

Automated Identification and Representation of Design Constraints

Scientific work to obtain the degree

Bachelor of Science (B.Sc.)

at the TUM School of Engineering and Design
of the Technical University of Munich.

Supervised by Prof. Dr.-Ing. André Borrmann
Jiabin Wu
Lehrstuhl für Computergestützte Modellierung und Simulation

Submitted by Elvira Khromykh [REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]

Submitted on 03. Sep 2023

Acknowledgements

This thesis has been written during my studies at the Lehrstuhl für Computergestützte Modellierung und Simulation at the TUM School of Engineering and Design. This work would not have been possible without the supervision of Prof. Dr.-Ing. André Borrmann and Jiabin Wu. I especially want to thank Jiabin Wu for supporting me in all my tasks, from the topic selection to the writing of the content. Your scientific guidance and advice were essential for me and the success of the presented work.

Abstract

The building design process becomes increasingly complicated every year, but employing digitised tools enables designers to consider diverse requirements for a project. For an architect, considering the relationships between different objects in the building model is as essential as it gets. These relationships are defined as conceptual ideas or specific requirements for a building and can be formalised as design constraints. Currently, design constraints are manually set by the user within the Building Information Modelling (BIM) authoring tools with a limited number of constraint types. Besides identifying design constraints, the appropriate representation is needed because not all information can be intelligibly displayed. To address these issues, the thesis presents a semi-automated approach for identifying and representing design constraints in the building model. The identification process employs statistical analysis. Constraint representation is based on a graph approach. The approach provides an automated tool to formalise design constraints according to the extracted design knowledge in the design model. Afterwards, the graph-based representation is employed and allows one to select a design constraint and display an explicit representation of the constraint. The developed methodology is tested in a case study. The constraints identification and representation processes are tested for several types of geometric constraints. The proposed method increases the efficiency of the design process and guides better application and interpretation of design constraints with digitised modelling methods.

Zusammenfassung

Der Gebäudeentwurfsprozess wird von Jahr zu Jahr komplizierter, aber der Einsatz digitalisierter Tools ermöglicht es Entwerfer, vielfältige Anforderungen an ein Projekt zu berücksichtigen. Für einen Architekten ist es äußerst wichtig, die Beziehungen zwischen verschiedenen Objekten im Gebäudemodell zu berücksichtigen. Diese Beziehungen werden als konzeptionelle Ideen oder spezifische Anforderungen an ein Gebäude definiert und können als Entwurfsbeschränkungen formalisiert werden. Derzeit werden Entwurfsbeschränkungen manuell vom Benutzer in den BIM (Building Information Modeling) Software mit einer begrenzten Anzahl von Beschränkungstypen festgelegt. Neben der Identifizierung von Entwurfsbeschränkungen ist auch die entsprechende Darstellung erforderlich, da nicht alle Informationen verständlich dargestellt werden können. Um diese Probleme anzugehen, stellt die Arbeit einen halbautomatischen Ansatz zur Identifizierung und Darstellung von Entwurfsbeschränkungen im Gebäudemodell vor. Der Identifizierungsprozess nutzt statistische Analysen. Die Darstellung von Beschränkungen basiert auf einem Graph-Ansatz. Der Ansatz bietet ein automatisiertes Tool zur Formalisierung von Entwurfsbeschränkungen entsprechend dem extrahierten Designwissen im Entwurfsmodell. Anschließend wird die graphbasierte Darstellung verwendet und ermöglicht die Auswahl einer Entwurfsbeschränkung und die Anzeige einer expliziten Darstellung der Beschränkung. Die entwickelte Methodik wird in einer Fallstudie getestet. Die Prozesse zur Identifizierung und Darstellung von Beschränkungen werden für verschiedene Arten geometrischer Beschränkungen getestet. Die vorgeschlagene Methode erhöht die Effizienz des Entwurfsprozesses und ermöglicht eine bessere Anwendung und Interpretation von Entwurfsbeschränkungen mit digitalisierten Modellierungsmethoden.

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Problems and goals	3
2	State of the art	6
2.1	Constraint classification	6
2.2	Constraints from architectural perspective	8
2.3	Analysis of constraints	10
2.4	Representation of constraints	12
3	Methodology	16
3.1	Design interpretation	16
3.2	Design constraint analysis	19
3.3	Customised constraint validation	20
3.4	Constraint representation	21
4	Approach	22
4.1	Instruments	22
4.1.1	BIM authoring tool	22
4.1.2	Graph database	24
4.2	Development of a constraint identifier	25
4.2.1	Collection panel	25
4.2.2	Constraint Panel	32
4.2.3	Graph Panel	37
5	Case study	41
5.1	Experimental setup	41
5.2	Constraint identification and representation	42
5.3	Results and Interpretation	43
6	Discussion	56
7	Conclusion & outlook	58
A	Design constraints implementation and representation	59
	Bibliography	81

Acronyms

3D	Three Dimensional
AI	Artificial Intelligence
API	Application Programming Interface
BIM	Building Information Modelling
BRep	Boundary Representation
CSV	Comma-Separated Values
DBMS	Database Management System
GRS	Graph Rewriting Systems
IFC	Industry Foundation Classes
LINQ	Language Integrated Query
ML	Machine Learning
NLP	Natural Language Processing
NoSQL	Non-relational SQL
PBG	Parametric Building Graph
QL4BIM	Query Language for Building Information Model
RAD	Rapid Application Development
SQL	Structured Query Language

Chapter 1

Introduction

1.1 Motivation

The building design process is complex and involves project participants being active during all design phases. Design must fulfil various requirements, regulations, and building codes and satisfy project goals and tasks. This is processed iteratively through all project phases, where each iteration makes improvements and corrections in the design (ZAHEDI et al., 2022). Nowadays, building models become increasingly complex, thus necessitating efficient methods and tools to verify the current design phase from the point of view of the requirements. In addition, the process of inspecting a project significantly influences the quality of the design (ABUALDENIEN and BORRMANN, 2021, LUO et al., 2022).

While designing the building, an architect or a project manager needs to have an overview of and control of relationships among objects in a building. The specific rules (such as the ones coming from governmental requirements or architectural concepts) that design must satisfy are called design constraints. Design constraints can describe not only regulations but also design ideas. Understanding these relationships is a complex problem. One crucial design constraint type is geometrical constraints, which define the component orientation and location information, including mutual positions.

Commonly, such constraints are manually set by designers with Building Information Modelling **BIM** authoring tools (NIEMEIJER et al., 2010). Sometimes, a designer needs to have an overview of the design constraints for the model without applying them. It allows management constraints or guides new participants into the design code, resulting in a list of project constraints or a project guide. Therefore, an automated identification kernel and representation of design constraints can save time and improve design accuracy. Correctly identifying and representing the constraints helps to avoid potential conflicts or design flaws and mitigate errors that might be unnoticed during the manual setup of constraints. This ensures greater design precision and contributes to a more coherent architectural outcome.

Currently, there are different methods to identify and represent the design constraints in the building model. A significant portion of research aims at identifying constraints using spatial query (BORRMANN and RANK, 2010, DAUM and BORRMANN, 2014, ISMAIL et al., 2017). Another instance is using a declarative spatial reasoning framework to generate geometrical constraints (BHATT et al., 2011). A more advanced identification approach invokes artificial intelligence (**AI**) techniques (SACKS et al., 2022). For example, ZAHEDI et al. (2022) proposed using the building code to set regulatory constraints

on the properties of an object in a model (a wall). LUO et al. (2022) invokes deep learning to classify BIM objects and consider the geometrical and spatial information. The representation of constraints is an essential part of every approach. The most common representation is a graph representation. ABUALDENIEN and BORRMANN (2021) proposes a Parametric Building Graph (PBG) for capturing design constraints (patterns), which allows one to select one of the patterns and transfer to another model based on the Graph Rewriting System (GRS). VILGERTSHOFER and BORRMANN (2017) uses graph to represent parametric constraints in the model and GRS to modify model. The highlighted identification and representation approaches can solve complex challenges of design constraints within the building model. These methods have the potential to recognise non-standard design patterns for guiding the building design.

1.2 Problems and goals

Understanding design constraints is a complicated problem. This results from the abstract definition of constraint and the geometrical complexity of the building model. Below, three major problems relevant to this work are described.

Problem 1: Studies primarily focus on geometric constraint identification (G. LEE et al., 2006) but not on capturing specific design constraints such as design ideas and concepts. Another research direction concentrates on the problem of capturing design knowledge in the form of geometrical and semantic information (LANGENHAN et al., 2013, ABUALDENIEN and BORRMANN, 2021, ZAHEDI et al., 2022). A design constraint can be expressed as elements' dimensions, position, or semantic information (e.g. material). However, the available BIM authoring tools support only basic usage of predefined constraints such as equality constraints.

Problem 2: One notable area of research addresses the representation of constraints. One interesting technique is the topological approach, which includes graph-based representation. In ARORA et al. (2021), graphs formalise semantic spatial configurations for further consistency checking. KIRCHNER and HUHNT (2018) use the interval method in a constraint graph to enable user's modifications. SCHULTZ et al. (2017) generates a parametric sketch that contains all constraints and allows improvement of the existing model based on the declarative spatial reasoning system called CLP(QS)¹. Although such works are numerous, there is no universal approach to design constraints representation. The appropriate solution should fulfil a specific design task and represent a building model in an abstract way (i.e. abstract from irrelevant numerical details, see SCHULTZ et al. (2017)). It means the more detailed the geometry, the more complex the abstract constraints representation.

Problem 3: Other works are related to the complications of a database implementation. Organising data properly and choosing an appropriate database based on the design problems is essential. Few studies focused on the relational structured query language

¹<http://hcc.uni-bremen.de/spatial-reasoning/index.php/systems-1>

(SQL) databases, and others focused on the object-oriented or even object-relational database, which is more stable and efficient and supports object-oriented queries (ZHOU et al., 2020). The latest research concentrates on cloud-based solutions such as non-relational (NoSQL) databases that provide quick access to building model elements. However, ZHOU et al. (2020) noticed that no efficient algorithm can work with complex spatial data models to prevent missing data.

The solution presented in this thesis aims to fill the gap in automatically identifying design constraints in a given model and representing it in an efficient manner. Improved identification and representation allow designers to easily manage the project's constraints, guide new participants into the project regulations, and increase the overall accuracy of the design during all phases.

The main goals of this work:

1. Conduct a literature review of state-of-the-art research articles and industrial reports to investigate and classify the design constraints from an architectural perspective (e.g., from basic design knowledge, general user requirements, and building codes).
2. Develop a methodology that enables the graph representation of constraints in building design.
3. Create a prototype to support customised design constraint extraction and application.

Thesis structure

The thesis introduces a method and an automated tool to identify and represent design constraints. The methodology of constraint identification lies in the statistical analysis of geometrical data, whereas the graph database approach handles the representation part. The methods implemented are condensed in an automated solution (tool) based on a BIM authoring tool that can accurately identify and afterwards represent design constraints.

The rest of this thesis is organised as follows:

- Chapter 2 presents state-of-the-art methods and related works. This chapter highlights the different methods and approaches of constraint identification and representation.
- Chapter 3 explains the methodology to develop and evaluate algorithms to identify and represent constraints in the building model.
- Chapter 4 describes the workflow of the proposed approach in detail.
- Chapter 5 is dedicated to the experimental setup and simulation results.
- Chapter 6 discusses the simulation results and the limitations of the proposed approach.

- Chapter 7 summarises the findings and presents an outlook for future research.

Chapter 2

State of the art

The **BIM** paradigm was introduced into the industry a few decades ago (EASTMAN et al. (2008)). It proposes numerous advantages to designers, architects, and engineers. To name a few related to this thesis, **BIM** is a framework in which one works with the object-based parametric design system. It allows one to quickly verify the consistency of design intent (requirements) and discover design errors early in the planning and building process (BORRMANN, HYVÄRINEN, et al., 2009; MARCOS ALBA, 2017; TAKIM et al., 2013). Still, there are several open research questions. One is the performance problem in data extraction from the **BIM** model, e.g., geometrical data. Other issues include efficient data storage and analysis methods and tools (ZHOU et al., 2020).

Design constraints constitute relationships between building objects that may satisfy the design intentions. Constraints play a significant role in architectural design. A building's purposes and functions may include requirements for different types of spaces, such as offices, classrooms, laboratories, or residential units (EASTMAN et al. (2008)). The design must consider the specific needs of the occupants, such as accessibility, safety, and comfort. Local building regulations (e.g. zoning requirements, fire codes, and mobility) can influence the building design (EASTMAN et al., 2009). The design must balance the requirements with the practical needs of the building. Therefore, design constraints must be carefully checked and addressed during the design process to create a successful building project.

The following sect. 2.1 provides a comprehensive literature review about constraint classification. Constraints from the architectural perspectives presented in the sect. 2.2 and summarised constraints classification, which can be useful during the design process. The constraint analysis is described in the sect. 2.3 and includes constraint identification and evaluation. In the following sect. 2.4, constraint representation is described. Constraints can be represented differently, but the main focus of this thesis lies in the graph representation approach.

