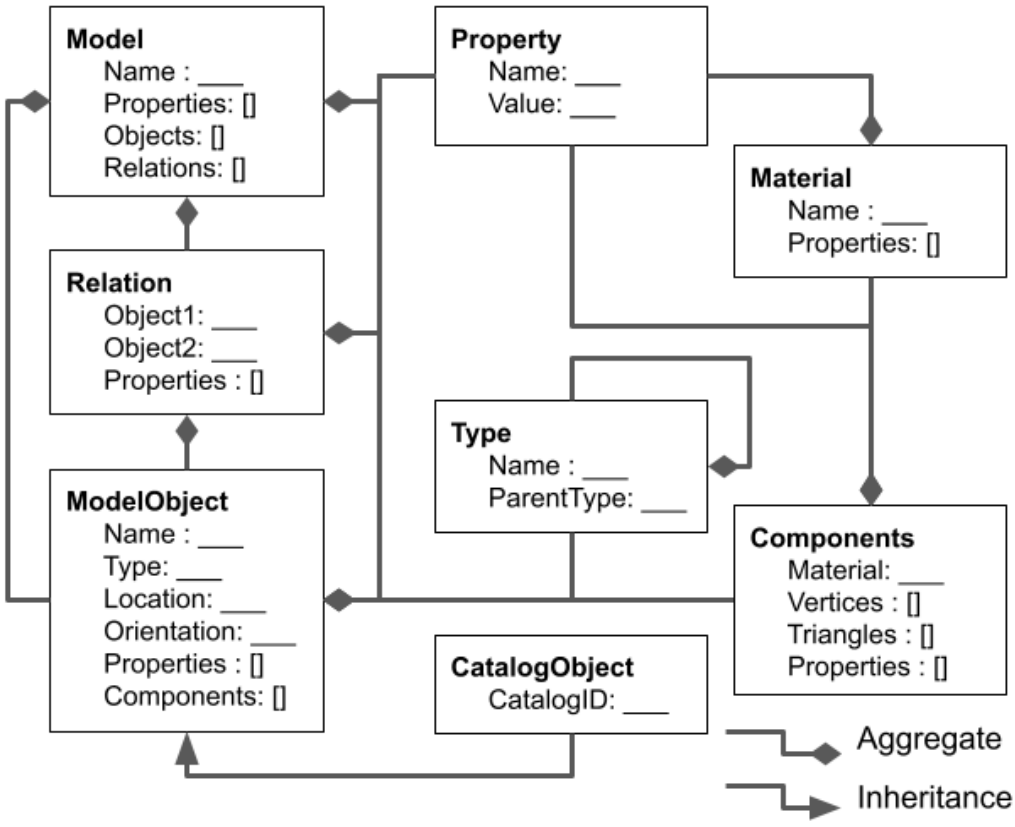


Figure 6.1: BIM-kit client-side data model.

BIM-kit data is stored in MongoDB as document classes, in BSON, a format similar to JSON. The MongoDB schema contains five core element

classes, namely *Model*, *Object*, *Material*, *Type*, and *User*. Client applications "see" the client-side schema, diagrammatically depicted in Figure 6.1.

The *Model* class is the root of the building model, identified by a unique ID and a name. It contains two types of objects, which are the items inside a building: *CatalogueObjects* and *ModelObjects*. On the client side, both will appear to be the same, however, *CatalogueObjects* in the MongoDB *Model* are merely references to the elements in the MongoDB object catalogue. Thus, on the client side *CatalogueObjects* contain the additional *CatalogueID* field, but when stored in the MongoDB *Model*, only contain information about where they should be placed in the space, i.e., (*Location* and *Orientation*). *ModelObjects*, on the other hand, are objects that cannot be found in the catalogue, either because they have been constructed for the specific building, or because they do not have a fixed geometry but rather one that depends on the geometry of other objects. These would generally include Walls, Floors, and Ceilings, but can also include what we define as Virtual Objects such as Rooms/Spaces, Paths, and Floor Levels. For this reason, *ModelObjects* must store the actual geometrical shape representations in both client-side and the MongoDB *Model* as a list of *Components* along with object *Type*. The *Model*, *CatalogueObjects*, and *ModelObjects* all have a *Properties* collection associated with them.

MongoDB catalogue objects are independent of specific building *Models*; their internal structure is similar to that of the custom *ModelObjects* since they describe a geometrical shape, basic geometrical properties, and the object's *Type*. However, each MongoDB catalogue object may be associated with a number of shape representations (also known as MeshRep), which are a list of *Components* for different LODs. When a model is requested and MongoDB catalogue objects are reinserted back into the client-side *Model*, the user specifies the LOD those objects in the model will be in. This LOD is particularly important for applications that are restricted to the size of the model due to performance limitations. For example, an automated design method may first reason about very coarse objects, such as bounding boxes, and switch to displaying the objects in high resolution when a final output solution is determined.

The idea of separating objects from a catalogue from building *Models* is motivated by three reasons. The first is storage efficiency: a single catalogue object can be referenced, instead of being cloned, in multiple models. At the same time, computational efficiency is supported by the multiple LOD for each object. The second reason is the need to standardize object usage as much as possible. Having a catalogue can streamline the model-checking process, by reducing the need to recalculate common object geometric properties repeatedly, reducing searches by geometry and eliminating the need for redundant recalculation. Finally, this design decision aligns with objects in the real world, and could potentially increase the relevance of such a tool to product retailers.

In addition to the objects, *Models* also include a list of *Relations* between two related objects. They contain a reference to the two objects, that can be either *CatalogueObjects* or *ModelObjects*, and a set of geometric relations that hold true between the two. Two common property examples would be a relation between two objects with a distance property, or a property that indicates one is an aggregate of another object [100].

The next two element types are meant to standardize the way items are labelled. *Types* define a tree structure, or a taxonomy, meaning each *Type* has a parent *Type*. Ideally, *ModelObjects* and *CatalogueObjects* should only use the leaf *Types*, with intermediate *Types* only being used for search.

The *Material* element standardizes the material names, such that rendering applications can expect certain material values and properties for display, while also having a list of properties for physical attributes.

User documents exist in order to restrict model access, by storing ownership and accessibility. For now, only users with administration access can add new *CatalogueObjects*, although in the future this can be changed such that *CatalogueObjects*, like models, have owners. Additionally, private objects could be used to save *ModelObjects* for each user, thus increasing object reusability.

6.2.2 BIM-kit Repository

The BIM-kit Repository is the service in charge of the storage and retrieval of data to and from the MongoDB store. While the data is stored in five separated document classes, the models and objects being sent to and received from the BIM-kit Repository take client-side schema in Figure 6.1. This is where the references to the MongoDB catalogue objects in the model are replaced with the actual stored objects, but only using the shape representation LOD defined in the request. Therefore, the BIM-kit Repository receives the model ID and LOD in the request and reconstructs the data to form what is then returned to the client.

The BIM-kit Repository is also responsible for validity in the model. For instance, if a model is uploaded that does not satisfy the structure, or invalid ID references, then the BIM-kit Repository will rectify those mistakes to the best of its ability or else it will not perform the request.

In its current state, the Building Model Editor in BIM-kit is not able to create walls and floors. Therefore, models in BIM-kit initially come from IFC files, exported from other BIM tools and converted to the BIM-kit client side data structure using the Data Conversion tool. Once a model is uploaded, BIM-kit provides a number of services and tools specific to the tasks of stakeholders. A key functionality of the BIM-kit Repository is the ability to reason about the model and add an additional layer of information to the model in the form of properties, relations, and Virtual Objects. This additional information layer can then be further used by other BIM-kit services to form a common semantic basis for performing a variety of tasks. This is achieved through a shared geometrical library which contains methods for adding property, relation, and Virtual Object information to a model.

