Technical University of Munich

TUM School of Engineering and Design

Chair of Computational Modeling and Simulation

Leveraging Parametric Modeling for Building Design Adaptation toward Code Compliance – the case of building egress requirements

Master thesis

for the Master of Science Course Civil Engineering

Author: Muhammad Ali Malik

Matriculation number: [redacted]

Supervisors: Prof. Dr.-Ing. André Borrmann

Jiabin Wu, M.Sc.

Date of issue: 01. April 2024

Submission date: 30. September 2024

Abstract

The continued advancement of Building Information Modelling (BIM) has enabled Automated Compliance Checking (ACC). Despite improvements in highly advanced code compliance checkers to ensure adherence to regulatory standards and adjust building designs for compliance, there remains a significant gap in automating and optimizing this process for complex and diverse design scenarios. Manually adjusting building models based on building code specifications takes time, effort, and is typically error-prone, reducing total project efficiency. This thesis proposes a code compliance framework to automate code checking and adjust violation-related parameters, ensuring code-compliant building designs, specifically focusing on building egress requirements. It also integrates the use of Generative Design (GD) and Genetic Algorithm (GA) with parametric BIM modeling to optimize the means of egress by finding the optimized parameter configuration that yields the least travel distances. It ensures that the building design not only complies with building regulations but is also optimized to enhance the safety of occupants. The proposed approach was validated using a case study. The results illustrate the significance of the proposed framework in automating compliance checking, adjusting violation-related parameters, and optimizing building design to enhance the safety of occupants during emergency scenarios.

Zusammenfassung

Die Weiterentwicklung der BIM hat die automatisierte Prüfung der Einhaltung von Bauvorschriften möglich gemacht. Trotz der Verbesserungen bei den hochmodernen Prüfprogrammen, die die Einhaltung gesetzlicher Normen sicherstellen und Gebäudeentwürfe auf ihre Konformität hin anpassen, gibt es immer noch eine erhebliche Lücke bei der vollständigen Automatisierung und Optimierung dieses Prozesses für komplexe und vielfältige Entwurfsszenarien. Die manuelle Anpassung von Gebäudemodellen auf der Grundlage von Bauvorschriften ist zeitaufwändig, erfordert einen erheblichen Arbeitsaufwand und ist oft fehleranfällig, was sich auf die Gesamteffizienz des Projekts auswirkt. In dieser Masterarbeit wird das „code compliance framework“ vorgeschlagen, um die Überprüfung von Codes zu automatisieren und verletzungsbedingte Parameter anzupassen, um codekonforme Gebäudeentwürfe zu gewährleisten, wobei ein besonderer Schwerpunkt auf den Anforderungen an die Fluchtwege von Gebäuden liegt. Es integriert auch die Verwendung von Generativem Design (GD) und Genetischem Algorithmus (GA) mit parametrischer BIM-Modellierung, um die Ausstiegsmöglichkeiten des Gebäudes zu optimieren, indem die optimale Parameterkonfiguration gefunden wird, die die geringsten Laufwege ergibt. Dadurch wird sichergestellt, dass das Gebäudedesign nicht nur den Bauvorschriften entspricht, sondern auch so optimiert ist, dass die Sicherheit der Bewohner erhöht wird. Der vorgeschlagene Ansatz wurde anhand einer Fallstudie validiert. Die Ergebnisse zeigen die Effektivität des vorgeschlagenen Rahmens bei der automatischen Überprüfung der Einhaltung von Vorschriften, der Anpassung von Parametern, die mit Verstößen zusammenhängen, und der Optimierung des Gebäudedesigns, um die Sicherheit der Bewohner in Notfallszenarien zu verbessern.

Table of contents

List of Figures

List of Tables

List of Abbreviations

ACC	Automated Compliance Checking
AED	Available Egress Distance
BIM	Building Information Modeling
GA	Genetic Algorithm
GD	Generative Design
IBC	International Building Code
MED	Maximum Egress Distance
OL	Occupant Load
VPL	Visual Programming Language

1 Introduction

1.1 Overview

The AEC industry follows specific standards to ensure that buildings are structurally sound, reliable, and usable. Compliance with these codes is crucial for the proper functioning of buildings and the safety of their users. In the past, ensuring adherence to building standards relied on manual processes, mainly using 2D drawings. However, due to a lack of automation, this approach is error-prone and time-consuming (Villaschi et al., 2022).

The emergence of BIM technology has made Automated Compliance Checking (ACC) possible. While current compliance checkers can verify adherence to regulatory standards and modify building designs for compliance, there is still a need to fully automate and optimize this process for complex and diverse design scenarios (Patlakas et al., 2018). As a result, architects and engineers manually change building models to meet code requirements, which is time-consuming, labor-intensive, and iterative, frequently resulting in errors and reducing the project's total profitability.

Many code compliance checkers use programmed codes to evaluate compliance. However, this method only shows the input and output data, not the actual steps taken to process it. As a result, designers don't know how the data is being processed and only see the final results, which makes it difficult to ensure accuracy and fairness. Also, designers find it challenging to make process changes to fit their specific needs without developer intervention, which can be time-consuming and costly (Preidel & Borrmann, 2018).

To address these challenges, the manual adjustment of building models to achieve code compliance can be automated by using generative design for parametric BIM models. This thesis aims to examine and adjust design alternatives by modifying violation-related parameters to streamline code-compliant building designs and identify the most efficient and compliant design solution using optimization algorithms. The GA has been applied to optimize the building design to minimize the available egress distance. The proposed framework can automate not only the process of checking code compliance but also the adjustment of BIM models to ensure that they are both code-compatible and optimized for the safety of the occupants.

1.2 Research Objectives

This thesis aims to create a framework that combines parametric modeling and advanced optimization algorithms to identify an efficient and code-compliant design solution for building egress requirements. It is essential to consider egress requirements in building design as they contribute to the safety of building occupants during emergencies and are essential for compliance with building regulations. This thesis aims to address the following research question:

> *"How can automated design support be provided through generative design and optimization algorithms to transform parametric models into code-compliant solutions that minimize egress distances and enhance occupant safety?"*

This thesis introduces the code compliance framework designed to correct the BIM models to ensure compliance with building codes. it automates code compliance checking for the building's egress requirements and can automatically adjust violation-related parameters to ensure code compliance. Using a visual programming language, the building code limitations are imported and compared with the data of the building elements in the BIM model to check code compliance. Once the non-compliant building parameters are identified, they are automatically adjusted within the specified range of building codes to ensure code compliance. This thesis aims to use generative design to optimize egress routes by identifying the design variant with the shortest egress distance, allowing quick evacuation in emergencies. This objective guarantees that the building design not only adheres to building regulations but is also optimized to enhance the safety of occupants. The proposed framework has been tested in a case study to check its accuracy and reliability. The case study considered in this thesis is the Physics Department (N6) Building of the Technical University of Munich (TUM). The obtained results are thoroughly explored, and potential enhancements are discussed.

1.3 Outline of Thesis

Chapter 2 investigates the current state of Automated Compliance Checking (ACC) practices. This includes exploring how ACC was developed and the challenges it faces today. The theoretical principles of parametric BIM modeling are also discussed, with examples provided to illustrate its current usage in benefiting various projects.

Furthermore, an in-depth analysis is carried out on generative modeling and optimization algorithms commonly utilized in the AEC industry.

Chapter 3 outlines the proposed methodology, including the workflow and practical application of the approach. An in-depth analysis of the building code, specifically used for egress requirements is presented and objective function is created. Further in this chapter, a detailed explanation is provided on the application of generative design and genetic algorithm for the automatic adjustment of a parametric design to optimize the egress distance of the building.

Chapter 4 presents the implementation of the code-compliance and optimization framework. It begins by introducing ten compliance checks selected from the IBC. The chapter then explains the process of creating a parametric BIM model and its application in extracting building information for code compliance checks. Furthermore, it provides a comprehensive overview of the workflow of the Dynamo, which is utilized for automating the adjustment of violation-related parameters. Finally, it showcases the utilization of a genetic algorithm to achieve optimized parameter configuration for the optimized egress distance.

Chapter 5 tests and validates the functionality of the proposed framework. This is accomplished through a case study of a building at TUM, for which a parametric BIM model is constructed using the parameters selected in the preceding chapter. Subsequently, the model undergoes a code compliance check. When the model is determined to be non-compliant, the proposed framework is employed to automatically modify the model to ensure it meets the required code standards. Furthermore, the proposed optimization framework is validated to reduce the egress distance while taking building layout constraints into account.

Chapter 6 provides a detailed analysis of the research results, and summarizes the contributions made by this thesis. It also outlines the limitations of the proposed methodology and discusses possibilities for its future improvement.

2 State of the Art

2.1 Automated Compliance Checking

ACC increases the effectiveness and precision of the inspection process while also providing opportunities to identify noncompliance with building standards. Building codes represent a collection of international established regulations and standards that aim to ensure the planning, construction and upkeep of buildings in a manner that prioritizes safety and sustainability. These building regulations typically specify the minimal prerequisites for building design, including prescribed materials, obligatory structural components and mandated safety provisions. In recent years, there has been a growing interest in structuring building codes to facilitate machine interpretation and application to improve the accuracy and efficiency of building code enforcement (Kincelova et al., 2020).

ACC is the subject of extensive research, focusing on leveraging BIM and knowledge graph technologies. For instance, Peng and Liu (2023) conduct a study on ACC based on BIM and knowledge graphs, aiming to automate the drawing review process. Using natural language processing technology, they propose a framework to transform specification provisions into a computer-recognizable structured language. This solution effectively overcomes the problems of manual reliance and inefficiency in the review procedure.

ACC techniques evaluate building designs through four stages: rule interpretation, building model preparation, rule execution, and rule check reporting (Luo & Gong, 2015). Rule interpretation involves examining the construction code and identifying pertinent rules and requirements. Building model preparation is the process of creating a digital representation of the building that integrates all relevant design elements and features. Rule execution is done through software that applies building code rules to a building model and tests various design scenarios against building code requirements. Rule check reporting involves generating a report that summarizes the results of the rule execution and highlights areas where the building design does not meet the code requirements. This report can subsequently be utilized to implement the necessary design modifications and ensure building compliance with the relevant building codes.

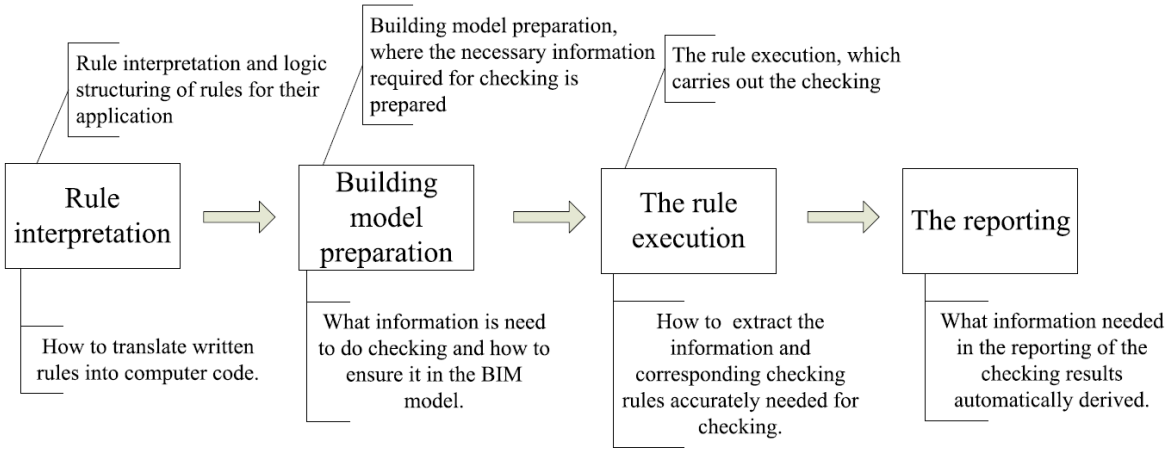


Figure 2.1: Four-stage process of ACC approaches (Luo & Gong, 2015)

In current code compliance checkers, human expert involvement is necessary to resolve reported design flaws. Designers manually adjust the building model to ensure conformance with codes, which is a laborious and iterative task. Moreover, implementing design alterations is often prone to error, as satisfying one building regulation may lead to violating other requirements (Wu et al., 2023).

2.2 Techniques in ACC Development

Research has been conducted to explore different methods of interpreting rules in various countries for ACC. These methods include rule-based, object-based, logical, and ontological approaches (Ismail et al., 2017). The appropriate approach selection depends on the problem's complexity and the accuracy required for ACC. This section outlines some commonly used strategies for comprehending rules for developing Automated Compliance Checking.

2.2.1 Software Applications

Commercial software applications are frequently employed for rule checking since they are accessible and simple to use. Some popular software for ACC includes Solibri Model Checker (SMC), BIM Assure, SMARTreview, and CORENET.

Solibri Model Checker (SMC) is a software application developed by the Finnish company Solibri Inc. that has become increasingly popular among architects, engineers and construction professionals (Solibri, 2024). The software is designed to help designers identify and resolve design errors before and during the construction process. One of the primary strengths of SMC is that it automates the design analysis and checking process, making it easier for designers to identify potential issues such as

clashes, missing components, or accessibility problems. Using SMC, designers can significantly reduce the risk of costly construction errors and delays, resulting in a more efficient and streamlined construction process.

SMC uses a classification strategy to combine data from multiple discipline models and BIM authoring tools. This strategy involves categorizing elements of construction based on particular details included inside the building model. The categorization technique is a critical step in the process as it allows information from various construction models to be filtered and prepared for the subsequent inspection procedure. The classification system helps to organize and structure the data, making it easier to identify and inspect the different components of the building model. This procedure guarantees that all data in the model is precise, full, and unified, lowering the likelihood of mistakes and conflicts.

The Solibri application's checking methods are based on the Ruleset Manager, which has a library of templates (Preidel & Borrmann, 2018). Each template provides a standard-checking approach that may be tailored to a set of parameters. Users can create or modify these rule templates to suit their specific requirements. Rule compositions can be saved as rule sets, making sharing and distributing specified rules possible. SMC also has several built-in features that help users quickly identify potential issues, such as highlighting areas where the design may not comply with accessibility standards.

The process of creating rules in SMC requires extensive knowledge of the rules themselves. Therefore, this process is usually performed by experts rather than users. As a result, most users depend on SMC's established criteria and focus on key design criteria such as information sufficiency or building component collision (Fig. 2.2). Despite this limitation, SMC remains a highly effective tool for identifying and resolving design errors, and its popularity among professionals in the construction industry continues to grow (Greenwood et al., 2010).

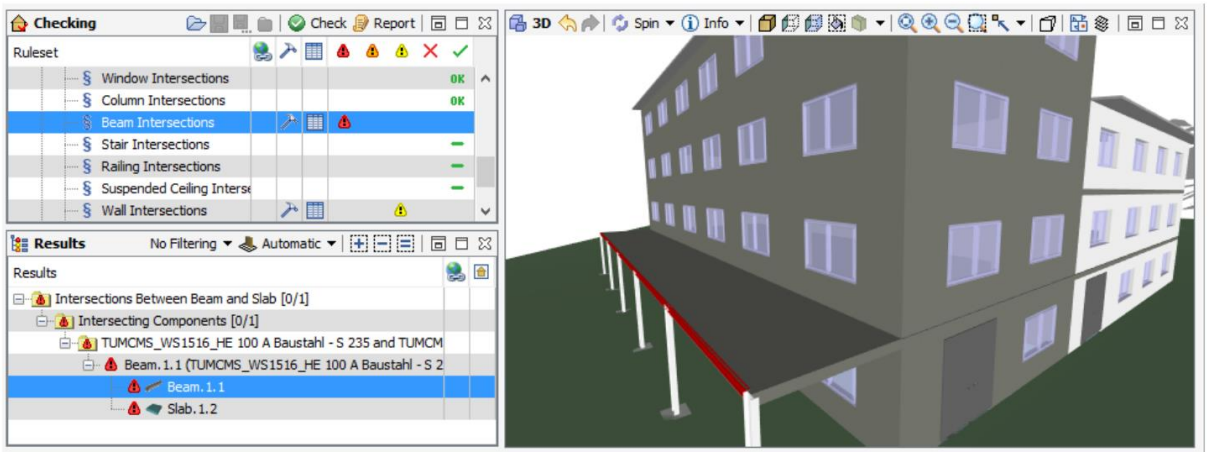


Figure 2.2: User interface of the Solibri Model Checker (Preidel & Borrmann, 2018)

Kim and Nguyen (2011) explore the effectiveness of a plug-in application for automatic code compliance checks. The approach requires using a plugin to retrieve the essential information from the building design to perform building code checks. This is achieved by utilizing a VB.Net plug-in to generate a conditional query. Although executing conditional queries is not directly possible in Revit, it can be accomplished using the Revit API. The framework possesses a static graphical model that exhibits building properties in both graphical and non-graphical formats. Their comprehensive analysis centers on Autodesk Revit Architecture to ensure compliance with the International Building Code (IBC). The focus is on three essential areas of compliance: firewall openings, fireproofing scores, and lateral consistency. Their technology allows the extraction of information not readily accessible in the building design. They develop additional parameters that encapsulate their understanding of the IBC building code, such as the dimensions of firewall openings. This step proves necessary as the building model alone does not furnish all the requisite information for compliance checks. The study reveals that Autodesk Revit Architecture is an optimal platform for implementing automated building design systems, including code compliance verification. Incorporating additional parameters based on the knowledge of building codes guarantees the availability of all necessary information for compliance checks.

2.2.2 Object-based approach

The object-based method is more appropriate for monitoring product quality and detecting design flaws. This approach utilizes the object-oriented attributes of BIM. A building model comprises distinct objects, each with unique characteristics and connections to other objects, which are compared to relevant building codes for compliance. The object-based approach is a strategy for organizing comprehension. This

structure is achieved by expressing categories of objects as units of information. The technique of modeling building codes using the object-oriented approach includes three parts (Yang & Li, 2001):

1. Categorization and Abstraction of Building Codes: This stage involves identifying all information in the building codes and classifying them. This stage aims to ensure that all significant information is encapsulated in the knowledge base.

2. Modeling of Rule Representation: This stage involves identifying all related objects and establishing connections between them and building code classes.

3. Development of Knowledge Base: This stage entails recording and preserving data and principles about construction regulations in a hierarchical style within an information base. The objective of this stage is to ensure that the knowledge base is current and precise.

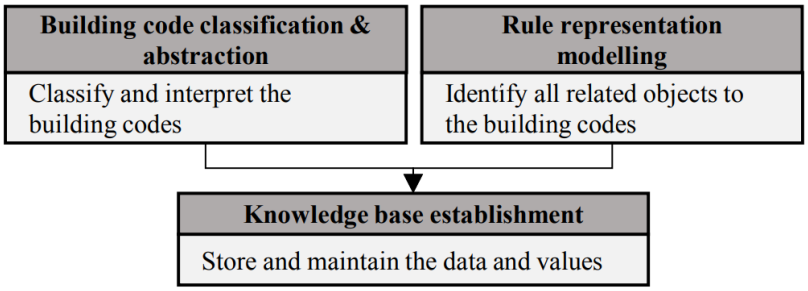


Figure 2.3: Three stages of the object-based approach (Yang & Li, 2001)

Some studies demonstrate that machine learning can surpass rule-based techniques under certain circumstances (Bloch & Sacks, 2018). Sacks et al. (2019) discuss artificial intelligence's strong, favorable implications in ACC. They highlight numerous possible applications for artificial intelligence approaches throughout the inspection process. Furthermore, it is feasible to execute the entire rule-checking procedure using a machine-learning algorithm (Bloch & Sacks, 2018). However, machine learning presents challenges, such as collecting training datasets and selecting data characteristics from an enormous quantity of building information. It is vital to establish or identify distinct data characteristics of buildings to evaluate different rules, as relevant and effective data features ensure high-accuracy outcomes from machine learning algorithms.

CORENET e-PlanCheck was formally adopted in Singapore in September 2000 as an example of an object-based methodology and automated code-checking system (Shih & Sher, 2014). CORENET project is the pioneer project in the automation of code-

checking. The technology increases the productivity and precision of the verification process, making it more reliable and faster. FORNAX is an independent platform and C++ object library. By extending the IFC model and extracting FORNAX objects, rules defined in plain language can be easily understood without requiring algorithm development (Eastman et al., 2009).

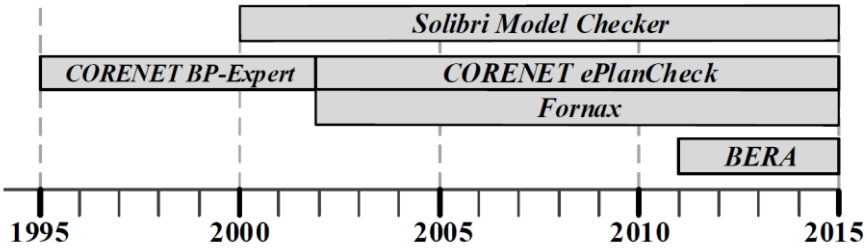


Figure 2.4: Recent approaches to automated compliance checking (Preidel & Borrmann, 2015)

DesignCheck, an automated checking system, can create complete design information with corresponding descriptions that map to building codes. Building codes are comprehended through an object-oriented description and subsequently encoded as object-oriented rules using the programming language. The tool enables designers to verify building models against certain clauses in the building code, or they may check individual item kinds or groups of items rather than the full building design.

2.2.3 Logical approach

Building rules, the product of human creation, can be interpreted through formal methodologies, such as converting logical constructs into statements in human language. The formal interpretation process entails decomposing intricate rules into smaller, more manageable components that can be expressed in a logical language. The language most commonly employed for natural language translation is first-order predicate logic, which allows for expressing complex ideas using simple symbols. First-order predicate logic is a formal language that uses symbols and rules to depict logical associations between objects and concepts. It is a powerful tool for expressing complex ideas because it creates precise and unambiguous statements that humans and machines can easily understand. Prolog, Datalog, and Answer Set Programming are commonly used logical programming languages.

There are certain advantages of using logic-based reasoning in checking code compliance (Zhang & El-Gohary, 2016). The binary nature of logic-based reasoning is one of the main reasons. Logic-based reasoning evaluates statements or conditions as true

or false, following a logical sequence to conclude. Because logic-based reasoning is binary, it may quickly assess if a building design is appropriate or in breach of specific standards or rules. In addition, complex building codes can be effectively captured and represented logically and structured using expressive logic-based reasoning, ensuring accurate interpretation and implementation of the building code.

Lee (2010) discover that predicate logic is helpful for logical validation of checks. It can reflect a verification technique, computation in the defined guidelines, numerous standard circumstances, and the ability to treat construction portions as predicate logic entities. By confirming the logical combinations of conditions, the verification can be completed, resulting in validation findings that are either 'true' or 'false'. Park and Lee (2016) provide KBimCode, which uses a logic rule-based mechanism to represent, define, and evaluate building codes.

In addition to predicate logic, the conceptual graph is another method used to convert rules into fundamental logic structures. It is a useful tool that allows experts to extract rules, construct objects, and describe the interaction between them, including any restrictions, without requiring programming knowledge. Translating rules into a conceptual graph involves four steps (Figure 5). Firstly, the central idea of a rule, such as "space", must be identified. Secondly, the individual sub-rules that make up each rule must be recognized, as each sub-rule is independent. Thirdly, the atomic limits and restrictions are determined. Finally, the most suitable conceptual graph is defined by establishing the relationships between all the pieces.

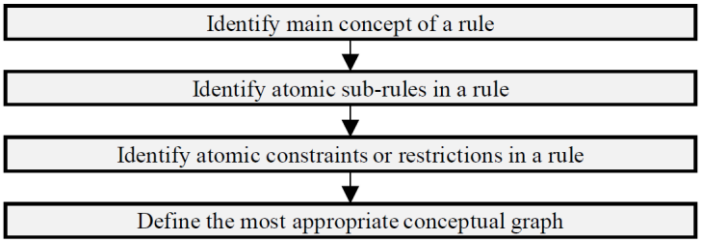


Figure 2.5: Four Stages in Creating a Conceptual Graph for Logical Interpretation (Yang & Li, 2001)

Salama and Gohary (2011) conducts a study on the logical approach and develops the deontology approach to make it easier to determine if a building model is authorized or banned under applicable rules. This approach provides complex knowledge representation and reasoning more effectively. The decision table technique, developed in 1969, is quite similar to the logical approach (Fenves et al., 1969). It was created to help understand the steel design criteria of a structure. This method involves recording logical rules as a parameter table and does not require computer programming. The

decision table organizes every consideration and judgment into an array, allowing intricate reasoning to be expressed concisely and clearly.

2.2.4 Ontological approach

Ontology is a representation of knowledge that offers a structured framework for presenting building objects, their attributes, and relationships in a particular domain (Preidel & Borrmann, 2018). Ontology may be used to represent the connections and restrictions that are present between various construction components and systems. For example, relationships between different building materials, such as those used for walls, roofs, and floors.

A graph effectively expresses complex ideas and relationships in the semantic network (Decker et al., 2000). It is a visual representation of multiple objects and their logical relationships. The Resource Description Framework (RDF) is usually utilized to generate such graphs. RDF is a language that allows you to specify graph structure by using statements and expressions to describe the links between resources. These statements, or RDF triples, comprise three parts: subject, predicate, and object. The subject represents the resource, the predicate indicates their relationship, and the object represents the other resource. All three components work together to provide a comprehensive and precise overview of the resource relationships.

Zarli et al. (2008) present a framework with four components contributing to a comprehensive conformity-checking approach. First, an information collection approach was developed utilizing RDF (Resource Description Framework) to define the framework and construction rules. In addition, a logic system was created to align the design with the construction codes. This method aids in detecting variations and flaws in the construction model. Finally, they were included in a working model termed the C3R framework to assist in developing the structure. The C3R system is a computer application that compares code compliance to the building model.

The ICC adopted the ontological approach for regulatory compliance by developing SMARTcodes in 2006 (Eastman et al., 2009). SMARTcodes are a set of codes that incorporate the principles of ontology to provide a more structured and consistent approach to regulatory compliance. These codes are used for various purposes, including building codes, fire codes, and energy codes. As part of the development of SMARTcodes, the ICC established an International Energy Conservation Code (IECC) dictionary. This dictionary is a knowledge acquisition method and a communication

platform between regulations and building models. Bouzidi et al. (2012) propose a framework that utilizes semantic web technologies to formalize building regulations. The framework is based on RDF Query Language (SPARQL). The framework incorporates an ontological approach called CQIEOntology to facilitate construction quality inspection and evaluation. This approach evaluates construction quality by assessing if required standards are met.