2.1 Constraint classification

Since constraints play a significant role in research, this section reaches an up-to-date classification of design constraints. Design constraints in the building model can be divided into several groups (fundamental types). BORRMANN, HYVÄRINEN, et al. (2009) distinguishes two fundamental constraints: consistency and requirement constraints. The former results from geometric or topological references in the **BIM** model. The latter represents regulations (norms and rules) and user requirements. KIRCHNER and HUHNT

(2018) proposed detailed types of requirements constraints such as architect and engineer requirements, client requirements, and administrative regulations (limited permission).

Other building constraints have an abstract meaning, including design ideas and semantic information. Designer's ideas (or patterns) are classified by BISKJAER et al. (2014) as "creativity constraints" (conceptual). Creativity constraints are defined as a designer's decisions during the design process. Creativity constraints can be used to manoeuvre and transform the design space. Spatial constraints can include semantic information of the building (BORRMANN, HYVÄRINEN, et al., 2009). By 'semantic' one means the non-geometric relationships (e.g., 'X is above Y,' 'X if made out of Y'). Spatial constraints indicate relations between building components and not only can be semantic ("below," "above", etc.) but also describe geometric relationships (min, max values of distance, etc.). In BORRMANN, HYVÄRINEN, et al. (2009), spatial constraints are divided into distance, directional, and topological constraints.

As mentioned above, spatial constraints also describe geometrical relationships between building components. These relationships are so-called geometric constraints. Geometric constraints are part of the spatial constraints and represent relationships between geometric objects. SCHULTZ et al. (2017) uses geometrical constraints to encode the high-level (semantic) relationships. Geometric constraints can be classified into parametric, non-parametric, and algebraic constraints. Parametric constraints require parameter values to create and edit the model's shape and define its dimensions. On the other hand, non-parametric constraints specify structural relationships between geometric objects and can be considered fixed-parameter parametric constraints. Algebraic constraints indicate relationships among parameters from different parametric constraints (TANG et al., 2022). Examples of algebraic constraints include topological, distance, dimensional, polynomial, numerical, placement, movement, parallel, perpendicular, point on the line, point on the segment, and many others (see SCHULTZ et al., 2017). KIRCHNER and HUHNT (2018) uses the interval constraints as a type of geometric constraint. Interval constraints enable a less restrictive way of modelling.

Constraints are additionally defined and classified in many BIM authoring tools. In Autodesk Revit 2023, there are three types of constraints: explicit (locked alignments and dimensions), looser (no locked alignments and sizes), and implied (such as a wall attached to a roof)¹. Autodesk AutoCAD 2024, on the other hand, indicates two types of constraints: geometric and dimensional². These constraints are typically user-defined and improve the design quality. The constraints enforce requirements when making changes (i.e. AutoCAD prohibits changes in the constrained geometry - distances and angles). Constraints enable the maintenance of design, include formulas within dimensional constraints, and change design by changing the value of a constraint. According to the official documentation, the design can be defined in three states: unconstrained (no restrictions in geometry), under-constrained (few constraints are applied), and fully constrained (all relevant constraints are applied). These states provide methods for working with constraints in the model

¹<https://help.autodesk.com/view/RVT/2023/ENU/?guid=GUID-91CBCCF3-66D1-496B-80B3-D893065D1A50>

²<https://help.autodesk.com/view/ACDLT/2024/ENU/?guid=GUID-899E008D-B422-4DF2-AC8D-1A4F5701ED4E>

depending on the design practices and the requirements of the discipline. The methods include working with underconstrained drawings or creating fully constrained drawings.

The Industry Foundation Classes Industry Foundation Classes (IFC) model constraints are presented as part of a non-standardized representation schema. Constraints ([IfcConstraints](#)) are divided into qualitative ([IfcObjective](#)) or quantitative ([IfcMetric](#))³ types. Constraints in the project may be subdivided into user-defined and system-defined constraints. [IfcConstraints](#) can be specified by the user-defined grades such as hard, soft, advisory, or undefined. The entity [IfcConstraintAggregationRelationship](#) enables one to relate to the individual constraints by applying the logical operators (BORRMANN, HYVÄRINEN, et al., 2009).

Another entity to define constraints in the IFC is [IfcResourceConstraintRelationship](#). This entity enables the constraints to be related to one or more resource-level objects and applies constraints to the properties. Therefore, property helps to control, identify and fulfil requirements⁴. BORRMANN, HYVÄRINEN, et al. (2009) also integrated spatial constraints into IFC. They proposed [IfcSpatialControl](#) (a subtype of [IfcControl](#)) with subtypes [IfcDirectionalControl](#), [IfcTopologicalControl](#) and [IfcProximityControl](#). New entities allow the control requirements and represent constraints in detail.

2.2 Constraints from architectural perspective

Every building project contains an array of design constraints. It is essential to manage constraints to prevent cost increases. On the one hand, the project should fulfil clients' requirements; on the other hand, too many constraints increase the occurrence of risk associated with a project delay, costs increase, and reduction in the quality of the project.

LAU and KONG (2022) studies the constraints and their impact on different projects. The paper classifies project constraints into categories:

- Economic constraints - the budget limit.
- Legal constraints - building regulations, fire safety, etc.
- Environmental constraints - air protection, traffic limit, noise control.
- Technical constraints - restrictive site area (storage space, transportation, equipment.)
- Social constraints - public concern, media pressure.

All these categories affect project performance, quality, and progress. LAU and KONG (2022) summarise their result in a scoring system and conclude that the legal constraints

³<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcConstraint.htm>

⁴<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcResourceConstraintRelationship.htm>

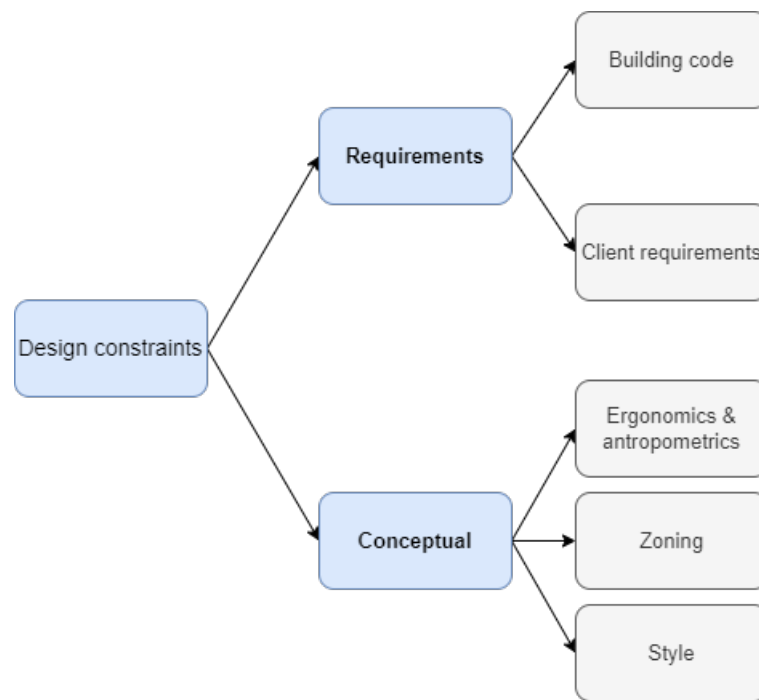


Figure 1: Design constraints from an architectural perspective

have a higher impact on the project, the technical - lower. The economic, legal, and environmental constraints are mainly used during the project.

Constraints from the architectural perspective can be classified into requirement (or requirements) (BORRMANN, HYVÄRINEN, et al., 2009) and conceptual (BISKJAER et al., 2014); see also the scheme in Fig. 1. "Requirements" are regulations or restrictions such as building codes, financial restrictions, and client requirements. "Conceptual" comes from architectural ideas or concepts. Conceptual constraints contain ergonomics & anthropometrics, zones, and style constraints. Conceptual constraints are often not readily identifiable with logical approaches and require intelligent algorithms such as machine learning (ML)(see JUNG and LEE, 2019, LUO et al., 2022, EMUNDS et al., 2022) and natural language processing (NLP) (see NIEMEIJER et al., 2014, ZAHEDI et al., 2022). The subdivision of those constraints is shown below.

Ergonomics & anthropometrics constraints concentrate on the comfortable design that fits the user. An example to consider may be the difference in people's height across the world⁵. Another instance, described in FONSECA et al. (2012), illustrates how the avoidance of the ergonomic rules can affect the work performance using two laboratories as examples.

Zones constraints require efficient connectivity between spaces in the building. How the spaces are grouped depends on several factors, such as usability, comfort, and building shape. Usually, spaces are grouped by functionality (working, eating, relaxing) and can form zones, e.g., a working zone containing working and meeting rooms, a kitchen, and sanitary space. The furniture can be connected to the specific zone, e.g., sanitary

⁵<https://ehs.oregonstate.edu/sites/ehs.oregonstate.edu/files/pdf/ergo/ergonomicsanddesignreferenceguidewhitepaper.pdf>

installation needs to be placed in the sanitary zone. For example, in KRAFT and WILHELMS (2005), the zone (referred to in the paper as the room class) contains different rooms as sections (room). The section connects with the semantic objects corresponding to design rules. EISENSTADT et al. (2021) define spatial information from the architectural plans as a "relation map," a set of semantic information. They create a graph where rooms are part of specific zones and have relationships, such as a type of connection.

Style constraints are formed from style elements such as form, construction method, design pattern, and material. For example, high-tech architecture requires aluminium, steel, glass, and concrete as materials in the building. In the design, open floor plans are dominant, and such elements as ducts, electrical equipment, and mechanical components are visible for users⁶.

2.3 Analysis of constraints

Constraint analysis is a complex task that includes constraint identification and evaluation. Constraints can be analysed differently depending on the geometrical data representation and task.

Geometric data is presented in various ways, e.g., bounding box approximation, octree decomposition, and topological surface boundary representation (ZHOU et al., 2020). ZHOU et al. (2020) inferred three widely used types of representation of spatial data: Point Clouds, Boundary representation model (so-called BRep(see e.g. BORRMANN, SCHRAUFSTETTER, et al., 2009, DAUM and BORRMANN, 2014)), and Solid Model. These types help to increase performance in spatial queries and constraint identification.

The first type, Point Clouds, presents geometry as point clouds. The second, BRep Model, works with triangulated meshes and handles complex geometrical situations more effectively. Solid models, in turn, use information about an object's inner and outer surface and can analyse simple primitive combinations such as union, intersection, and difference.

After retrieving spatial data from the model, the constraint identification and analysis can be processed. Identification is a process of recognising relationships between building components. These can be done with the spatial query such as Spatial Query language introduced in BORRMANN and RANK (2010). Spatial queries are used for model abstraction and need to label spatial relationships. By applying spatial operators, the user can easily query different information from the model. Spatial operators (BORRMANN and RANK, 2010) were developed to analyse the model and used to identify building components that fulfil requirements. They include metric, directional, and topological operators.

Another example of a spatial query is the Query Language for Building Information Models (QL4BIM, DAUM and BORRMANN, 2014), created for filtering specific objects and relationships, which provides metric, directional, and topological operators for expression with spatial semantics. The authors proposed an integrated mechanism (LINQ) to optimise

⁶<https://www.dezeen.com/2019/11/20/anthony-hunt-high-tech-architecture-engineer/>

performance and object networks. They also conclude that BRep representation for topological operators works more effectively with spatial data.

Focusing on the IFC⁷ representation, there is a standard for supporting filtering in the IFC model with the data modelling language EXPRESS (NIEMEIJER et al., 2009). EXPRESS language operates efficient algorithms to process spatial operators (constraints) based on the geometric representation of building elements and space objects. The essential geometrical representations are expressed by: *IfcFacetedBRep*, *IfcAdvantagedBRep*, *IfcTriangulatedFaceSet*. ISMAIL et al. (2017) converts IFC EXPRESS schema and IFC-SPF⁸ files into graph and stores it in Neo4j graph database. Using Cypher language, one can query building data for topology analysis.

Instead of focusing on the IFC data structure, SOLIHIN et al. (2017) introduce BIMRL, a data warehouse-like schema for the BIM data. The approach is user-oriented, and the information stored in the database is most spatial related, such as information about building objects, their property, location, and how objects interact. The schema is defined using a relational database structure. The approach uses standard SQL, integrated with spatial operators, to access data and enable support of spatial queries.

A substantial part of the research is aimed at spatial rule-checking approaches. Rule-checking applies rules, constraints, or conditions to a project and identifies design errors and mistakes (EASTMAN et al., 2009). Spatial rules play an essential role in the design of the building. Since several rules/constraints in the model are already defined, the problem is how to validate the compliance and consistency of these elements. BORRMANN and BEETZ (2010) introduces spatial calculus. Based on the topology, orientation, shape, size, and distance, constraints can be expressed differently (for example, as a graph) and then applied for consistency or compliance checking. NIEMEIJER et al. (2010) also highlights constraint solving. Constraint solving means taking the constraints as input and finding an appropriate design. The solving approach generates possible variations of the design. The appropriate combination needs to meet all given constraints. Usually, the solving method results in more computational time because the process needs to check all combinations and validate them (NIEMEIJER et al., 2010).

Since the research scope of this paper lies in the design constraints, it is important to understand how to identify these constraints out of the design model, e.g. building code, from the user, artificial Intelligence (AI) -based techniques (SACKS et al., 2022).