6.3 Use Cases

Figure A.1 shows the current state of the BIM-kit architecture and the flow of data between the repository, and integrated services, and applications. The following subsections describe each of the components of the ecosystem around

the BIM-kit Repository, and the task(s) they perform.

6.3.1 Building Model Editor

A key functionality for any design tool is a visual editor for users to interact with the design artifact. For building designs, editing is typically done in a 3D environment, and we constructed our editor using Unity [101] taking advantage of the out-of-the-box features such as 3D rendering (and in future iterations Augmented Reality (AR)/Virtual Reality (VR) and physics). The editor uses the BIMKitApi library, containing methods for sending requests to the BIM-kit Repository, for data retrieval from the BIM-kit Repository. Edits to the model are made locally and only once the model is saved is it uploaded to the BIM-kit Repository. Currently, the model editor is only able to add catalogue objects to an existing model.

The model editor will over time add plugins for accessing external services. We will also implement VR and AR features such as the work done in our previous work - see [102].

6.3.2 Rule Management Service

The Rule Management Service is responsible for managing sets of rules, relevant to building designs. It relies on a rule repository, also implemented in MongoDB, accessible to three different editors for users to specify rules. The schema of the Rule Management Service repository, derived from Sydora and Stroulia [1], can be seen in Figure 6.2.

Rules store all the information needed to compile into executable code that can be invoked by the BIM-kit Model Checking Service. **Rulesets** are collections of references to **Rules**: meaning if a rule changes, all the rulesets that include this rule are automatically updated to point to the latest version of that rule. This ensures that as building codes evolve or the experts' understanding of ergonomics deepens, changes to individual rules are propagated in all sets that consider this rule as relevant, thus enabling consistency. When a ruleset is requested, the Rule Management Service constructs the ruleset by finding and linking with referenced rules. For the Rule Management Service

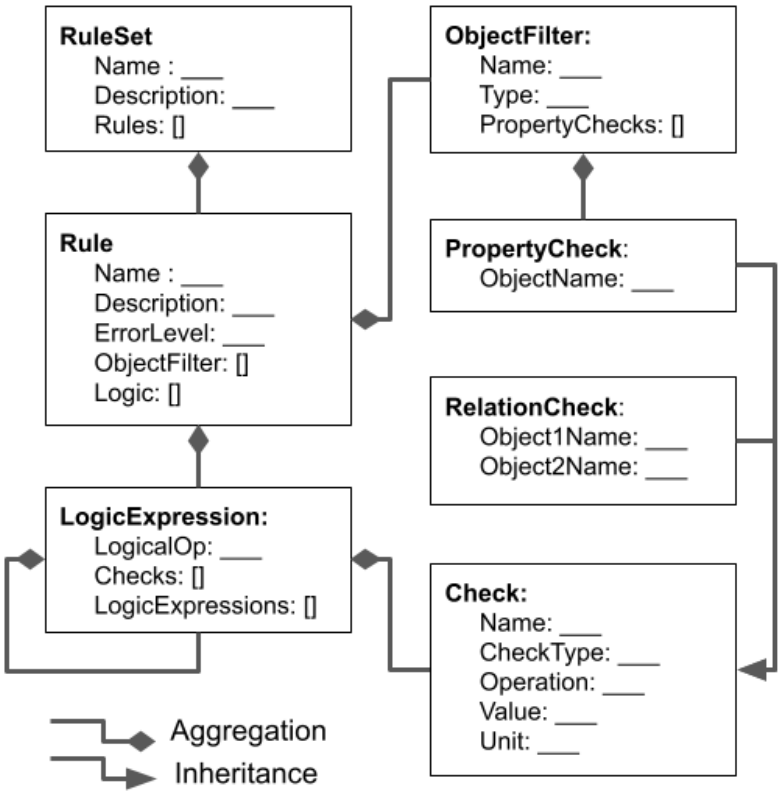


Figure 6.2: Rule Management Service data storage model.

prototype security is limited to only a username, and only the user's public name is saved in the model. The Rule Management Service provides RESTful API calls for making changes to individual rules and rulesets. Additionally, it performs validity checks to ensure proper authorization and completeness.

As with the BIM-kit Repository, the Rule Management Service has a corresponding client-side RMSApi library which provides methods for sending rule requests and parsing the results. Also included is the Rule Administration Application which is an application for managing the Rule Management Service.

6.3.3 Rule Editors

There are currently three Rule Editors (RE) available in BIM-kit for creating rules, to be used for model checking and generative design: a visual editor based on [14] (BlocklyRE), a textual editor based on a domain-specific struc-

tured language (DslRE), and an editor based on natural language processing (NlpRE). All three editors create and edit rule objects in the Rule Management Service, but each one was designed to support a different user-interaction model, to appeal to different types of users with different types of technical backgrounds.

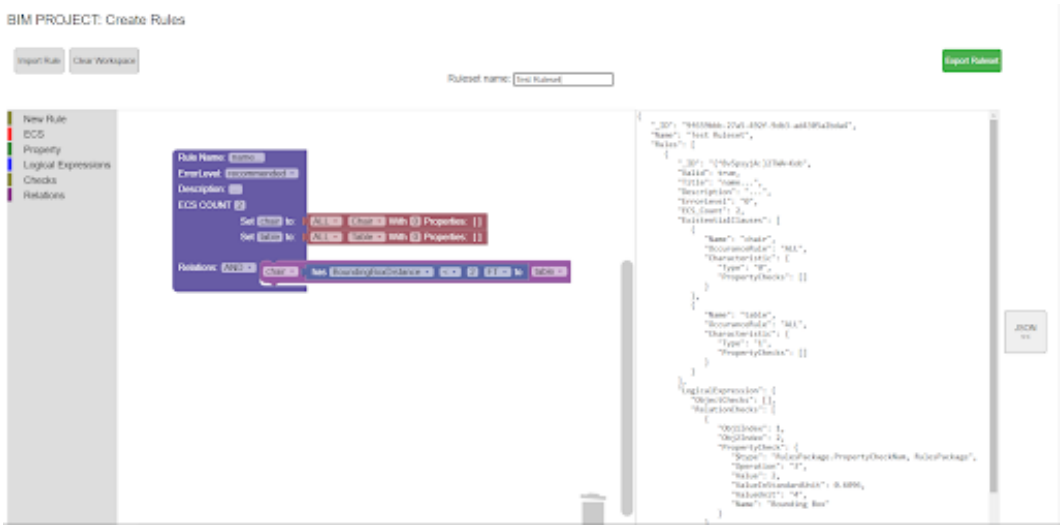


Figure 6.3: Blockly Rule Editor.

The BlocklyRE, as seen in Figure 6.3, uses puzzle-like pieces, each one corresponding to an element of the rule object schema, to construct the rules. The DslRE is a .NET WindowsForm application that guides users' actions by only providing a selectable list of valid options to develop a textual representation of the rules. Finally, the NlpRE enables a user to describe in (a restricted) natural language a rule and automatically parses this description into a rule object, as expected by the Rule Management Service. The user can then make modifications to the parsed rule to better reflect the input text's intended meaning. The comparison of these three editors is left to a future study, however, each has the same overall functionality in creating, editing, and uploading rules.

Each rule editor communicates directly with the BIM-kit Repository to retrieve the available object **Types** so that they can be included in the specification of the objects to which the edited rules may apply.

6.3.4 Rule Leaning Application

The rule learning application is developed using the rule learning algorithm from Chapter 4 of this thesis. The workflow is an input dataset is selected and passed through the rule learner. The user then specified the rules to be learnt, either a specific set of rules for two object types or the set of all possible rules for all type pairs in the examples. Rules in the RuleDSL format are learnt and returned to the user which can be used in any RuleDSL supporting tool: the Rule Editors, Model Checking, and Generative design.