ACC also established the ontology concept through the RASE framework (Hjelseth & Nisbet, 2011). The RASE approach organizes and structures specification rules, which are transformed into a machine-readable format for automated compliance assessment. These four components were employed as model-checking operators, and each rule is divided into four-element groups. A requirement is a set of specified criteria a building design must fulfill to comply with the building code. Requirements often begin with 'shall' or 'shall not'. Applicability assesses whether a certain rule applies to the building design. It involves identifying many sentences that relate to the same idea. If a requirement is met, the selection element specifies the exact component to which it applies. Exceptions are the opposite of Applicability; they account for any unique circumstances in which the requirement may not apply, even if it usually applies to the building design. Exceptions often begin with 'unless'.

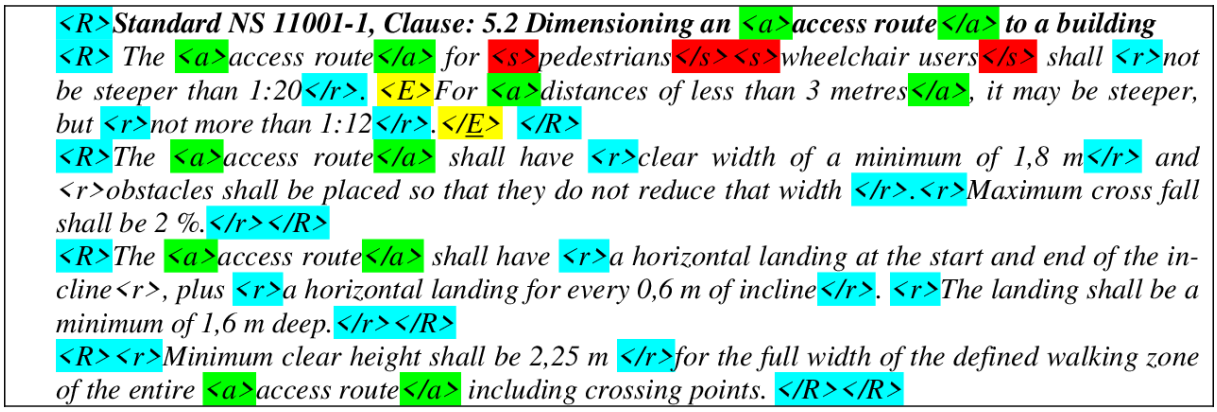


Figure 2.6: Establishment of the Norwegian code with the RASE syntax (Hjelseth & Nisbet, 2011)

2.2.5 Challenges encountered in ACC

With the advancement of technology, data can now be accessed and processed automatically in machine-readable format. However, it is challenging to obtain what one needs from an extensive number of written material. Natural language processing (NLP) is widely utilized to fill the separation between machines and human languages, enabling successful human-machine communication. NLP is a discipline of computer science that studies the interaction between human and machine language. The

objective is for trained automated machines to understand, interpret, and develop human language. NLP enables machines to analyze large amounts of written text, discover patterns, and extract usable information. Zhang & El-Gohary (2017) propose a framework for the ACC that addresses the challenge of manual coding rules. The suggested system uses NLP to automate the process of coding rules, lowering the chance of human mistakes. With this framework, the ACC achieves greater efficiency and accuracy in its rule-making process. Moreover, Zhong et al. (2020) present an artificial learning model that combines NLP with data extraction. This model helps analyze and understand complex building regulations. By combining natural language processing (NLP) and information retrieval, this model extracts critical information from large sets of building codes, providing actionable insights to stakeholders.

The fundamental goal of ACC is to convert rules and guidelines into machine-usable language. There are two distinct methods for this translation (Preidel & Borrmann, 2018). The translation process can be substantially simplified by shifting the evaluation procedure directly to a coded program. In this approach, the digitization of code or guideline materials focuses on defining machine-readable algorithms often hidden from users. This concept is referred to as the Black-Box method, and it is a process that displays incoming and exiting data, not the actual computation operation. However, the fundamental benefit of this method is an extremely small overall rate of mistakes due to the code-checking system's closedness and explicit use of internal data structures. The Black-Box approach works well when the outcome is more important than the procedure. However, the user's capacity to interpret the translated rules is limited due to the hidden approaches used in the process. As a result, additions and modifications can only be implemented via integration with the software developer.

White-Box methods are a type of testing approach that allows the internal processing steps of a system to be visible and comprehensible to the user. Unlike hidden procedures, which conceal the inner workings of a system, White-Box methods provide transparency that enables users to understand and retrace the steps of the checking procedure at any time. To reach this degree of accountability, each section of the translated rule or procedure must be clear by both the computer and the person using it. This necessitates a code representation system comprising a set of signs and regulations for describing objects, methods, and relationships clearly and comprehensibly to the user.

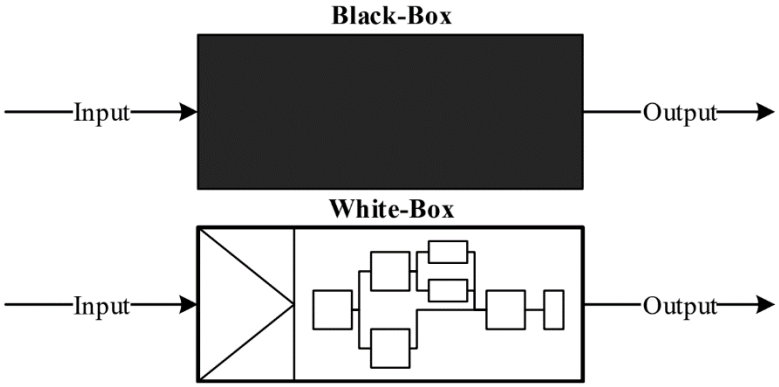


Figure 2.7: Representation of the black-box and white-box approaches (Preidel & Borrmann, 2018).

The fundamental purpose of White-Box techniques is not just to encompass all potential data that an instruction or regulation may provide but additionally to allow the consumer to track progress step by step. This means the regulations must be interpreted using a user-friendly code representation system, allowing people to understand how the system processes information at any given time. Despite creating and executing such an approach that needs substantially more time than the closed verification technique, its benefits for carrying out an inspection assignment are significant. By providing users with a clear understanding of the inner workings of a system, White-Box methods can help improve system performance, identify potential issues more quickly, and ultimately lead to a better user experience.

Considering the rising level of digitization and outsourcing in the building sector, accountability for the success of each process step ultimately falls on the planning engineers or regulatory authorities due to legal constraints. Automated processes must be cautiously approached, and their results must be manually checked for accuracy and plausibility. In the AEC industry, it is common practice to periodically check results manually by performing calculations or comparing them with established rules of thumb. However, because organizing experts are often not software developers, tests are not conducted using black-box approaches owing to a lack of openness. This lack of openness frequently causes a lack of faith in the outcomes and raises concerns about the trustworthiness of automated systems (Gross, 1996). As a result, a white-box approach is preferred to meet the main objectives. This approach allows for transparency and observability of individual processing steps, reducing the risk of hidden procedures and errors. However, this raises the concern of whether total machinery automation without human intervention or input is realistic. Guidelines may include confusing semantics that a human with the necessary expertise, wisdom, and accountability must interpret. Therefore, it is advised that a partially automated technique be

employed. This strategy allows for user interaction and input while retaining the benefits of automation.

2.3 Parametric BIM Modeling

Parametric BIM models are constructed utilizing relationships and restrictions, resulting in an adaptable framework which can rapidly and effectively adjust to forthcoming or evolving circumstances (Borrmann et al., 2018). A “parameter” defines building objects' geometric and semantic information and helps establish interactive dependencies. Using parametric modeling, architects and engineers can gain precise control over building component information. Parametric modelling helps planners to swiftly examine a variety of construction alternatives by integrating factors that represent the limits and requirements of building regulations. It enables seamless modification of component geometric and semantic data through parameters, allowing professionals to automate the model adjustment process to ensure code compliance of building models.

In parametric modeling, the first step is to create a 2D sketch that includes all the geometric elements required to produce the final object (Borrmann & Berkhahn, 2018). Afterward, the geometric elements are linked using two kinds of constraints: dimensional and geometric. Dimensional constraints specify the dimensional value of geometric elements such as length, width, or thickness, while geometric constraints ensure that the geometric elements remain parallel or perpendicular. After assigning the constraints, the 2D parametric sketch can be used for operations, including extrusion or rotation, to create the final 3D parametric body. Figure 2 illustrates that the rectangular and round shapes confine an identical region due to dimension constraints.

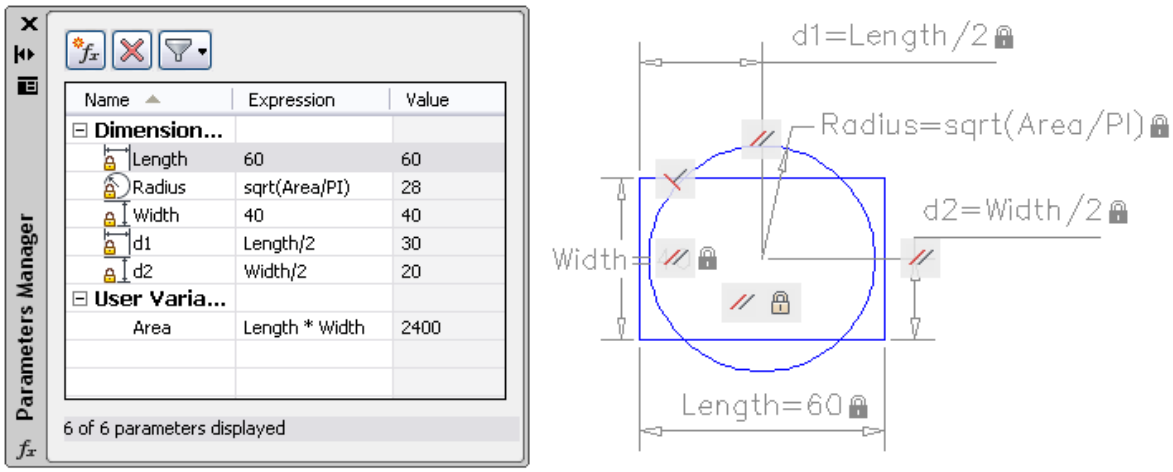


Figure 2.8 Instance of a parametric sketch (Borrmann & Berkhahn, 2018)

Parametric modeling based on geometry involves creating shapes using rules that are based on parameters. For example, a sketch is created with specific dimensional, equational, and geometric constraints. This sketch is then combined with a procedural geometric description, such as extrusion, to create a three-dimensional shape. This method is considered an implicit representation because the entire construction history is saved, allowing for modifications at each step of construction by adjusting parameters. On the other hand, Parametric BIM modeling limits the flexibility of parametric modeling because it relies on predefined object types and constraints within the BIM application. The parametric approach in BIM operates on two levels: first, by establishing geometric construction objects such as walls and stairs, and second, by defining their positions within the overall building complex. This approach requires defining the object's position within the building complex and often includes predefined constraints between object types. These parameters involve considerations such as parallelism, orthogonality, alignment, and distance between objects (Borrmann et al., 2018).

Parameters are crucial in defining the relationships between different building elements and how they affect each other (Edmonds et al., 2022). It implies that any alteration in a parameter will impact its dependent parameters, thereby influencing the overall design. For instance, an alteration in the parameter representing the width of a room would automatically update its dependent parameter, i.e., the length of the wall. Moreover, a parameter could be established to ensure that the height of a door always remains at 80% of the height of the wall it is set in. Consequently, any changes in the wall height would automatically trigger adjustments in the door height to preserve this relationship. This concept of parametric interdependency facilitates the development of intelligent BIM models capable of automatic adjustments in response to changes. This not only diminishes the time and effort required for design updates but also enhances the overall efficiency of projects.

A number of advances have been made to assist parametric modelling. The tightly coupled method connects systems via the Application Programming Interface (API). In this scenario, graph-based systems interact using the API, which directly generates geometry in the BIM model each time that the graph-based model is performed. One of the examples is Dynamo, which utilizes the Revit API. The loosely coupled technique connects platforms by exchanging models. The graph-based approach frequently generates data in a typical file structure that can be put directly into the BIM platform. An example of the loosely coupled technique is Grasshopper, which uses IFC as the

interchange format. These techniques are continually emerging. Nevertheless, of the two options, the loosely coupled method based on sharing files has the essential benefit of being process agnostic, allowing individuals to connect tools and systems to enable multiple types of collaboration and exchange. For instance, while GeometryGym generates an IFC file, users may link to any BIM platform capable of importing IFC files (Janssen, 2015).

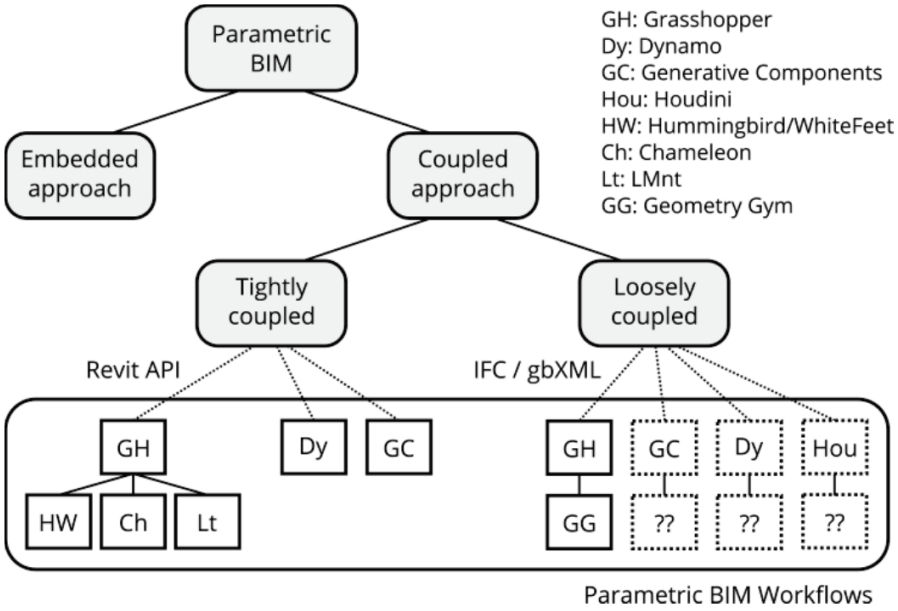


Figure 2.9: Integration of graph-based systems with parametric BIM modeling (Janssen, 2015).

Holzer (2015) explains the benefits of integrating BIM with parametric modeling. BIM involves creating and maintaining digital representations of a project's physical and functional qualities, resulting in a collaborative environment for all stakeholders engaged in a building's lifetime. On the other hand, parametric design is a technique for intelligently developing architectural objects based on relationships and rules defined through parameters. The combination of BIM and parametric design provides a solid foundation for tackling difficult design challenges. It allows architects and designers to develop flexible models that can respond to changing design needs without having to begin from the start.

Parametric models facilitate performance analysis early in the design process. By integrating analysis tools with parametric models, designers can assess performance criteria such as daylighting, thermal comfort, and structural stability, enabling informed design decisions. Károlyfi and Szép (2023) use parametric BIM modeling to generate various structural solutions and evaluate their embodied environmental impact during the conceptual design phase. The study showcases the effectiveness of parametric

BIM modeling through a case study of an unheated warehouse constructed using steel frames, analyzing 48 different design solutions. The study concludes that parametric design can be a valuable tool for conducting comprehensive environmental impact evaluations.

Júnior et al. (2023) investigate the use of parametric modeling in BIM for the identification of potential building pathologies. A parametric digital twin of an existing building is created to facilitate a comprehensive analysis. It permits the linkage of relevant information to the operational duration of products and equipment in development. It provides a proactive strategy for building maintenance, potentially leading to significant savings in costs and resources. Yang et al. (2022) presents a framework that utilizes parametric modeling to calculate the construction expenses. A construction cost estimation model is developed employing a wide range of characteristics, emphasizing the importance of financial viability as well as expense assessment during the initial design phase of initial-stage construction tasks.

Barazzetti and Banfi (2022) study the use of parametric BIM and GIS information in infrastructure design. Geospatial data is used extensively in infrastructure and land administration projects. GIS softwares are great at geospatial analyses but lacks parametric modeling tools while BIM softwares excels at parametric modeling but lacks geospatial tools. They suggest that BIM and GIS are complementary technologies, and ongoing research is aimed at enhancing their interoperability, particularly at the building level. Integration of geoinformation and parametric modeling streamlines design workflow. They provide simulated and real examples, showing that integration is feasible at specific scales.

2.4 Building Egress Requirements

A means of escape is a continuous and unobstructed route between any point in the structure to a public route or a designated area of safety (Shen, 2006). A building's technique for escape is made up of three parts. Exit access is the path from any location within a building to an outlet. Exit access is characterized as the portion of the way of escape that connects to the entrance of an escape and is included in the total distance traveled to reach an exit. The exit shall be defined as the component of the means of escape that has protection from the zone of incidence and offers a safe passage to the exit discharge. An exit is generally a door that leads to the outdoors or, in a multi-story structure, an enclosed exit staircase. The exit discharge shall include any

part of the route between the exit's termination and the exterior or the refuge area (Bukowski & Tubbs, 2016).

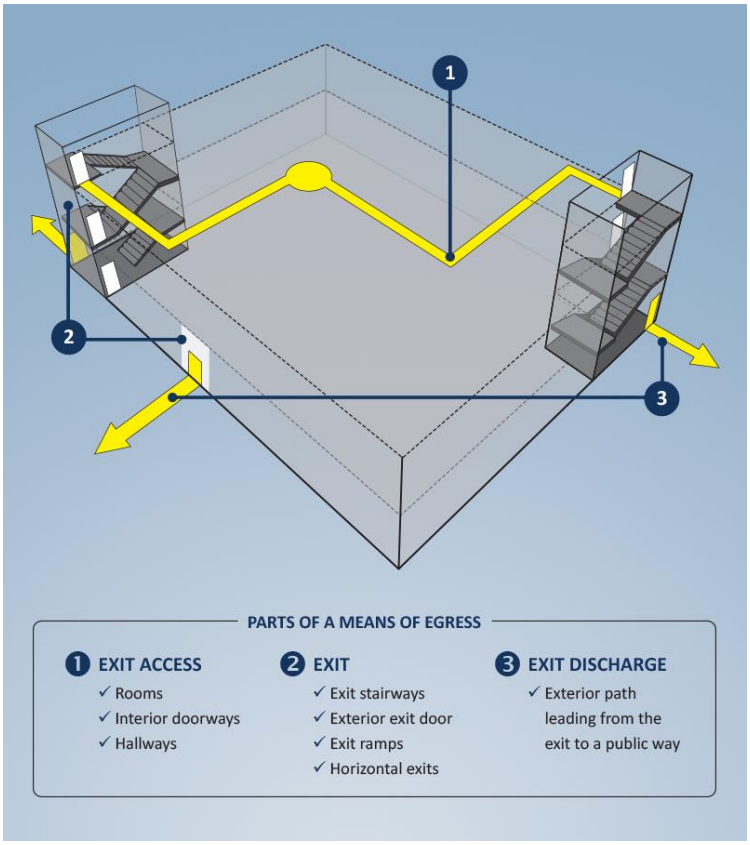


Figure 2.10: Three components of means of egress (Shen, 2006)

Sun and Turkan (2019) used BIM to simulate fire situations and human behavior, enabling the comparison of available and required evacuation times. The study highlights the importance of including building layout, fire characteristics, and human behavior in evacuation simulations to improve the means of egress. The findings suggest that by comparing available and required evacuation times, one can select an optimal building architecture that reduces the required evacuation time, thereby enhancing occupant safety. Kodur et al. (2020) investigate evacuation techniques inside a thirty-two-story standard workplace during various fire-induced scenarios. He discovers that the two most important factors determining evacuation duration are the location of the stairs within the structure and the floors where the explosion of flame begins. The study also finds that using situational awareness in urgent evacuation protocols can increase evacuation efficiency, resulting in up to a 24% reduction in time.

Usman et al. (2020) proposes a computational method for automating semantic rule testing of flame escape situations in the architectural layout of a building. The tool uses dynamic crowd simulations to consider space semantics and the impact of design

space on human safety during possible egress evacuations, going beyond static geometric information. Integrating spatial semantics with dynamic crowd simulations enables a more thorough study of egress scenarios, leading to safer architectural solutions. The case study demonstrates that standard egress planning approaches lacking spatial semantics or dynamic human behavior may violate IBC regulations and put human safety at risk. Nourkojouri et al. (2023) address the critical issue of emergency evacuation in building design. The study acknowledges the challenges of analyzing the various factors that affect evacuation and the time-consuming nature of simulation-based assessments during the early design stages. To address this, the researchers use two deep learning algorithms, Pix2Pix and XGBoost, to assess the evacuation process. The Pix2Pix model accurately generates heat maps showing potential congested routes, while the XGBoost model effectively predicts evacuation times with an average inaccuracy of 36 seconds. The study suggests that this approach offers a fast, reliable alternative to the typical time-consuming evacuation simulations during the early stages of design, enabling more efficient and safe architectural planning.

Wang et al. (2017) conduct a study to enhance evacuation efficiency during emergencies in complex infrastructure. The research addresses problems arising from competitive behavior and congestion in narrow corridors during evacuations, which can significantly reduce evacuation rates and exit efficiency. The study introduces a unique technique that combines network analysis with BIM to create a decision-making framework for evacuation routes based on graph theory. This strategy also integrates psychological elements and simulation studies to influence evacuation behavior, considering how flames, dust, and psychological stress affect the desire of individuals to flee. The study demonstrates that integrating advanced micro-pedestrian models, computer-aided techniques, and psychological insights can optimize evacuation procedures.

2.5 Generative Design

Generative design is a paradigm shift in the AEC industry that leverages algorithms to generate an array of design possibilities, all of which adhere to a set of predefined criteria and constraints (Filippo, 2021). Generative design learns and evolves with each cycle, refining subsequent iterations and increasing efficiency. Traditional design processes in architectural and engineering design have been predominantly characterized by linearity and determinism, with the scope of solutions frequently constrained by subjective experiences. Generative design deviates from this paradigm by adopting a non-

linear and investigative approach. It automates the generation of design variations, facilitating a comprehensive exploration of the design space that would be otherwise time-consuming for a human designer to replicate manually.

Zarzycki (2012) explains the use of parametric modeling for generative design. Parametric modeling is a crucial methodology that allows for creating complex architectural forms by adjusting algorithmically defined parameters and constraints. This method helps establish variables that control design characteristics, which are then interconnected through computational rules to generate numerous design variations. Furthermore, the iterative process can be fine-tuned by integrating with analytical tools to optimize the design for specific performance metrics.

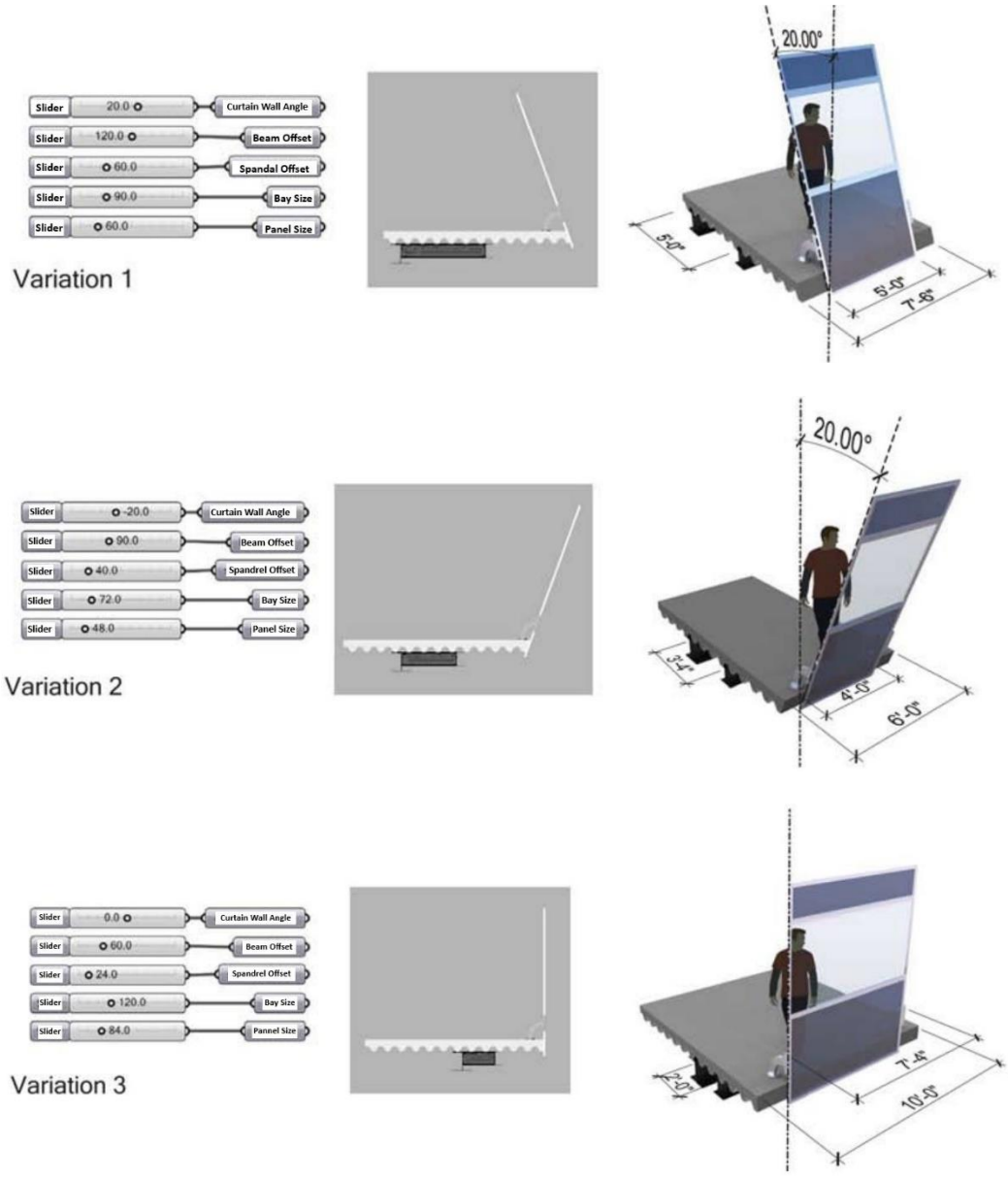


Figure 2.11: Generative design using parametric modeling achieved with Grasshopper (Zarzycki, 2012)

BIM models play a crucial role in modern architecture and construction projects but creating them may be laborious and susceptible to user mistakes and inaccuracies. Moreover, making changes to finished models can be a challenging task. In this context, generative design is a promising solution to address these issues. Generative design aims to enhance efficiency, minimize the risk of errors, and increase design flexibility by automating tasks and interconnecting components. It also offers the advantage of using an implemented parametric model as a template for future projects or for comparing different design versions quickly. This means that designers can define a set of parameters and generate many solutions in a fraction of the time it would take manually. In the initial stages of the design process, generative design allows for more iterations and enables faster implementation of changes than traditional design models (Fischer et al., 2012).