The understanding of design from the building code and the user can be processed with natural language (understandable by a human). Two possibilities exist to convert natural language into a machine-readable format: deterministic parser and natural language processing (a machine learning technique). The first is the simplest to implement and is used in several domains. The parser can recognise simple sentences with familiar structure and syntax. Typically, the parsing process consists of three steps: lexical, syntactic, and semantic analysis. Lexical analysis splits the input sentence into separate

⁷<https://ifc43-docs.standards.buildingsmart.org/>

⁸<https://technical.buildingsmart.org/standards/ifc/ifc-formats/>

elements (tokens). Then, in the syntactic analysis, each token will be converted into a data structure to check the syntax and grammar. In the last step, the resulting data structure is evaluated (NIEMEIJER et al., 2010).

Another option is natural language processing (NLP). NLP provides a computer-readable representation of a natural text. In contrast to deterministic parses, NLP can check the context of a word and choose the correct interpretation. NLP can work with complicated tasks but is limited in ambiguous word meaning, computational complexity, and linguistic and non-linguistic content (e.g. tacit knowledge, information about stakeholders) (NIEMEIJER et al., 2014). Constraints can be extracted from regulation texts such as a building code or norm and translated in a computer-readable way.

Several pieces of research cover constraint identification with NLP. ZAHEDI et al. (2022) proposed an approach based on the BIM and NLP to meet client requirements and regulations in the design concept. A particular requirement can link to specific properties or constraints of a selected element in the model. JUNG and LEE (2019) use NLP and machine learning techniques to analyze BIM-related tasks. The natural language parser was presented in NIEMEIJER et al. (2014), recognising and transforming natural language into computable constraints. The recognised constraints can be applied to the building model. As a result, the number of elements that match the given constraint will be highlighted. YIN et al. (2023) present ontology-based semantic parser to automatically map comprehensive queries into computer-readable code for querying complex building models.

AI-based technique as a design constraint identification option is novel in BIM applications. It requires a large dataset of building models to recognise design patterns. For example, LUO et al. (2022) uses a deep learning framework (a subset of machine learning) to classify BIM objects and utilise geometrical and semantic information. Another research presents a neural network model based on sparse convolution for classification building IFC-based geometry and semantic enrichment (making implicit information explicit) of BIM model (EMUNDS et al., 2022).

2.4 Representation of constraints

After the identification process, design constraints should be prepared for representation. By representation, one means the form of information ready to be digested by a human (e.g., an architect) or a machine (e.g. a table or a graph). This is essential because the relationships in the building models are very complex, and not all information can be presented in an easy-to-understand manner. There are several ways to represent constraints. The standard methods include mathematical expression, expression using data formats, e.g., XML, JSON, EXPRESS, natural language, and graph representation (ARORA et al., 2021; BORRMANN and RANK, 2010; NIEMEIJER et al., 2009).

The universal representation is mathematical equations and inequalities (BORRMANN and RANK, 2010; NIEMEIJER et al., 2009). The example of constraint distance between two objects (O1 and O2) has the following representation:

$$\text{distance}(O1, O2) := 1000 \text{ mm} \quad (2.1)$$

This representation means "The distance between object 1 and object 2 is equal to 1000 mm". Considering representation, this meaning can be interpreted as a natural language representation of constraints. Regarding the data formats expression Y.-C. LEE et al. (2016) introduce the ontology framework based on the XML⁹ schema. Ontology knowledge is used to represent object properties and relationships. To verify relationships and accuracy from the ontology, semantic reasoning is used.

The graph is a standard method to represent complex data structures. A graph consists of nodes and edges. In our case, nodes represent objects, and edges display relationships. The nodes can have different labels, attributes, and types. Edges can also have individual properties. Both nodes and edges have a unique identifier. The graph structure can be used as a database (graph database) to represent and store data. There are three generic use cases for graphs (POKORNÝ, 2015):

- CRUD (create, read, update, delete) possibility
- data querying, warehousing, and analysis
- data discovery (collection and evaluation)

The Database Management System (DBMS) manages the graph databases. There is a host of graph DBMSs. POKORNÝ (2015) define general and special (e.g. Web, InfoGrid) graph purpose databases. Examples of general databases are:

- Neo4j¹⁰
- Sparksee¹¹
- GraphDB¹²

Regarding the constraints representation of the BIM model, the graphs are a suitable way to manage and visualise constraints, store and update information, and analyse and discover data (BRAUN et al., 2015, VILGERTSHOFER and BORRMANN, 2017, SCHULTZ et al., 2017). Neo4j or GraphDB graph databases would be a relevant solution to serve these goals (POKORNÝ, 2015, LÓPEZ and CRUZ, 2015, KOTIRANTA et al., 2022). Such a database efficiently stores, manages, and edits graphs (SASAKI et al., 2018).

⁹<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/xml/doc/>

¹⁰<https://neo4j.com/>

¹¹<https://www.sparsity-technologies.com/#sparksee>

¹²<https://www.ontotext.com/products/graphdb/>

There are a lot of works about building data representation in a graph way, e.g., representation of constraints, semantic information, requirements, geometry (LANGENHAN et al., 2013, VILGERTSHOFER and BORRMANN, 2017, SCHULTZ et al., 2017, ABUALDENIEN and BORRMANN, 2021). The graph approach helps to understand the complex data structure of the building, identify objects, modify the parametric model, or even evaluate spatial information.

The graph representation of building information can be classified as follows (ABUALDENIEN and BORRMANN, 2021):

- Space connectivity graph. A space connectivity graph is used for space allocation problems (LANGENHAN et al., 2013, SCHULTZ et al., 2017)
- Navigation graph. The navigation graphs are used for navigation in the building (DUBEY et al., 2020) or for routing simulations (KNEIDL et al., 2012).
- IFC model graphs. These graphs represent building model from IFC and contain geometrical representation and attributes related to the objects (EXNER et al., 2019, YIN et al., 2023)
- Knowledge representation graphs. The knowledge representation graph is based on a combination of multiple types of data structures, such as geometrical and semantic information (VILGERTSHOFER and BORRMANN, 2017, SOLIHIN et al., 2017).

The conceptual design information is presented as a graph in KRAFT and WILHELMS (2005). Here, the design rules are offered in a graph as relationships. BRAUN et al. (2015) introduce the approach to compare actual and target states of the building from photogrammetry. The graph helps to identify objects that are on the site but are temporarily occluded in a photo. ABUALDENIEN and BORRMANN (2021) uses graphs to capture design patterns (geometrical and semantic information) and automatically transfer those patterns to new projects. The proposed solution benefits architects and engineers in sharing regulations and design patterns for new projects.

VILGERTSHOFER and BORRMANN (2017) create an automation mechanism for building two-dimensional sketches into a 3D procedural geometry model. The research uses graph and graph rewriting methods (GRS) to represent and modify the model. The nodes presented in a graph correspond to geometric elements (point, line, spline, circle, arc). At the same time, the edges display the parametric constraints (geometric and dimensional constraints) and procedural dependencies (workplane, extrusion, sweep). They mentioned the advantages of graph representation and graph rewriting in capturing design knowledge and enabling model maintenance.

ARORA et al. (2021) introduces an automatic constraint-based approach that enables the evaluation of the consistency of the constraint graph. The spatial configuration is represented as a graph, where nodes are room types, and the edges are connection types. As a result, each arrangement obtains a consistency score, which helps the architect

to decide and approve the configuration. This approach can evaluate the coherency of semantic spatial configuration and use this information in further deep-learning algorithms.

Chapter 3

Methodology

The proposed methodology involves developing and evaluating algorithms to identify and represent the design constraints in the building model. The methodology aims to create an automated solution to accurately identify and represent design constraints systematically and efficiently.

Fig. 2 describes the constraint identification and representation process. The process contains several steps: collection and calculation, analysis, constraint recognition and improvement, and representation in a graph database. The steps are presented in detail below in the following sect. 3.1 describes the building structure, data collection, and calculation processes. Sect. 3.2 presents data analysis methods, including separation and aggregation. Sect. 3.3 and 3.4 presents constraint identification and representation respectively.

3.1 Design interpretation

The data structure plays an essential role in collecting the data from the building model. The IFC schema ensures a neutral environment for the multi-platform exchange of BIM models. According to the official IFC¹ documentation, the building contains a spatial hierarchy of the building components (Fig. 3). The objectified relationship `IfcRelAggregates` connects the spatial elements. The building can be divided into levels (`IfcBuildingStorey`) and can have components (elements) presented by the `IfcProduct`.

Data in BIM can be defined by a domain layer². Domain layer includes domain schemes containing entity definitions that are specific to a certain product, process or discipline. It consists of the following domains:

- Building Controls Domain (`IfcBuildingControlsDomain`)³ represent concepts of building automation, control, instrumentation, and alarm.
- Plumbing Fire Protection Domain (`IfcPlumbingFireProtectionDomain`)⁴ includes plumbing and fire protection services for a building.

¹<https://ifc43-docs.standards.buildingsmart.org/>

²<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/content/introduction.htm>

³<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcbuildingcontrolsdomain/content.html>

⁴<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcplumbingfireprotectiondomain/content.html>

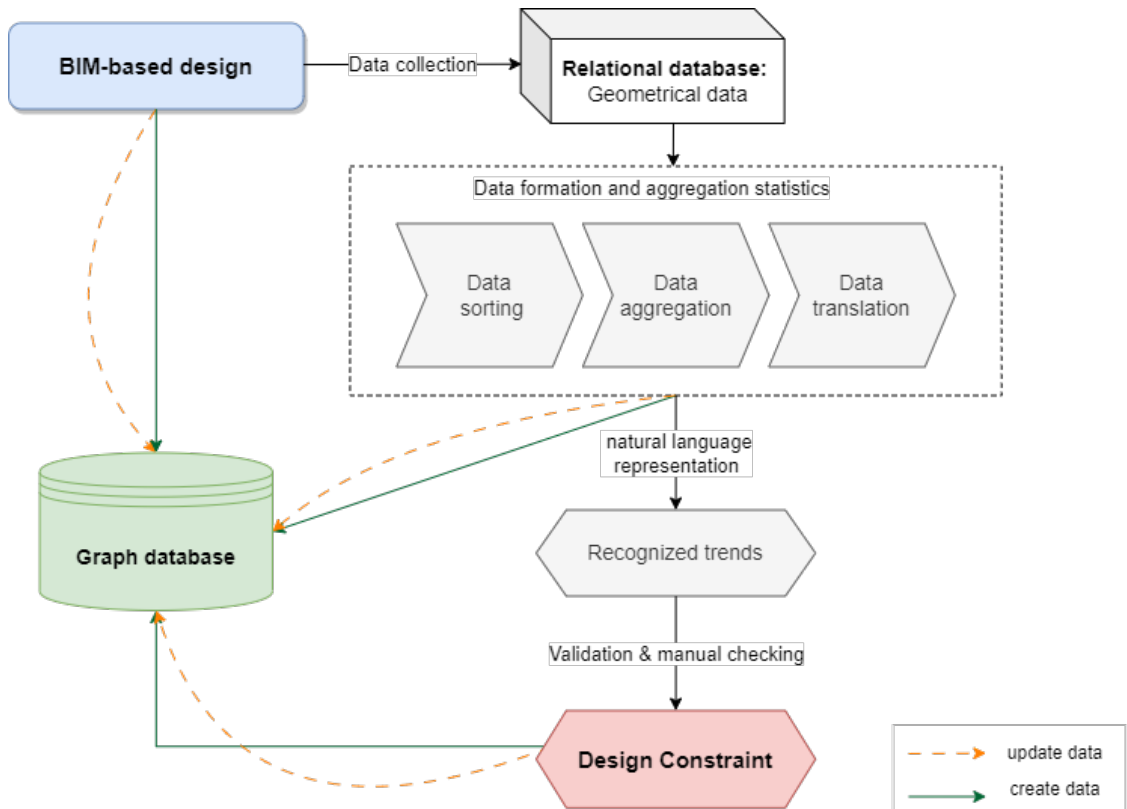


Figure 2: Process of constraint identification and representation.