6.3.5 Model Checking Service

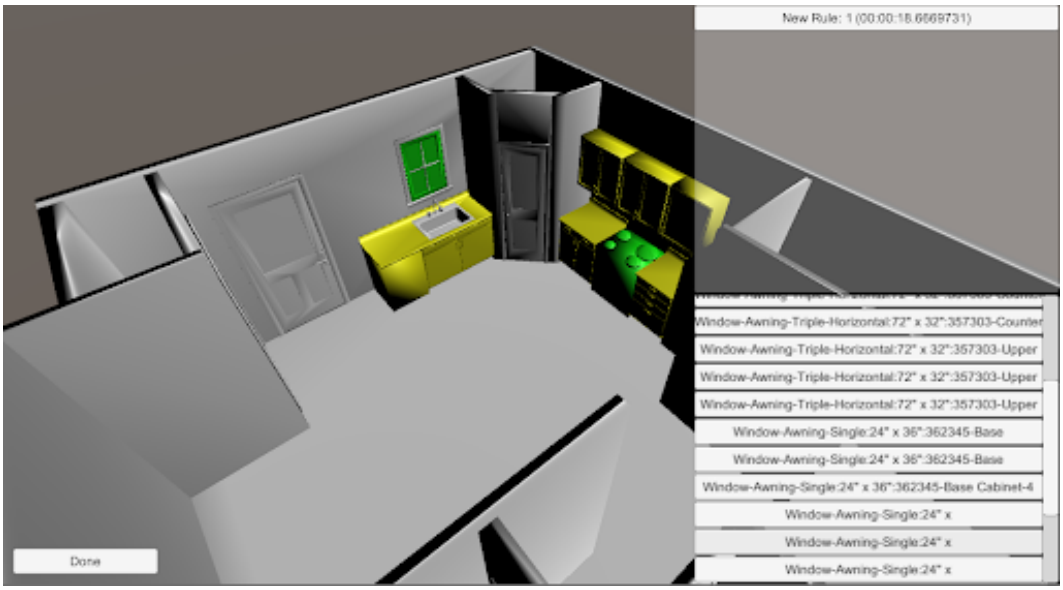


Figure 6.4: BIM-kit Model checking extension in the Building Model Editor.

The Model Checking Service is an automated service that can be invoked by any tool in the BIM-kit ecosystem, or by other third-party client applications. The invoking application must provide a user's credentials as access tokens, in order to ensure that the end user of the invoking application is authorized to access the requested building model and validate it against the chosen ruleset. The Model Checking Service retrieves the model from the BIM-kit Repository and the rules from the Rule Management Service, which it then compiles into executable methods. As a result of executing these methods on the building

model, a check result is created and returned to the invoking client. The check result is a list of rule results, which includes a reference to the rule and to the object(s) in the building model that is responsible for the rule result. As an example, Figure 6.4 shows the results from a rule, with the yellow objects being objects relevant to the rule, and the green objects indicating an instance of the rule that has passed the rule check. The interested reader can find an in-depth description of the model-checking method in Sydora and Stroulia [1].

A key feature of the Model Checking Service is that, in order to evaluate the rules in the context of the building model, it typically calculates the values of a variety of geometrical properties and relations, namely these properties and relations of interest to the invoked rules. These values are stored with the building model, and cross-referenced with their associated objects. In this manner, they can be reused by subsequent model checks. If, at any point, the model objects on which these properties and relations depend move or change, the values become obsolete and are, therefore, deleted.

6.3.6 Generative Design Service

Relying on the Model Checking Service, we also developed an automated Generative Design Services that automates the placement of BIM-kit catalog objects in a building model such that the placements validate all the relevant design rules. As with the Model Checking Service, the Generative Design Service acts as a proxy for the client application and also accesses the model and rules from the BIM-kit Repository and Rule Management Service. In addition, the end user provides as input a list of desired objects from the MongoDB catalogue to be included in the building model.

The Generative Design Service included in the BIM-kit contains the three generation algorithms described in Chapter 3 of this thesis. When a configuration solution is found, a copy of the model with the object placement configuration is created by the Generative Design Service and shared with the user. The user can then save the configuration over the existing model or discard the Generative Design Service solution.

6.3.7 Model Occupancy Simulator

In 2014, Wong et al. [103] reported that very few CloudBIM projects focused on operation and facility management. In addition to design, building models can be used to manage buildings and simulate activity. In that vein, we recently developed a prototype building-occupancy simulator, based on BIM-kit models. The simulator uses the model data to construct a set of spaces and paths that are accessible to the simulation agents. In our case, the goal was to simulate the risks associated with an infectious pathogen under different building operation scenarios, such as restricting room or floor access or limiting building occupancy capacity. More generally, however, the simulator acts as an additional evaluation in which time is a factor as elements of the model and the use of the spaces are dynamically changing due to the movements of building occupants.

The simulator is a Unity [101] application that accesses a building model from the BIM-kit Repository and renders the scene, using the same methodology as the Building Model Editor. A navigation mesh (NavMesh), which represents the walkable areas of the building is constructed based on the floor components in the model. Paths can be generated by a combination of the NavMesh and adjacency relations extracted from the model, as the NavMesh alone is unable to recognize level-to-level transitions such as elevators. Finally, the simulator takes as input a series of time-stamped agent activities, generated by a special-purpose agent-trace generation tool given a start location and an end location, and simulates the agent's behaviour. During the simulation, the simulator records a variety of interesting indicators, including agent interactions, and occupancy levels at the individual room and zone levels.

A case study on the occupancy simulator for predicting occupant infectious disease risks can be seen in Sydora et al. [104].

6.4 Discussion

6.4.1 Advancing Interoperability

BIM-kit is conceived to address the challenge of maintaining a consistent representation of a building model, while at the same time, making it available to a variety of tools for different types of reasoning and design activities. This is a primary problem with IFC, which oftentimes is referred to as a "view only" version of all building information; modification of an IFC file has to be requested from the designer rather than done directly on the model. On the other hand, BIM-kit allows direct modification of the model, ensuring its integrity through its well-defined modular schema. At the same time, the BIM-kit Repository can ingest and export IFC models, thus ensuring a degree of interoperability with current IFC-based software.

Compared with previous cloud-based BIM approaches, the underlying BIM-kit schema is concise in its representation, but as demonstrated, not limited to a single application and use case.

6.4.2 Semantic Modeling

The underlying semantic enrichment and rule-checking follow much of the same motivations as the work of Pauwels and Zhang [20], Sacks et al. [105], and Hagedorn and König [106]. BIM-kit has been designed with the expectation that models have little to no information outside of geometrical information. Therefore, the power of BIM-kit is to support the enrichment of the models with semantically meaningful information, such as, for example, the evaluation of interesting geometric relationships between model objects. Such relationships, e.g., alignment, distance in different dimensions, etc, are useful to users exploring the models through the editor and necessary to the model-checking service evaluating the rulesets to which the model may adhere.

In BIM-kit, the Rule Management Service, the Rule Editors, and the Model Checking Service provide key functionality, essential for building-model management, namely the definition of logical and geometrical constraints and their validation against building models. This set of tools, in effect, supports the

specification of operational semantics behind the model representation.

6.5 Conclusion

In this Chapter, we discussed our work on BIM-kit, a set of tools supporting a number of reasoning tasks about building models. The BIM-kit set of federated tools relies on a central repository of building models, represented in a well-defined modular and extendible schema.

The BIM-kit schema covers the same information as Industry Foundation Classes (IFC) but is fundamentally different from IFC because it is modular, i.e., it defines a set of simple, highly cohesive classes, cross-referenced with each other. The modular nature of building representation in BIM-kit supports the management of the overall consistency of the models: because the repository contents are manipulated through well-defined APIs, building models can be more easily shared across different tools. Also because of its modularity, the BIM-kit schema supports extendibility. At the same time, the BIM-kit model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set cross-referenced objects, and it can export an IFC model by collecting the relevant classes corresponding to a single model.