Generative design is a programming-centered approach that empowers designers to use computer programs to autonomously generate potential solutions for a given problem or task. This methodology is particularly pertinent to the AEC industry, which often entails a diverse range of potential solutions for a single problem and involves numerous factors that influence the optimal solution. By leveraging generative design, professionals can explore various design possibilities and efficiently navigate the complex landscape of design considerations (Kalkan et al., 2018).

Based on generative design, a generalized framework called Model Healing has been suggested, which presents algorithmic approaches for parametric construction planning and model-based presentation in solution domains to automatically modify inconsistent building plans (Wu et al., 2022). A healing metric is used as an assessment indication throughout the adjustment procedure to choose the design version that complies with building rules while deviating the least from the original design. An initial version of a responsive construction layout for the German norm is developed to show the framework's applicability, ensuring emissions escape from exit points. The fundamental variant is "healed" to a conforming layout by lightly increasing the size of the emissions ports and installing one extra emissions release to the roof. In recent years, the use of AI methods in generative design has emerged as a significant advancement. Li et al. (2024) provides an autonomous structure planning method that streamlines the method from generative design to preparation stages for precast structures. The system uses the BIM model to develop new design choices that properly anticipate

construction efficiency under wind conditions. This is accomplished using a trained Graph Convolutional Neural Network (GCNN) to optimize design choices efficiently.

2.6 Optimization Algorithms

Optimization algorithms represent a class of computational methodologies designed to systematically search for the optimal solution within a predefined solution space for a given problem (Stork et al., 2022). An optimization algorithm aims to decrease or raise a certain objective function while satisfying certain constraints, thereby addressing various real-world problems across diverse domains (Gad, 2022).

Defining an objective function is a crucial part of any optimization process because it outlines the objectives of a particular problem. The objective function is a function that is either maximized or minimized based on the optimization goal. The feasible region is defined based on the constraints. It represents the region where all objective function constraints are satisfied, leading to the decision variables being located at the corners of the feasible region. The optimum solution is the values of decision variables at which the objective function is maximized or minimized.

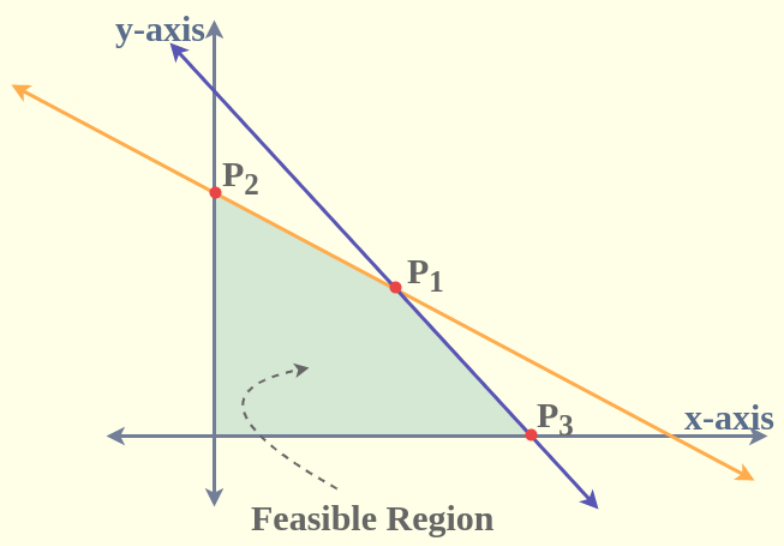


Figure 2.12: Graphical representation of feasibility region of an objective function (Berhe, 2012)

Optimization algorithms are effective tools for addressing challenging issues that are frequently utilized in a variety of industries. Every approach contains unique advantages and drawbacks. Therefore, the algorithm selection is typically determined by the individual situation. For example, a study showed that PSO provides the shortest distance solutions, while SA is more efficient in execution time (Mirsadeghi & Khodayifar, 2020).

2.6.1 Genetic Algorithm

Holland and his fellow researchers created the notion of the GA in the early 1960s (Sivanandam & Deepa, 2008). The GA is motivated by the concept of natural selection. Natural selection causes inadequate and unsuitable creatures in their habitat to become obsolete. Powerful individuals have a better chance of carrying on their DNA to subsequent generations through breeding. Over time, organisms with the proper mix of genes dominate their populations. The GA is frequently employed to develop excellent approaches for optimization and query issues by employing ecologically based drivers. The GA was first designed to populate the population with random candidate solutions and evolve the ideal answer from generation to generation.

Zhou et al. (2022) introduce a novel framework using the GA for the optimum performance of the hospital. The framework's objective is to adapt the spatial configuration of healthcare facilities to fulfill varying operational demands, ensuring superior effectiveness under typical conditions and mitigating danger amid pandemics. The study first analyzes the key parameters influencing hospital layout. Subsequently, to address the dual objectives of efficiency and safety, the study establishes constraints in the form of a nearby fitness rating and probability of an infection index. Finally, an autonomous design process produces numerous building layout schemes, from which the best hospital building plan is chosen.

Zhao et al. (2019) propose a framework that utilizes GA to estimate construction project costs. The framework has been validated by evaluating the construction costs of 20 building projects, demonstrating its effectiveness and reliability. Zhang (2010) study GA to develop maintenance schedules to minimize user maintenance plan delays. It is observed that a larger population size led to a better spread of solutions. He et al. (2019) introduces an optimization model based on GA that assesses the factors and constraints leading to delays and increased costs in construction projects. The model utilizes BIM to simulate real construction projects before they commence, enabling the identification of all factors contributing to delays and cost overruns. Attia et al. (2023) develop a plugin for BIM authoring tools to reduce the time and cost of building construction projects using GA. Validation has shown that the plugin can reduce project time by approximately 20% and save project costs.

Tafraout et al. (2019) propose a framework for automating the generation of the structural layout of a building using a Genetic Algorithm (GA). A Multi-Objective Genetic

Algorithm (MOGA) has been utilized to generate multiple potential floor layouts that comply with structural and seismic building codes. Each floor layout generated by the GA is analyzed to determine its fitness score. The fittest layouts are then selected for the next generation through crossover and mutation. Finally, the best individual from the last generation is chosen as the optimal architectural floor layout for the building.

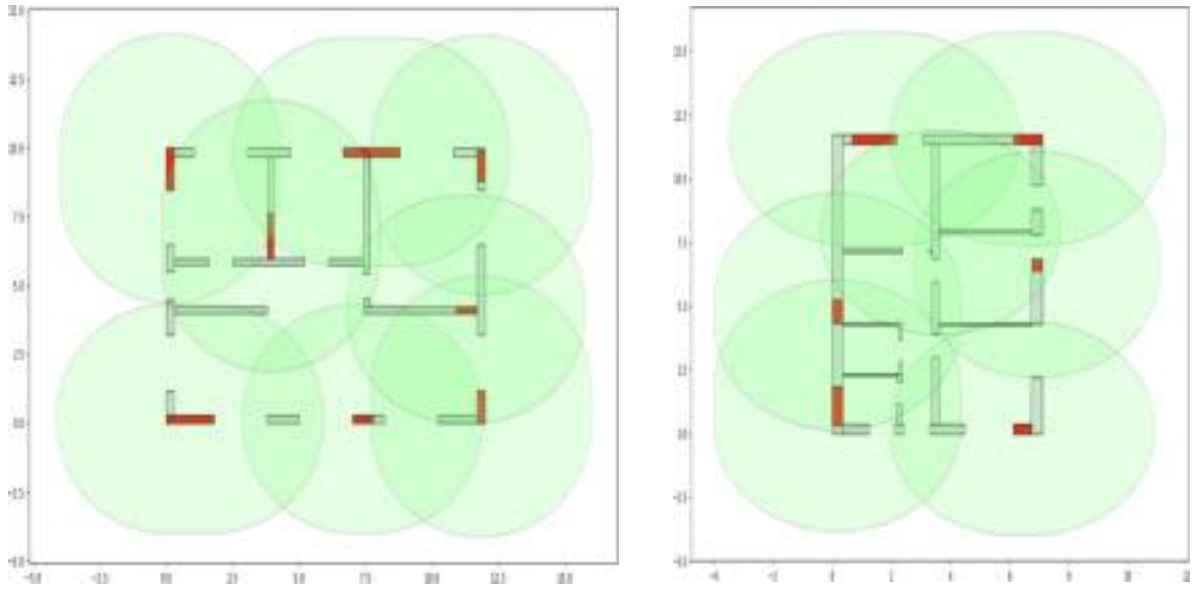


Figure 2.13: Generated floor plans using GA

2.6.2 Simulated Annealing

Kirkpatrick et al. (1983) have used simulated annealing in combinatorial optimization for the first time. The simulated annealing technique relies on the annealing process in manufacturing, which involves rapidly heating a metal to extreme temperatures and then slowly cooling it. The simulated annealing techniques tackle solo and multiobjective optimization queries by hiding a desired global minimum/maximum amid multiple local minima/maxima.

In SA, the procedure of search begins with an intense phase (a starting solution) and slowly reduces the temperature (a control variable) till it reaches a low-energy state (the ideal solution). The key benefit of simulated annealing is its capacity to flee regional minimums and settle on a single global minimum. The simulated annealing is also simple to construct and requires no previous understanding of exploring regions (Chopard & Tomassini, 2018). The simulated annealing is utilized to propose a classification system of existing ACC approaches by considering various criteria such as framework development concepts, industry application suitability, and open standards compatibility (Doukari et al., 2022).

2.6.3 Particle Swarm Optimization

Kennedy and Eberhart developed the particle swarm optimization algorithm, which replicates interpersonal interactions (Juneja & Nagar, 2016). Particle swarm optimization was initially developed for continuous problems, but it cannot handle discrete problems. The two developed an independent binary variant of the particle swarm optimization to overcome this issue. As the name asserts, this method draws inspiration from the swarm of insects. Analogous to how insects identify an optimal location within a swarm, PSO employs the same principle to ascertain the most suitable solution within a given search space. Insect swarms demonstrate cooperative behavior in their hunt for food, which is replicated in the particle swarm optimization method. Each part of the colony adapts its search pattern depending on its unique learning experiences and those of other members, optimizing the collective search process. This dynamic adaptation of search patterns is a crucial feature of the particle swarm optimization algorithm, contributing to its efficacy in solving complex optimization problems.

Han et al. (2019) implements the particle swarm optimization algorithm using a visual programming language (VPL) to facilitate multiple-objective building design optimization to reduce building energy consumption. An office space in the building is chosen to investigate how the opening, depth, and window-wall ratio affect the space's sunlight heat transfer. After determining the impact of these design parameters, a Multi-Objective Particle Swarm Optimization (MOPSO) solver is created and utilized to optimize the aforementioned structure layout parameters to improve building performance, such as lowering solar radiation heat gain and increasing daylight factor.

3 Methodology

This chapter presents the proposed framework for the ACC and the optimization of the parametric model, as shown in Figure 3.1. The framework is structured into a four-step process. It begins with a comprehensive review of building codes, focusing on two main objectives: identifying the required values for building egress compliance and deriving the relevant objective function. Next, all essential parameters required for the egress requirements are introduced in the BIM model during the parametric modeling. The compliance checking is then carried out by comparing the egress requirements mentioned in the building regulations with the parametric values obtained from the BIM model. In case of non-compliance, the parameters related to the violation are automatically adjusted to ensure the building model is code-compliant. Finally, the framework concludes with an optimization phase, where the most efficient design solution is identified to provide optimum security for occupants during emergencies.

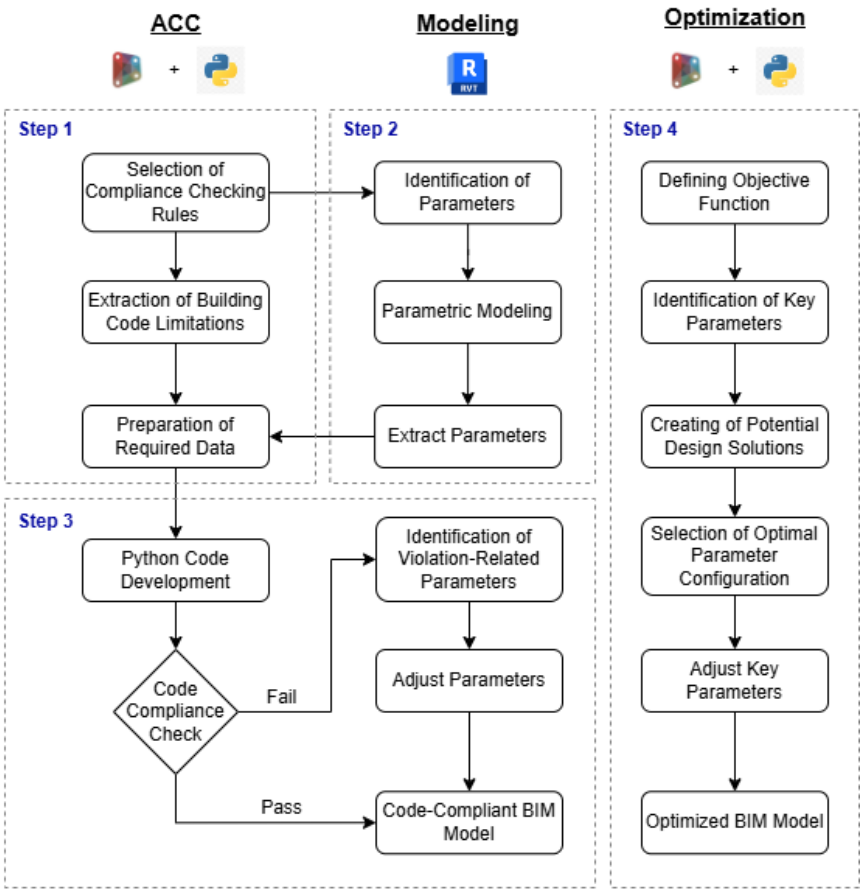


Figure 3.1: Illustration of the proposed framework for ACC and building design optimization. Step 1: Review of building codes. Step 2: Parametric BIM modeling. Step 3: Compliance checking. Step 4: Optimization.

3.1 Code Compliance Framework

The code compliance framework is designed to correct the BIM models to ensure compliance with building codes. It can automate checking code compliance for a building's means of egress and automatically adjust violations to ensure code compliance. After a thorough review of the IBC, ten compliance-checking rules specifically related to egress requirements were selected (Section 4.1.2). These rules were chosen with the aim of addressing design requirements that are associated with the means of egress. Based on these selected rules, key parameters for modeling the case study were identified and created. The case study considered in this thesis lies in the category of education occupancy type; therefore, the building model must follow the following requirements.

Table 3.1: Summary of the necessary values of education occupancy type as stated in the IBC

Building Section	Required Value	IBC Section
Minimum Corridor Width	72 inches (1828.8 mm)	1020.2
Required Capacity of Corridor	OL * 0.2 inch (5.1 mm)	1005.3.2
Minimum Stairway Width	44 inches (1118 mm)	1011.2
Required Capacity of Stairway	OL * 0.3 inch (7.6 mm)	1005.3.1
Minimum Ceiling Height	7 feet, 6 inches (2286 mm)	1003.2
Minimum Door Width	32 inches (813 mm)	1010.1.1
Minimum Door Height	80 inches (2032 mm)	1010.1.1
Minimum Space between two Doors	48 inches (1219 mm)	1010.1.8
Minimum Number of Exits per Story	2	1006.3.2
Egress Distance	250 feet (76.2 meters)	1017.2

**OL = Occupant Load*

A parametric BIM model is designed to incorporate all required information for code compliance. Parameters and constraints are defined at the element level, such as door width and height, as well as between different building elements, for example, corridor width and space between two doors. Furthermore, certain required values indicated in

the IBC (Table 3.1) can be retrieved directly from the BIM model using the design authoring tool's API. This eliminates the need to create separate parameters. For instance, data such as the number of exits per story, egress distance, and ceiling height can be accessed through the API. After the parametric modeling is completed, the necessary information for code compliance is extracted from the design authoring tool, which serves as input for the Python node. The checking functions are created with Python to verify if the building model parameters lie within the specified range of building code or not. If the checking function fails, the difference is automatically calculated, and the non-compliant parameter value is adjusted to the required value of IBC, and if the checking function passes, the parameter value remains unchanged (Figure 4.13). Once all the checking functions have validated the parameter values, a dictionary is created with the “Parameter_Name: Parameter_Value” pairs with the adjusted values of parameters. Dynamo utilizes this dictionary to assign the corresponding parameters to the code-compliant values, ensuring the building model complies with the IBC regulations.

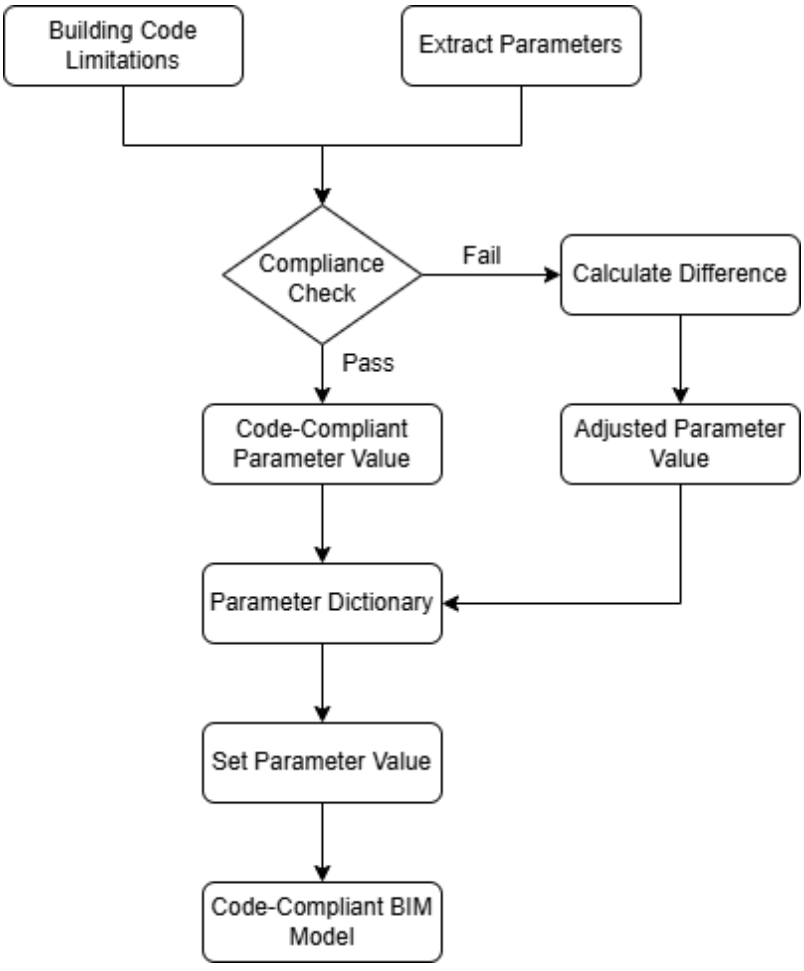


Figure 3.2: Workflow of the Code Compliance Framework

3.2 Optimization Framework

This proposed framework optimizes the egress distance while considering building layout constraints. Therefore, during the optimization process, only those parameters will be considered that affect the egress distance, while other parameters used during code compliance will not be considered. The exit access travel distance, also known as the maximum egress distance, is specified in Table 1017.2 of the IBC. The distance is determined based on the building's occupancy classification. To reduce the occupants' travel distance, the objective function can be defined as

$$\text{Maximize } f(x) = \sum_{i=1}^{n} (d_m - d_a) \qquad (1)$$

Subject to: $d_a \leq d_m$

d_m is the maximum egress distance (IBC – Table 1017.2). d_a is the available egress distance. n is the number of building stories.

To calculate the available egress distance, the floor plans of each building story are generated using the API of the design authoring tool. The room doors serve as the points of origin, and the floor exits (door/stairway) serve as the points of destination (Figure 4.19). A developed tool is used to find the shortest distance, which includes a custom node called "VisibilityGraph.ShortestPath" (Ortega, 2018). This node requires three inputs: the floor layout, points of origin, and points of destination. It computes the distance from each origin point to each destination point and subsequently identifies the shortest path by comparing these calculated distances.

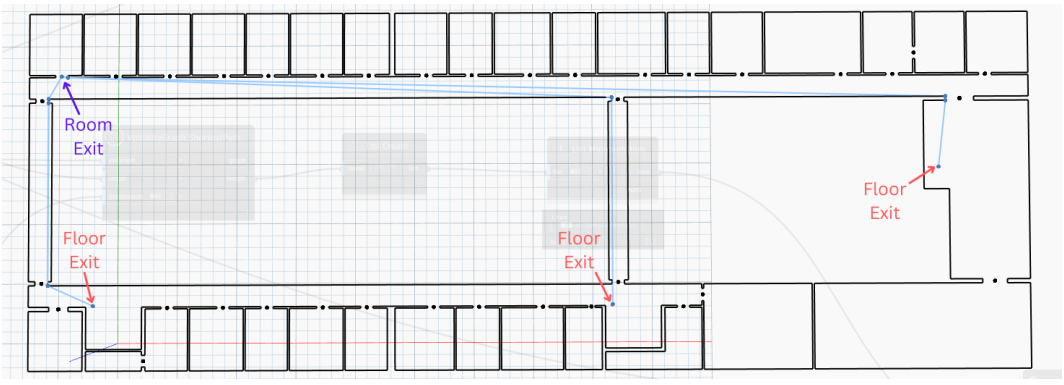


Figure 3.3: Illustration of calculating the shortest path of a room

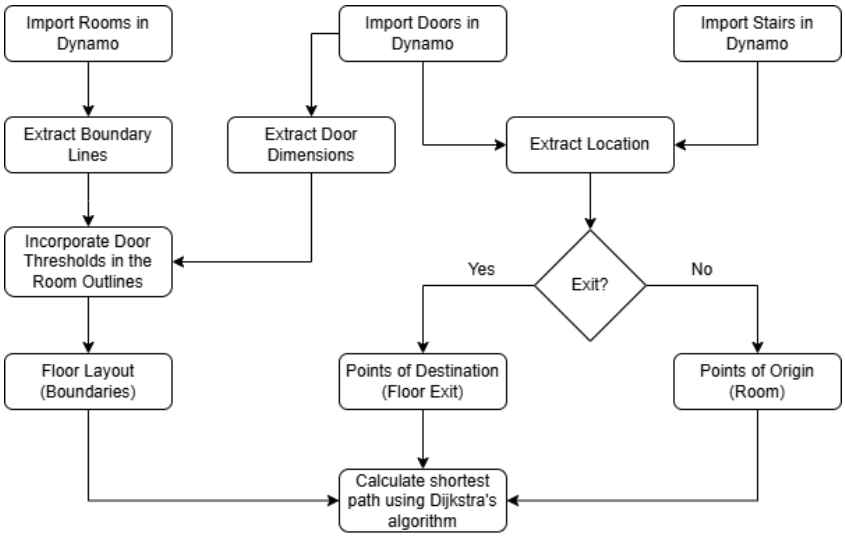


Figure 3.4: Workflow of calculating the shortest path

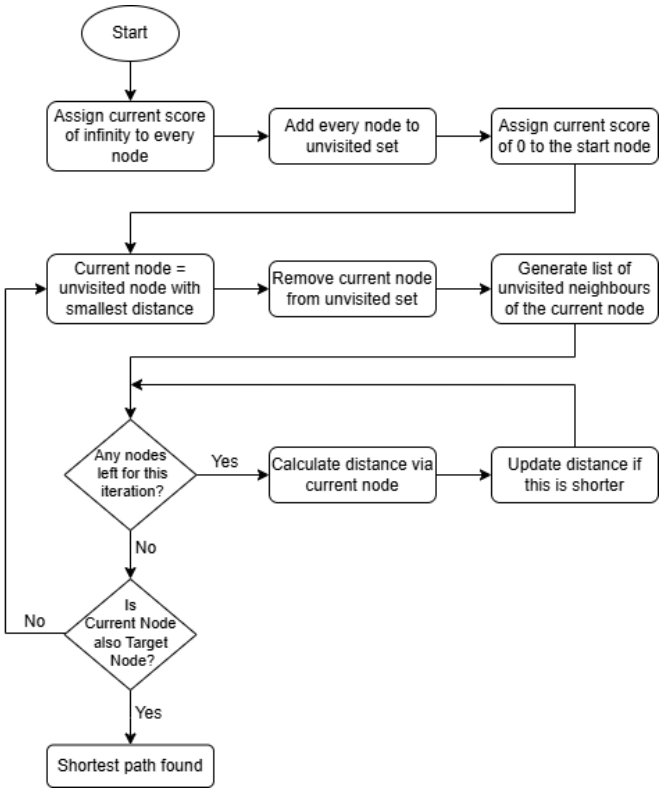


Figure 3.5: Dijkstra's Algorithm to calculate the shortest path

Real-coded genetic algorithm is used for the optimization of egress distance. A total of ten design variables consisting of floating-point numbers are defined. Each design variable represents the parameter that affects the egress distance.

$$x = \{x_1, x_2, x_3, \dots, x_{10}\} \quad (2)$$

The move limits of design variables are defined for the generation of design space. The lower limit is based on the required minimum values of IBC, while the constraints to preserve the initial design topology determine the upper limit (Table 5.3). These

move limits are set according to the requirements of IBC to ensure that the building model remains code-compliant during the optimization process.

$$x_L \leq x \leq x_U \quad (3)$$

x_L is the lower limit of the parameter range. x_U is the upper limit of the parameter range.

The roulette wheel selection, also known as the fitness-proportionate selection, is used for the genetic algorithm's selection process. This method effectively balances rewarding higher-fitness individuals and preserving diversity in the population. Even lower-fitness individuals still have a chance to be selected, encouraging genetic diversity, which is crucial for preventing the algorithm from getting stuck in the local optima.

The initial population is created by selecting a floating-point number within the defined design space for each variable. Parents are then chosen from the population using the roulette wheel selection method. This selection method selects individuals based on their fitness scores. Each individual's chance of being picked is proportional to their fitness score. Initially, the fitness score of all individuals is calculated, and then a random number is generated between 0 and the total fitness. The selection function iterates over the population, summing up the fitness scores cumulatively. The corresponding individual is selected when this cumulative sum exceeds the random pick. Individuals with higher fitness scores occupy the more prominent segment and thus have more chances of being selected.