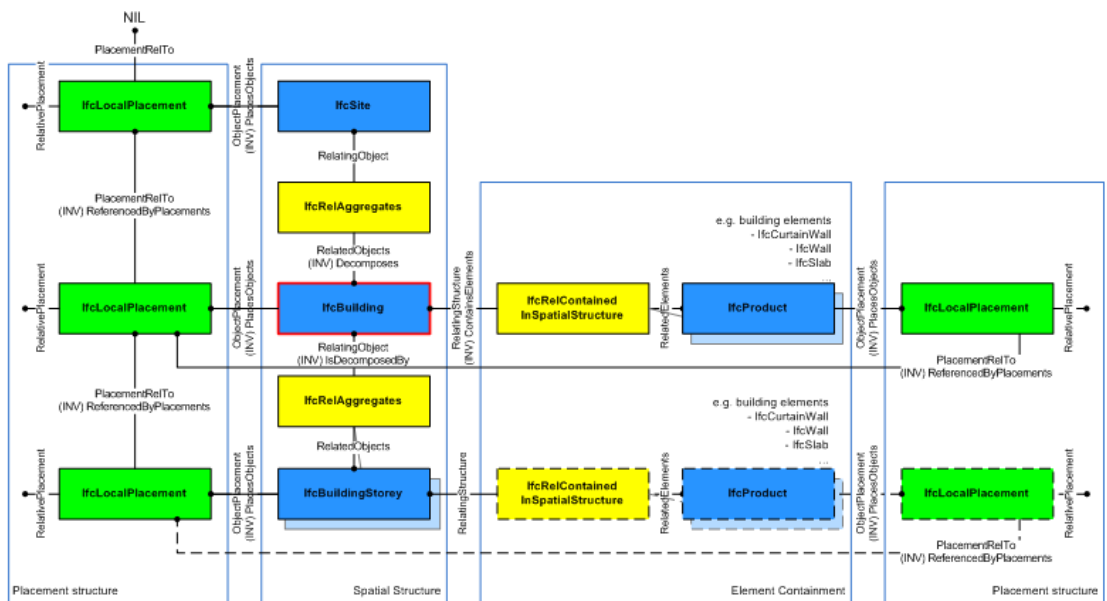


Figure 3: IfcBuilding hierarchy ^a

^a<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/lexical/IfcBuilding.htm>

- Structural Elements Domain ([IfcStructuralElementsDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcstructuralelementsdomain/content.html))⁵ contains structural components of the building such as foundation, different kinds of explicit reinforcement parts.
- Structural Analysis Domain ([IfcStructuralAnalysisDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcstructuralanalysisdomain/content.html))⁶ defines structural analysis model. This includes elements specification and model definition (spatial or/and planar).
- HVAC Domain ([IfcHvacDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifchvacdomain/content.html))⁷ supports objects and concepts required for interoperability within the heating, ventilating and air condition systems.
- Electrical Domain ([IfcElectricalDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcelectricaldomain/content.html))⁸ defines objects and systems for the building's electrical equipment (inc., data, telephone, audio-visual system, etc.). The domain contains methods for supporting and carrying cables.
- Architectural Domain ([IfcArchitectureDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcarchitecturedomain/content.html))⁹ defines basic objects used in architectural domain. Most elements from the architectural domain are shared with other domains.
- Construction Management Domain ([IfcConstructionMgmtDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcconstructionmgmtdomain/content.html))¹⁰ is used to exchange information between construction management applications (it may take place in a single application or on a set of products). This domain uses specific information (e.g., cost, time, productivity) and resources (e.g., material, product, crew) to improve management within the project.
- Ports And Waterways Domain ([IfcPortsAndWaterwaysDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcportsandwaterwaysdomain/content.html))¹¹ contains objects and types of ports and waterways infrastructure.
- Rail Domain ([IfcRailDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcrailldomain/content.html))¹² supports specific rail elements such as rails, sleeper, derailer, etc.
- Road Domain ([IfcRoadDomain](https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcroaddomain/content.html))¹³ defines concept specific to road infrastructure.

In this thesis, the information is used primarily from the architectural domain. Data from the architectural domain contains core information for further operations and can be defined as a **core database**. Each building component should have the following information:

- Identification number
- Location (level and room/space)

⁵<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcstructuralelementsdomain/content.html>

⁶<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcstructuralanalysisdomain/content.html>

⁷<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifchvacdomain/content.html>

⁸<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcelectricaldomain/content.html>

⁹<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcarchitecturedomain/content.html>

¹⁰<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcconstructionmgmtdomain/content.html>

¹¹<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcportsandwaterwaysdomain/content.html>

¹²<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcrailldomain/content.html>

¹³<https://ifc43-docs.standards.buildingsmart.org/IFC/RELEASE/IFC4x3/HTML/ifcroaddomain/content.html>

- Category (walls, windows, doors, etc.)
- Name and Type of the element

The next step after data collection is data calculation. Calculations provide information that cannot be extracted from the [BIM](#) and needs to be calculated by accessing element geometry. An example of this information contains:

- Component dimensions (width, height)
- Horizontal and vertical distances (between components)
- Semantic information (parallelism and perpendicularity)

A new database **geometrical data** is extracted based on the calculations above. The database collects geometrical data and includes distances, sizes, angles, and other spatial relationships of objects. The data is organised in a structured manner to facilitate the analysis process.

3.2 Design constraint analysis

The next step is the analysis of design constraints. There are different options to extract design constraints from the building data. The novel option uses machine learning (ZABIN et al., 2022), allowing intelligence constraint recognition. [ML](#) algorithms can recognise design patterns from a building model and work with semantic information. Design constraint analysis with machine learning is accurate but requires a large data set containing building projects for training models. Another option is the usual statistical analysis (e.g. BORRMANN, HYVÄRINEN, et al., 2009, NIEMEIJER et al., 2010, BORRMANN and RANK, 2010). Statistical analysis can analyse the constraints with classical statistics descriptive tools (such as min, mean, max, standard deviation, quantiles, and mode). Statistical methods do not require computer-heavy calculations and provide necessary information about measurements in the building model. Typically, design constraints are specified by min and max values, whereas mean helps an architect to understand the expected value of a given building dataset. Mode presents the value that appears most often.

This thesis analyses building data with statistical functions (second approach). This requires separating geometrical data into several groups and aggregating min, mean, max, and mode (Fig. 4). Data separation is accomplished by clustering data by common characteristics. The arranged groups are based on architecture logic and may be an element category (Windows, Walls, etc.), a geometrical location (level, room/space), or a building component name ("M_Desk", etc.). In the next step, a dataset is built from the aggregated element of the building. The statistical values are computed, and the result is translated into natural language (human-readable text). The results are marked

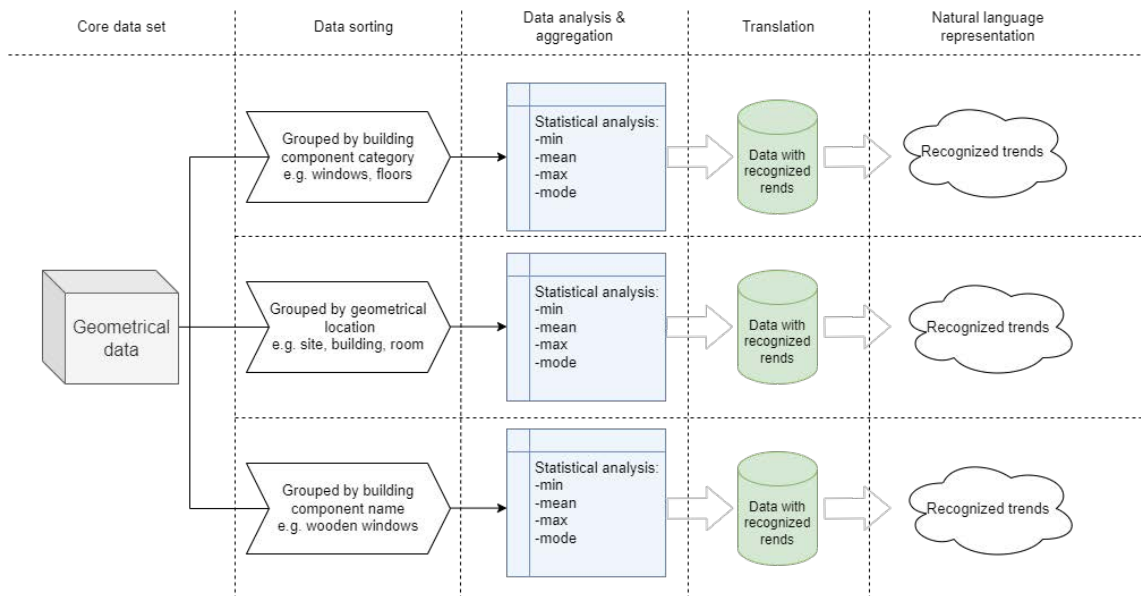


Figure 4: Data formation and aggregation statistics

as “recognised trends.” An example of a recognised trend is “ *All windows of the family M_Fixed have [min, mean, max, mode] [0.193, 2.595, 10.215, 0.6] horizontal distances to the edges.*”

3.3 Customised constraint validation

The analysis may reveal several potential constraints that should be translated into design constraints. The potential constraints are called ‘trends’. And there are a few options to formulate design constraints. First is natural language parsing. The regulations or design prescriptions can be translated into a machine-readable format and compared with statistical data to automatically validate and identify potential constraints as real design constraints. The second option is ML techniques, which take advantage of the historical data (from previous building projects). Machine learning also provides constraint validation automatically. The third option, which is used in this thesis, is the manual selection. By that, a user is expected to choose an appropriate design constraint for a given candidate from the list of trends. Since the aggregated data presents only statistical values and cannot display design patterns, manual selection accurately translates recognised constraint candidates. The manual selection allows one to specify constraints by **characteristic** and by **type**. The **types** are:

- Requirements
- Conceptual

The Requirement type refers to building codes and regulations or requirements. Requirements constraints are often obligatory and describe the design that must be achieved. Conceptual constraints (‘Conceptual’ above) are ideas or design intent that represent the

conceptual solutions of an architect. Conceptual constraints are not obligatory but are recommended.

Characteristics are defined by an interval that describes possible constraint values. For example, the constraint “All windows in the rooms called “Breakout space” have min 0.00 m and max 11.00 m horizontal distance to the edges” can have values in the interval [0.00, 11.00] in case of closed interval or [0.00,11.00) in case of half-closed interval.

3.4 Constraint representation

Design constraints can be represented differently. The proposed representation in this thesis is a graph representation (Fig. 5). The building data is presented in the graph database. This process is accomplished by transferring the data into a graph’s nodes (building components) and edges (relationships between them). The graph database allows the manipulation and analysis of the building information more efficiently compared to traditional databases due to the large number of connections between building components. The constraint representation in a graph database helps one to understand the complex data structure of the building and evaluate spatial information.

After recognition and validation, a design constraint will be converted into a machine-readable format. This conversion is needed to translate constraint into a graph. Commonly, the translation is done with the declarative query language. The declarative graph query language allows efficient data querying in a property graph.

The constraints are presented in a graph format as edges between vertices. The designer can visually convey complex information with the data representation in a graph format. This information can capture design knowledge, identify potential errors, and create transparency in the project life cycle.

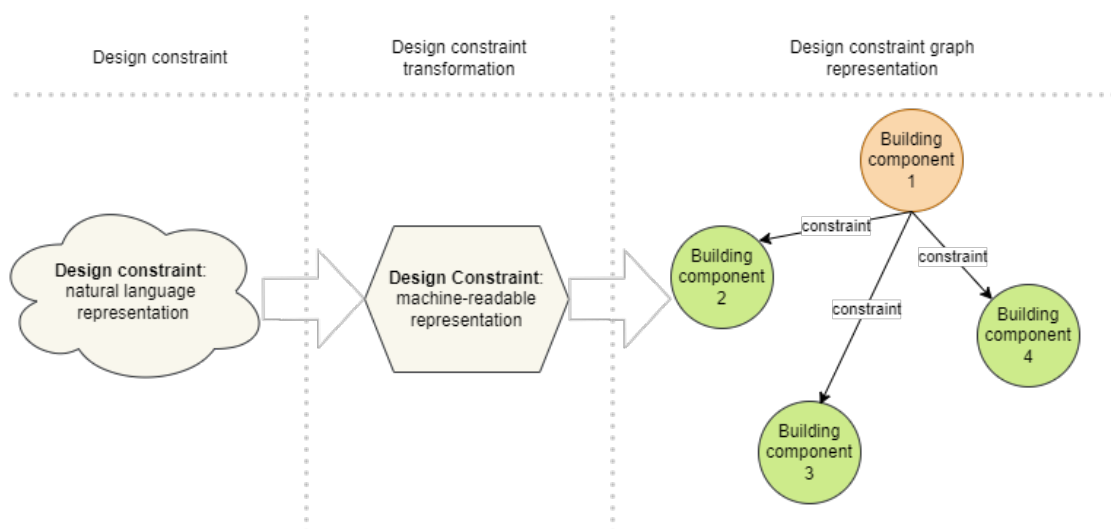


Figure 5: Types of design constraint. The design constraint as natural language is translated into the machine-readable format and then presented in a graph.

Chapter 4

Approach

This chapter explains the retrieval of geometrical information from a building model and the interpretation and representation of design constraints in a property graph. In this thesis, design constraints are requirements or design intentions. The focus lies on geometrical design constraints such as distance, angles, and dimensions.

As mentioned above, the core processes are interpretation/identification and representation of design constraints. Interpretation involves geometrical analysis of the BIM model, including calculations and data evaluation. The geometrical analysis includes estimating distances, angles, dimensions, etc., and collecting elements' properties in a geometrical data set. Data evaluation comes after, which is done by statistical aggregation and manual user selection. A property graph provides a representation of design constraints. These steps are done within an automated tool (plugin) based on PyRevit¹ and Revit API (application programming interface)² within Autodesk Revit 2022. Building components and their relationships, including design constraints, are written in a Neo4j graph database³ using the Cypher declarative query language⁴.

The following chapter is organised as follows. Section 4.1 explains the instruments (Autodesk Revit, Revit API, PyRevit) for developing an automated tool to interact with BIM model and to represent design constraints in a graph (Neo4j). Section 4.2 illustrates the automated tool structure - collection, constraints, and graph panels.