The variety of tools we have developed around the BIM-kit Repository constitutes strong and persuasive evidence of the effectiveness of the underlying representation. The six interactive and automated tools around the BIM-kit Repository cover a wide range of design activities, across the building lifecycle, from interactive manipulation, to rule-based validation, to automatic generation of design alternatives, to usage simulation.

Chapter 7

Conclusion

The goal of this thesis is to develop and evaluate new algorithms for automating a variety of reasoning tasks around interior building design, from model checking to automated layout generation, to learning design rules. The algorithms presented in this thesis use or create rules in the form of RuleDSL a user-friendly DSL for describing rules on building components' geometric relationships. Specifically, this thesis investigated and addressed the following research questions.

Question 1: Two key criteria for automated layout generation algorithms are (i) the quality of the layouts in terms of the search goal, in our case the rule compliance score and (ii) how fast a search algorithm converges on a high-quality layout configuration i.e., its search efficiency. One grid-based methodology and two continuous random sampling algorithms are compared based on these two metrics. **How do output layout quality and layout configuration search efficiency trade-offs differ among the layout generation algorithms?**

Answer 1: In Chapter 3 of this thesis, I present a new search algorithm called Jump Search (JS). Unlike the grid-based algorithms, such as the previous GS algorithm, JS does not depend on a predefined grid but rather uses random samples from a user-specified radius for its potential placement locations. A Simulated Annealing (SA) alternative is also tested as it selects locations more cautiously as opposed to the greedy selection in JS. To compare,

each algorithm is given a budget of search iterations and the final layout they produced is evaluated for rule compliance. The experimental results demonstrate that, given the same budget, the JS algorithm produced equal to or better rule-compliant layouts than the GS and SA algorithms. Additionally, because of its grid reliance, GS is found to provide little variation in layouts which may be considered a negative aspect.

Contribution: The contribution of Chapter 3 is a new continuous space search algorithm, Jump Search (JS). I have demonstrated the good performance of this novel algorithm, and its effectiveness in delivering high-quality layouts in many different problem scenarios, by comparing it against a previous grid-based method, Grid Search (GS), and a non-greedy continuous space search algorithm based on Simulated Annealing (SA).

Question 2: Design rules are typically edited through a special-purpose syntax-aware editor. To date, three RuleDSL editors have been developed. One is a Windows form-based editor where rules are created through a series of dynamic drop-down and data-input widgets. The others are prototypes for a VPL using BLOCKLY and NLP text-to-code tokenizer which are still being tested and evaluated. Coding rules is difficult and while the focused RuleDSL vocabulary on interior design eases the coding process, the syntax and meaning behind relation terms can still be the root of some challenges. A potentially simpler rule creation process might be to only provide examples where the rules are adhered to and ideally have the rules extracted automatically. My algorithm approach looks at the relation patterns in the examples and the range of their values, rules can be formulated on the bounds of the relation ranges and converted to the RuleDSL. **How well does the design-rule learning algorithm capture the quality of the layouts of its input examples?**

Answer 2: The answer to this question is the focus of Chapter 4. In this chapter, a rule-learning algorithm is presented, using relation value ranges as

the basis for learning patterns in the examples. To evaluate how well the proposed algorithm captures expert knowledge, a dataset of synthetic layouts of different room scenarios is used. The advantage of using synthetic layouts is that the synthetic layouts follow known expert rules. This leads to the intuition that a high-quality rule learner would be able to learn rules whose check scores are highly correlated to the check scores of the expert rules that the input examples followed. In short, a high rule score correlation implies the rule learner can create rules that capture expert rule information. From the rule correlation analysis, the rule learning algorithm can capture much of the expert rule information. Performance is dependent on the variety and room scenario of the input.

Contribution: The contribution of Chapter 4 is a novel rule learning algorithm whose quality and effectiveness at capturing known rule information is evaluated through a rule correlations analysis.

Question 3: Rule compliance is not the only metric for evaluating the quality of a room layout. Often users will have preferences for their arrangements that they may not be able to express in geometrical terms like the rules. Therefore, in addition to capturing rule information, the ability of the rule learning algorithm to capture the implicit non-trivial user preferences and aesthetics is also of interest to someone who may use the rules to automatically evaluate or generate layouts similar to a set of examples. **How well is implicit user-perceived quality captured and recreated using the same rule learning algorithm?**

Answer 3: To answer this question, user-created layouts are used as input to the rule learning algorithm from Chapter 4. The learnt rules are then used in the automated layout generation for their respective room scenarios to create new learnt-rule layouts. A survey is developed to evaluate the perceived quality of the original layouts and the learnt-rule layouts for each room scenario. The results show that the perceived quality of the learn layouts is generally lower

than that of the user-created layouts but the extent of which depends on the room scenario and how many user-created layouts went into the rule learning.

Contribution: The contribution of Chapter 5 is an evaluation of how well the rule learning algorithm in Chapter 4 performs at capturing and recreating perceived quality.

Question 4: The final contribution of this thesis relates to the synthesis of all BIM reasoning applications as web services, in a coherent software platform. **What software architecture can be employed to support the RuleDSL and other BIM reasoning application Application Programmable Interface (API)s?**

Answer 4: Chapter 6 outlines the development environment I created to address some of the issues regarding interoperability and resource sharing among BIM reasoning applications. IFC is a verbose format meaning ingestion from one application to another can be inconsistent. I address this by using a basic model representation that relies only on key model components and relationships. Through multiple use cases, I show that the lightweight representation contains enough information to perform meaningful tasks.

Contribution: Chapter 6 makes the following two contributions to the state-of-the-art. First, it puts forward a well-defined, extendable BIM model; the model is compatible with IFC, in that it can import a complete IFC model and create a corresponding set of cross-referenced objects. At the same time, because the model consists of a number of distinct objects that are manipulated through well-defined APIs, the model can be more easily shared across different tools. The repository design enables the extension of the model with additional objects, all of which are supported by a core set of geometrical definitions. Second, it demonstrates the usefulness of the above model through the implementation of a variety of interactive and automated tools that, together, cover a wide range of activities that reason about building models. I envision BIM-kit will be a valuable research testbed for building design applications

with reduced training overhead.

7.1 Future Work

The body of research on this topic is far from complete. Many more interesting questions and applications remain which each take this work in new constructive directions. First, the expressiveness of RuleDSL itself can be extended. New rules and new application domains offer a unique opportunity to push the boundary of how far the RuleDSL can be used in practice. The RuleDSL, with only some new type and relation definitions, has been used in a case study on highway code checking [107]. This shows promise for the RuleDSL to be used outside the interior domain of this thesis and is evidence of the usefulness of the software and APIs in developing new prototypes in a short time.

The layout generation algorithms are only some of the possible methods that could be used for the placement of objects. I specifically use sequential object placement due to the higher likelihood of converging on results faster. Multi-object placement search algorithms could have some potential to improve results (likely as a higher budget) which might be worth comparing against. Alternatively, hierarchically grouping objects to form rule-compliant object clusters that can be added and moved as a group could improve this further. Parallel evaluations and better sampling would also improve algorithm efficiency. In short, there are a seemingly endless number of alternative search methods that could be tested with the RuleDSL as the evaluation step.

Next, the rule learning algorithm relies on a template of pre-selected relations which are deemed most likely to be relevant to furniture arrangement. This significantly limits the learnt rule expressiveness which is far from the full expressiveness the RuleDSL affords. An interesting problem would be how to build the learnt rule structure as opposed to limiting the structure. Possible approaches to this problem could be found in the research area of Program Synthesis [108].