Table 3.2: Example of roulette wheel selection

S.No.	Fitness Score	Accumulated Fitness	Occupied (Segment)	Occupied (Percentage)
1.	16	16	0 - 16	17%
2.	3	19	17 - 19	3%
3.	25	44	20 - 44	27%
4.	40	84	44 - 84	42%
5.	10	94	85 - 94	11%

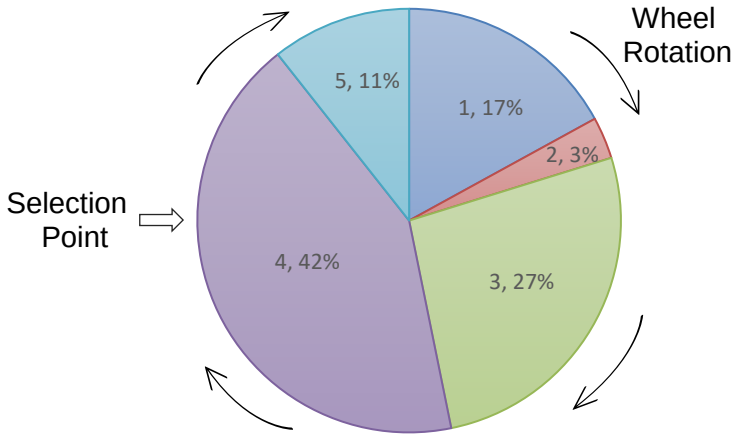


Figure 3.6: Illustration of roulette wheel selection based on fitness scores

```
def select_individual(population, fitness_scores):
    total_fitness = sum(fitness_scores)
    pick = random.uniform(0, total_fitness)
    current = 0
    for individual, score in zip(population, fitness_scores):
        current += score
        if current > pick:
            return individual
```

Figure 3.7: Python code for the roulette wheel selection

Genetic algorithm has different types of crossovers, among which the uniform crossovers method has been utilized to mix chromosomes. In uniform crossover, each child's gene is selected randomly from one of the corresponding genes of the parent chromosomes with a 50% crossover. Each child's gene is equally likely to be inherited by any of the parents.

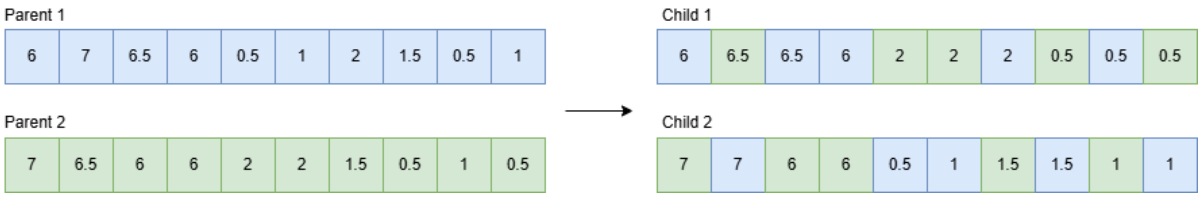


Figure 3.8: Uniform crossover of parent chromosomes

```
def crossover(parent1, parent2):
    child = {}
    for key in parent1.keys():
        child[key] = parent1[key] if random.random() < 0.5 else parent2[key]
    return child
```

Figure 3.9: Python code for the uniform crossover

Mutation is introduced to create diversity among the offspring, preventing the algorithm from getting stuck in local optima or converging prematurely and promoting better parameter space exploration. The random resetting approach of mutation with a 10%

probability has been used, which assigns a random value from the defined parameter range.

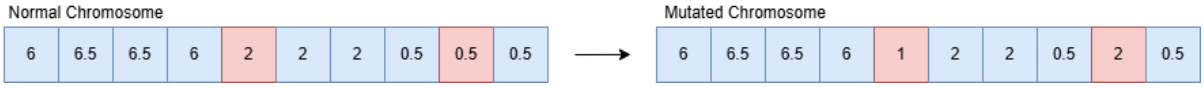


Figure 3.10: Random resetting mutation

```
MUTATION_RATE = 0.1

def mutate(individual):
    for key in individual.keys():
        if random.random() < MUTATION_RATE:
            if key == "Corridor_Width":
                individual[key] = random.choice(range_Corridor_Width)
    return individual
```

Figure 3.11: Python code for random resetting of genes with 10% mutation rate

The children then replace the current population to form the next generation. This process is repeated until the maximum number of generations is reached. Once the last generation is created, the best individual is identified from it, and the genetic algorithm is stopped. The chromosomes of this individual are termed as the optimized parameter configuration and applied to the BIM model, leading to an optimized building design that enhances occupant safety.

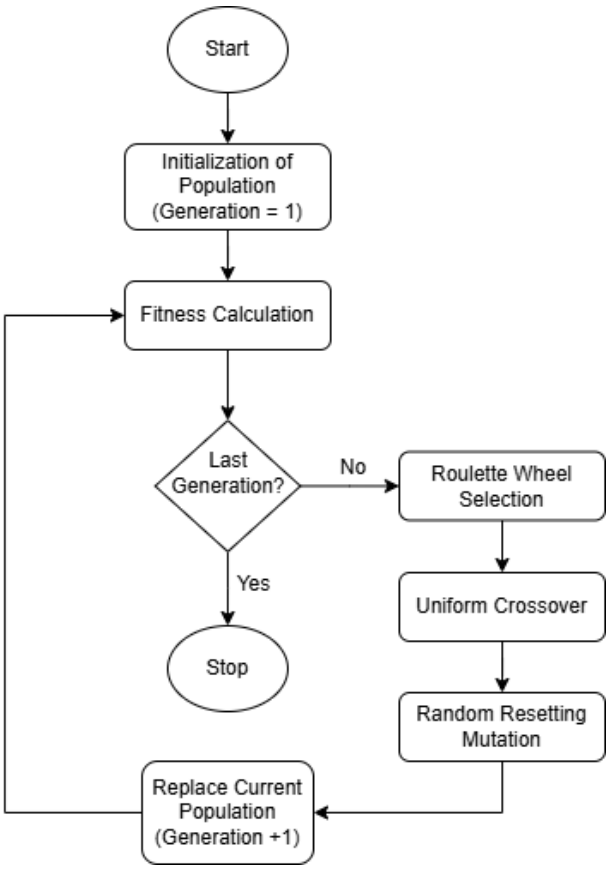


Figure 3.12: Workflow of genetic algorithm used for the optimization of building design

4 Implementation

4.1 Input Requirements from IBC

The IBC is a comprehensive collection of principles developed by the International Code Council (ICC) and is chosen as the regulatory framework for this thesis. The IBC establishes baseline criteria for various aspects of building design and construction, including material selection, structural system design, fire safety implementation, and the provision of accessibility features (IBC, 2018). These standards are basically intended to protect the welfare of building occupants and others in the community.

Chapter 10 of the IBC defines the fundamental standards for the design of egress systems, which are positioned as the principal method for protecting humans within built environments by allowing for the rapid relocation or evacuation of inhabitants. This chapter includes descriptive and performance-oriented terminology to lay the groundwork for developing a safe egress system that applies to all occupancies. It includes all parts of the egress system, including exit access, exits, and exit discharge, as well as the design criteria and laws that govern individual components. The criteria include precise specifications for the size, arrangement, quantity, and protection of egress components. Furthermore, the chapter describes the functional and operational qualities that allow for safely using these components without requiring specialized expertise or effort.

4.1.1 Occupancy Classification and Use

Chapter 3 of the IBC outlines the criteria for classifying buildings and structures into different use groups and occupancies (Appendix C). Occupancy classification is the official categorization of a building's primary purpose or a specific part of it. Buildings are classified into one or more occupancy groups based on the potential hazards and risks associated with their intended use. Different occupancy classes and uses involve varying danger levels and risks to building occupants. The process of occupancy classification is crucial in determining many aspects of construction, including means of egress. This classification is an important tool in ensuring the safety and functionality of built environments.

4.1.2 Selection of Code Compliance Rules

After a detailed review of Chapter 10 of the IBC, ten checking rules are finalized upon which code compliance will be conducted. Each of these rules is related to the egress path and ensures the safety of occupants.

1. Minimum Corridor Width

One of the most important parameters in case of egress requirements is the width of corridors, which will be used to evacuate occupants in an emergency. The width of the corridors should be sufficient to accommodate the maximum number of possible occupants. Section 1020 of the IBC describes the requirements for the design and construction of corridors in detail. Table 1020.2 provides the minimum corridor width for different types of occupancies. As the next chapter of this thesis explains the case study of the N6 building of the Technical University of Munich (TUM), therefore the occupancy classification of “Educational” (Group E) is considered while taking the egress requirements from the code. The main corridor in the case study should have a minimum width of 72 inches (1828.8 mm) since the occupant load is over 100. Side corridors with an occupant load of less than 50 must have a minimum width of 36 inches (914.4 mm) to ensure safe evacuation space for occupants.

2. Required Capacity of Corridors

Section 1005.3.2 of the IBC explains the corridor's required capacity based on occupant load. The required capacity of egress components other than stairways shall be calculated by multiplying the occupant load served by the egress component by the egress capacity factor, i.e., 0.2 inch (5.1 mm) per occupant.

$$Corridor\ Capacity = \frac{Occupant\ Load}{Egress\ Capacity\ Factor \approx 0.2\ inch\ /\ 5.1\ mm} \quad (4)$$

Where:

$$Occupant\ Load = \sum_{i=1}^{n} \frac{(Room\ Area)_i}{(Occupant\ Load\ Factor)_i} \quad (5)$$

$$n\ =\ Number\ of\ Rooms\ Connected\ with\ the\ Corridor$$

The occupant load factor is the maximum floor space permitted per occupant. It is used to calculate occupant load, which is the maximum number of occupants that can occupy a building or a section of a building at any given moment. The occupancy load factor is determined based on the function of the space. If it is for a space where usually

many occupants are present, it will have a lower occupant load factor. On the other hand, if the function of space is such that at any given time, usually there are few occupants present, then such space will have a higher occupancy load factor. Table 1004.5 of the IBC provides the values of different occupant load factors. After examination of the considered case study, a total of six different functions of spaces were identified, and their occupancy load factor values are taken from Table 1004.5.

3. Minimum Stairway Width

Stairways are one of the essential egress components that must be designed to accommodate the occupant load in an emergency. Section 1011.2 of the IBC explains that the width of stairways should not be less than 44 inches (1118 mm). To elaborate on the requirements governing stairway landings, Section 1011.6 of the IBC states that a floor or landing must be provided at the top and bottom of each stairway. The width of these landings, measured perpendicular to the direction of travel, must at least match the width of the stairways they service. Doors that open onto a landing shall not reduce the landing to less than half of its minimum width. The door cannot protrude more than 7 inches (178 mm) into the landing when completely opened. These standards protect the safety and accessibility of stairways in various building types.

4. Required Capacity of Stairways

Section 1005.3.1 of the IBC explains the required capacity of stairways based on occupant load. The required capacity of stairways shall be calculated by multiplying the occupant load served by stairways by the egress capacity factor, i.e., 0.3 inch (7.6 mm) per occupant. When stairways serve many levels, the required capacity of the stairways serving each story should be determined entirely by the occupant load of each story.

$$Stairway\ Capacity = \frac{Occupant\ Load}{Egress\ Capacity\ Factor \approx 0.3\ inch\ /\ 7.6\ mm} \quad (6)$$

Where:

$$Occupant\ Load = \sum_{i=1}^{n} \frac{(Room\ Area)_i}{(Occupant\ Load\ Factor)_i} \quad (7)$$

$$n = Number\ of\ Rooms\ Connected\ with\ the\ Egress\ Path$$

5. Minimum Ceiling Height

Section 1003.2 of the IBC specifies building ceiling heights to allow enough headroom for safe occupant evacuation. According to this specification, the egress passage shall have a ceiling height of at least 7 feet 6 inches (2286 mm) above the finished surface of the floor.

6. Minimum Width of Doors

Section 1010.1.1 of the IBC explains the minimum width of the door. After considering the function of space and occupant load, it was decided that a minimum clear opening width of 32 inches (813 mm) should be provided. In the context of doorways equipped with swinging doors, the clear opening width is quantified by measuring the distance between the face of the door and the stop when the door is positioned at an angle of 90 degrees. In instances where an opening is comprised of two door leaves without a mullion, it is mandated that one leaf should furnish a minimum clear opening width of 32 inches (813 mm). This requirement ensures adequate access through the doorway.

7. Minimum Height of Doors

The height of each building door is critical to ensure the safety and efficient evacuation of occupants. According to Section 1010.1.1 of the IBC, the minimum clear opening height for building doors is specified to be 80 inches (2032 mm). This requirement is essential to accommodate each room's corresponding occupant load and facilitate safe and quick evacuation in an emergency.

8. Minimum Space Between Two Doors

Section 1010.1.8 of the IBC, titled "Door Arrangement" provides specific guidelines for the layout of doors, particularly focusing on series configurations. This section outlines that the minimum distance between two successive doors in a series should be at least 48 inches (1219 mm), along with the additional width required for a door that swings into this space. Furthermore, it stipulates that all doors in a sequence must swing in the same direction or away from the space between them. This minimum space between two doors ensures adequate space for safe and efficient movement through doorways, particularly in emergencies.

9. Minimum Number of Exits per Story

It is important to define the minimum number of separate exits that must be provided per building story or occupied roof to accommodate the exit discharge. Occupant load

is an important factor in determining the minimum number of exits. Section 1006.3.2 of the IBC classifies the occupant load per story into 3 groups, and a minimum number of exits is defined for each group.

10. Maximum Travel Distance

Establishing a maximum travel distance within a building while considering the means of egress is critical in ensuring the safety and efficiency of built environments. This key parameter considers the building's layout, size, and occupancy to determine the distance a person would need to travel to reach a safe exit. By carefully assessing these factors, building designers and safety professionals can ensure that occupants can swiftly and safely evacuate during emergencies, minimizing the potential for injury or loss of life. Section 1017.2 of the IBC provides the value of this key parameter based on the occupancy type and whether the building is equipped with a sprinkler system. A sprinkler system in a building can influence the “Exit Access Travel Distance” due to its role in fire suppression and safety. A sprinkler system can control a fire in its early stages, slowing its spread and reducing the smoke and heat produced, providing occupants additional time to evacuate the building safely.

4.2 Parametric Model Setup

In Revit, parameters and constraints can be defined on the element level (family parameters) and between different building elements (global parameters). Family parameters are employed in the creation of specific building elements. These parameters are specific to the family (i.e., the type of element, such as a wall, door, window, etc.) and control the properties of the elements within that family. For instance, a wall can be created by defining its width, height, and length as parameters. On the other hand, global parameters are established across various building components and can control dimensions or relationships between elements within the project. For example, a global parameter can control the distance between a wall and its reference grid. Global parameters facilitate coordinated updates across multiple elements, enhancing the design process's efficiency and accuracy.

Assignment of global parameters is possible in two ways; the first approach uses a global parameter as a label for a dimension string. When a label is assigned to a dimension string, the global parameter determines the value of the dimension. This approach is usually used to adjust the size of various elements in a model. The second approach is to add a global parameter directly to an element at the instance or type level. This

can be accomplished by choosing the element and locating the instance or type parameter. After clicking the button to the right of the parameter value, an associate global parameter dialogue opens, and a global parameter from the list is selected and assigned to the selected parameter.

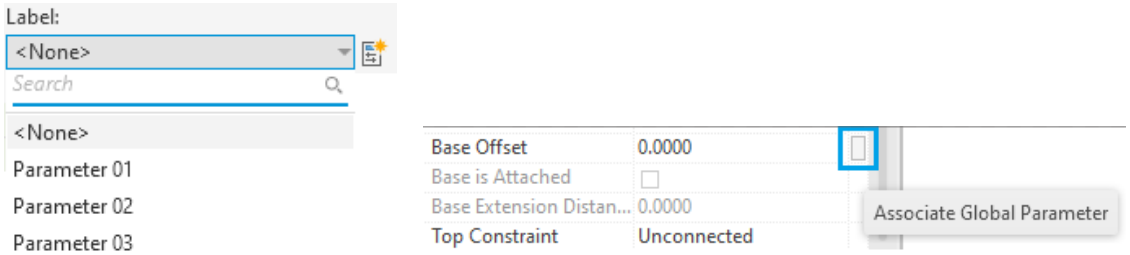


Figure 4.1: Two approaches of assigning a global parameter

4.2.1 Shared Parameters

Autodesk Revit offers the ability to make use of shared parameters. These are parameter sets that can be extended to various families or projects. In Autodesk Revit, shared parameters are saved as a text file (.txt) that is not associated with any family file or Revit project. This enables several families or projects to use the same set of parameters. The utilization of shared parameters provides significant benefits. Once created, this document can be used with other projects, avoiding establishing these parameters from scratch for any project requiring a means of egress evaluation. This promotes reusability and efficiency in the design process.

Shared parameters help generate custom tags. For instance, after creating a shared parameter for occupant load, it can be used inside the tag family to show the maximum number of occupants in each room. This makes it more feasible to view critical information within the building model. Using shared parameters improves the efficiency and accuracy of egress evaluations in building design. Shared parameters are an effective tool in the BIM process since they allow for reusability and uniformity. Considering the benefits of shared parameters, all the parameters necessary for evaluating the means of egress are defined as shared parameters.

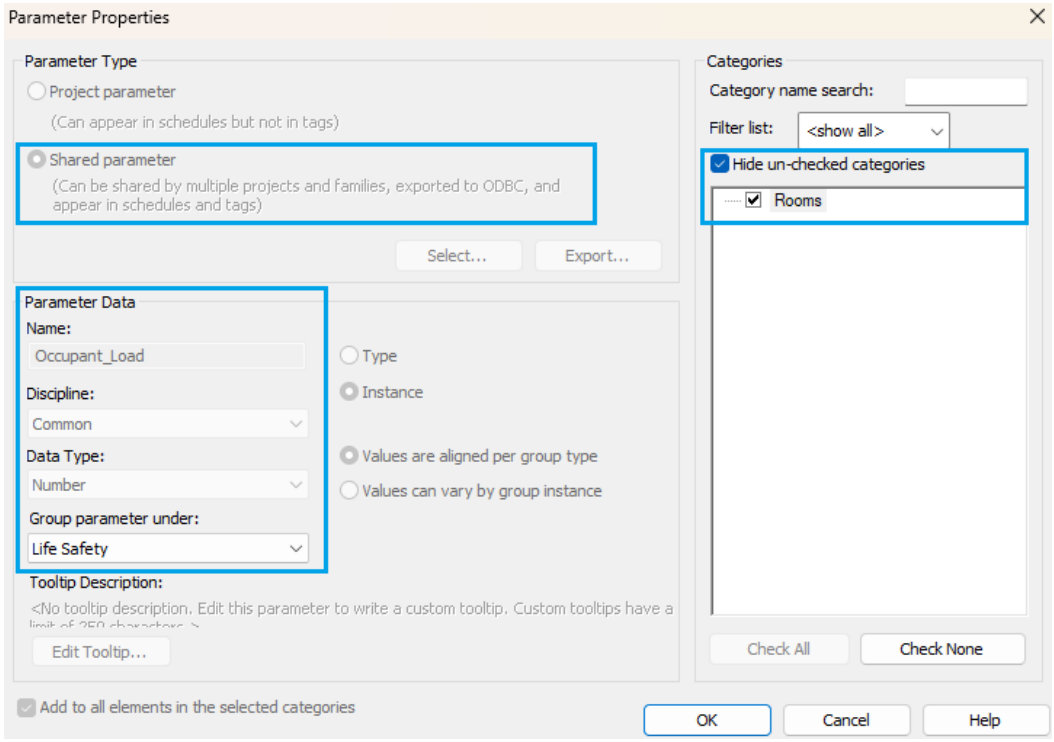


Figure 4.2: Defining occupant load of rooms as a shared parameter in Autodesk Revit

4.2.2 Setting Constraints for Parametric Modeling

In parametric modeling, it is essential to define a set of principles that will be followed during parametric modeling because, without any future consideration, it can lead to lots of confusion during the modeling process. Building elements are linked to one another using two kinds of constraints: dimensional and geometric. Dimensional constraints specify the dimensional value of geometric elements such as length, width, or thickness, while geometric constraints ensure that the geometric elements remain parallel or perpendicular to each other.

One of the important constraints in parametric modeling is the boundary constraint. After creating the boundary walls of a building, they need to be constrained so that any future changes in other parameters will not cause the boundary walls to exceed the defined constraints. In Revit, boundary constraints can be defined by using grids. Grids must be defined on the boundary, and the boundary walls must be locked with these defined grids to create the boundary constraints of a building.

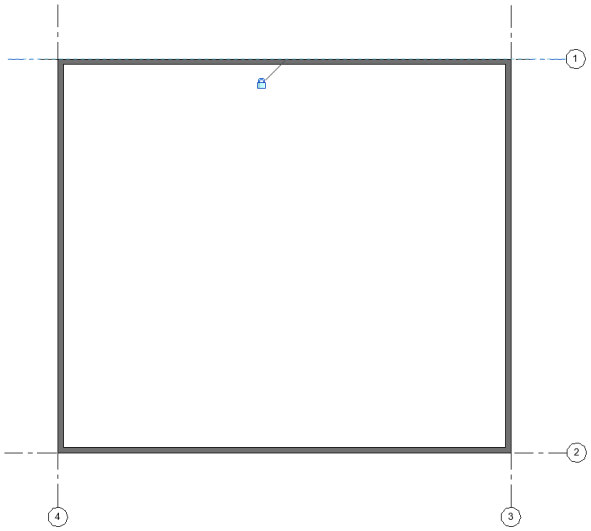


Figure 4.3: Defining boundary constraints of a building using grids

To evaluate and optimize the means of egress, one of the key parameters that should be considered is the width of the corridor. If the width is adjusted, the corridor walls will move from their original position, so it's important to constrain other attached building elements, such as adjacent walls, columns, stairways, or railings, to move along with the corridor walls.

4.2.3 Creation of Room Elements

During the code compliance check, each room area is required to calculate the required capacity of corridors and stairways (Section 4.1.2). For the optimization stage, each room's boundaries are required to generate one surface for finding the shortest path (Section 4.4.2). Considering these requirements of the thesis, it is necessary to define room elements in Revit for each closed space. Room elements in Autodesk Revit are 3D model elements used to define a closed space's boundaries and properties. These boundaries of the closed space are defined by certain types of building elements, including walls, floors, and ceilings. Additionally, room separation lines can also be used to modify the boundaries of the room according to the user's requirements. The properties of each building element can be defined whether it is a room-bounding element or not; it allows the user to accurately define room boundaries and to ignore elements that should not define the room boundaries. Once the boundaries of the room element are defined, its parameters are automatically computed, including room area, volume, perimeter, and unbounded height. One of the main advantages of room elements in parametric modeling is that the room's boundaries will change with the change of parameter values. As a result, the properties of each room will

automatically update according to the new boundaries, eliminating the need to manually adjust room areas for calculations of the required capacity of corridors and stairways.

Rooms can also aid in the visualization of the egress path of the building with the help of the “Color Scheme” built-in parameter of the floor plan view. This parameter assigns different colors to the rooms in the floor plan based on their properties. To assign a specific color to the egress path of the building, all rooms of the egress path should be given a unique name, and with the help of a color scheme built-in parameter, the rooms with this unique name will be given a different color than other rooms.

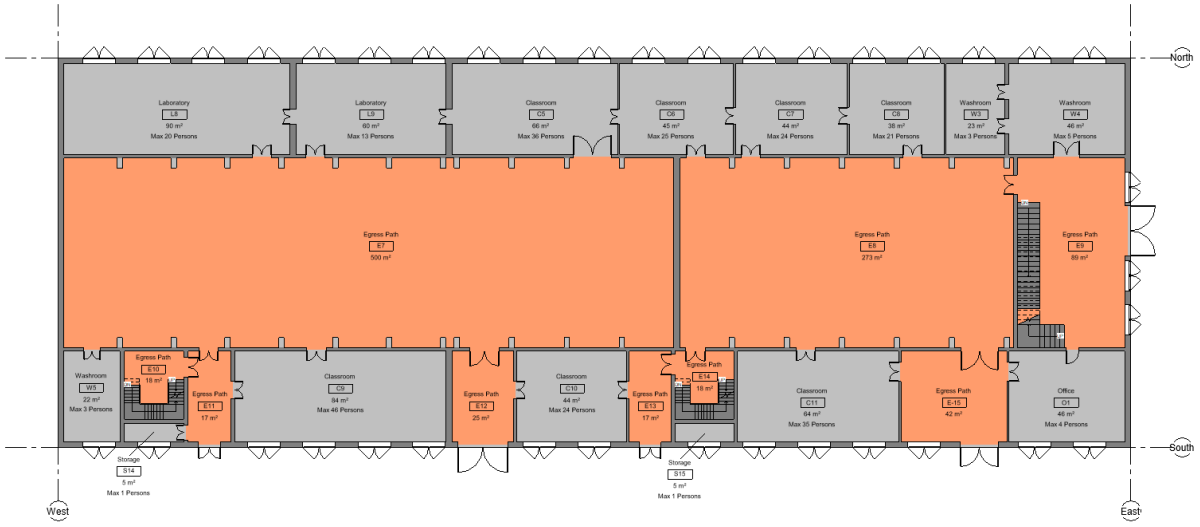


Figure 4.4: Using color schemes to identify the building's egress path (Orange: Egress Path, Grey: Rooms)

Rooms can also be imported in Dynamo, where a room of each building story can be classified, and their boundaries are used for defining one combined surface of the floor plan. Another advantage of using room boundaries in defining the floor surface is that it will automatically include the barriers that occupants face while going out of a building, as room boundaries are defined by walls and other obstacles that come in the egress path.

4.2.4 Determination of Required Capacity of Egress Path

The required capacity of corridors and stairways can be determined by the help of room schedules. By applying filters, specific building elements can be isolated, providing a focused view while concealing unnecessary information. Moreover, schedules offer the functionality to create calculated values through the use of formulas derived from different columns, enabling efficient analysis and decision-making.

To consider the rooms of different building stories separately, a separate room schedule is created for each building story, and for each schedule, a filter is applied to ignore the rooms of other building stories. Additionally, as the rooms of the egress path do not contribute to the calculations of their required capacity, these rooms will be filtered out. For example, in the case of the room schedule of the ground floor, the following filter can be used to consider only those rooms that will be considered for the calculation. This filter allows the selection of only ground floor rooms, excluding the egress path, as it will not be considered for the calculation of the required capacity of corridors and stairways. Furthermore, if there are more than one corridor in a building story, an additional parameter of “Location” could be introduced to filter out the connected rooms for each corridor.