4.1 Instruments

4.1.1 BIM authoring tool

The data collection processes the existing Revit model directly and involves accessing the Revit API. Data collection includes retrieving necessary information to calculate geometrical and semantical relationships, which can be further recognised as constraints. To collect information from Revit, the approach should be selected. There are several possible ways:

- Visual programming using Dynamo⁵ script in a combination with python scripts

¹<https://github.com/eirannejad/pyRevit/releases>

²<https://www.revitapidocs.com/>

³<https://neo4j.com/>

⁴<https://neo4j.com/developer/cypher/>

⁵<https://dynamobim.org/>

- Traditional programming using C#, .NET, C++, or Python languages

The former approach is the most common. Dynamo is easy to use and comes preinstalled with Revit as a plug-in. However, unlike traditional programming, Dynamo needs more flexibility to efficiently solve problems stated in this work. According to the official Autodesk Revit documentation, C# is the best option to develop plugins for Revit. On the other hand, Python is more suited for scientific computing and, most importantly, data analysis. In this thesis, Python with the PyRevit environment is used.

Next, it is essential to understand the structure of the Revit API. [API](#) is a set of namespaces, classes, methods, and properties provided by Autodesk for the developers to interact with and extend the functionality of Autodesk Revit. Revit API contains namespaces that organise programs into logical groups. Examples of some of the namespaces include:

- [Autodesk.Revit](#) – the root namespace.
- [Autodesk.Revit.DB](#) - the common namespace in Revit to interact with building elements. Elements in Revit are single components and abstract categories such as view or floor type.
- [Autodesk.Revit.UI](#) – the namespace to create the custom user interface.
- [Autodesk.Revit.ApplicationServices](#) – the namespace to manage the Revit application.

Classes in Revit represent various objects, elements, or concepts. Important classes include:

- [Document](#) – represents a project or a document.
- [Element](#) – a base class for the elements in Revit.
- [Parameter](#) – represents a parameter associated with an element.
- [Transaction](#) – manages operations in Revit.

Each class has methods and properties to interact with and manipulate elements, parameters, and other components. Fig. 6 shows an example of class structure.

PyRevit

In this thesis, PyRevit⁶ rapid application development ([RAD](#)) environment for Autodesk Revit is used to create plugins. PyRevit enables one to quickly and comfortably create plugins using Python.

To create a plugin, it is important to follow the folder structure⁷ (Fig. 7):

⁶<https://github.com/eirannejad/pyRevit/releases>

⁷<https://pyrevitlabs.notion.site/Create-Your-First-Command-2509b43e28bd498fba937f5c1be7f485>

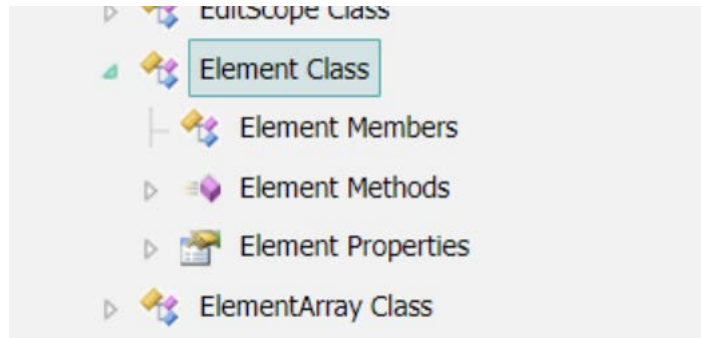


Figure 6: Element class structure with methods and properties inside the class

```

MyExtensions/
├── MyFirstExtension.extension/
│   ├── MyTools.tab/
│   └── MyTools.panel/

```

Figure 7: PyRevit folder structure to create plugin

PyRevit supports two types of Python interpreters: CPython⁸ and IronPython⁹. CPython is the standard Python interpreter. Due to their limitations within the PyRevit environment, both are used. Most notably, IronPython allows data analysis libraries such as pandas, whereas CPython is suitable for creating user interfaces.

4.1.2 Graph database

In this thesis, Neo4j was chosen as a graph database to store building data and represent design constraints. Neo4j¹⁰ is a native graph database to store and manage data. The graph database is a property graph. The data elements are nodes and edges (relationships) with attributes. The elements are nodes, and relationships between elements (including attributes) are edges of the graph. Nodes and edges can be labelled and used to narrow the search.

To write/query data to/from Neo4j graph Cypher language¹¹ query language is used. Cypher uses an ASCII-art type of syntax. The nodes in queries are expressed in rounded brackets, and relationships are shown in squared brackets. Cypher, as a programming language, has keywords for specific actions. The most common are:

- **MATCH**: searches for existing nodes, relationships, labels, or properties in the database.

⁸<https://github.com/python/cpython>

⁹<https://ironpython.net/>

¹⁰<https://neo4j.com/>

¹¹<https://neo4j.com/developer/cypher/>

- **RETURN**: returns nodes or relationships.

4.2 Development of a constraint identifier

The plugin will be presented in Revit as a separate tab called "Super-Constraints" (Fig. 8). The tab contains panels related to processes such as data interpretation, constraint identification, and representation. Each panel includes one or more push buttons. Pushbutton, in turn, consists of the script with Python code, icon image, and additional information, such as tables or text files. The structure is presented in Fig. 9.

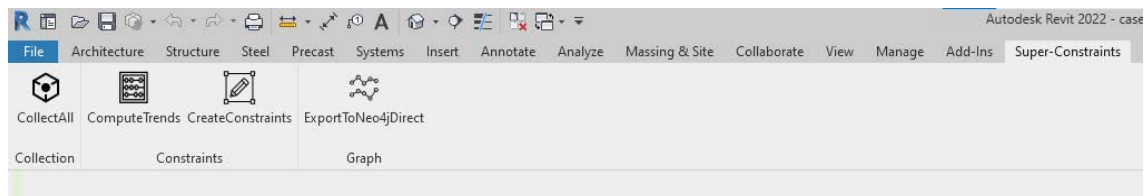


Figure 8: Tab representation in Autodesk Revit 2022

The collection panel (described in sect. 4.2.1) is responsible for the analysis process. Here, the data will be collected and interpreted as tables. In the constraints panel (sect. 4.2.2) constraints will be identified and applied to the project constraints. Graph panel (sect. 4.2.3) is the representation part of the plugin. Elements and their relationships including applied constraints will be represented in a Neo4j graph database.

4.2.1 Collection panel

The collection panel contains a button called "CollectAll.pushbutton". Besides the script and icon image, the pushbutton includes the table "Revit-Categories-2022-arc.csv" (Fig. 10) with Revit categories filtered for an architectural discipline by a user. Types supported by our plugin should be marked by a user as "Yes" in the column "Support." Additionally, the column "Furniture" includes categories related to the furniture. In the current thesis, constraint identification is only supported for specific groups. The support of other types is in future work.

The main script has a structure presented in Fig. 11. In the first step, elements from the Revit model will be collected and sorted. The building is divided into groups of elements (rooms) to find relationships within groups. In the next step, each group's geometrical data will be processed. Finally, all data will be saved into tables.

The script starts with the collection of rooms and elements from the model. After that the elements are separated into groups. These groups are:

- Windows
- Doors

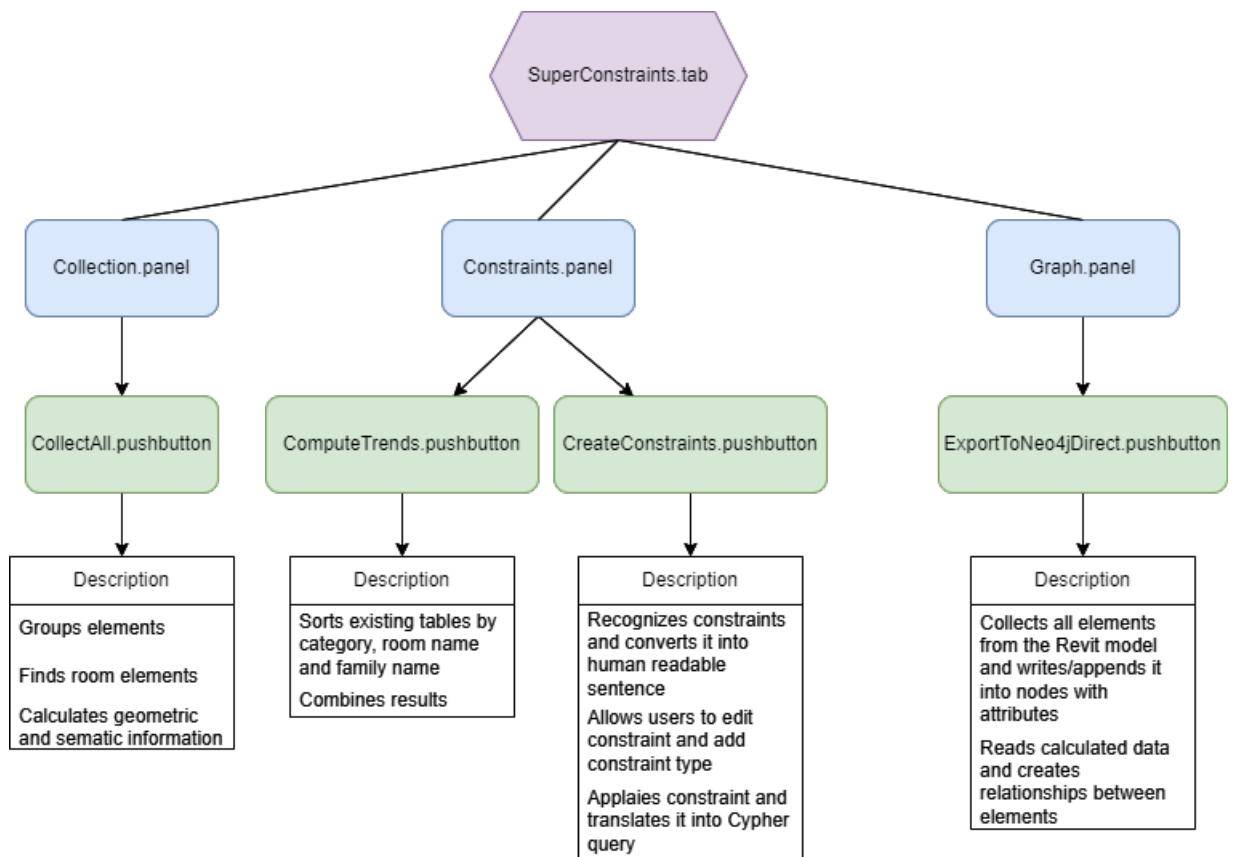


Figure 9: Tab representation in Autodesk Revit 2022

- Floors
- Walls
- Furniture

Once the rooms are defined and elements are separated into one of the groups above, the process of sorting elements into rooms starts. There are several options to find a room for a certain element:

- Find the location point of the element and call the method [GetRoomAtPoint¹²](https://www.revitapidocs.com/2022/656d34c2-1e53-7278-ab83-fefaff7f40a4.htm) of the class [Document](#). The methods return the room containing the point.
- Find the location curve of the element and check the intersection between this curve and the room solid.
- Find the intersection between element solid and room solid.
- For walls, floors, and other elements that bound the room, the room can be defined from the room boundary information (method [GetBoundaryFaceInfo¹³](https://www.revitapidocs.com/2022/150ca07e-90b0-506f-9b9c-fd39d194a7ea.htm) of the class [SpatialElementGeometryResults](#)).

¹²<https://www.revitapidocs.com/2022/656d34c2-1e53-7278-ab83-fefaff7f40a4.htm>

¹³<https://www.revitapidocs.com/2022/150ca07e-90b0-506f-9b9c-fd39d194a7ea.htm>

BuiltIn Category Name	Category ID	English	Suppc	Furniture
OST_StructuralColumns	-2001330	Structural Columns	Yes	
OST_PlumbingFixtures	-2001160	Plumbing Fixtures	Yes	Yes
OST_LightingFixtures	-2001120	Lighting Fixtures	Yes	Yes
OST_ElectricalFixtures	-2001060	Electrical Fixtures	Yes	Yes
OST_GenericModel	-2000151	Generic Models	Yes	Yes
OST_Stairs	-2000120	Stairs	Yes	
OST_Columns	-2000100	Columns	Yes	
OST_Furniture	-2000080	Furniture	Yes	Yes
OST_Ceilings	-2000038	Ceilings	Yes	
OST_Floors	-2000032	Floors	Yes	
OST_Doors	-2000023	Doors	Yes	
OST_Windows	-2000014	Windows	Yes	
OST_Walls	-2000011	Walls	Yes	
		Analytical Columns	Yes	

Figure 10: Part of Table “Revit-Categories-2022-arc.csv” with supported Revit categories from architectural discipline

An appropriate method will be used for each element depending on its type (e.g., a wall or a piece of furniture).

Once the elements are sorted into rooms, the calculation process starts. The calculation will be described separately for each group of elements.

Walls

Walls belong to the class `Wall` and have curves as locations. Before calculating distances, it is essential to separate walls into parallel and perpendicular pairs. In this thesis, the distances between parallel elements and angles between nonparallel are calculated. An example setup is shown in Figure 6. The location curve of wall #1 (Figure 6, left) has the start and end points, and the direction is shown as an arrow. The first step is determining whether other walls (#2,3,4) are parallel or perpendicular to wall #1. To find this the method `Intersects`¹⁴ from `Curve` class is used. This method has one of the following outputs:

- `Overlap` – one or more intersections occur.
- `Subset` - two curves are parallel and have one intersection point.
- `Superset` – the input curve is entirely within another curve.
- `Disjoint` – no intersections found.
- `Equal` – the curves are identical.