The emergence of Large Language Models (LLMs) has drastically changed the research landscape in the last few years. The ability and usage of these

tools are still being heavily investigated but their ability to generate code based on natural language prompts has major implications for the work in this thesis. In effect, these tools may soon be able to drastically help designers write Automated Code Checking (ACC) code, however, to understand the code, a DSL would still provide explainability and readability to the user. Therefore, investigating an LLM's ability to generate DSL code will be investigated in the future.

Image generators have also improved drastically over the last few years thanks to Generative Adversarial Networks (GANs) [109] and more recent diffusion models [85]. These tools have been shown to generate high-quality building and interior images and as they continue to improve, their use in design evaluation and layout generation will continue to be analyzed in the future.

Finally, BIM-kit is an ongoing development project but its applicability to research projects and usage outside of interior reasoning has been growing. I believe that BIM-kit's simple data model and growing vocabulary of BIM terms could be of great value to anyone who is learning, testing, and validating their new and exciting BIM-related research.

References

[1] C. Sydora and E. Stroulia, "Rule-based compliance checking and generative design for building interiors using bim," *Automation in Construction*, vol. 120, p. 103 368, 2020.

[2] C. Sydora and E. Stroulia, "Comparative analysis of room generation methods using rule language-based evaluation in bim," in *European Conference on Computing in Construction (EC3)*, European Council on Computing in Construction, vol. 4, 2023.

[3] C. Sydora and E. Stroulia, "Bim-kit: An extendible toolkit for reasoning about building information models," in *European Conference on Computing in Construction (EC3)*, 2021, pp. 107–114.

[4] R. Sacks, C. Eastman, G. Lee, and P. Teicholz, *BIM handbook: A guide to building information modeling for owners, designers, engineers, contractors, and facility managers*. John Wiley & Sons, 2018.

[5] C. Eastman, J.-m. Lee, Y.-s. Jeong, and J.-k. Lee, "Automatic rule-based checking of building designs," *Automation in construction*, vol. 18, no. 8, pp. 1011–1033, 2009.

[6] J. K. Lee, "Building environment rule and analysis (bera) language and its application for evaluating building circulation and spatial program," *Doctoral dissertation, Georgia Institute of Technology*, 2011.

[7] J.-K. Lee, C. M. Eastman, and Y. C. Lee, "Implementation of a bim domain-specific language for the building environment rule and analysis," *Journal of Intelligent & Robotic Systems*, vol. 79, pp. 507–522, 2015.

[8] W. Solihin and C. Eastman, "A simplified bim model server on a big data platform," in *Proceedings of the 33rd CIB W78 Conference*, 2016.

[9] W. Solihin, J. Dimyadi, Y.-C. Lee, C. Eastman, and R. Amor, "Simplified schema queries for supporting bim-based rule-checking applications," *Automation in Construction*, vol. 117, p. 103 248, 2020.

[10] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, and V. Koltun, "Interactive furniture layout using interior design guidelines," *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, 87:1–10, 2011.

[11] M. Fisher, D. Ritchie, M. Savva, T. Funkhouser, and P. Hanrahan, "Example-based synthesis of 3d object arrangements," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 6, pp. 1–11, 2012.

[12] J. Betker, G. Goh, L. Jing, *et al.*, "Improving image generation with better captions," 2023.

[13] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 3836–3847.

[14] *Google blockly*, Accessed: August 15, 2024. [Online]. Available: `https://developers.google.com/blockly`.

[15] A. Borrmann, M. König, C. Koch, and J. Beetz, *Building information modeling technology foundations and industry practice: Technology foundations and industry practice*, 2018.

[16] J. Dimyadi and R. Amor, "Automated building code compliance checking–where is it at?" *Proceedings of the 19th CIB World Building Congress*, pp. 172–185, 2013.

[17] D. Greenwood, S. Lockley, S. Malsane, and J. Matthews, "Automated compliance checking using building information models," in *The Construction, Building and Real Estate Research Conference of the Royal Institution of Chartered Surveyors*, 2010.

[18] A. S. Ismail, K. N. Ali, and N. A. Iahad, "A review on bim-based automated code compliance checking system," *International Conference on Research and Innovation in Information Systems (ICRIIS)*, pp. 1–6, 2017.

[19] E. Hjelseth, "Bim-based model checking (bmc)," *Building Information Modeling–Applications and Practices*, pp. 33–61, 2015.

[20] P. Pauwels and S. Zhang, "Semantic rule-checking for regulation compliance checking: An overview of strategies and approaches," *Proceeding of the 32rd international CIB W78 conference*, 2015. [Online]. Available: `http://hdl.handle.net/1854/LU-6890589`.

[21] W. Solihin, J. Dimyadi, and Y.-C. Lee, "In search of open and practical language-driven bim-based automated rule checking systems," *Advances in Informatics and Computing in Civil and Construction Engineering*, pp. 577–584, 2019.

[22] *Solibri*, Accessed: August 15, 2024. [Online]. Available: `https://www.solibri.com/`.

[23] *Statsbygg bim manual, version 1.2.1(sbm1.2.1)*, 2013. [Online]. Available: `https://dok.statsbygg.no/wp-content/uploads/2020/06/statsbyggs-bim-manual-1-2-1_en_20131217.pdf`.

[24] *Nova group, corenet eplancheck*, Accessed: August 15, 2024. [Online]. Available: `https://www.nova-hub.com/e-government/`.

[25] L. Ding, R. Drogemuller, M. Rosenman, D. Marchant, and J. Gero, "Automating code checking for building designs-designcheck," *Clients Driving Innovation: Moving Ideas into Practice*, pp. 1–16, 2006.

[26] *Autodesk, revit*, Accessed: August 15, 2024. [Online]. Available: `https://www.autodesk.com/products/revit/overview`.

[27] *Graphisoft, archicad.* Accessed: August 15, 2024. [Online]. Available: `https://graphisoft.com/solutions/archicad`.

[28] *Autodesk, dynamo*, Accessed: August 15, 2024. [Online]. Available: `https://dynamobim.org/`.

[29] S. Davidson, *Grasshopper*, Accessed: August 15, 2024. [Online]. Available: `https://www.grasshopper3d.com/`.

[30] *Bimserver*, Accessed: August 15, 2024. [Online]. Available: `http://bimserver.org/`.

[31] P. Pauwels, D. Van Deursen, R. Verstraeten, *et al.*, "A semantic rule checking environment for building performance checking," *Automation in Construction*, vol. 20, no. 5, pp. 506–518, 2011.

[32] W. Mazairac and J. Beetz, "Bimql–an open query language for building information models," *Advanced Engineering Informatics*, vol. 27, no. 4, pp. 444–456, 2013.

[33] S. Wawan, "A simplified bim data representation using a relational database schema for an efficient rule checking system and its associated rule checking language," *Doctoral dissertation, Georgia Institute of Technology*, 2016.

[34] W. Solihin, J. Dimyadi, Y.-C. Lee, C. Eastman, and R. Amor, "The critical role of the accessible data for bim based automated rule checking system," *Proceedings of the Joint Conference on Computing in Construction (JC3)*, vol. 1, pp. 53–60, 2017.

[35] H. Lee, J.-K. Lee, S. Park, and I. Kim, "Translating building legislation into a computer-executable format for evaluating building permit requirements," *Automation in Construction*, vol. 71, pp. 49–61, 2016.

[36] S. Park, Y.-C. Lee, and J.-K. Lee, "Definition of a domain-specific language for korean building act sentences as an explicit computable form," *Journal of Information Technology in Construction (ITcon)*, vol. 21, pp. 422–433, 2016.

[37] J. Choi and I. Kim, "A methodology of building code checking system for building permission based on openbim," *Proceedings of the 34th International Symposium on Automation and Robotics in Construction (ISARC)*, vol. 34, pp. 945–950, 2017.