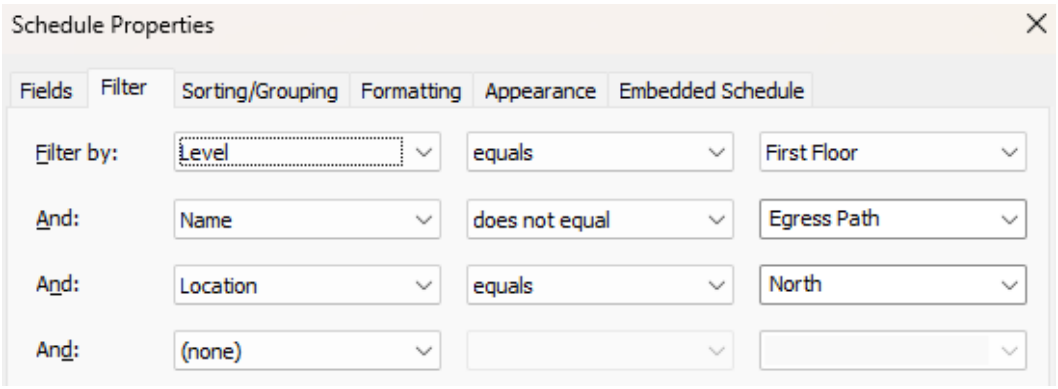


Figure 4.5: Using filters to include rooms in the schedule that are required for the calculations

The initial step of the calculation of the required capacity of corridors and stairways is the calculation of occupant load, which can be calculated by dividing the room area by the occupant load factor. The occupant load, representing the number of people permitted in a particular space, must be specified as an integer value. To ensure increased safety, a "roundup" keyword is used to round the occupant load to the nearest whole number.

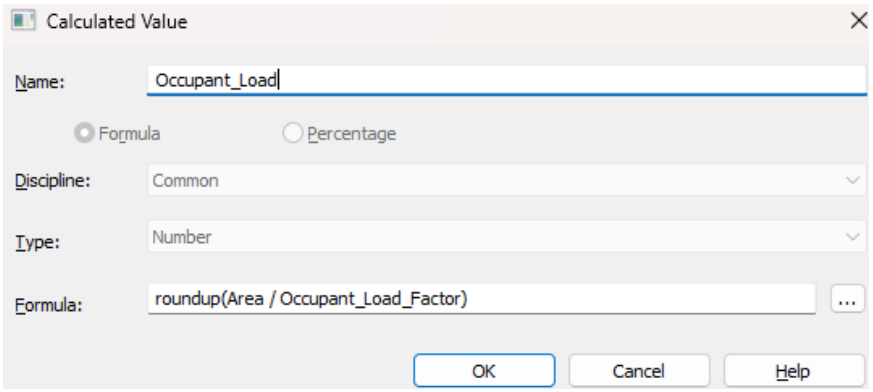


Figure 4.6: Calculation of occupant load of a room

4.2.5 Custom Tag for Occupant Load

Room elements in Autodesk Revit are 3D model elements used to define a closed space's boundaries and properties (Section 4.2.3). However, room tags are required to display the information of these 3D elements in floor plans, elevation, or section views. Room tags are annotation elements used to display specific information about room elements for a better understanding of users, such as room name, number, height, area, volume, etc. With the help of shared parameters, room tag families can be customized to calculate and display extra information about rooms. In the case of means of egress, occupant load is one of the important parameters used for code compliance and optimization; that's why custom tags are created for the calculation and display of the occupant load of each room element. Initially, parameters required to calculate the occupant load are identified, including the room's area and occupant load factor. Area parameters are available by default for each room element; however, shared parameters will be used to import occupant load factor into the custom tag. As all parameters of this thesis are shared parameters for the purpose of reusability and increased efficiency in the design process, there is no need to create a new parameter for the occupant load factor in the custom tag.

After identifying and importing the parameters required for calculating the custom tag of occupant load, a label is created to use these parameters to calculate and display the room's occupant load. A label is an annotation element just like a text, but instead of displaying a single piece of information, labels can use parameters to create equations to calculate and display the custom information for each room. To calculate occupant load, the room area is divided by the occupant load factor (Figure 4.7). After the creation of the label, the custom tag can be imported into the project and used to display the occupant load of any created room in a building.

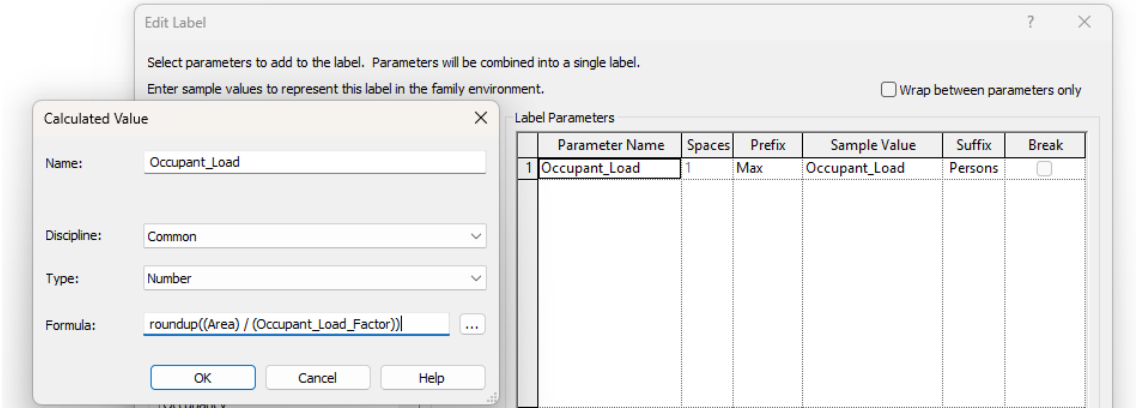


Figure 4.7: Creating a custom tag for the occupant load of building rooms

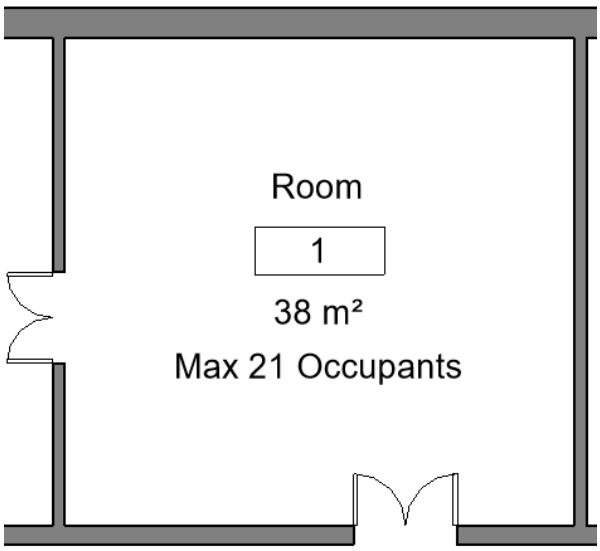


Figure 4.8: Illustration of a custom tag for occupant load of a room

4.3 Code Compliance Checking

After selecting code compliance checks and creating the BIM model, this section will explain how to automate the code checking of the chosen rules. It's important to consider various criteria and limitations to ensure accurate results. Dynamo is a visual programming language that presents code in a graphical format and accesses necessary data through Revit API, allowing direct interaction with the building model's information. This approach visually represents the rules and makes them easily understandable, which is particularly beneficial for editors with limited programming expertise. Therefore, dynamo (version 2.18.1) is used to automate the code compliance process in this thesis.

Ensuring that the BIM model is enriched with all the necessary parameters for the proposed framework to function correctly is essential. This thesis considers ten checking rules, and their corresponding parameters have already been included in the BIM model for the framework's functionality (Section 3.1). The process of automating code compliance is divided into four steps. The first step involves importing the specified code limits of different building elements in the IBC into the Dynamo. The second step includes importing the relevant information from the BIM model into the Dynamo. The third step is the checking phase, in which the code values are compared with the corresponding values of the BIM model. The last step is the adjustment of violation-related parameters; the difference between the required values of the building code and the corresponding building data will be calculated and adjusted to make the building model code compliant.

4.3.1 Importing Requirement Specification

The first step in automating code compliance is to import all required values from the building code into the Dynamo script. These required values can be categorized into two types. The first type consists of minimum or maximum values specified in the building code, such as the maximum egress distance for an educational occupancy classification building equipped with a sprinkler system, which is 250 feet (76.2 meters) as per Table 1017.2. These required values are defined as variables inside the Python node of the Dynamo script and further compared with the corresponding building data.

```
#International_Building_Code_Requirements
Minimum_Corridor_Width = 1.828
Minimum_Stairways_Width = 1.118
Minimum_Ceiling_Height = 2.286
Minimum_Door_Width = 0.813
Minimum_Door_Height = 2.032
Minimum_Space_between_Two_Doors = 1.219
Minimum_Number_of_Exits_per_Story = 2
Maximum_Egress_Distance = 76.2
```

Figure 4.9: Creating variables in the Python node for the required values of the IBC

The second category of building code values consists of formulae that need to be calculated, rather than numerical values. For example, the required capacity of corridors and stairways is calculated by dividing the occupant load by the egress capacity factor (Section 4.1.2). When calculating the area of rooms, the areas of egress paths need to be excluded as they are designed according to the occupant load of the connected rooms. Considering these conditions, schedules in Revit filter out the rooms that need to be considered, and then the required capacity of corridors and stairways is calculated. These calculated values from schedules are then imported directly into the Python node of the Dynamo script for further comparison with the corresponding dimensions of corridors and stairways.

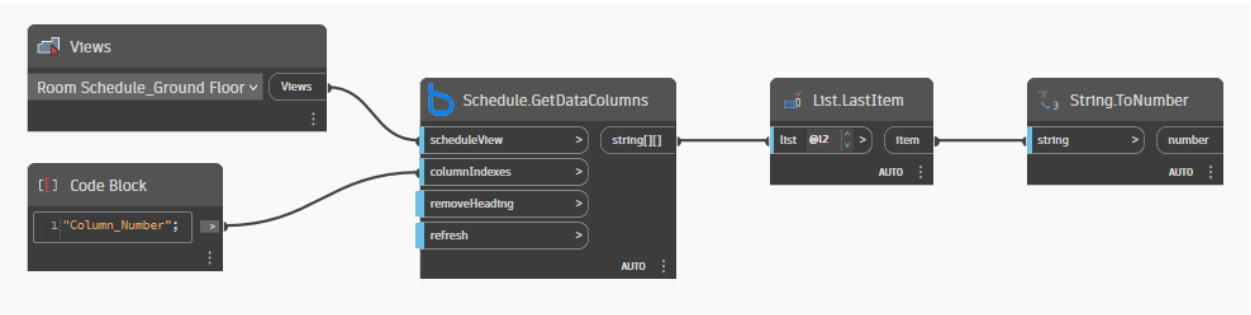


Figure 4.10: Dynamo script for importing the calculated required capacity of the ground floor corridor from a schedule

4.3.2 Importing BIM Model Information

After importing all the required values into the Dynamo script, it is necessary to have the corresponding building data for checking. Building model information can be imported into the Dynamo script using parameters. The parameters of the families used to create the building model elements are imported into the Dynamo script. For example, below is a Dynamo script for importing the family parameter "Door Width" of all the doors in the building model.

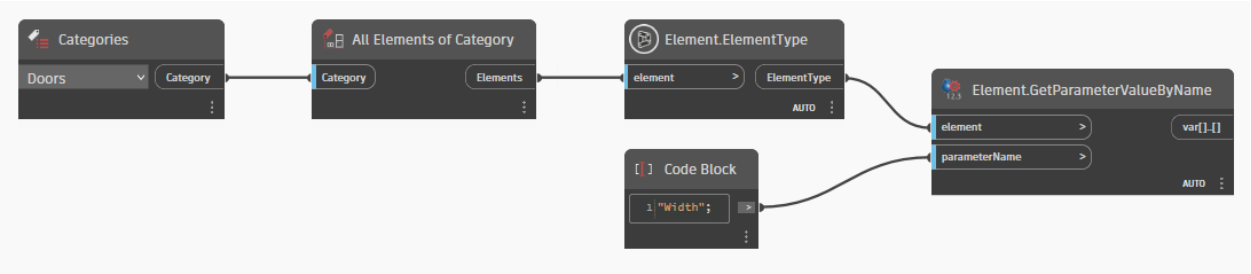


Figure 4.11: Dynamo script for importing family parameter of door width

Secondly, the building code includes certain values that aren't specifically for individual building elements but rather for relationships between elements, such as the width of a corridor. In these cases, it's necessary to define global parameters to manage and adjust them based on the building code's required values. Below is an example of a Dynamo script for importing global parameters.

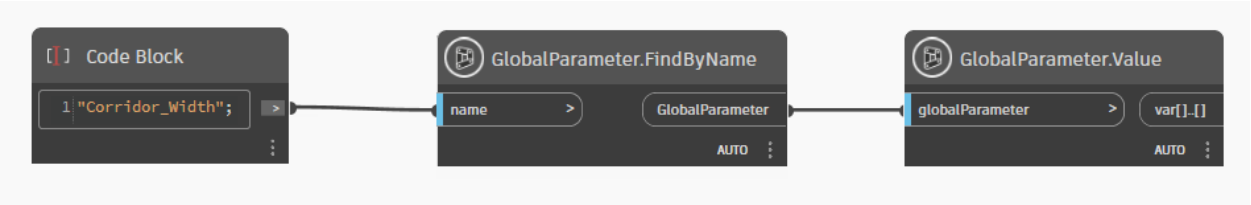


Figure 4.12: Dynamo script for importing global parameters into Dynamo GUI

4.3.3 Compliance Checking

Once all the required data for code compliance is imported into Dynamo, it is collected in the form of lists, which are then imported into the Python node. This thesis has defined 10 selection rules, and a checking function is created for each compliance checking rule. Each checking function imports 2 parameters: parameter value and the required value. Additionally, variables "Result" and "Difference" are created. The Result variable stores either "Pass" or "Fail" depending on the comparison between the parameter value and the required value. The Difference variable stores the difference between the parameter and the required value in case of non-compliance.

After importing the function parameters and creating the necessary variables, a check is performed to verify if the building model parameter falls within the specified range for the building code. If the parameter value is within the specified range, then "Pass" is stored in the result variable. If the parameter value is outside the range of the building code, the difference between the parameter value and the required value is calculated and stored in the difference variable, and “Fail” is stored in the result variable and the required value of IBC is assigned to the parameter value. At the end of the checking function, these two variables ("Result" and "Parameter Value") are returned. They are further used in Dynamo to show code compliance results and adjust the parameter values to make the building model code-compliant.

```
def Check_Minimum_Door_Width(Minimum_Door_Width, Door_Width):
    Result = 'Check(Minimum Door Width): Pass'
    for i in range(len(Door_Width)):
        if Minimum_Door_Width > Door_Width[i]:
            difference = round(Minimum_Door_Width - Door_Width[i], 3)
            Door_Width[i] = Minimum_Door_Width
            Result = f'Check(Minimum Door Width): Fail - Door Width has increased by {difference} meters'
    return Result, Door_Width
```

Figure 4.13: Checking function for the minimum width of the doors (Appendix A1)

It is important to gather all the results from the checking functions and organize them into a single data structure. In Python, this can be achieved using a dictionary. A dictionary is a robust data structure that stores values in "key:value" pairs. Similar to a traditional dictionary where meanings are assigned to words, numerical values can be assigned to unique keys in Python when defining a dictionary. This makes it easier to access the values using the unique keys. To better organize the results of the checking functions, a dictionary is created to store the adjusted parameter values with their parameter names as keys.

4.3.4 Adjustment of Violation-Related Parameters

After checking the code compliance, violation-related parameter values are adjusted to make the building model code compliant. The output of the Python node consists of two data structures. The first data structure comprises a list of outcomes from 10 compliance checks, providing visibility into successful and unsuccessful checks. The "difference" variable allows for an analysis of the extent to which parameter values require adjustment. Meanwhile, the second data structure consists of a dictionary featuring "Parameter_Name : Parameter_Value" pairs. These specific parameter values have been adjusted to align with building code specifications and will subsequently serve the purpose of refining the building model.

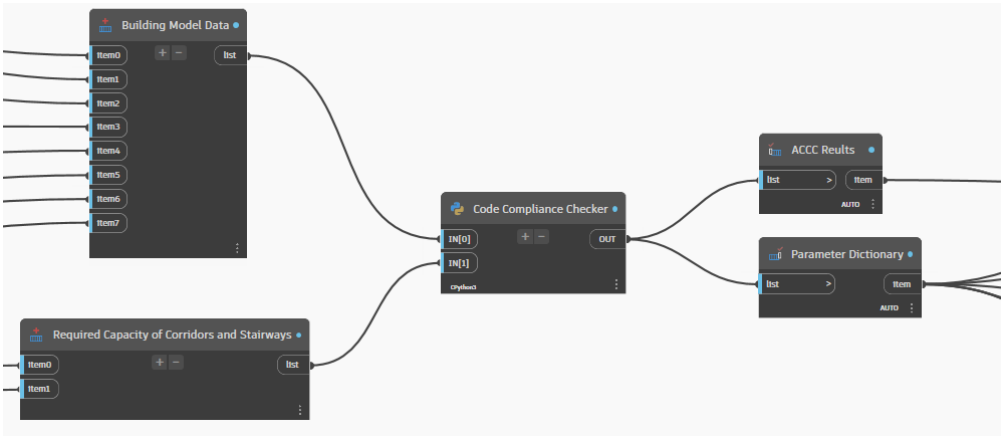


Figure 4.14: Dynamo script for checking code compliance and adjusting violation-related parameter values

The parameter dictionary is used to access all the adjusted parameter values. These values are then used to update parameters not meeting the building code requirements. As previously discussed, there are two types of parameters - family parameters and global parameters. Dynamo provides the ability to modify both types of parameters.

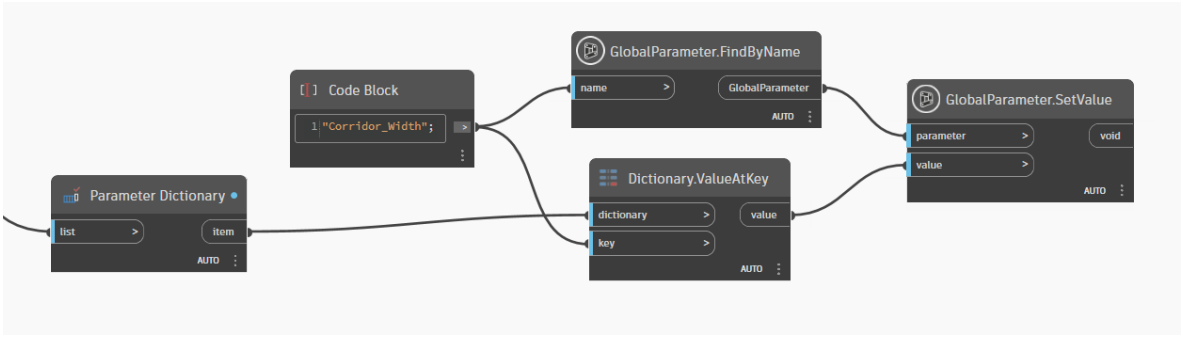


Figure 4.15: Adjusting the global parameter of corridor width using the parameter dictionary

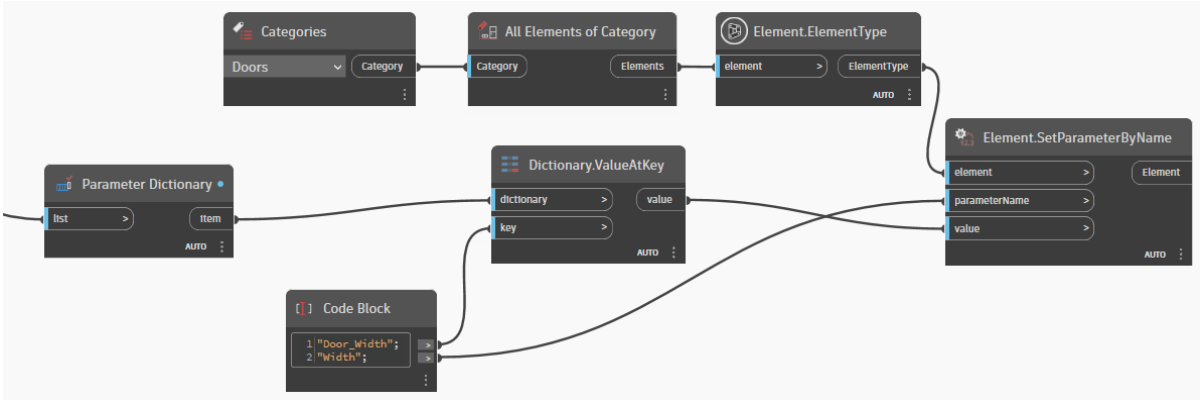


Figure 4.16: Adjusting the family parameter of door width using the parameter dictionary

When adjusting violation-related parameters, it is crucial to consider the specific type of parameter being adjusted. For instance, if a parameter is labeled as "length" in Revit, assigning it an integer value will not result in any adjustments. In such cases, in order to adjust the parameter, it is necessary to assign a value of type double or float to it.

Therefore, it is important to consider the type of each parameter before making any adjustments.

4.4 Optimization

The proposed optimization framework is designed to minimize the distance occupants need to travel to reach the building exit, while also taking into account the layout constraints of the building. This means that only those parameters that impact the egress distance will be taken into consideration during the optimization process, while other parameters related to code compliance will not be included in the analysis. The optimization of egress distance will guarantee that the building model not only meets the code requirements but is also optimized for the safety of its occupants.

4.4.1 Identification of Potential Parameters

The objective function maximizes the difference between the available and maximum egress distances (Section 3.2). The key parameters that will have the most effect on the egress distances can now be identified. The width of the corridor is an important parameter that can affect the egress distance of each room. Changing the width of the main or side corridors can increase the egress distance of some rooms while decreasing it for others. Therefore, it is essential to calculate the optimum value of this parameter to maximize the objective function. Another crucial factor affecting the egress distance is the location of doors. Doors should be positioned to allow occupants to exit the room easily, reducing the overall egress distance. Introducing parameters such as the distance of a door from its adjacent wall can help change the location of doors effectively.

When making adjustments to potential parameters for building layout optimization, it is important to establish upper and lower limits. The move limits of parameters are defined for the generation of design space. The lower limit is based on the required minimum values of IBC, while the building layout consideration of education occupancy type determines the upper limit. These move limits are set according to the requirements of IBC to ensure that the building model remains code-compliant during the optimization process.

4.4.2 Determining Available Egress Distance

According to Chapter 10 of the IBC, available egress distance is defined as the shortest and unobstructed route from any point in a building story to the closest exit in an emergency. This distance represents the actual path that an occupant would take to reach the nearest exit as quickly as possible, ensuring their safety. In Dynamo, calculating the available egress distance requires three key components: a complete floor layout, origin points (Room Doors), and destination points (Exit Doors). Considering these factors, it is possible to determine the shortest path from any point of the building to the nearest exit.

To import the floor layout into the Dynamo script, the room elements can be utilized to outline the layout boundaries by using the "Room.FinishBoundary" node. From these room boundaries, a surface is created; however, as the room elements are closed polylines, only closed spaces are generated without door openings. It is crucial to include door openings in the floor layout to calculate available egress distance. The door elements are imported into the Dynamo script to incorporate door thresholds in the room outlines. Each door's location is denoted by a point using the "Element.GetLocation" node. Once the location of each door is determined, it should be translated to the width and thickness of the door. Consequently, a surface is formed for each door dimension, which is then merged with the surface of the rooms to create one complete polysurface of the floor layout. The final floor layout is derived from this polysurface using the "Surface.PerimeterCurves" node.

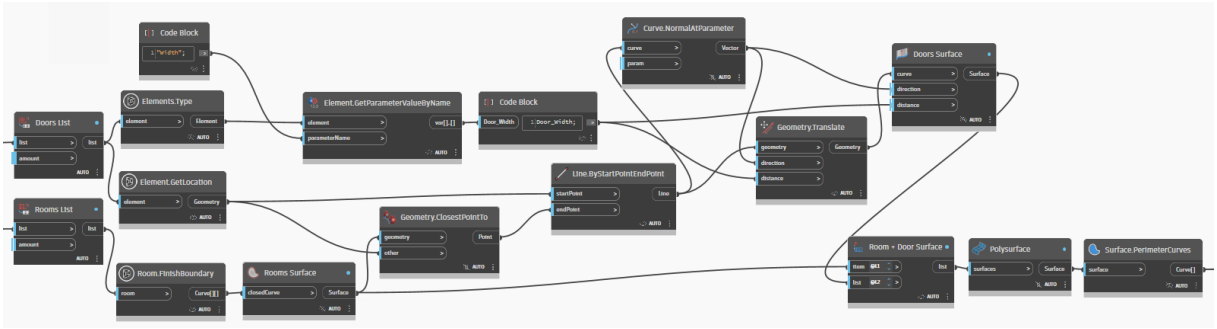


Figure 4.17: Dynamo script for creating floor layouts.

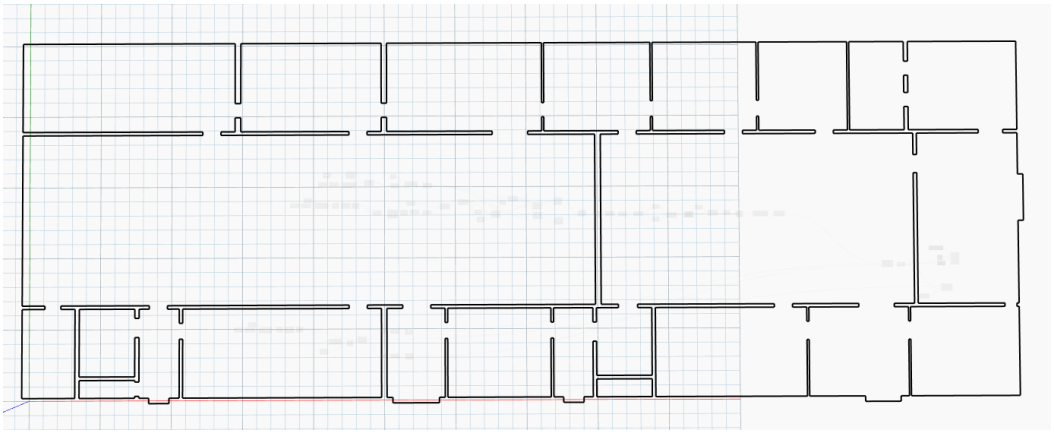


Figure 4.18: Generation of building floor layout in dynamo

After creating the floor layout, the origin and destination points are identified. Each door and stairway have been assigned a parameter called "Exit_Type". If it is a destination point, the parameter is labeled "Floor Exit," and if it's an origin point, the parameter is labeled "Room Exit". Using this semantic data, Dynamo filters the points of origin and destination and determines their locations with the "Element.GetLocation" node.