If two walls are potentially perpendicular (#1 and #3 in the left panel of Fig. 12) the result of the method `Intersect` returns “Overlap”. Additionally, the angles between the two walls need to be checked.

If two walls are parallel (#1 and #3, Fig. 12), right panel) the output is “Disjoint” with the angle between them 0 degrees. To exclude the case when two disjoint walls are perpendicular (walls #1 and #4, right panel in Fig. 12), two unbounded lines will be

¹⁴<https://www.revitapidocs.com/2017/90e86110-9bce-6e43-c18a-4d67380008bb.htm>

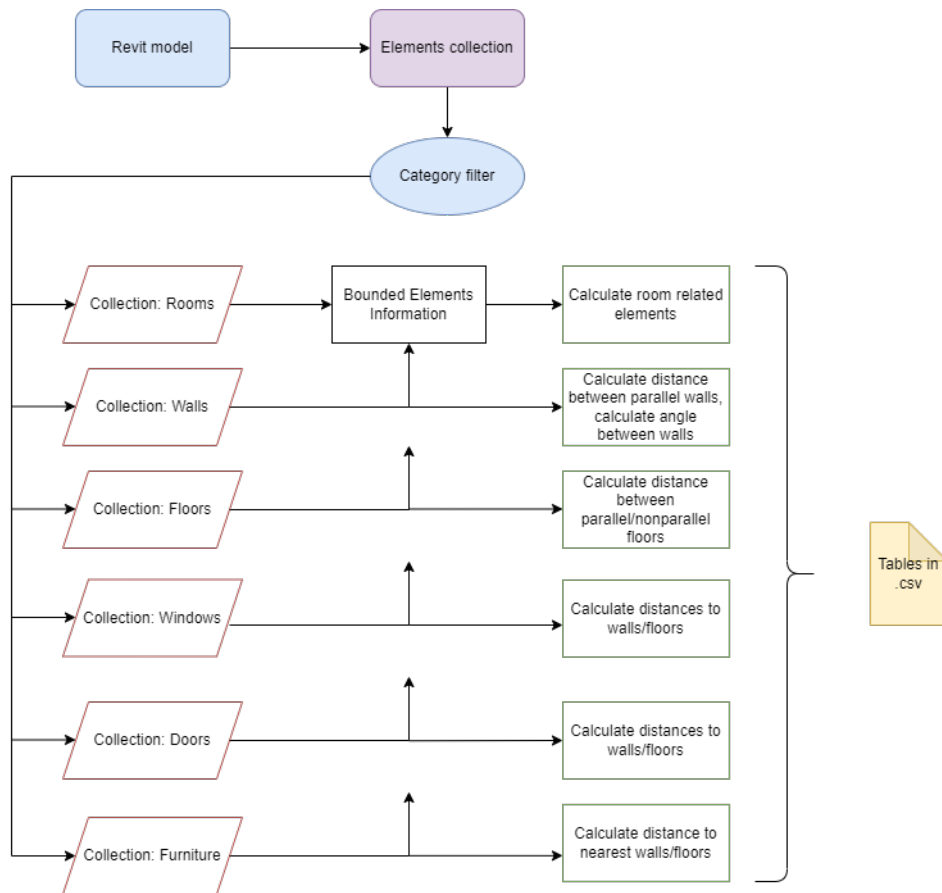


Figure 11: Structure of the CollectAll button

created (from the curves of walls #1 and #4) and checked the intersection and angle between them.

Finally, to find the distance between two parallel walls one point from the first location curve needs to be projected into the other location curve. The points can be accessed from the [Curve](#) class as start or endpoints. The curve is converted into an unbound line for projection. Before calculating the distance, the projection point should correspond to the selected point on the first curve (start or end point). If the points do not match, then the direction of the second line needs to be changed (Fig. 13).

Floors

Floor elements belong to the classes [Floor](#) and [Ceiling](#). From floors, it is impossible to extract the location, neither the location point nor the location curve; therefore, finding a suitable way to calculate distances is necessary. There are different options:

- Use room bounding box properties.
- Convert floors into solid geometry and calculate distances between floors and room location points.
- Convert floors into solid geometry and extract endpoints from edges. Then, calculate minimal distances between opposite floors.

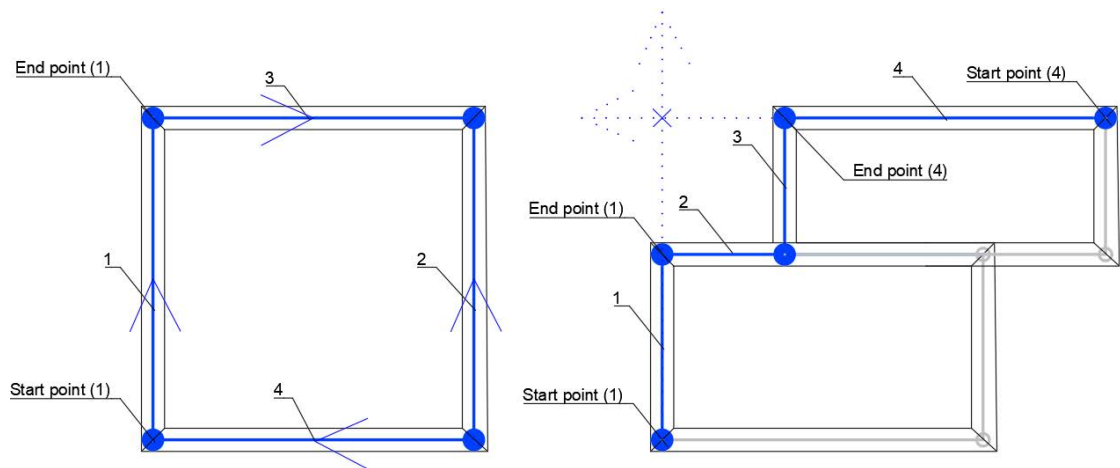


Figure 12: Walls with the location of lines in the middle of each wall (blue line). The start and endpoints are marked with blue circles. Arrows in the left figure display directions from a line's start to the endpoint. Dotted lines in the right figure present unbound lines from the origin point in the given direction (walls #1 and #4)

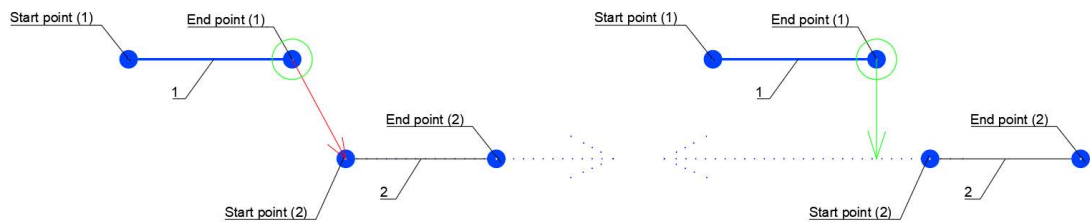


Figure 13: Projection endpoint (in green circle) on line #2. On the left figure, the projection point corresponds to the start point on line #2 and cannot be used for distance calculation. The direction (dotted blue line) of the line #2 was changed on the right figure. The projection point is now in the right place (green arrow).

The first method is the simplest and is suitable if the room has one floor and one ceiling. In this case, converting a room into a bounding box is necessary, and then using the bounding box properties "Min" and "Max" to find the room's minimum and maximum z points. The method enables calculation for tilted surfaces (roofs). The distance between two elements is the difference between minimum and maximum points (Fig. 14).

The second method suits a room with multiple floors, such as false floors or ceilings. The method transforms each floor or ceiling into a solid geometry, creating a normal vector from the floor surface to the room location point. The distances between opposite floors should be added, and the distances between parallel floors (with the same normal vectors) should be subtracted. The calculated distances are presented in Fig. 15.

The third method is the combination of previous methods. The method gives more accurate results and will be used as a primary method in the calculation. For this method, only the z coordinate of each edge of the floor/ceiling face can be extracted, and the minimum

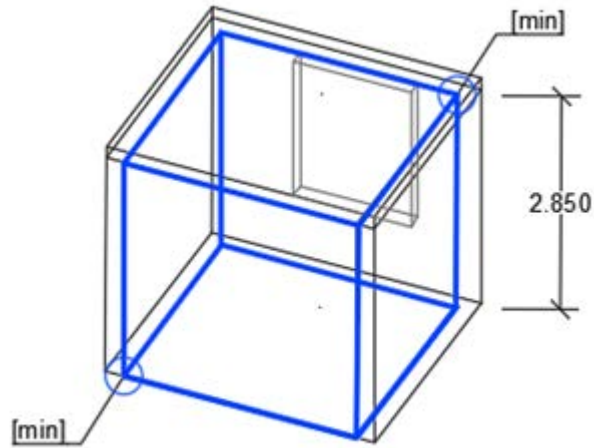


Figure 14: Minimum and Maximum points of the room bounding box.

and maximum values can be found. The distance is the smallest difference between two points. Fig. 16 presents an example of the calculation.

Doors and Windows

The calculations for windows/doors and nearby elements involve extracting the opening cut, as shown in Fig. 17. There are two options to extract the opening cut. The first option is to access the class `ExporterIFCUtils`¹⁵ which contains several methods such as `GetInstanceCutoutFromWall`¹⁶. This method gets the curve loop corresponding to the hole in the wall made by the element.

Another option is to extract cut edges directly from the intersection between the wall solid and the window or door solid.

The closest distance from the opening cut to the nearest elements can be found as follows:

- For the nearest walls distance is calculated with the method `ComputeClosestPoint`¹⁷ of class `Curve`. The method returns the closest points between two curves and, correspondingly, can be used only with walls. Then, the distance between the two points can be calculated
- The nearest floor distance is calculated with the floor's solid geometry. The distance is found between the curve point and the nearest solid face.

Furniture

Because of the geometrical complexity of the furniture, the distance to be found is the distance from the furniture element location point to the room-bounding elements (Fig. 18). Other constraints for furniture are not implemented in this thesis.

¹⁵<https://www.revitapidocs.com/2018/e0e78d67-739c-0cd6-9e3d-359e42758c93.htm>

¹⁶<https://www.revitapidocs.com/2018/07529283-96a7-8aca-5edf-906d8ddd3b7d.htm>

¹⁷<https://www.revitapidocs.com/2022/04ab73d1-bc85-9b87-aace-4272a0c7c3e4.htm>

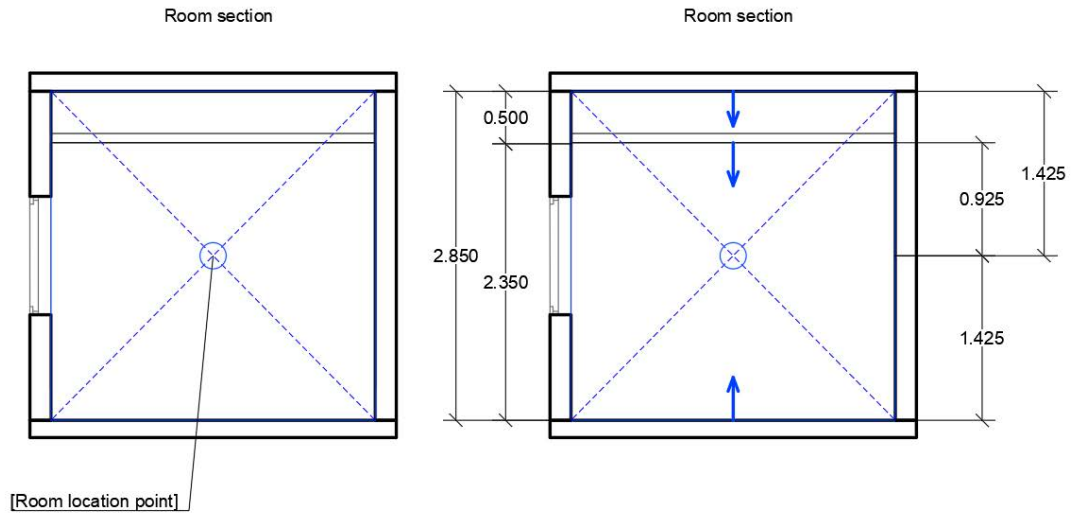


Figure 15: Calculation process for Floor type elements. On the left panel, the room location point is displayed. The faces normal to the room location point are presented on the right panel.

Data storage

After collecting geometrical information, it is important to save all data for further operations. All data will be converted into tables using **CSV** (Comma-Separated Values) format. The output is in the following tables:

- "room-elements-bounds.csv" - contains room bounding elements
- "room-elements-walls.csv" - contains calculated distances between parallel walls, perpendicular walls id, and nearest walls ids.
- "room-elements-floors.csv" - contains distances between nonparallel and parallel floors and ceilings
- "room-elements-doors.csv" - contains distances from the door opening cut to floors and walls
- "room-elements-furniture.csv"- contains distances from window opening cut to floors and walls
- "room-elements.csv" - the main dataset.

The main dataset contains information about the level, room location, element name, type, and category (Fig. 19). Each component has an identification number, such as an ID and a unique ID. The rest of the data files contain query information about the room location and geometrical data (Fig. 20).