[38] R. Niemeijer, B. De Vriès, and J. Beetz, "Check-mate: Automatic constraint checking of ifc models," *Managing IT in construction/Managing construction for tomorrow*, pp. 479–486, 2009.

[39] C. Preidel and A. Borrmann, "Automated code compliance checking based on a visual language and building information modeling," *Proceedings of the 32nd International Symposium on Automation and Robotics in Construction and Mining (ISARC)*, vol. 32, pp. 1–8, 2015.

[40] C. Preidel and A. Borrmann, "Towards code compliance checking on the basis of a visual programming language," *Journal of Information Technology in Construction (ITcon)*, vol. 21, no. 25, pp. 402–421, 2016.

[41] C. Preidel and A. Borrmann, "Refinement of the visual code checking language for an automated checking of building information models regarding applicable regulations," *ASCE International Workshop on Computing in Civil Engineering*, pp. 157–165. 2017.

[42] H. Kim, J.-K. Lee, J. Shin, and J. Choi, "Visual language approach to representing kbimcode-based korea building code sentences for automated rule checking," *Journal of Computational Design and Engineering*, vol. 6, no. 2, pp. 143–148, 2019.

[43] E. Hjelseth, "Converting performance based regulations into computable rules in bim based model checking software," *In book: eWork and eBusiness in Architecture, Engineering and Construction*, pp. 461–469, 2012.

[44] E. Hjelseth and N. Nisbet, "Capturing normative constraints by use of the semantic mark-up rase methodology," *Proceedings of CIB W78-W102 Conference*, pp. 1–10, 2011.

[45] J. Zhang and N. M. El-Gohary, "Extending building information models semiautomatically using semantic natural language processing techniques," *Journal of Computing in Civil Engineering*, vol. 30, no. 5, p. C4016004, 2016.

[46] S.-H. Zhang, S.-K. Zhang, Y. Liang, and P. Hall, "A survey of 3d indoor scene synthesis," *Journal of Computer Science and Technology*, vol. 34, no. 3, pp. 594–608, 2019.

[47] R. Akase and Y. Okada, "Automatic 3d furniture layout based on interactive evolutionary computation," *Seventh International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 726–731, 2013.

[48] P. Kán and H. Kaufmann, "Automatic furniture arrangement using greedy cost minimization," in *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2018, pp. 491–498.

[49] Y. Li, X. Wang, Z. Wu, G. Li, S. Liu, and M. Zhou, "Flexible indoor scene synthesis based on multi-object particle swarm intelligence optimization and user intentions with 3d gesture," *Computers & Graphics*, vol. 93, pp. 1–12, 2020.

[50] Y. Liang, S.-H. Zhang, and R. R. Martin, "Automatic data-driven room design generation," in *Next Generation Computer Animation Techniques: Third International Workshop7*, 2017, pp. 133–148.

[51] L.-F. Yu, S. K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. Osher, "Make it home: Automatic optimization of furniture arrangement," *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH*, vol. 30, no. 4, 86:1–12, 2011.

[52] S.-K. Zhang, W.-Y. Xie, and S.-H. Zhang, "Geometry-based layout generation with hyper-relations among objects," *Graphical Models*, vol. 116, p. 101 104, 2021.

[53] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, and P. Hanrahan, "Synthesizing open worlds with constraints using locally annealed reversible jump mcmc," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–11, 2012.

[54] W. Xu, B. Wang, and D.-M. Yan, "Wall grid structure for interior scene synthesis," *Computers & Graphics*, vol. 46, pp. 231–243, 2015.

[55] Z. S. Kermani, Z. Liao, P. Tan, and H. Zhang, "Learning 3d scene synthesis from annotated rgb-d images," in *Computer Graphics Forum*, vol. 35, 2016, pp. 197–206.

[56] Q. Fu, X. Chen, X. Wang, S. Wen, B. Zhou, and H. Fu, "Adaptive synthesis of indoor scenes via activity-associated object relation graphs," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 6, pp. 1–13, 2017.

[57] Y. Liang, F. Xu, S.-H. Zhang, Y.-K. Lai, and T. Mu, "Knowledge graph construction with structure and parameter learning for indoor scene design," *Computational Visual Media*, vol. 4, pp. 123–137, 2018.

[58] K. Wang, Y.-A. Lin, B. Weissmann, M. Savva, A. X. Chang, and D. Ritchie, "Planit: Planning and instantiating indoor scenes with relation graph and spatial prior networks," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–15, 2019.

[59] Y. Zhou, Z. While, and E. Kalogerakis, "Scenegraphnet: Neural message passing for 3d indoor scene augmentation," in *IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7384–7392.

[60] M. Li, A. G. Patil, K. Xu, *et al.*, "Grains: Generative recursive autoencoders for indoor scenes," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 2, pp. 1–16, 2019.

[61] M. Keshavarzi, A. Parikh, X. Zhai, M. Mao, L. Caldas, and A. Y. Yang, "Scenegen: Generative contextual scene augmentation using scene graph priors," *arXiv preprint arXiv:2009.12395*, 2020.

[62] A. Chang, M. Savva, and C. D. Manning, "Learning spatial knowledge for text to 3d scene generation," in *Conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 2028–2038.

[63] M. Fisher, M. Savva, Y. Li, P. Hanrahan, and M. Nießner, "Activity-centric scene synthesis for functional 3d scene modeling," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–13, 2015.

[64] R. Ma, H. Li, C. Zou, Z. Liao, X. Tong, and H. Zhang, "Action-driven 3d indoor scene evolution," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–13, 2016.

[65] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu, "Human-centric indoor scene synthesis using stochastic grammar," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5899–5908, 2018.

[66] Q. Fu, H. Fu, H. Yan, B. Zhou, X. Chen, and X. Li, "Human-centric metrics for indoor scene assessment and synthesis," *Graphical Models*, vol. 110, p. 101 073, 2020.

[67] K. Wang, M. Savva, A. X. Chang, and D. Ritchie, "Deep convolutional priors for indoor scene synthesis," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, pp. 1–14, 2018.

[68] B. Yang, L. Li, C. Song, Z. Jiang, and Y. Ling, "Automatic furniture layout based on functional area division," in *2019 international conference on cyberworlds (CW)*, 2019, pp. 109–116.

[69] D. Ritchie, K. Wang, and Y.-a. Lin, "Fast and flexible indoor scene synthesis via deep convolutional generative models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6182–6190.

[70] B. Yang, L. Li, C. Song, Z. Jiang, and Y. Ling, "Automatic interior layout with user-specified furniture," *Computers & Graphics*, vol. 94, pp. 124–131, 2021.

[71] P. Kán and H. Kaufmann, "Automated interior design using a genetic algorithm," in *Proceedings of the 23rd ACM symposium on virtual reality software and technology*, 2017, pp. 1–10.

[72] T. Germer and M. Schwarz, "Procedural arrangement of furniture for real-time walkthroughs," *Computer Graphics Forum*, vol. 28, no. 8, pp. 2068–2078, 2009.

[73] P. Kan, A. Kurtic, M. Radwan, and J. M. L. Rodriguez, "Automatic interior design in augmented reality based on hierarchical tree of procedural rules," *Electronics*, vol. 10, no. 3, p. 245, 2021.

[74] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, "Semantic scene completion from a single depth image," *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[75] H. Fu, B. Cai, L. Gao, *et al.*, "3d-front: 3d furnished rooms with layouts and semantics," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 933–10 942.

[76] K. Xu, K. Chen, H. Fu, W.-L. Sun, and S.-M. Hu, "Sketch2scene: Sketch-based co-retrieval and co-placement of 3d models," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–15, 2013.

[77] S.-H. Zhang, S.-K. Zhang, W.-Y. Xie, C.-Y. Luo, Y.-L. Yang, and H. Fu, "Fast 3d indoor scene synthesis by learning spatial relation priors of objects," *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 9, pp. 3082–3092, 2021.