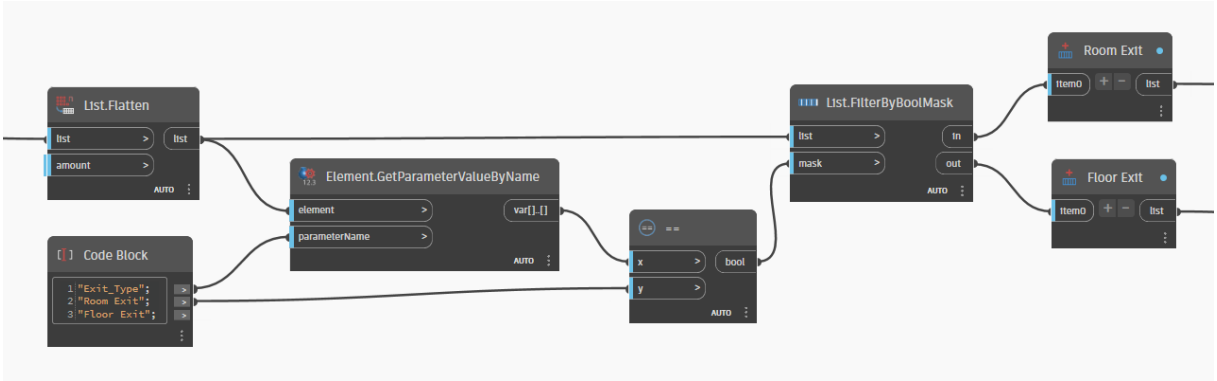


Figure 4.19: Dynamo script for identifying the points of origin and destination

A custom "Graphical" plugin is used to find the shortest distance. This plugin includes a node, "VisibilityGraph.ShortestPath," that requires three inputs: the floor layout, points of origin, and points of destination (Ortega, 2018). It then calculates the distance from each point of origin to each destination point. By comparing these distances, the shortest path can be determined. For instance, when a floor layout has three exits, the distance from the room exit to all three exits is calculated and compared, and the shortest distance among them is selected (Figure 3.3).

4.4.3 Achieving Optimized Parameter Configuration

After creating the dynamo script to calculate the available egress distance, it can be further developed for optimization. Real-coded genetic algorithm is used to identify the optimized parameter configuration. The following functions are necessary to execute

GA (Appendix A2). The first function is “initialize_population,” which generates a list of individuals with parameters assigned random numbers from the specified parameter ranges. Another function named “fitness_scores” is used to calculate the fitness score of each individual in the population, playing a crucial role in the selection process.

To select two individuals with higher fitness scores from the entire population, another function named “select_individual” has been created. These two individuals are referred to as “Parent1” and “Parent2.” After identifying the parents, the “crossover” function is used to create offspring, where each offspring contains mixed genes/parameter values from both parents. Following the generation of offspring, the “mutate” function introduces variability in the offspring through the mutation process. It prevents the algorithm from local optima or premature convergence. This process continues until the specified number of generations is reached, at which point the best individual in the final population is identified.

The Revit API enables users to modify the document (opened project) by employing the concept of a transaction. A transaction consists of a series of steps, such as creating, deleting, or modifying elements, ensuring that all changes are executed as a single step, which can then be saved or discarded if not needed. To calculate the shortest path and make changes to the project, necessary Revit libraries and a custom dynamo package of “Graphical” are imported into the Python node (Ortega, 2018). The first step is to access the opened Revit document from the Python node to extract all global parameters from it. Two functions are defined for the interaction of the Python node with the Revit API. The first function, "set_global_parameters," applies the given parameter values to the corresponding global parameters. The second function, "Shortest_Path," calculates the available egress distance of the current design solution.

After a thorough iterative process, the genetic algorithm explores all generations. Once the specified number of generations has evolved, the best individual in the final population is identified. Its parametric configuration is saved as the optimized parameter configuration and assigned to the global parameters of the opened Revit document, resulting in an optimized BIM model for improved occupant safety.

5 Case Study

To verify the validity of the proposed framework, the physics department building (N6) of the Technical University of Munich (TUM) is considered. The initial step involves creating a parametric model of the building and integrating all the essential parameters into the BIM model.

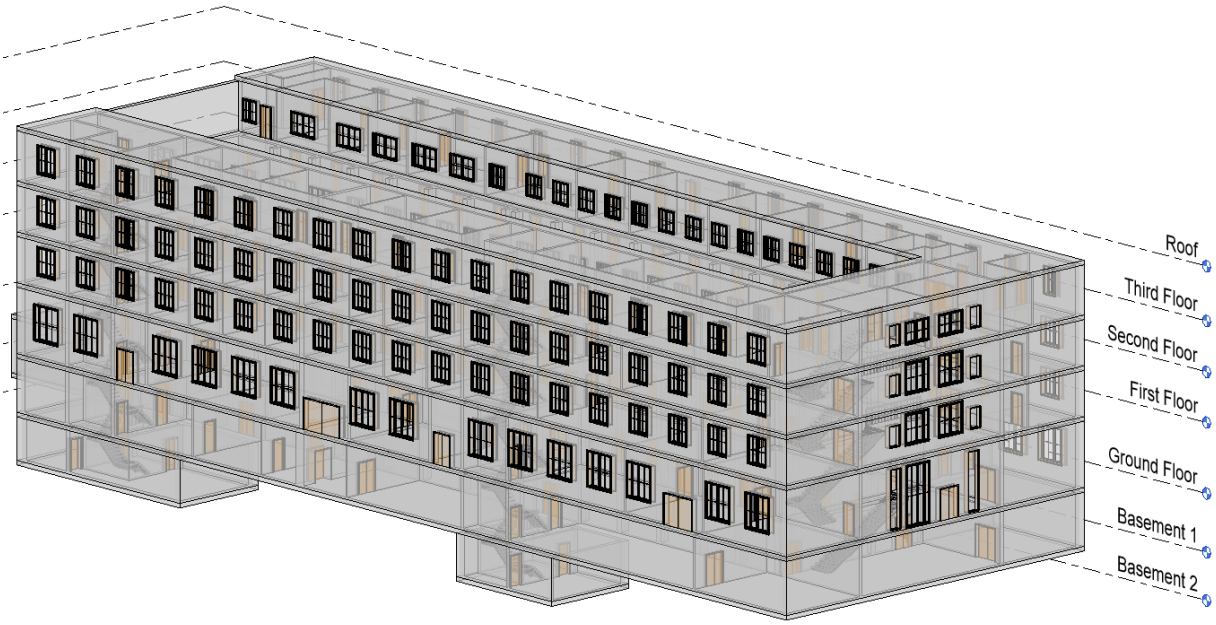


Figure 5.1: 3D-View of physics department building (N6) of TUM

Following the building modeling, sensitivity analysis will be performed to assess the impact of the parameters on the available egress distance. Moreover, the code compliance framework will be applied to the building model to check the code compliance with the IBC and automatically adjust parameters related to code violations to achieve code compliance. Subsequently, optimization will be conducted to ascertain the best parameter configuration, providing maximum safety for occupants by minimizing travel distances. Additionally, the impact of adding an extra exit to the building is examined, and the optimal location for the additional door is determined. The building model consists of 6 floors in total; after the modeling of the building model, each floor's occupant load and available egress distance are calculated.

Table 5.1: Building information of the physics department (N6), TUM

Floor	No. of Rooms	Occupant Load	Max AED (m)	Σ AED (m)
Basement 2	4	5	7.99	21.96
Basement 1	22	251	25.87	336.03
Ground Floor	15	272	23.96	369.66
First Floor	32	192	35.86	566.08
Second Floor	32	186	36.76	622.80
Third Floor	29	167	40.20	853.54

5.1 Impact of Corridor Width

The width of the main corridor is a crucial factor that influences the available egress distance in any building. Therefore, evaluating this parameter's impact on egress distance is important. The change in corridor width may increase or decrease the egress distance depending on the building layout. The corridor width varies from 9 to 15 meters in 0.5-meter increments to observe its impact on the travel distance.

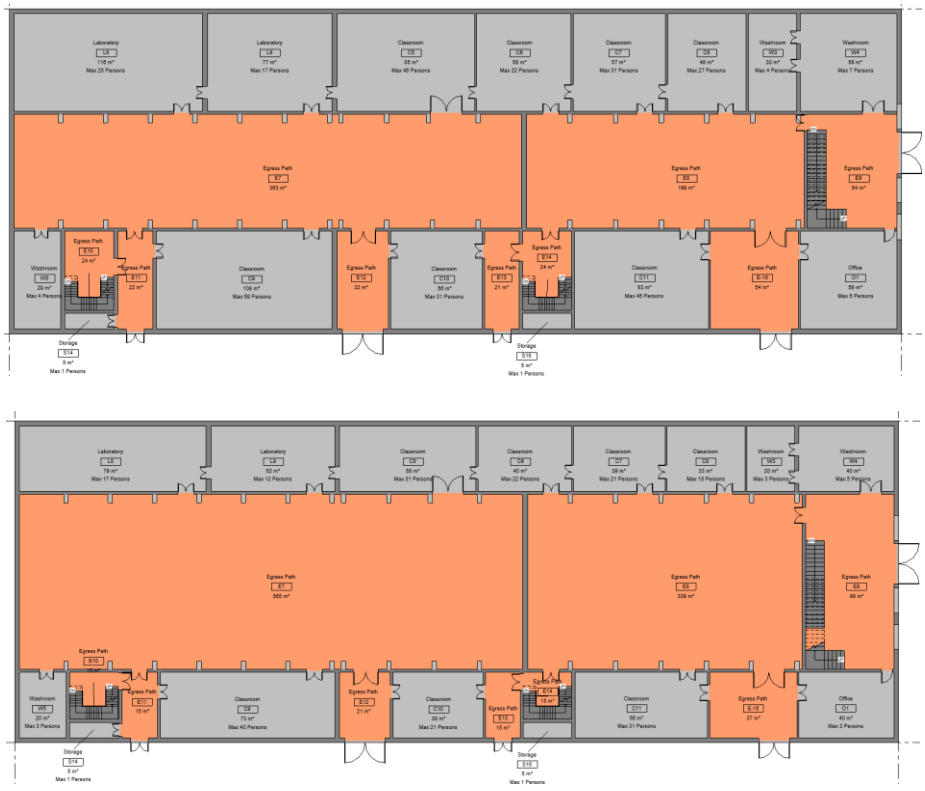


Figure 5.2: Minimum corridor width of 9m (Top), Maximum corridor Width of 15m (Bottom)

Changing the corridor width affects the travel distance of rooms to the nearest building exit. The egress distance of rooms on the north side of the building increases, while the distance for rooms on the south side of the building decreases. This is because 4 out of the 5 building exits are located in the southern part of the building. As a result, the sum of all travel distances (Σ AED) remains more or less the same, without significant changes. However, as the maximum available egress distance is calculated from the north side of the building, the maximum travel distance is increased with an increase in the corridor width.

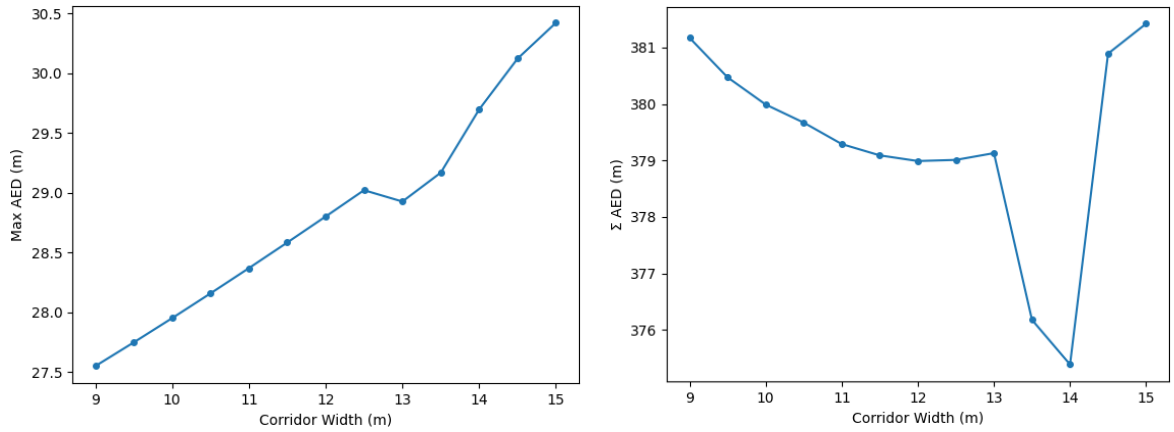


Figure 5.3: Effect of changing corridor width on the available egress distance

5.2 Impact of Door-Wall Clearance

Another significant factor affecting the available egress distance is the proximity of the door to its adjacent wall. The door-wall clearance is varied from 0.5 to 3 meters in 0.5-meter increments to observe its impact on the travel distance. It is observed that increasing the door-wall clearance results in an increase in both the sum of available egress distances (Σ AED) and the maximum available egress.

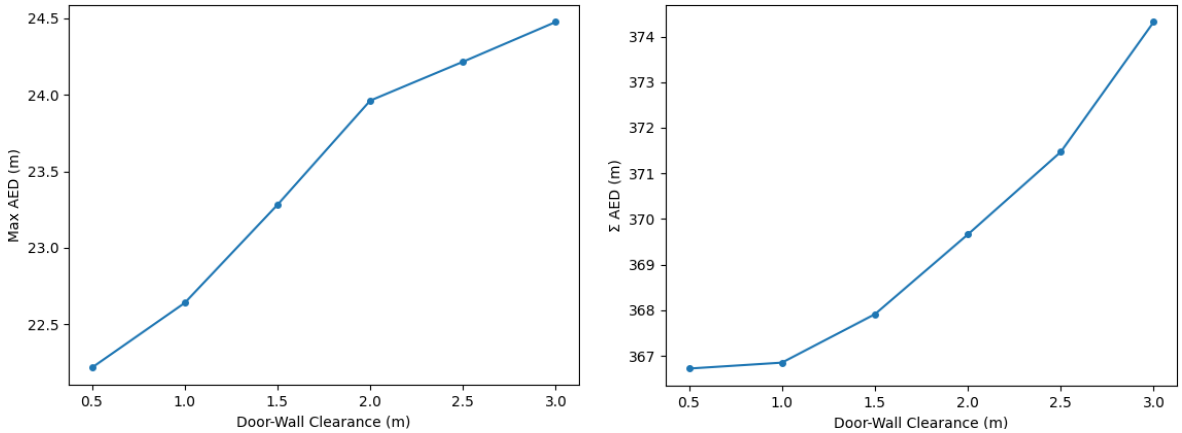


Figure 5.4: Effect of changing door-wall clearance on the available egress distances

5.3 Validating Code Compliance

The BIM model is used to gather building information required for code checking, which is used as input for a Python node, where 10 compliance checks are conducted (Section 4.3.3). Out of these, 7 compliance checks passed, while 3 checks failed.

Table 5.2: Results of automated compliance checking

S.No.	Checking Function	Result	Difference (cm)
1.	Minimum Corridor Width	**Fail**	20.8
2.	Required Capacity of Corridor	**Pass**	-
3.	Minimum Stairway Width	**Pass**	-
4.	Required Capacity of Stairway	**Pass**	-
5.	Minimum Ceiling Height	**Pass**	-
6.	Minimum Door Width	**Fail**	1.3
7.	Minimum Door Height	**Pass**	-
8.	Minimum Space between two Doors	**Fail**	3.9
9.	Minimum Number of Exits per Story	**Pass**	-
10.	Egress Distance	**Pass**	-

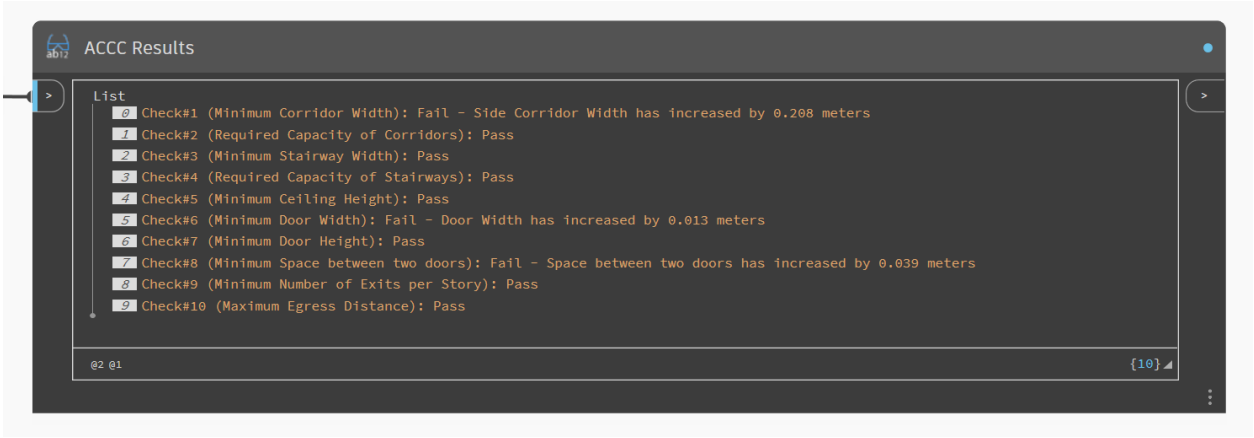
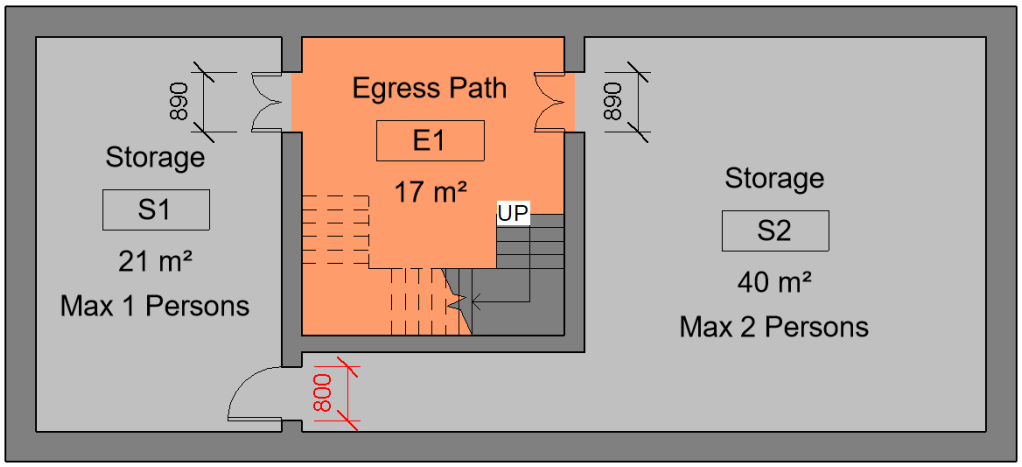


Figure 5.5: Results of automated compliance checking in Dynamo UI

The first adjustment involves an increase in the width of the side corridor. The building code specifies a minimum corridor width of 1.83 meters. However, the building model initially had a corridor width of only 1.62 meters. As a result, the first compliance check failed, and the width of the side corridor was automatically adjusted to ensure code compliance.

Figure 5.6: Adjusting the width of the side corridor according to the IBC standards

The second adjustment is being made to the door width in the basement of the building model. The door originally had a width of 800 mm, but the building code specifies a minimum door width of 813 mm.



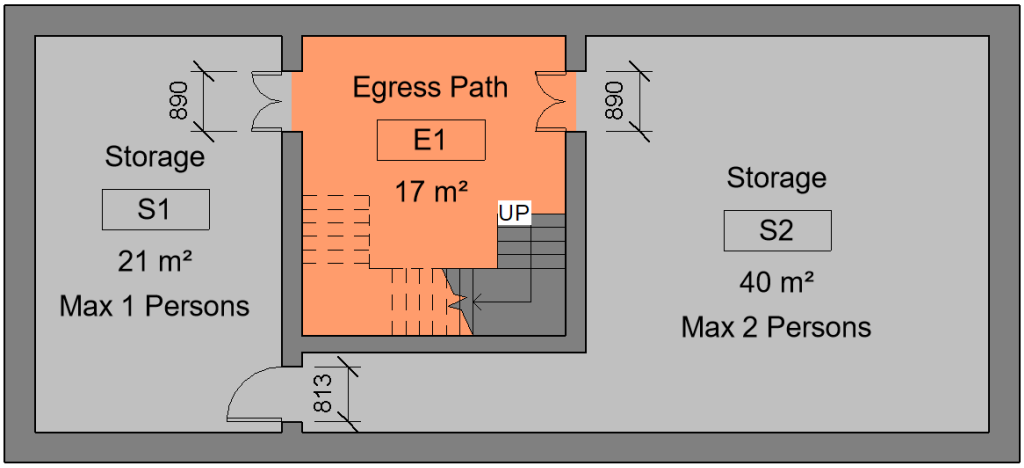


Figure 5.7: Adjusting the width of the door according to the IBC standards

The third adjustment involves the space between two adjacent doors. According to the IBC, a minimum space of 1.22 meters should be provided between adjacent doors. However, in the washroom on the ground floor, the space between two doors was only 1.18 meters.

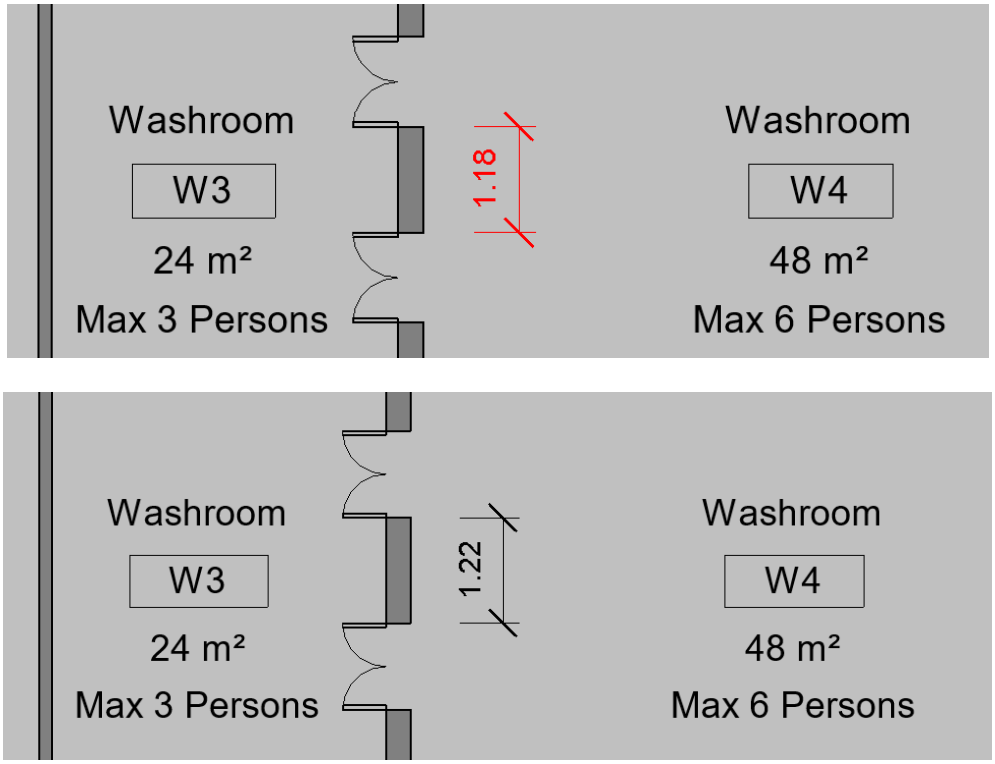


Figure 5.8: Adjusting the space between adjacent doors according to the IBC standards

With the help of the code compliance framework, the building model was checked for code compliance, and any violation-related parameters were automatically adjusted. This is one of the main advantages of parametric modeling. When non-compliance is identified, the building model can be automatically adjusted, saving time and money compared to manual adjustments to the BIM model.

5.4 Optimization

After ensuring that the building meets all code requirements, the building model is further analyzed to optimize the available egress distance. Only those parameters that impact the egress distance will be taken into consideration during the optimization process, while other parameters related to code compliance will not be included in the analysis. Ten parameters are selected for the optimization process, which affects the available egress distance. Custom ranges of each parameter have been defined to create a variation space; the lower limit is based on the required minimum values of IBC, while the constraints to preserve the initial design topology determine the upper limit. These move limits are set according to the requirements of IBC to ensure that the building model remains code-compliant during the optimization process.