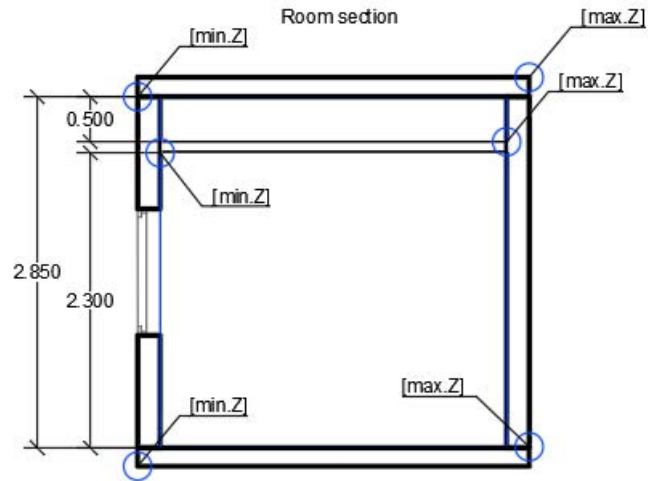


Figure 16: An example of the distance calculation with the floor's minimum and maximum z coordinate.

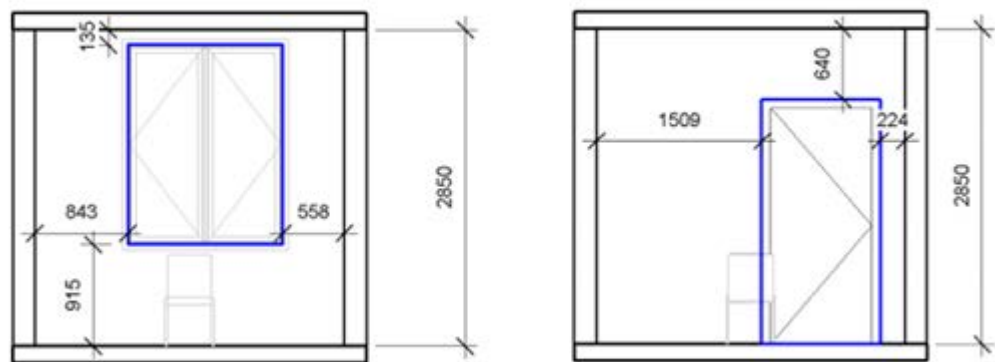


Figure 17: Window and door opening cut marked in blue with dimensions to the nearest walls and floors.

4.2.2 Constraint Panel

The Constraint panel consists of two pushbuttons [ComputeTrends](#) and [CreateConstraints](#). This panel analyses geometrical information and recognises trends that allow users to choose and apply constraints.

[ComputeTrends](#) pushbutton performs analysis of the geometrical information. The script converts all tables into two-dimensional tabular data. For the conversion and further analysis, [pandas](#)¹⁸, [NumPy](#)¹⁹, and built-in statistics package for Python are used. The data is grouped by certain values. Usually, in architectural practice, elements can be grouped depending on their position (e.g., floor, meeting room), object name (e.g., brick wall or concrete wall), and category (e.g., walls, windows). The groups are:

¹⁸<https://pandas.pydata.org/>

¹⁹<https://numpy.org/>

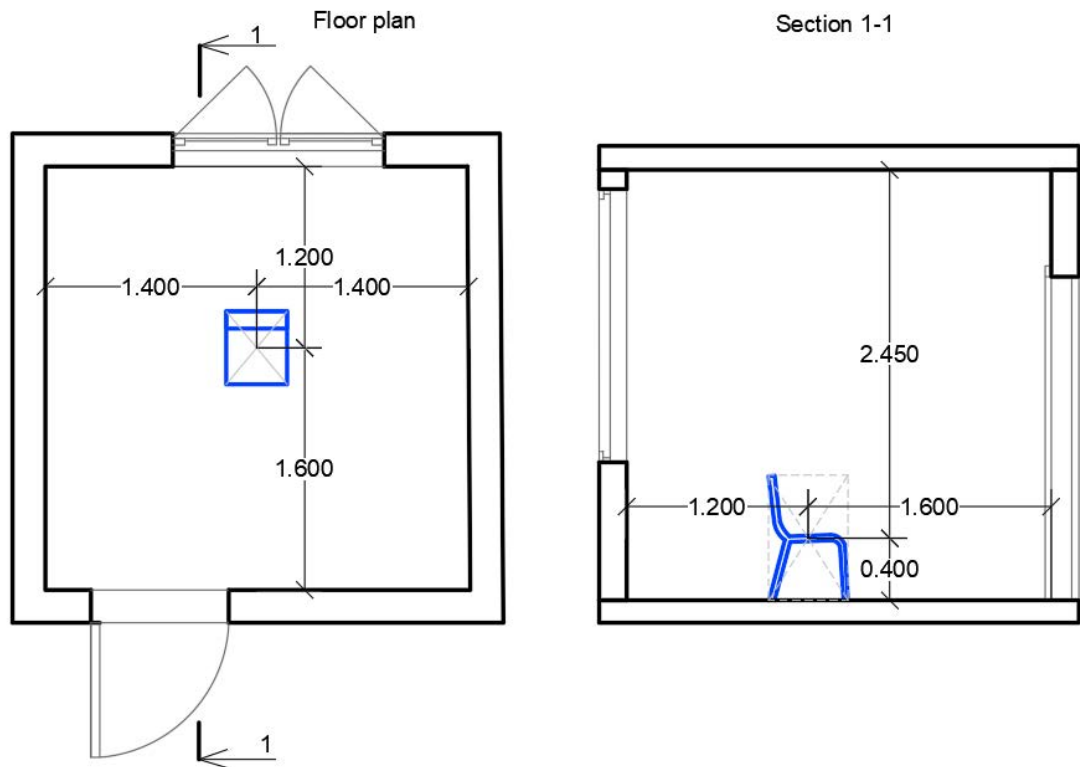


Figure 18: Floor plan (left) and section 1-1 (right) with dimensions of the chair.

- Category(as listed above)
- Room name
- Family name

Each category of data needs to be analysed separately. This analysis aims to apply statistical functions (e.g. min, mean, max and mode) to the building components. For example, it is important to calculate the windowsill's possible minimum and maximum height. As shown in Fig. 21, the room has windowsills of three different windows on different heights, and interpreting only the minimum and maximum values from these three windowsills can be incorrect. In this case, it is better to group all windows not by room but by window family name (in our case, the names are 1,2,3) and calculate the value of the windowsill. Design patterns can be recognised by calculating the most common value (mode). The most common value can be the distance between the neighbour's window or the windowsill height.

Geometrical interpretations of windows and doors are similar. They build an opening cut in the wall, and the distances between this cut and other elements (walls, floors and nearest window) must be calculated. The distance between the window cut and the wall is horizontal distance ("Distance_to_edges_hor"), and the distance between the cut and floor/ceiling is vertical distance("Distance_to_edges_vert"). The distance between the following windows is defined as "Distance_to_next." While grouping data into the abovementioned groups uses global minimum, mean, maximum, and mode values.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Level_Id	Level	Room_uni	Room_uni	Room_id	Room_number	Room_name	ElementId	Element_uni	Family	Typ	Category
2	0	311	Level 1	69a69155-af3a-4	393139	5	Corridor	359726	69a69155-af3a-4a	M_Door-Passage-Single-Flush	M_Door-Passage-Single-Flush	Doors
3	1	311	Level 1	69a69155-af3a-4	393139	5	Corridor	359780	69a69155-af3a-4a	M_Door-Passage-Single-Flush	M_Door-Passage-Single-Flush	Doors
4	2	311	Level 1	69a69155-af3a-4	393139	5	Corridor	362868	69a69155-af3a-4a	M_Door-Passage-Uneven-Flush	1350 x 2000mm	Doors
5	3	311	Level 1	69a69155-af3a-4	393139	5	Corridor	372407	69a69155-af3a-4a	M_Door-Passage-Single-Two_Lite	1050 x 2000mm	Doors
6	4	311	Level 1	69a69155-af3a-4	393139	5	Corridor	372476	69a69155-af3a-4a	M_Door-Passage-Single-Two_Lite	1050 x 2000mm	Doors
7	5	311	Level 1	69a69155-af3a-4	393139	5	Corridor	372678	69a69155-af3a-4a	M_Single-Flush	0915 x 2134mm	Doors
8	6	311	Level 1	69a69155-af3a-4	393139	5	Corridor	372808	69a69155-af3a-4a	M_Single-Flush	0915 x 2134mm	Doors
9	7	311	Level 1	69a69155-af3a-4	393139	5	Corridor	373041	69a69155-af3a-4a	M_Single-Flush	0915 x 2134mm	Doors
10	8	311	Level 1	69a69155-af3a-4	393139	5	Corridor	373253	69a69155-af3a-4a	M_Door-Passage-Single-Two_Lite	1050 x 2000mm	Doors
11	9	311	Level 1	69a69155-af3a-4	393139	5	Corridor	387727	69a69155-af3a-4a	M_Door-Exterior-Double-Two_Lite	2100 x 2000mm	Doors
12	10	311	Level 1	69a69155-af3a-4	393139	5	Corridor	391619	69a69155-af3a-4a	M_Fixed	1200 x 1830mm	Windows
13	11	311	Level 1	69a69155-af3a-4	393139	5	Corridor	391847	69a69155-af3a-4a	M_Fixed	0915 x 1830mm	Windows
14	12	311	Level 1	69a69155-af3a-4	393139	5	Corridor	391985	69a69155-af3a-4a	M_Fixed	0915 x 1830mm	Windows
15	13	311	Level 1	69a69155-af3a-4	393139	5	Corridor	349688	69a69155-af3a-4a	Floor	Generic 150mm	Floors
16	14	311	Level 1	69a69155-af3a-4	393139	5	Corridor	349649	69a69155-af3a-4a	Floor	Generic 150mm	Floors
17	15	311	Level 1	69a69155-af3a-4	393139	5	Corridor	348774	69a69155-af3a-4a	Basic Wall	Generic - 300mm	Walls
18	16	311	Level 1	69a69155-af3a-4	393139	5	Corridor	349983	69a69155-af3a-4a	Basic Wall	Interior - 138mm Partition (1-hr)	Walls
19	17	311	Level 1	69a69155-af3a-4	393139	5	Corridor	387704	69a69155-af3a-4a	Basic Wall	Interior - 138mm Partition (1-hr)	Walls
20	18	311	Level 1	69a69155-af3a-4	393139	5	Corridor	349065	69a69155-af3a-4a	Basic Wall	Generic - 300mm	Walls
21	19	311	Level 1	69a69155-af3a-4	393139	5	Corridor	351958	69a69155-af3a-4a	Basic Wall	Generic - 200mm	Walls
22	20	311	Level 1	69a69155-af3a-4	393139	5	Corridor	351821	69a69155-af3a-4a	Basic Wall	Generic - 200mm	Walls
23	21	311	Level 1	69a69155-af3a-4	393139	5	Corridor	348890	69a69155-af3a-4a	Basic Wall	Generic - 300mm	Walls