[78] R. Ma, A. G. Patil, M. Fisher, *et al.*, "Language-driven synthesis of 3d scenes from scene databases," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–16, 2018.

[79] Z. Zhang, Z. Yang, C. Ma, *et al.*, "Deep generative modeling for scene synthesis via hybrid representations," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 2, pp. 1–21, 2020.

[80] W. R. Para, P. Guerrero, N. Mitra, and P. Wonka, "Cofs: Controllable furniture layout synthesis," in *ACM SIGGRAPH 2023 Conference Proceedings*, 2023, pp. 1–11.

[81] Q. A. Wei, S. Ding, J. J. Park, *et al.*, "Lego-net: Learning regular rearrangements of objects in rooms," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 19 037–19 047.

[82] H. Xie, W. Xu, and B. Wang, "Reshuffle-based interior scene synthesis," in *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, 2013, pp. 191–198.

[83] A. Raistrick, L. Mei, K. Kayan, *et al.*, "Infinigen indoors: Photorealistic indoor scenes using procedural generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 21 783–21 794.

[84] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10 850–10 869, 2023.

[85] L. Yang, Z. Zhang, Y. Song, *et al.*, "Diffusion models: A comprehensive survey of methods and applications," *ACM Computing Surveys*, vol. 56, no. 4, pp. 1–39, 2023.

[86] J. Beetz, L. van Berlo, R. de Laat, and P. van den Helm, "Bimserver. org–an open source ifc model server," in *Proceedings of the 27th CIP W78 conference*, 2010.

[87] M. Das, J. C. Cheng, and S. Shiv Kumar, "Bimcloud: A distributed cloud-based social bim framework for project collaboration," in *International Conference on Computing in Civil and Building Engineering*, 2014, pp. 41–48.

[88] S. Logothetis, E. Karachaliou, E. Valari, and E. Stylianidis, "Open source cloud-based technologies for bim," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 42–2, 2018, pp. 607–614.

[89] *Autodesk bim 360*, Accessed: August 15, 2024. [Online]. Available: `https://www.autodesk.com/bim-360/`.

[90] *Graphisoft bimcloud*, Accessed: August 15, 2024. [Online]. Available: `https://graphisoft.com/solutions/products/bimcloud`.

[91] L. Ma and R. Sacks, "A cloud-based bim platform for information collaboration," in *33rd International Symposium on Automation and Robotics in Construction (IAARC)*, 2016, pp. 581–589.

[92] J.-R. Lin, Z.-Z. Hu, J.-P. Zhang, and F.-Q. Yu, "A natural-language-based approach to intelligent data retrieval and representation for cloud bim," *Computer-Aided Civil and Infrastructure Engineering*, vol. 31, no. 1, pp. 18–33, 2016.

[93] Y. Jiao, Y. Wang, S. Zhang, Y. Li, B. Yang, and L. Yuan, "A cloud approach to unified lifecycle data management in architecture, engineering, construction and facilities management: Integrating bims and sns," *Advanced Engineering Informatics*, vol. 27, no. 2, pp. 173–188, 2013.

[94] *Buildingsmart ifc*, Accessed: August 15, 2024. [Online]. Available: `https://technical.buildingsmart.org/standards/ifc`.

[95] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[96] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[97] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[98] *Buildingsmart industry foundation classes release 4 (ifc4) documentation*, Accessed: August 15, 2024. [Online]. Available: `https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/`.

[99] K. Afsari, C. M. Eastman, and D. R. Shelden, "Cloud-based bim data transmission: Current status and challenges," in *Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)*, 2016.

[100] C. Sydora and E. Stroulia, "Towards rule-based model checking of building information models," in *Proceedings of the International Symposium on Automation and Robotics in Construction (ISARC)*, vol. 36, 2019, pp. 1327–1333.

[101] *Unity technologies*, Accessed: August 15, 2024. [Online]. Available: `https://unity.com/`.

[102] C. Sydora and E. Stroulia, "Augmented reality on building information models," *9th International Conference on Information, Intelligence, Systems and Applications (IISA)*, pp. 1–4. 2018.

[103] J. Wong, X. Wang, H. Li, G. Chan, and H. Li, "A review of cloud-based bim technology in the construction sector," *Journal of Information Technology in Construction*, vol. 19, pp. 281–291, 2014.

[104] C. Sydora, F. Nawaz, L. Bindra, and E. Stroulia, "Building occupancy simulation and analysis under virus scenarios," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 8, no. 3, pp. 1–20, 2022.

[105] R. Sacks, L. Ma, R. Yosef, A. Borrmann, S. Daum, and U. Kattel, "Semantic enrichment for building information modeling: Procedure for compiling inference rules and operators for complex geometry," *Journal of Computing in Civil Engineering*, vol. 31, no. 6, p. 04 017 062, 2017.

[106] P. Hagedorn and M. König, "Rule-based semantic validation for standardized linked building models," in *International Conference on Computing in Civil and Building Engineering*, 2020, pp. 772–787.

[107] F. Momeni Rad, C. Sydora, and K. El-Basyouny, "Leveraging generative design and point cloud data to improve conformance to passing lane layout," *Sensors*, vol. 24, no. 2, p. 318, 2024.

[108] S. Gulwani, O. Polozov, R. Singh, *et al.*, "Program synthesis," *Foundations and Trends in Programming Languages*, vol. 4, no. 1-2, pp. 1–119, 2017.

[109] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

Appendix A

Background Material

A.1 Room Scenarios

Different room types present different layout challenges. Each room type is associated with different objects, variable room and object dimensions, and unique rules governing furnishing arrangements. In this thesis, I have selected the following four room types: living rooms, kitchens, bedrooms, and bathrooms. While there are many other categories of room types (offices, game rooms, etc.), I believe these four also represent a range of features such as dependency amount on movable objects (furnishing) versus dependency amount on static objects (walls) and room layout density. Of course, many more possible room scenarios could and should be tested in the future.

A.1.1 Expert Rulesets

Different publications consider different sets of rules for the above room types. In this thesis, the four rule sets were developed based on my examination of numerous examples I observed from design websites and general intuition of where the objects should be placed (against the wall, away from doors, etc.). The expert rules used depend on the room scenario but in practice could vary significantly depending on the room furnishing designer. Multiple rules can be related to the same object types and there is no guarantee that rules do not conflict with one another or that the space provided is large enough to perfectly accommodate all rules. Therefore, note that a perfect evaluation score may not be achievable in all cases.

A.1.2 Initial Empty Layouts

The initial layout includes all static objects in the scene, which in practice are objects that moving or altering require drastic renovation. The most basic of these object types are the floor, walls, windows, and doors which I include in my initial models. All initial rooms have one door with some rooms containing windows while others do not. In general, there are Mechanical, Electrical, Plumbing (MEP) factors of room design, such as electrical outlets and plumbing fixtures, that would affect design choices which were not included in my layout models. I aim to include these in future iterations.

My experiments contain two different shapes of living rooms: Rectangular or L-shaped. The rectangular rooms are further broken up into small and large rooms. L-shaped rooms pose a more challenging layout problem as the number of walls and the curved shape of the room make furnishing placement less intuitive. As a room can take many shapes and sizes, a challenge is in creating compliant arrangements for all possibilities.

A.1.3 Furnishing Objects

For the research presented in this thesis, the furnishing types for each are based on common objects tested in prior research and commonly found in each room type. The precise number and type of movable furnishing objects were selected based on standard room types and what would reasonably fit in the space. Specific object selection was also based on publicly available furnishing objects 3D models.