Table 5.3: Design variables/parameters used for the optimization of available egress distance

S.No.	Design variable	Location	Lower Limit	Upper Limit
1	Main Corridor Width	North	6	7
2		South	6	7
3	Side Corridor Width	North	1	2
4		South	1	2
5	Door-Wall Clearance	Classroom	0.5	2
6		Laboratory	0.5	2
7		Meeting Room	0.5	2
8		Office	0.5	2
9		Storage	0.5	2
10		Washroom	0.5	2

```
def custom_range(start, end, step):
    while start <= end:
        yield start
        start += step

range_Main_Corridor_Width_Half_North = list(custom_range(6, 7, 0.5))
range_Main_Corridor_Width_Half_South = list(custom_range(6, 7, 0.5))
range_Side_Corridor_Width_North = list(custom_range(1, 2, 0.5))
range_Side_Corridor_Width_South = list(custom_range(1, 2, 0.5))
range_Door_Wall_Clearance_Classroom = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Laboratory = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Meeting_Room = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Office = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Storage = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Washroom = list(custom_range(0.5, 2, 0.5))
```

Figure 5.9: Creation of variation space of all parameters

With the help of GA, instead of checking each potential design solution individually, the two design solutions with higher fitness scores are selected from each generation, and new design solutions are created after the process of crossover and mutation (Section 3.2). During the optimization, 20 generations have been generated, with a population size of 10 individuals each. Once the maximum number of specified generations is analyzed, the fitness score of the last population is calculated, and the individual with the highest fitness score comprises the optimized parameter configuration, which is automatically assigned to the BIM model, resulting in an optimized BIM model.

```python
# Initialize Population
population = initialize_population()

for generation in range(GENERATIONS):
    fitness_scores = [fitness_function(individual) for individual in population]

    new_population = []
    for _ in range(POPULATION_SIZE // 2):
        parent1 = select_individual(population, fitness_scores)
        parent2 = select_individual(population, fitness_scores)

        child1 = crossover(parent1, parent2)
        child2 = crossover(parent1, parent2)

        child1 = mutate(child1)
        child2 = mutate(child2)

        new_population.extend([child1, child2])

    population = new_population

# Selecting Optimal Parameter Configuration
fitness_scores = [fitness_function(individual) for individual in population]
best_individual = population[fitness_scores.index(max(fitness_scores))]

# Applying Optimal Parameter Configuration to the BIM Model
set_global_parameters(doc, best_individual)

OUT = best_individual
```

Figure 5.10: Python code for using genetic algorithm to identify the best individual (Appendix A2)

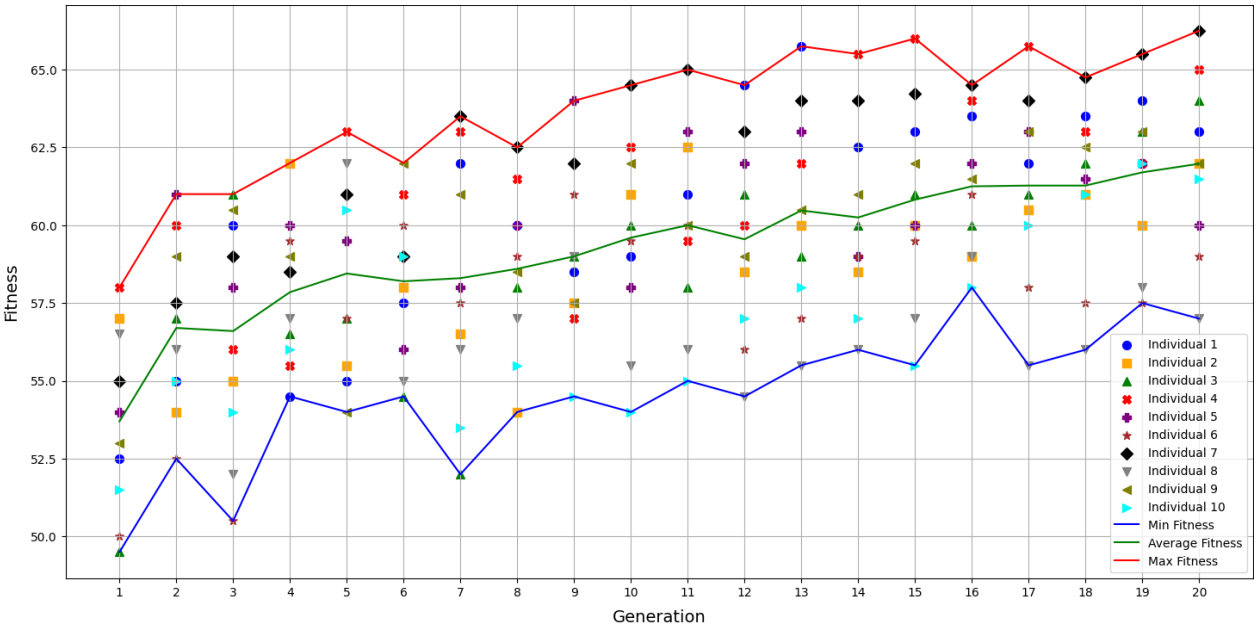


Figure 5.11: Fitness score evolution

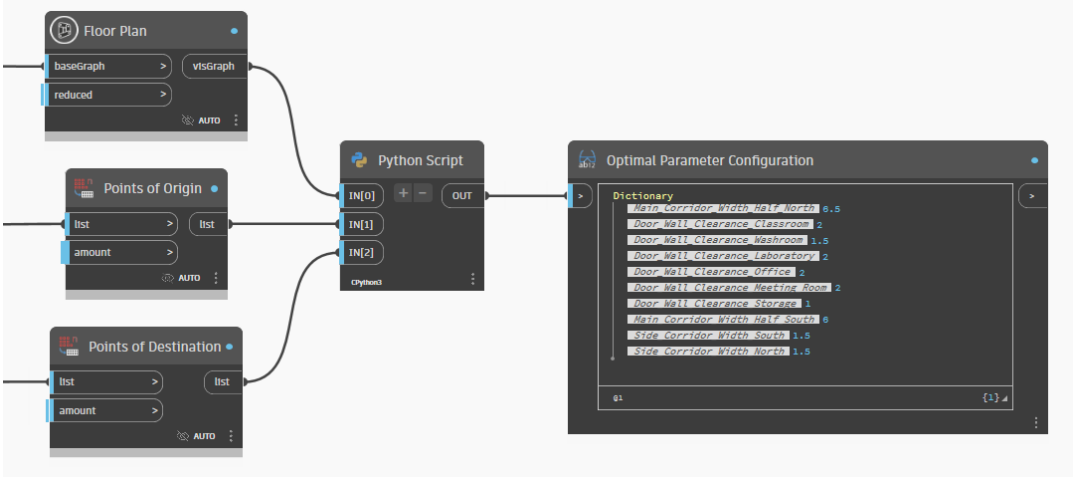


Figure 5.12: Optimized parameter configuration of the best individual (Individual 7)

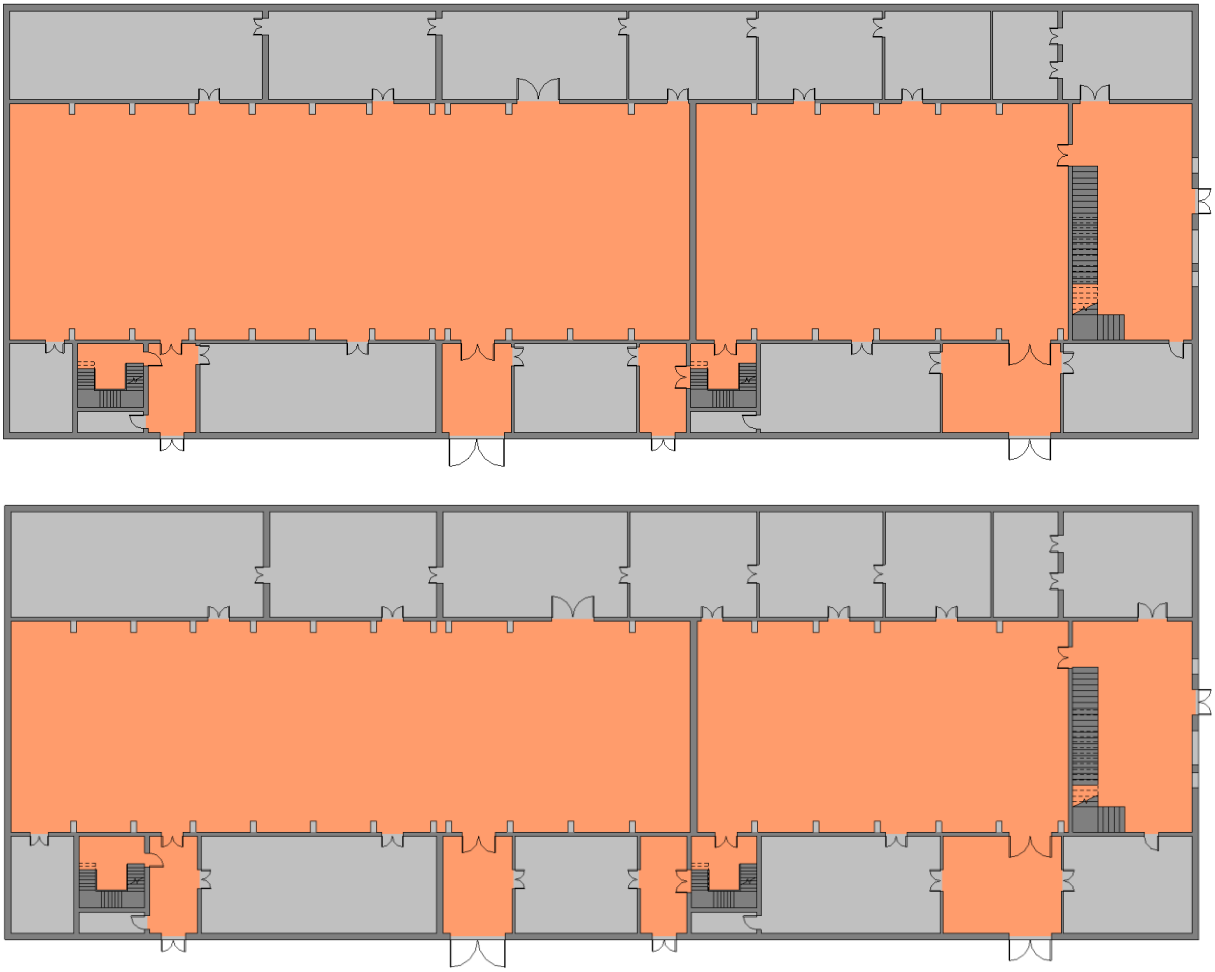


Figure 5.13: Comparison of floor layout. Initial design (Top), Generated floor plan using GA (Bottom)

After identifying the optimized parameter configuration, adding an extra building exit is another important factor to consider in enhancing occupant safety. The impact on the available egress distance by adding an additional exit is observed on each building floor, and the best location for providing an additional exit is determined. In the considered case study, building exits are already located on the south and east sides of the

building. Therefore, additional exits are provided on the north and west sides of the building, and their impact is considered.

Table 5.4: The impact of an extra exit on the Available Egress Distance (AED)

Building Story	Location	Max AED (m)	Σ AED (m)
Ground Floor	West	23.96	351.78
	North	5.23	290.59
First Floor	West	25.25	496.48
	North	14.65	465.38
Second Floor	West	26.92	546.81
	North	17.97	494.94
Third Floor	West	35.26	795.17
	North	19.38	704.46

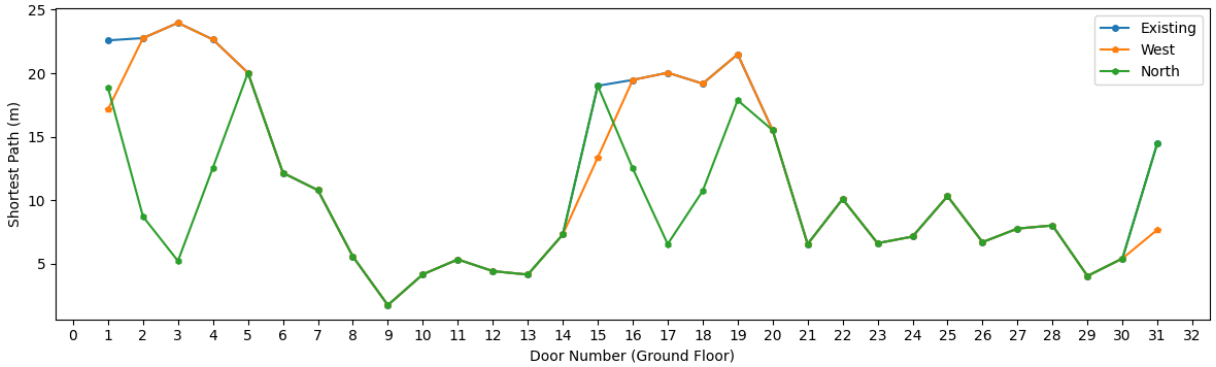


Figure 5.14: The impact of additional extra exit on the available egress distance

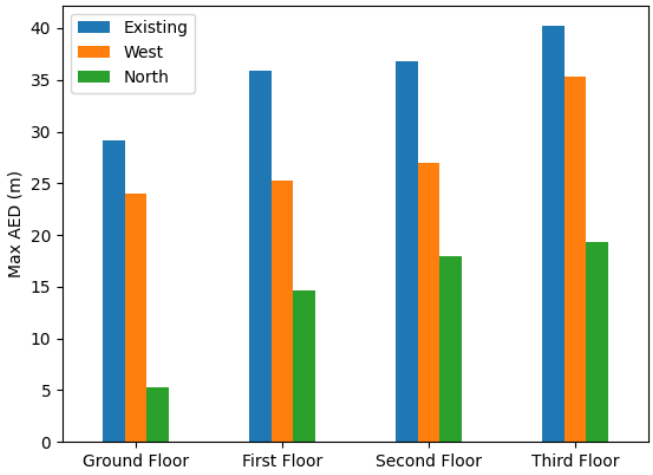


Figure 5.15: Comparison of available egress distances after the implementation of an additional exit

After adding an additional building exit, the shortest path of many points of origin to the building exit has changed. The points of origin on the east or south side of the building

did not experience any changes in their available egress distance; however, the points of origin located on the northern and eastern sides of the building, their shortest path, have been altered due to the provision of additional exits. Given the building layout, many rooms are positioned on the northern side instead of the eastern side. As a result, adding a door on the north side has impacted the travel distances for most of the points of origin. The same trend has been observed on the other floors of the building. Hence, it can be concluded that the optimal location for an additional building exit is on the northern side of the building. Similarly, this framework can also be applied to determine the optimal location for an extra building exit during the planning phase of any construction project, ensuring the safety of occupants is maximized.

6 Discussion

6.1 Contribution

This paper makes an essential contribution to the Building Information Modeling (BIM) field by introducing a BIM-based method for automatically checking code compliance and adjusting parameters related to violations. Additionally, it presents an optimization framework that can be utilized to minimize the available egress distance using parametric modeling to improve occupant safety.

State-of-the-art code compliance checkers have advanced to confirm adherence to regulatory standards and adjust building designs for compliance. However, a significant gap exists in fully automating and optimizing this process for complex and diverse design scenarios. As a result, architects and designers have to manually modify the building model to meet code requirements. This process takes a lot of time and effort and involves multiple iterations, often resulting in errors during the design and construction stages.

Many code compliance checkers use the Black-Box method, which utilizes coded programs to evaluate code compliance. This approach only presents the incoming and outgoing information, not the actual processing procedure. Consequently, users remain unaware of the processing steps and are only provided with the results, leading to a lack of transparency. Without transparency, verifying the processing steps' correctness and fairness becomes challenging. Moreover, users cannot modify or optimize the process to meet specific needs or preferences. Any changes or improvements require developer intervention, leading to potential time and cost implications.

This thesis introduces a framework that can automate not only the process of checking code compliance but also the adjustment of BIM models to ensure they are code-compatible and optimized for the safety of the occupants. The proposed framework has been successfully implemented in the physics department building (N6) at the Technical University of Munich (TUM), demonstrating its practicality and effectiveness. A detailed analysis comprising ten code compliance checks was carried out, resulting in the passage of 7 checks, while 3 checks did not meet the required standards. To rectify the violations, the parameters of the BIM model were automatically adjusted. These

adjustments encompassed widening side corridors, increasing door width, and ensuring the specified spacing between two doors adhered to the IBC regulations.

The impact of changing corridor width and door-wall clearance on the available egress distance was analyzed. The corridor width varied from 9 to 15 meters in 0.5-meter increments, and the door-wall clearance varied from 0.5 to 3 meters in 0.5-meter increments. The analysis revealed that changes in the corridor width did not significantly affect the sum of available egress distances (ΣAED) due to the specific layout of the considered case study. Since the rooms were distributed on the north and south sides of the building, and most of the exits were located on the southern side, altering the corridor width resulted in increased egress distance for northern rooms and decreased distance for southern rooms, leading to a negligible overall change in the sum. However, adjusting the door-wall clearance was observed to increase the sum of egress distances, indicating its significant impact.

GA was utilized to optimize the available egress distance. A total of ten parameters were adjusted within specified ranges. Each generation was examined to create a corresponding floor layout, origin, and destination points. Two individuals with higher fitness scores from each generation were chosen for the crossover and mutation process, and this process was repeated for subsequent generations. After analyzing 20 generations with a population size of 10 individuals, the individual in the last generation with the highest fitness score was identified. Its parametric configuration was saved as the optimized parameter configuration and automatically assigned to the BIM model, resulting in an optimized building design.

The proposed framework can be used in building projects' planning and design phases to automate code compliance checking and adjust the building model. This eliminates the need for manual modification of the BIM model to meet code requirements. Using this framework will save time and cost compared to the traditional method. Besides, it provides full transparency into the processing steps, allowing users to verify the correctness and fairness of the process. Users can also modify or optimize the framework to fulfill their specific needs without relying on the developers of a specific code compliance checker.

6.2 Limitations

The proposed framework has been successfully implemented in the case study, demonstrating its practicality and effectiveness. However, there are a few limitations that need to be considered. When calculating the available egress distance for exiting a building, only the room boundaries and door thresholds are considered for generating floor plans, and furniture within the building is not considered. However, in real-life situations, occupants will only use the available space. Therefore, the actual travel distance will be slightly longer than the calculated egress distance.

The point of origin is currently taken from the room's door. However, in reality, occupants begin evacuation from inside the room. In future work, instead of using the semantic information of doors to determine the point of origin, an approach that uses the centroid of room elements to calculate the point of origin from the middle of the room could be employed to more realistically compute egress distances.

The proposed framework requires the BIM model to be enriched with all the necessary parameters for complete functionality. This thesis considers ten checking rules, and their corresponding parameters are already included in the BIM model to make the proposed framework work. If additional checking rules need to be considered, the required parameters must be manually added to the BIM model. Depending on the building design's complexity, this process could be time-consuming.

There is a heavy reliance on custom parameters to meet the dynamo script requirements for calculating the available egress distance. This is because not all the necessary information is present in the BIM model by default. For example, Revit cannot automatically detect whether a created building door is the floor exit door or a room exit door. This determination is essential for finding the points of origin and the point of destination. Therefore, to determine the location of the starting point and the destination, a custom parameter named "Exit_Type" has been created for the corresponding doors and stairways. For the starting point, the value of this parameter is "Room Exit" and for the destination, it is "Floor Exit". Dynamo scripts then utilize this semantic information to distinguish between starting points and destinations. The user must ensure that the specified custom parameter contains the same information used in the dynamo script. For instance, if the user inputs "Floor exit" instead of "Floor Exit", the dynamo script will not be able to recognize the destination points.

When adjusting violation-related parameters, the proposed framework calculates the difference between the actual building data and the required values of IBC. Afterward, it automatically adjusts these parameters without providing further details about the building component. For example, users can view the adjusted parameter values, but they won't see specific details about the building component, such as the building floor, type of building component, or element ID of the adjusted building component.

6.3 Future Work

There are some improvements through which the proposed framework can be further enhanced. In the scope of this thesis, ten compliance-checking rules are being finalized, and they could be further expanded to allow the framework to check the building design in more detail. Additionally, the level of detail of generated floor plans could be improved such that in addition to considering walls and doors, it could also consider the furniture and other possible obstacles.

The location of stairways should be considered when calculating the egress distance as they serve as the destination points for each building story. To maximize the safety of occupants, stairways should be strategically placed so that occupants have to travel the shortest distance from any room to reach the nearest stairway. One way to achieve this is by creating a parameter to control the placement of the stairways and then identifying the optimized configuration for this parameter during the optimization phase.

To improve user experience, a custom plugin could be created to allow even those without experience using Dynamo or Python to utilize the proposed frameworks. Additionally, it could be designed so that when the plugin is run, the user would automatically receive a comprehensive list of information about adjusted building components and be able to view them directly in Revit with just a single click.

Bibliography

Alzara, M., Attia, Y., Mahfouz, S., & Yosri, A. (2023). Building a Genetic Algorithm-Based and BIM-Based 5D Time and Cost Optimization Model. *IEEE Access, 11*, 122502-122515. doi:https://doi.org/10.1109/ACCESS.2023.3317137

Anton, I., & Tănase, D. (2016). Parametric Modelling and Energy Analysis in Early Stages of Design. *Energy Procedia, 85*, 9–16. doi:https://doi.org/10.1016/j.egypro.2015.12.269

Berhe, H. W. (2012). Penalty function methods using matrix laboratory. *African Journal of Mathematics and Computer Science, 5*(13), 209-246. doi:http://dx.doi.org/10.5897/AJMCSR12.027

Bloch, T., & Sacks, R. (2018). Comparing machine learning and rule-based inferencing for semantic enrichment of BIM models. *Automation in Construction, 91*, 256-272. doi:https://doi.org/10.1016/j.autcon.2018.03.018

Borrmann, A., & Berkhahn, V. (2018). Principles of Geometric Modeling: Technology Foundations and Industry Practice. *Building Information Modeling*, 27-41. doi:http://dx.doi.org/10.1007/978-3-319-92862-3_2

Borrmann, A., König, M., Koch, C., & Beetz, J. (2018). *Building Information Modeling: Technology Foundations and Industry Practice.* doi:https://doi.org/10.1007/978-3-319-92862-3

Bouzidi, K. R., Fies, B., Zucker, F., & Zarli, A. (2012). Semantic Web Approach to Ease Regulation Compliance Checking in Construction Industry. *Future Internet, 4*(3), 830-851. doi:https://doi.org/10.3390/fi4030830

Bukowski, R. W., & Tubbs, J. S. (2016). Egress Concepts and Design Approaches. *Handbook of Fire Protection Engineering*. doi:https://doi.org/10.1007/978-1-4939-2565-0_56

Chopard, B., & Tomassini, M. (2018). Simulated Annealing. *An Introduction to Metaheuristics for Optimization*, 59-79. doi:https://doi.org/10.1007/978-3-319-93073-2

Decker, S., Mitra, P., & Melnik, S. (2000). Framework for the semantic Web: an RDF tutorial. *IEEE Internet Computing, 4*(6), 68-73. doi:https://doi.org/10.1109/4236.895018

Dimyadi, J., & Amor, R. (2013). Automated Building Code Compliance Checking - Where is it at? *19th International CIB World Building Congress*. doi:http://dx.doi.org/10.13140/2.1.4920.4161

Doukari, O., Greenwood, D., Rogage, K., & Kassem, M. (2022). Object-centred automated compliance checking: a novel, bottom-up approach. *Journal of Information Technology in Construction (ITcon), 27*, 335-362. doi:https://doi.org/10.36680/j.itcon.2022.017

Eastman, C. M., Lee, J.-m., Jeong, Y.-s., & Lee, J.-k. (2009). Automatic rule-based checking of building designs. *Automation in Construction, 18*(8), 1011-1033. doi:https://doi.org/10.1016/j.autcon.2009.07.002

Edmonds, A., Mourtis, T., & Boyle, M. (2022). Parametric Design—A Drive Towards a Sustainable Future. *Innovation in Construction. A Practical Guide to Transforming the Construction Industry*, 221–257. doi:https://doi.org/10.1007/978-3-030-95798-8_10

Fenves, S., Gaylord, E., & Goel, S. (1969). Decision Table Formulation of The 1969 Aisc Specification. *Univ-Dept Civ Eng-Structural Research*.

Filippo, A. (2021). Generative Design for project optimization. *The 27th International Conference on Distributed Multimedia Systems*. doi:http://dx.doi.org/10.18293/DMSVIVA21-014

Fischer, T., Biswas, K. D., Ham, J., & Naka, R. (2012). Beyond Codes and Pixels: Proceedings of the 17th International Conference on Computer-Aided Architectural Design Research. *Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, 537–546.

Gad, A. (2022). Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review. *Archives of Computational Methods in Engineering, 29*, 2531–2561. doi:https://doi.org/10.1007/s11831-021-09694-4

Greenwood, D., Lockley, S., Malsane, S., & Matthews, J. (2010). Automated compliance checking using building information models. *The Construction, Building and Real Estate Research Conference of the Royal Institution of*

Chartered Surveyors. doi:http://www.rics.org/site/download_feed.aspx?fileID=7953&fileExtension=PDF

Gross, M. D. (1996). Why can't CAD be more like Lego? CKB, a program for building construction kits. *Automation in Construction, 5*(4), 285-300.

Han, Z., Cao, N., Liu, G., & Yan, W. (2019). MOPSO for BIM: a multi-objective optimization tool using particle swarm optimization algorithm on a BIMbased visual programming platform. *18th International Conference, CAAD Futures 2019*, 39-51.

He, W., Shi, Y., & Kong, D. (2019). Construction of a 5D duration and cost optimisation model based on genetic algorithm and BIM. *Journal of Engineering, Design and Technology, 17*(5), 929-942. doi:https://doi.org/10.1108/JEDT-12-2018-0214

Hjelseth, E., & Nisbet, N. N. (2011). Capturing normative constraints by use of the semantic mark-up RASE methodology. *Computer Science*.

Holzer, D. (2015). BIM and Parametric Design in Academia and Practice:The Changing Context of Knowledge Acquisition andApplication in the Digital Age. *International Journal of Architectural Computing, 13*(1), 65-82. doi:https://doi.org/10.1260/1478-0771.13.1.65

IBC. (2018). *International Building Code (IBC).* International Code Council (ICC).

Ismail, A. S., Ali, K. N., & Iahad, N. (2017). A Review on BIM-based automated code compliance checking system. *5th International Conference on Research and Innovation in Information Systems (ICRIIS)*. doi:http://dx.doi.org/10.1109/ICRIIS.2017.8002486

Janssen, P. (2015). Parametric BIM Workflows. *20th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA)*, 437-446.