All furnishing objects are described in terms of their 3D shapes (or mesh representations) and various associated metadata, including type and facing orientation. Without the rules, furnishing objects have no inherent restriction on how and where they can be placed in the layout model (i.e. no location "snapping"). I also assume that all object sizes are fixed and cannot be expanded.

One caveat is that kitchens in practice are constructed based on appliance location with cabinets filling in the gaps between and around appliances. In my

case, standard cabinet-sized blocks were used to represent where the cabinets might go. A more realistic cabinet generation for Kitchens and Bathrooms is left to future work.

A.1.4 Layout Scenario Dataset

To evaluate my proposed methods' adaptability to higher constrained layout problems, I conducted a variety of room types as well as separated living rooms and kitchens into easy and hard scenarios. The basic assumption here is that higher density makes the scenario harder, while less density makes the problem "simple". I assume this because, with more objects and less available space, there are more decisions to be made and more variable outcomes with potentially different quality.

The difference between the simple and dense living room scenarios is the dense living room problems have additional objects, namely one extra side table and two extra armchairs. The dense kitchen contains an additional two chairs and a table with additional rules for both.

In summary, I evaluated the proposed methods in this thesis on nine layout scenarios: Bathroom 1 (Small), Bedroom 1 (Large), Bedroom 2 (Small), Kitchen 1 (Simple), Kitchen 2 (Dense), Living room 1 (L-Shape, Simple), Living room 2 (L-Shape, Dense), Living room 3 (Rectangular, Simple), Living room 4 (Rectangular, Dense). Table A.1 describes each of the layout scenarios by their initial layouts, movable furnishing objects, and briefly the expert rules used.

Scenario	Initial Layout	Object List	Rule List
Bathroom 1 (Small)	Small rectangular windowed room	Sink Shower Toilette Bathtub	Distribution Elements away from Doors. Nothing in front of Distribution Elements. Sanitary Terminals against Wall. Shower in Corner. Toilet in Corner.
Bedroom 1 (Large)	Large rectangular windowless room	Large bed Cabinet Armchair Side table (x2)	Armchair in Corner facing bed. Bed Location Center. Cabinet facing Bed. Cabinets against Wall. Furnishing away from Doors. Side table next to Bed.
Bedroom 2 (Small)	Small rectangular windowed room	Desk Chair Small bed Side table Shelf	
Kitchen 1 (Simple)	Large rectangular windowless room	Base corner cabinet Base cabinets (x3) Fridge Oven/Range Sink	Cabinets against Wall. Cabinets on sides of Corner Cabinet. Corner Cabinet in Corner. Distribution Elements against Wall. Distribution Elements aligned with Corner Cabinet. Distribution Elements next to Cabinet. Distribution and Furnishing Elements away from Doors. Nothing in front of Cabinet. Oven Fridge Distance. Oven Sink Distance. Sink Fridge Distance.
Kitchen 2 (Dense)	Large rectangular windowless room	Base corner cabinet Base cabinets (x3) Fridge Oven/Range Sink Dining table Chair (x2)	

Table A.1: Room Scenarios

Scenario	Initial Layout	Object List	Rule List
Living room 1 (L-Shape, Simple)	L-shape windowed room	Couch TVstand Coffee table Armchair Side table Shelf (x2) Plant	Side table Next to Couch. Couch Facing TV. Shelf against Wall. Coffee Table In front of Couch. Armchair Coffee table. TV Stand Against Wall and Centered. Armchair Couch Rule. Shelf in Corner. Armchair Next to Side Table. Plant in Corner or Beside Couch. Furnishing away from Doors.
Living room 2 (L-Shape, Dense)	L-shape windowed room	Couch TVstand Coffee table Armchair (x3) Side table (x2) Shelf (x2) Plant	
Living room 3 (Rectangular, Simple)	Large rectangular windowless room	Couch TVstand Coffee table Armchair Side table Shelf (x2) Plant	
Living room 4 (Rectangular, Dense)	Large rectangular windowless room	Couch TVstand Coffee table Armchair (x3) Side table (x2) Shelf (x2) Plant	

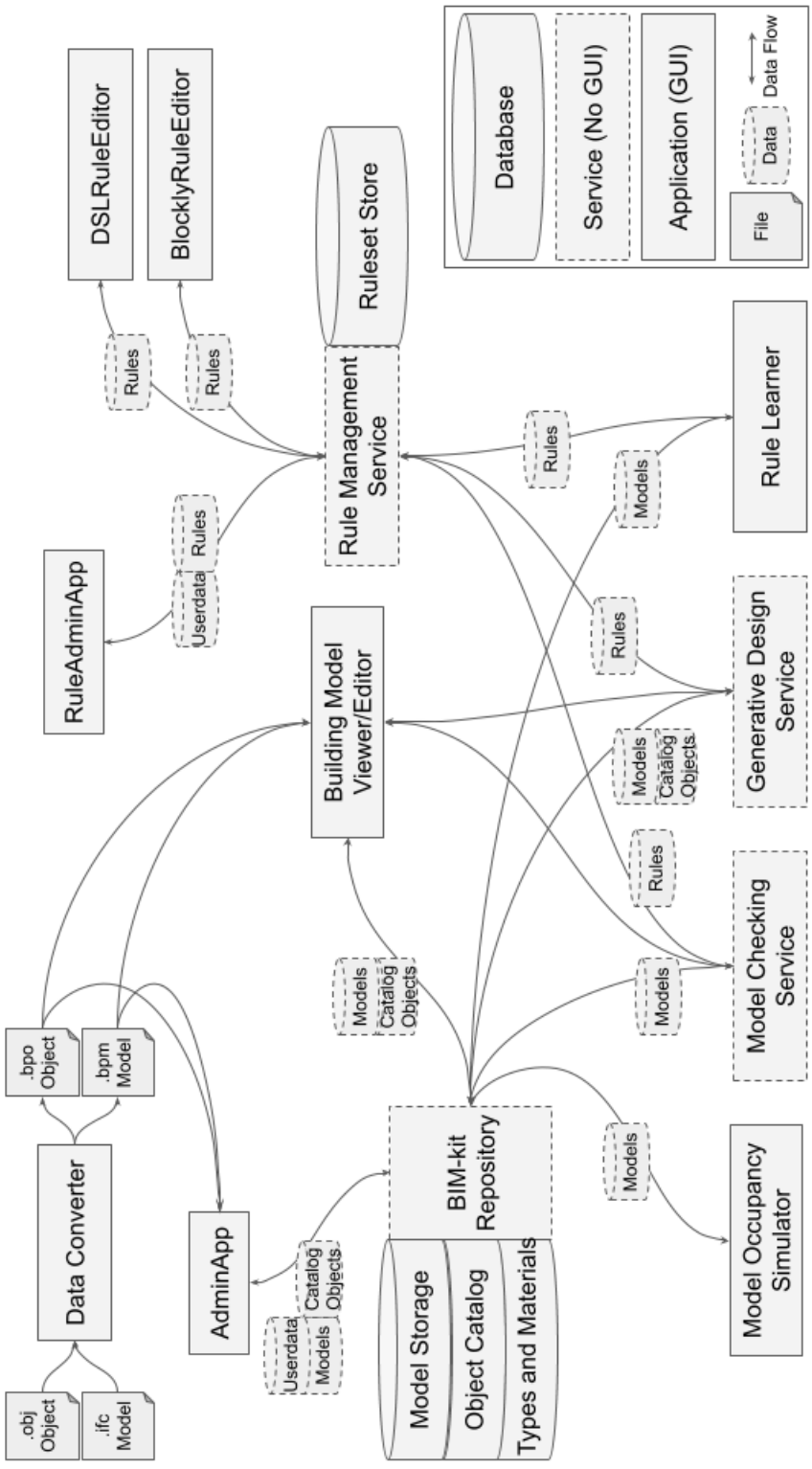


Figure A.1: Current State of BIM-kit's application and service architecture.