Juneja, M., & Nagar, S. K. (2016). Particle swarm optimization algorithm and its parameters: A review. *International Conference on Control, Computing, Communication and Materials (ICCCCM)*, 1-5. doi:https://doi.org/10.1109/ICCCCM.2016.7918233

Júnior, J., Santos, J., & Santos, M. (2023). Parametric modeling using the BIM methodology for the process of pathology identification in buildings. *Journal of Building Pathology and Rehabilitation, 8*(62). doi:https://doi.org/10.1007/s41024-023-00311-4

Kalkan, E., Okur, F. Y., & Altunışık, A. C. (2018). Applications and usability of parametric modeling. *Journal of Construction Engineering, Management & Innovation, 1*(3), 139-146. doi:https://doi.org/10.31462/jcemi.2018.03139146

Károlyfi, K. A., & Szép, J. (2023). A Parametric BIM Framework to Conceptual Structural Design for Assessing the Embodied Environmental Impact. *Sustainability, 15*(15). doi:https://doi.org/10.3390/su151511990

Kim, J., & Nguyen, T.-H. (2011). Building code compliance checking using BIM technology. *Winter Simulation Conference*. doi:http://dx.doi.org/10.1109/WSC.2011.6148035

Kincelova, K., Boton, C., Blanchet, P., & Dagenais, C. (2020). Fire Safety in Tall Timber Building: A BIM-Based Automated Code-Checking Approach. *Buildings, 10*(7). doi:https://doi.org/10.3390/buildings10070121

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680. doi:https://doi.org/10.1126/science.220.4598.671

Kodur, V. K., Venkatachari, S., & Naser, M. Z. (2020). Egress Parameters Influencing Emergency Evacuation in High-Rise Buildings. *Fire Technology, 56*, 2035–2057. doi:https://doi.org/10.1007/s10694-020-00965-3

Lee, J. (2010). Automated checking of building requirements on circulation over a range of design phases. *Georgia Institute of Technology*. doi:http://hdl.handle.net/1853/34802

Li, K., Gan, V., Li, M., Gao, M. Y., Tiong, R., & Yang, Y. (2024). Automated generative design and prefabrication of precast buildings using integrated BIM and graph convolutional neural network. *Developments in the Built Environment, 18*. doi:https://doi.org/10.1016/j.dibe.2024.100418

Luo, H., & Gong, P. (2015). A BIM-based Code Compliance Checking Process of Deep Foundation Construction Plans. *Journal of Intelligent & Robotic Systems, 79*(3), 549–576. doi:https://doi.org/10.1007/s10846-014-0120-z

Mirsadeghi, E., & Khodayifar, S. (2020). Hybridizing particle swarm optimization with simulated annealing and differential evolution. *Cluster Computing, 24*, 1135–1163. doi:https://doi.org/10.1007/s10586-020-03179-y

Nourkojouri, H., Dehnavi, A. N., Bahadori, S., & Tahsildoost, M. (2023). Early design stage evaluation of architectural factors in fire emergency evacuation of the buildings using Pix2Pix and explainable XGBoost model. *Journal of Building Performance Simulation, 16*(4), 415-433. doi:https://doi.org/10.1080/19401493.2022.2163422

Ortega, A. (2018). *Graphical Package for Dynamo*. Retrieved from https://github.com/alvpickmans/GraphicalDynamo

Park, S., & Lee, J. (2016). KBimCode Based Applications for the Representation, Definition and Evaluation of Building Permit Rules. *The International Association for Automation and Robotics in Construction*, 720–728. doi:https://doi.org/10.22260/ISARC2016/0087

Patlakas, P., Livingstone, A., Hairstans, R., & Neighbour, G. (2018). Automatic code compliance with multi-dimensional data fitting in a BIM context. *Advanced Engineering Informatics*, 216–231. doi:https://doi.org/10.1016/j.aei.2018.07.002

Peng, J., & Liu, X. (2023). Automated code compliance checking research based on BIM and knowledge graph. *Scientific Reports, 13*(1). doi:http://dx.doi.org/10.1038/s41598-023-34342-1

Preidel, C., & Borrmann, A. (2015). Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling. *Proceedings of the 32nd ISARC, Oulu, Finland*, 1–8. doi:https://doi.org/10.22260/ISARC2015/0033

Preidel, C., & Borrmann, A. (2018). BIM-Based Code Compliance Checking. In A. Borrmann, M. König, C. Koch, & J. Beetz, *Building Information Modeling* (pp. 367-381). Springer. doi:http://dx.doi.org/10.1007/978-3-319-92862-3_22

Sacks, R., Bloch, T., Katz, M., & Yosef, R. (2019). Automating Design Review with Artificial Intelligence and BIM: State of the Art and Research Framework. *Computing in Civil Engineering*, 353-360. doi:https://doi.org/10.1061/9780784482421.045

Salama, D., & Gohary, N. M. (2011). Semantic Modeling for Automated Compliance Checking. *International Workshop on Computing in Civil Engineering*. doi:http://dx.doi.org/10.1061/41182(416)79

Shen, T. S. (2006). Building Egress Analysis. *Journal of Fire Sciences, 24*. doi:https://doi.org/10.1177/0734904106052549

Shih, S.-Y., & Sher, W. (2014). Development of Building Information Modelling Enabled Code Checking Systems for Australia. *Proceedings of the 17th International Symposium on Advancement of Construction Management and Real Estate*, 1003–1010. doi:http://dx.doi.org/10.1007/978-3-642-35548-6_103

Sivanandam, S. N., & Deepa, S. N. (2008). Genetic Algorithm Optimization Problems. *Introduction to Genetic Algorithms*, 165–209. doi:https://doi.org/10.1007/978-3-540-73190-0_7

Solibri. (2024). Retrieved from A Nemetschek Company: https://www.solibri.com/

Stork, J., Eiben, A. E., & Beielstein, T. B. (2022). A new taxonomy of global optimization algorithms. *Natural Computing, 21*, 219–242. doi:https://doi.org/10.1007/s11047-020-09820-4

Sun, Q., & Turkan, Y. (2019). A BIM Based Simulation Framework for Fire Evacuation Planning. *Advances in Informatics and Computing in Civil and Construction Engineering*, 431–438. doi:https://doi.org/10.1007/978-3-030-00220-6_51

Tafraout, S., Bourahla, N., Bourahla, Y., & Mebarki , A. (2019). Automatic structural design of RC wall-slab buildings using a genetic algorithm with application in BIM environment. *Automation in Construction, 106*. doi:https://doi.org/10.1016/j.autcon.2019.102901

Usman, M., Mao, Y., Schaumann, D., Faloutsos, P., & Kapadia, M. (2020). From semantic-based rule checking to simulation-powered emergency egress analytics. *Proceedings of the 11th Annual Symposium on Simulation for Architecture and Urban Design*.

Villaschi, F., Carvalho, J., & Bragança, L. (2022). BIM-Based Method for the Verification of Building Code Compliance. *Applied System Innovation, 5*(64). doi:https://doi.org/10.3390/asi5040064

Wu, J., Dubey, R. K., Abualdenien, J., & Borrmann, A. (2022, Sep). Model Healing: Toward a Framework for Building Designs to Achieve Code Compliance. *Proceedings of the 14th European Conference on Product and Process Modelling, ECPPM*, 450–457. doi:https://doi.org/10.1201/9781003354222-58

Wu, J., Nousias, S., & Borrmann, A. (2023, June). Parametrization-based solution space exploration for Model Healing. *Proc. of the 30th Int. Conference on Intelligent Computing in Engineering (EG-ICE)*.

Yang, Q. Z., & Li, X. (2001). Representation and Execution of Building Codes for Automated Code Checking. *Computer Aided Architectural Design Futures*, 315-329. doi:http://dx.doi.org/10.1007/978-94-010-0868-6_24

Yang, S.-W., Moon, S.-W., Jang, H., Choo, S., & Kim, S.-A. (2022). Parametric Method and Building Information Modeling-Based Cost Estimation Model for Construction Cost Prediction in Architectural Planning. *Applied Sciences, 12*(19). doi:https://doi.org/10.3390/app12199553

Zarli, A., Yurchyshyna, A., Thanh, N., & Zucker, C. (2008). Towards an ontology-based approach for formalizing expert knowledge in the conformity-checking model in construction. In *eWork and eBusiness in Architecture, Engineering and Construction* (pp. 447-456). doi:http://dx.doi.org/10.1201/9780203883327.ch50

Zarzycki, A. (2012). Parametric BIM as a generative design tool. *100th ACSA Annual Meeting Proceedings, Digital Aptitudes*.

Zhang, J., & El-Gohary, N. M. (2016). Semantic-Based Logic Representation and Reasoning for Automated Regulatory Compliance Checking. *Journal of Computing in Civil Engineering, 31*(1). doi:https://doi.org/10.1061/(ASCE)CP.1943-5487.0000583

Zhang, J., & El-Gohary, N. M. (2017). Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. *Automation in Construction, 73*, 45-57. doi:https://doi.org/10.1016/j.autcon.2016.08.027

Zhang, Y. (2010). Genetic Algorithms for Bridge Maintenance Scheduling. *Technical University of Munich*.

Zhao, L., Zhang, W., & Wang, W. (2019). Construction Cost Prediction Based on Genetic Algorithm and BIM. *International Journal of Pattern Recognition and Artificial Intelligence, 34*(7). doi:https://doi.org/10.1142/S0218001420590260

Zhong, B., He, W., Huang, Z., Love, P., Tang, J., & Luo, H. (2020). A building regulation question answering system: A deep learning methodology. *Advanced Engineering Informatics, 46*. doi:https://doi.org/10.1016/j.aei.2020.101195

Zhou, Y., Wang, Y., Li, C., Ding, L., & Wang, C. (2022). Automatic generative design and optimization of hospital building layouts in consideration of public health emergency. *Engineering, Construction and Architectural Management, 31*(4). doi:https://doi.org/10.1108/ECAM-08-2022-0757

Appendix A1: Prototype for Automated Compliance Checking

```
import sys
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this python node are stored as a list in the IN variable
dataEnteringNode = IN

# Building_Model_Data:
Corridor_Width = IN[0][0]
Stairways_Width = IN[0][1]
Ceiling_Height = IN[0][2]
Door_Width = IN[0][3]
Door_Height = IN[0][4]
Space_between_Two_Doors = IN[0][5]
Number_of_Exits_per_Story = IN[0][6]
Available_Egress_Distance = IN[0][7]

# International_Building_Code_Requirements
Minimum_Corridor_Width = 1.828
Minimum_Corridor_Width_Based_on_OL = IN[1][0]    #OL = Occupant Load
Minimum_Stairways_Width = 1.118
Minimum_Stairways_Width_Based_on_OL = IN[1][1]
Minimum_Ceiling_Height = 2.286
Minimum_Door_Width = 0.813
Minimum_Door_Height = 2.032
Minimum_Space_between_Two_Doors = 1.219
Minimum_Number_of_Exits_per_Story = 2
Maximum_Egress_Distance = 76.2

# Check 01: Minimum Corridor Width
def Check_Minimum_Corridor_Width(Minimum_Corridor_Width, Corridor_Width):
    Result = 'Check#1 (Minimum Corridor Width): Pass'
    for i in range(len(Corridor_Width)):
        if Minimum_Corridor_Width > Corridor_Width[i]:
            difference = round(Minimum_Corridor_Width - Corridor_Width[i], 3)
            Corridor_Width[i] = Minimum_Corridor_Width
            Result = f'Check#1 (Minimum Corridor Width): Fail - Side Corridor Width has increased by {difference} meters'
    return Result, Corridor_Width

# Check 02: Required Capacity of Corridors
def Check_Required_Capacity_of_Corridors(Minimum_Corridor_Width_Based_on_OL, Corridor_Width):
    Result = 'Check#2 (Required Capacity of Corridors): Pass'
    for i in range(len(Minimum_Corridor_Width_Based_on_OL)):
        min_width = Minimum_Corridor_Width_Based_on_OL[i]
        for j in range(len(Corridor_Width)):
            if min_width > Corridor_Width[j]:
                difference = round(min_width - Corridor_Width[j], 3)
                Corridor_Width[j] = min_width
                Result = f'Check#2 (Required Capacity of Corridors):
```

```
                    Fail - Corridor Width has increased by {difference}
                    meters'
51                  return Result, Corridor_Width
52          return Result, Corridor_Width
53
54      # Check 03: Minimum Stairway Width
55      def Check_Minimum_Stairways_Width(Minimum_Stairways_Width, Stair
        ways_Width):
56          Result = 'Check#3 (Minimum Stairway Width): Pass'
57          for i in range(len(Stairways_Width)):
58              if Minimum_Stairways_Width > Stairways_Width[i]:
59                  difference = round(Minimum_Stairways_Width - Stair
                    ways_Width[i], 3)
60                  Stairways_Width[i] = Minimum_Stairways_Width
61                  Result = f'Check#3 (Minimum Stairway Width): Fail -
                    Stairway Width has increased by {difference} meters'
62          return Result, Stairways_Width
63
64      # Check 04: Required Capacity of Stairways
65      def Check_Required_Capacity_of_Stairways(Minimum_Stair
        ways_Width_Based_on_OL, Stairways_Width):
66          Result = 'Check#4 (Required Capacity of Stairways): Pass'
67          for i in range(len(Minimum_Stairways_Width_Based_on_OL)):
68              min_width = Minimum_Stairways_Width_Based_on_OL[i]
69              for j in range(len(Stairways_Width)):
70                  if min_width > Stairways_Width[j]:
71                      difference = round(min_width - Stairways_Width[j],
                        3)
72                      Stairways_Width[j] = min_width
73                      Result = f'Check#4 (Required Capacity of Stairways):
                        Pass'
74                      return Result, Stairways_Width
75          return Result, Stairways_Width
76
77      # Check 05: Minimum Ceiling Height
78      def Check_Minimum_Ceiling_Height(Minimum_Ceiling_Height, Ceil
        ing_Height):
79          Result = 'Check#5 (Minimum Ceiling Height): Pass'
80          for i in range(len(Ceiling_Height)):
81              if Minimum_Ceiling_Height > Ceiling_Height[i]:
82                  difference = round(Minimum_Ceiling_Height - Ceil
                    ing_Height[i], 3)
83                  Ceiling_Height[i] = Minimum_Ceiling_Height
84                  Result = f'Check#5 (Minimum Ceiling Height): Fail -
                    Ceiling Height has increased by {difference} meters'
85          return Result, Ceiling_Height
86
87      # Check 06: Minimum Door Width
88      def Check_Minimum_Door_Width(Minimum_Door_Width, Door_Width):
89          Result = 'Check#6 (Minimum Door Width): Pass'
90          for i in range(len(Door_Width)):
91              if Minimum_Door_Width > Door_Width[i]:
92                  difference = round(Minimum_Door_Width - Door_Width[i],
                    3)
93                  Door_Width[i] = Minimum_Door_Width
94                  Result = f'Check#6 (Minimum Door Width): Fail - Door
                    Width has increased by {difference} meters'
95          return Result, Door_Width
96
97      # Check 07: Minimum Door Height
98      def Check_Minimum_Door_Height(Minimum_Door_Height, Door_Height):
99          Result = 'Check#7 (Minimum Door Height): Pass'
100         for i in range(len(Door_Height)):
```

```
101             if Minimum_Door_Height > Door_Height[i]:
102                 difference = round(Minimum_Door_Height - Door_Height[i],
                    3)
103                 Door_Height[i] = Minimum_Door_Height
104                 Result = f'Check#7 (Minimum Door Height): Fail - Door
                    Height has increased by {difference} meters'
105         return Result, Door_Height
106
107     # Check 08: Minimum Space between Doors
108     def Check_Minimum_Space_between_Two_Doors(Minimum_Space_be
    tween_Two_Doors, Space_between_Two_Doors):
109         Result = 'Check#8 (Minimum Space between two doors): Pass'
110         if Minimum_Space_between_Two_Doors > Space_between_Two_Doors:
111             difference = round(Minimum_Space_between_Two_Doors -
                Space_between_Two_Doors, 3)
112             Space_between_Two_Doors = Minimum_Space_between_Two_Doors
113             Result = f'Check#8 (Minimum Space between two doors): Fail -
                Space between two doors has increased by {difference} me
                ters'
114         return Result, Space_between_Two_Doors
115
116     # Check 09: Minimum Number of Exits per Story
117     def Check_Minimum_Number_of_Exits_per_Story(Minimum_Number_of_Ex
    its_per_Story, Number_of_Exits_per_Story):
118         Result = 'Check#9 (Minimum Number of Exits per Story): Pass'
119         for i in range(len(Number_of_Exits_per_Story)):
120             if Minimum_Number_of_Exits_per_Story > Number_of_Ex
                its_per_Story[i]:
121                 difference = round(Minimum_Number_of_Exits_per_Story -
                    Number_of_Exits_per_Story[i], 3)
122                 Number_of_Exits_per_Story[i] = Minimum_Number_of_Ex
                    its_per_Story
123                 Result = f'Check#9 (Minimum Number of Exits per Story):
                    Fail - {difference} additional Exits are required'
124         return Result, Number_of_Exits_per_Story
125
126     # Check 10: Maximum Travel Distance
127     def Check_Maximum_Egress_Distance(Maximum_Egress_Distance, Availa
    ble_Egress_Distance):
128         Result = 'Check#10 (Maximum Egress Distance): Pass'
129         for i in range(len(Available_Egress_Distance)):
130             if Maximum_Egress_Distance < Available_Egress_Distance[i]:
131                 difference = round(Available_Egress_Distance[i] - Maxi
                    mum_Egress_Distance, 3)
132                 Result = f'Check#10 (Maximum Egress Distance): Fail -
                    Egress Distance exceeds by {difference} meters '
133         return Result, Available_Egress_Distance
134
135     # Applying Check Functions and Organizing the Results
136     Results_List = []
137     Building_Model_Data_List = []
138
139     check_functions = [
140         Check_Minimum_Corridor_Width, Check_Required_Capacity_of_Corri
             dors, Check_Minimum_Stairways_Width,
141         Check_Required_Capacity_of_Stairways, Check_Minimum_Ceil
             ing_Height, Check_Minimum_Door_Width,
142         Check_Minimum_Door_Height, Check_Minimum_Space_be
             tween_Two_Doors, Check_Minimum_Number_of_Exits_per_Story,
143         Check_Maximum_Egress_Distance
144     ]
145
146     parameters = [
```

```
147         (Minimum_Corridor_Width, Corridor_Width),
148         (Minimum_Corridor_Width_Based_on_OL, Corridor_Width),
149         (Minimum_Stairways_Width, Stairways_Width),
150         (Minimum_Stairways_Width_Based_on_OL, Stairways_Width),
151         (Minimum_Ceiling_Height, Ceiling_Height),
152         (Minimum_Door_Width, Door_Width),
153         (Minimum_Door_Height, Door_Height),
154         (Minimum_Space_between_Two_Doors, Space_between_Two_Doors),
155         (Minimum_Number_of_Exits_per_Story, Number_of_Exits_per_Story),
156         (Maximum_Egress_Distance, Available_Egress_Distance)
157     ]
158 
159     for check_function, params in zip(check_functions, parameters):
160         result, updated_data = check_function(*params)
161         Results_List.append(result)
162         if not Building_Model_Data_List or updated_data != Build
            ing_Model_Data_List[-1]:
163             Building_Model_Data_List.append(updated_data)
164 
165     # Creating Dictionary of Parameters
166     keys = [
167         "Corridor_Width", "Stairways_Width", "Ceiling_Height",
             "Door_Width", "Door_Height", "Space_between_Two_Doors",
             "Number_of_Exits_per_Story", "Available_Egress_Distance"
168     ]
169 
170     Building_Model_Data_Dict = {key: value for key, value in zip(keys,
    Building_Model_Data_List)}
171 
172     # Assigning output to the OUT variable
173     OUT = Results_List, Building_Model_Data_Dict
```

Appendix A2: Prototype for Optimization

```
import GraphicalDynamo as S
import clr
import time
import random

clr.AddReference('RevitServices')
clr.AddReference('RevitAPI')
clr.AddReference('RevitAPIUI')

from RevitServices.Persistence import DocumentManager
from Autodesk.Revit.DB import *

POPULATION_SIZE = 10
GENERATIONS = 20
MUTATION_RATE = 0.1
MED = 76.2

doc = DocumentManager.Instance.CurrentDBDocument
global_params =
FilteredElementCollector(doc).OfClass(GlobalParameter).ToElements()

def custom_range(start, end, step):
    while start <= end:
        yield start
        start += step

range_Main_Corridor_Width_Half_North = list(custom_range(6, 7, 0.5))
range_Main_Corridor_Width_Half_South = list(custom_range(6, 7, 0.5))
range_Side_Corridor_Width_North = list(custom_range(1, 2, 0.5))
range_Side_Corridor_Width_South = list(custom_range(1, 2, 0.5))
range_Door_Wall_Clearance_Classroom = list(custom_range(0.5, 2,
0.5))
range_Door_Wall_Clearance_Laboratory = list(custom_range(0.5, 2,
0.5))
range_Door_Wall_Clearance_Meeting_Room = list(custom_range(0.5, 2,
0.5))
range_Door_Wall_Clearance_Office = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Storage = list(custom_range(0.5, 2, 0.5))
range_Door_Wall_Clearance_Washroom = list(custom_range(0.5, 2, 0.5))

def set_global_parameters(doc, values):
    transaction = Transaction(doc, "Set Global Parameters Values")
    transaction.Start()
    try:
        for param in global_params:
            if param.Name in values:
                new_value = DoubleParameterValue(values[param.Name]
                * 3.28084)    # Converting meters to feet
                param.SetValue(new_value)
        transaction.Commit()
    except Exception as e:
        transaction.RollBack()
        print("Specified global parameter was not found")
        raise e

def ShortestPath(Visibility_Graph, Origins, Destinations):
```

```
52          results = []
53          for origin in Origins:
54              sub_list = []
55              for destination in Destinations:
56                  shortest_path =
                    S.Graphs.VisibilityGraph.ShortestPath(Visibility_Graph,
                    origin, destination)
57                  shortest_path_length = shortest_path["length"]
58                  sub_list.append(shortest_path_length)
59              results.append(sub_list)
60          return results
61
62      def initialize_population():
63          population = []
64          for _ in range(POPULATION_SIZE):
65              individual = {
66                  "Main_Corridor_Width_Half_North":
                    random.choice(range_Main_Corridor_Width_Half_North),
67                  "Main_Corridor_Width_Half_South":
                    random.choice(range_Main_Corridor_Width_Half_South),
68                  "Side_Corridor_Width_North":
                    random.choice(range_Side_Corridor_Width_North),
69                  "Side_Corridor_Width_South":
                    random.choice(range_Side_Corridor_Width_South),
70                  "Door_Wall_Clearance_Classroom":
                    random.choice(range_Door_Wall_Clearance_Classroom),
71                  "Door_Wall_Clearance_Laboratory":
                    random.choice(range_Door_Wall_Clearance_Laboratory),
72                  "Door_Wall_Clearance_Meeting_Room":
                    random.choice(range_Door_Wall_Clearance_Meeting_Room),
73                  "Door_Wall_Clearance_Office":
                    random.choice(range_Door_Wall_Clearance_Office),
74                  "Door_Wall_Clearance_Storage":
                    random.choice(range_Door_Wall_Clearance_Storage),
75                  "Door_Wall_Clearance_Washroom":
                    random.choice(range_Door_Wall_Clearance_Washroom),
76              }
77              population.append(individual)
78          return population
79
80      def fitness_function(individual):
81          set_global_parameters(doc, individual)
82          time.sleep(1)
83          Visibility_Graph = IN[0]
84          Origins = IN[1]
85          Destinations = IN[2]
86          shortest_paths = ShortestPath(Visibility_Graph, Origins,
            Destinations)
87          current_shortest_paths = [min(sublist) for sublist in
            shortest_paths]
88          AED = max(current_shortest_paths)
89          fitness = MED-AED
90          return fitness
91
92      def select_individual(population, fitness_scores):
93          total_fitness = sum(fitness_scores)
94          pick = random.uniform(0, total_fitness)
95          current = 0
96          for individual, score in zip(population, fitness_scores):
97              current += score
98              if current > pick:
99                  return individual
100
```

```
101     def crossover(parent1, parent2):
102         child = {}
103         for key in parent1.keys():
104             child[key] = parent1[key] if random.random() < 0.5 else
                parent2[key]
105         return child
106 
107     def mutate(individual):
108         for key in individual.keys():
109             if random.random() < MUTATION_RATE:
110                 if key == "Main_Corridor_Width_Half_North":
111                     individual[key] =
                        random.choice(range_Main_Corridor_Width_Half_North)
112                 elif key == "Main_Corridor_Width_Half_South":
113                     individual[key] =
                        random.choice(range_Main_Corridor_Width_Half_South)
114                 elif key == "Side_Corridor_Width_North":
115                     individual[key] =
                        random.choice(range_Side_Corridor_Width_North)
116                 elif key == "Side_Corridor_Width_South":
117                     individual[key] =
                        random.choice(range_Side_Corridor_Width_South)
118                 elif key == "Door_Wall_Clearance_Classroom":
119                     individual[key] =
                        random.choice(range_Door_Wall_Clearance_Classroom)
120                 elif key == "Door_Wall_Clearance_Laboratory":
121                     individual[key] =
                        random.choice(range_Door_Wall_Clearance_Laboratory)
122                 elif key == "Door_Wall_Clearance_Meeting_Room":
123                     individual[key] =
                       random.choice(range_Door_Wall_Clearance_Meeting_Room)
124                 elif key == "Door_Wall_Clearance_Office":
125                     individual[key] =
                        random.choice(range_Door_Wall_Clearance_Office)
126                 elif key == "Door_Wall_Clearance_Storage":
127                     individual[key] =
                        random.choice(range_Door_Wall_Clearance_Storage)
128                 elif key == "Door_Wall_Clearance_Washroom":
129                     individual[key] =
                        random.choice(range_Door_Wall_Clearance_Washroom)
130         return individual
131 
132     # Initialize Population
133     population = initialize_population()
134 
135     for generation in range(GENERATIONS):
136         fitness_scores = [fitness_function(individual) for individual in
            population]
137 
138         new_population = []
139         for _ in range(POPULATION_SIZE // 2):
140             parent1 = select_individual(population, fitness_scores)
141             parent2 = select_individual(population, fitness_scores)
142 
143             child1 = crossover(parent1, parent2)
144             child2 = crossover(parent1, parent2)
145 
146             child1 = mutate(child1)
147             child2 = mutate(child2)
148 
149             new_population.extend([child1, child2])
150 
151         population = new_population
```

```
152
153     # Selecting Optimized Parameter Configuration
154     fitness_scores = [fitness_function(individual) for individual in
        population]
155     best_individual = population[fitness_scores.index(max(fit
        ness_scores))]
156
157     # Applying Optimized Parameter Configuration to the open Revit docu
        ment
158     set_global_parameters(doc, best_individual)
159
160     OUT = best_individual
```

Appendix B: IBC Requirements

Table C.1: Occupancy classification of buildings (Section 302.1)

Occupancy Classification	Group
Assembly	A-1, A-2, A-3, A-4 and A,5
Business	B
Education	E
Factory and Industrial	F-1, F-2
High Hazard	H-1, H-2, H-3, H-4 and H-5
Institution	I-1, I-2, I-3 and I-4
Mercantile	M
Residential	R-1, R-2, R-3 and R-4
Storage	S-1, S-2
Utility and Miscellaneous	U

Table C.2: Minimum Corridor Width (Section 1020.2)

Occupancy	Minimum Width (inches)
Any facility not listed in this table	44
Access to and utilization of mechanical, plumbing or electrical systems or equipment	24
With an occupant load of less than 50	36
Within a dwelling unit	36
In Group E with a corridor having an occupant load of 100 or more	72
In corridors and areas serving stretcher traffic in ambulatory care facilities	72
Group I-2 in areas where required for bed movement	96

Table C.3: Occupant load factor (Section 1004.5)

Function of Space	Occupant Load Factor (ft^2 / Occupant)	Occupant Load Factor (m^2 / Occupant)
Classroom	20	1.858
Laboratory	50	4.645
Meeting Room	15	1.393
Office	2 - 4 *	2 - 4 *
Washroom	100	9.290
Storage	300	27.870

* When:

$$(Office\ Area < 15\ m^2) \rightarrow (OLF = 4)$$

$$(Office\ Area \geq 15\ m^2) \rightarrow (OLF = 2)$$

Table C.4: Minimum number of exits based on the occupant load (Section 1006.3.2)

Occupant Load per Story	Minimum Number of Exits
1 - 500	2
501 – 1000	3
More than 1000	4

Table C.5: Exit access travel distance (Section 1017.2)

Occupancy	Without Sprinkler System (feet)	With Sprinkler System (feet)
A, E, F-1, M, R, S-1	200	250
I-1	Not Permitted	250
B	200	300
F-2, S-2, U	300	400
H-1	Not Permitted	75
H-2	Not Permitted	100
H-3	Not Permitted	150
H-4	Not Permitted	175
H-5	Not Permitted	200
I-2, I-3	Not Permitted	200
I-4	150	200

Affirmation

Hereby I declare to have written the Master Thesis autonomously. Only the cited sources and means have been used. Verbally or semantically transferred intellectual property I distinguished as such.

Further I assure not to have handed in the Thesis for another examination.

München, 30. September 2024

Malik, Muhammad Ali