Figure 19: An example of the data table "room-elements.csv"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Room_id	Room_uni	ElementId	Element_uni	Window_width	Window_height	ElementId	ElementId	Distance_1	Distance_1	Distance_1	Distance_1	Distance_1	Distance_1	Distance_1	Distance_1	Distance_to_1	ElementId	Distance_to_next_win_min
2	0	393139	69a69155-	391619	69a69155-	1.2	1.83	[349688, 3]	[349983, 3]	[1.673, 0.4]	[2.545, 0.3]	0.357	3.862	10.157	0.305	1.425	2.545		0
3	1	393139	69a69155-	391847	69a69155-	0.915	1.83	[349688, 3]	[348774, 3]	[5.697, 6.6]	[2.545, 0.3]	0.2	3.395	6.612	0.305	1.425	2.545	391985	1.486
4	2	393139	69a69155-	391985	69a69155-	0.915	1.83	[349688, 3]	[348774, 3]	[6.867, 7.7]	[2.545, 0.3]	0.256	3.395	7.782	0.305	1.425	2.545	391847	1.486
5	3	393142	69a69155-	391444	69a69155-	0.915	1.83	[349688, 3]	[349065, 3]	[2.565, 1.6]	[2.545, 0.3]	0.235	1.4	2.565	0.305	1.425	2.545	391493	1.687
6	4	393142	69a69155-	391493	69a69155-	1.2	1.83	[349688, 3]	[349065, 3]	[0.193, 1.3]	[2.545, 0.3]	0.193	1.401	2.608	0.305	1.425	2.545	391444	1.687
7	5	393484	69a69155-	391028	69a69155-	1.35	1.8	[349688, 3]	[363293, 3]	[2.265, 0.4]	[0.6, 2.25]	0.465	1.365	2.265	0.6	1.425	2.25		0
8	6	393485	69a69155-	391134	69a69155-	1.35	1.8	[349688, 3]	[348890, 3]	[2.281, 0.4]	[0.6, 2.25]	0.481	1.381	2.281	0.6	1.425	2.25		0
9	7	393486	69a69155-	391060	69a69155-	1.35	1.8	[349688, 3]	[363351, 3]	[2.215, 0.4]	[0.6, 2.25]	0.415	1.315	2.215	0.6	1.425	2.25		0
10	8	393487	69a69155-	392011	69a69155-	0.915	1.83	[349688, 3]	[350176, 3]	[0.6, 1.515]	[2.545, 0.3]	0.6	2.078	3.556	0.305	1.425	2.545	392077	0.2
11	9	393487	69a69155-	392077	69a69155-	0.915	1.83	[349688, 3]	[350176, 3]	[1.715, 2.6]	[2.545, 0.3]	1.526	2.078	2.63	0.305	1.425	2.545	392011	0.2
12	10	393487	69a69155-	392173	69a69155-	0.915	1.83	[349688, 3]	[350176, 3]	[2.83, 3.74]	[2.545, 0.3]	0.411	2.078	3.745	0.305	1.425	2.545	392077	0.2
13	11	393488	69a69155-	392211	69a69155-	0.915	1.83	[349688, 3]	[363416, 3]	[0.6, 1.515]	[2.545, 0.3]	0.6	2.053	3.506	0.305	1.425	2.545	392337	0.2
14	12	393488	69a69155-	392337	69a69155-	0.915	1.83	[349688, 3]	[363416, 3]	[1.715, 2.6]	[2.545, 0.3]	1.476	2.053	2.63	0.305	1.425	2.545	392211	0.2
15	13	393488	69a69155-	392401	69a69155-	0.915	1.83	[349688, 3]	[363416, 3]	[2.83, 3.74]	[2.545, 0.3]	0.361	2.053	3.745	0.305	1.425	2.545	392337	0.2
16	14	393670	69a69155-	392702	69a69155-	1.35	1.8	[349712, 3]	[373953, 3]	[2.265, 0.4]	[1.325, 0.5]	0.465	1.365	2.265	0.55	1.6	2.675		0
17	15	393671	69a69155-	392704	69a69155-	1.35	1.8	[349712, 3]	[349628, 3]	[2.281, 0.4]	[1.325, 0.5]	0.481	1.381	2.281	0.6	1.6	2.675		0
18	16	393672	69a69155-	392703	69a69155-	1.35	1.8	[349712, 3]	[373954, 3]	[2.215, 0.4]	[1.325, 0.5]	0.415	1.315	2.215	0.55	1.6	2.675		0
19	17	393673	69a69155-	392710	69a69155-	0.915	1.83	[349712, 3]	[373952, 3]	[0.6, 1.515]	[3.27, 2.49]	0.6	2.078	3.556	0.305	1.718	3.27	392711	0.2
20	18	393673	69a69155-	392711	69a69155-	0.915	1.83	[349712, 3]	[373952, 3]	[1.715, 2.6]	[3.27, 2.49]	1.526	2.078	2.63	0.305	1.718	3.27	392710	0.2
21	19	393673	69a69155-	392712	69a69155-	0.915	1.83	[349712, 3]	[373952, 3]	[2.83, 3.74]	[3.27, 2.49]	0.411	2.078	3.745	0.305	1.718	3.27	392711	0.2
22	20	393674	69a69155-	392713	69a69155-	0.915	1.83	[349712, 3]	[373955, 3]	[0.6, 1.515]	[3.27, 2.49]	0.6	2.053	3.506	0.305	1.718	3.27	392714	0.2

Figure 20: An example of the data table with windows geometrical information "room-elements-windows.csv"

The furniture, walls and floors are calculated with distances or angles between surfaces. In the case of furniture, the spaces ("Distance_to_nearest") can be calculated between the furniture location point and the nearest wall or floor surface. The distances between walls are measured between parallel walls. Knowing the narrowest and widest distances between parallel walls is vital for an architect. Examples may include the corridors (the narrowest is essential) and the gym (the largest distance is essential). Those distances are defined in the analysis as "Distance_to_parallel_walls". Additionally, angles between walls are calculated. On the other hand, the ceiling height is fixed within a room or level. The parallel distance is "Distance_to_parallel" and nonparallel (e.g. a sloping roof) is "Distance_to_nonparallel". New tables with aggregated data will be created after analysing all categories and grouping them into specific groups.

The [CreateConstraints](#) pushbutton has the primary information (script, icon image) and also user interface data in the form of the "XAML" script. The user interface is presented in [Fig. 22](#).

The main script starts with loading data (element 1 in [Fig. 22](#)). The data should be aggregated and computed in the previous script [ComputeTrends](#). After loading data, the algorithm reads the selected table and converts each row into human-readable form (2 in [Fig. 22](#)). The user is allowed to choose one of these recognised trends and move (3 in [Fig. 22](#)) it into the design constraint section (8 in [Fig. 22](#)). In this section, the user can

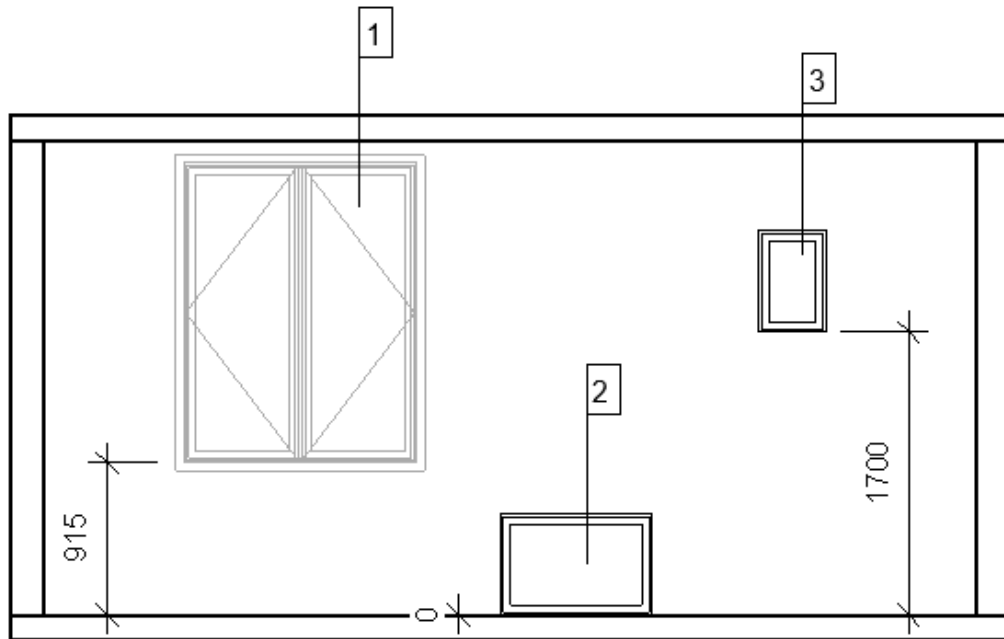


Figure 21: An example of different windowsill heights in one room.

correct values in the brackets if needed. Then constraint should be characterised (4 in Fig. 22). Mathematical intervals are used as characteristics. Interval describes the limits of the constraint. Intervals are:

- Closed interval includes all limit points and is denoted with square brackets.
- Half-open interval contains only one endpoint and is denoted by square and round brackets.

In element 6 from Fig. 22, there is a selection for the first and second limit points. The possible limits are:

- "min" – minimum value.
- "mean" - mean value.
- "max" - maximum value.
- "mode" - mode value.
- "inf" - infinity.

Element 5 in Fig. 22 sets the type of constraint. In this thesis, there are two defined types: requirement and conceptual. After adding characteristics, limits, and types of constraint, the constraint needs to be applied to the list of project constraints and transformed into a query language (7 in Fig. 22). Since Neo4j is a graph database, the query language is Cypher (Fig. 23).

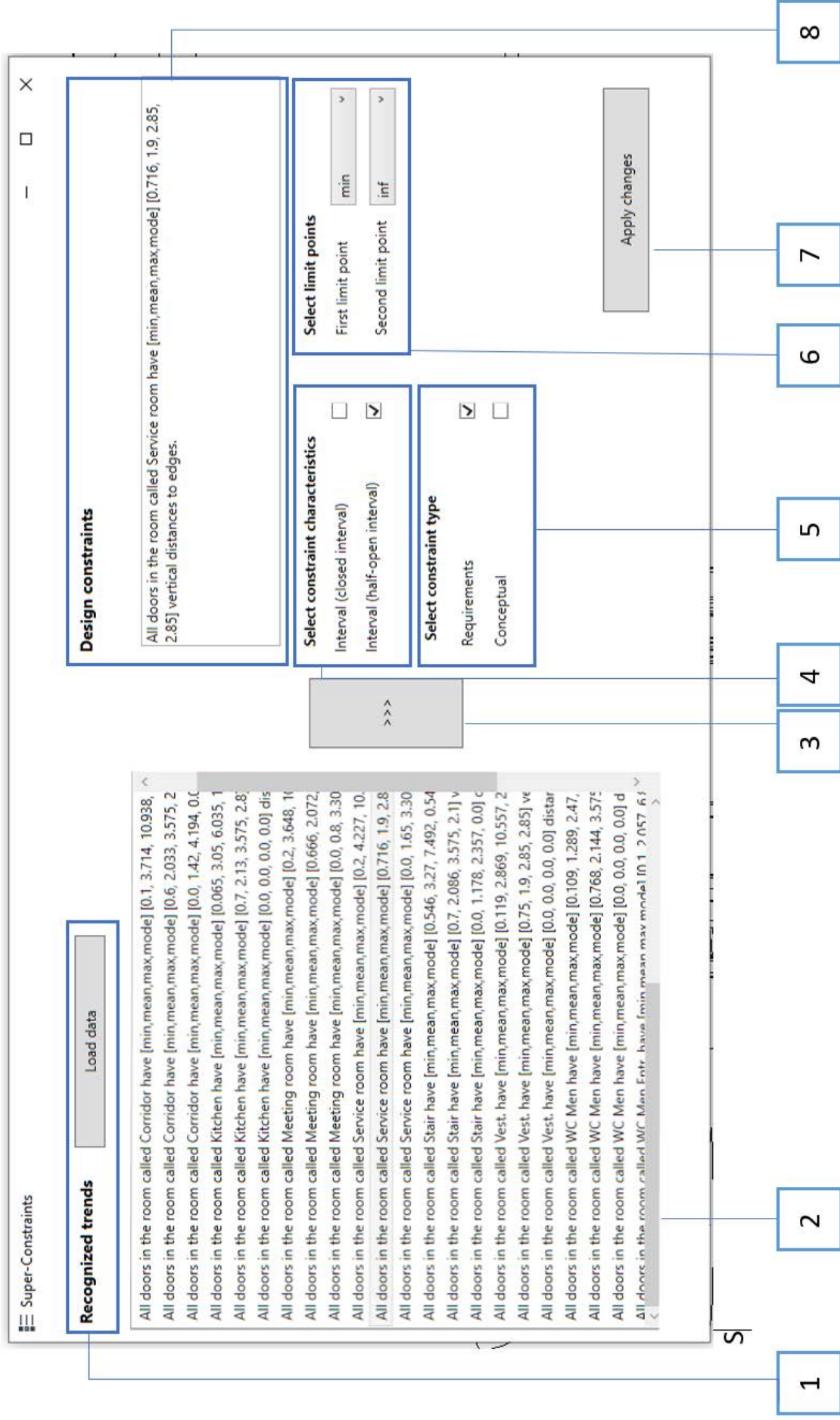


Figure 22: The user interface of the pushbutton CreateConstraints. 1 - load data, 2 - list of potential constraints (recognised trends), 3- button to move selected recognised trend for edition in (8), limit point (6) and type (5). Button (7) transforms constraint into query language and writes into a text file.

Constraint	Cypher query
<i>All windows in the room called Office have [min, mean, max, mode] [0.361, 2.065, 3.745, 0.6] horizontal distances to edges., "conceptual"</i>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE n.room_name = "Office" AND m.category = 'Windows' SET m.constr_distance_horizontal_min= 0.361, m.constr_distance_horizontal_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.361,distance_hor_max:"inf",constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>

Figure 23: Design constraint transformation into Cypher query language

4.2.3 Graph Panel

The Graph panel contains pushbutton [ExportToNeo4jDirect](#). Within the button's folder is an additional text file with Neo4j logging information (login, password) for the Neo4j database, which the user should put in. This pushbutton is responsible for the representation part. Here, elements from the Revit model will be presented as nodes with labels (e.g., Window, Furniture) and geometrical information in relationships between nodes (e.g., horizontal or vertical distances). After running the script, the user has two options: merge new information or delete all existing nodes from the graph. The merging process updates existing nodes and adds new data from the model. The deletion removes old data and inputs new nodes and vertices.

The main script starts with collecting all elements directly from the model with the category filter as described in the 4.2.1. The elements structure is presented in Fig. 24.

While creating new (or updating existing) nodes in Neo4j, properties of nodes are added. For the Document node, the properties are the document name and path. All other elements have category name, identification number ([Id](#)), unique identification number ([UniqueId](#)), and additional information for each type as properties.

Once the collection is completed, relationships can be defined. The first relationships are created between rooms and levels, then between documents and levels.

The next step is to load all data from the [CollectAll](#) pushbutton. From the "room-elements.csv" file, the relationships "CONTAINS" (Fig. 25) and "BOUNDS" are defined. The relationships between windows and doors are presented as "DISTANCE_VERT" and "DISTANCE_HOR". The distances between parallel walls create "DISTANCE_PAR", "PARALLEL", and "PERPENDICULAR" vertices. Floors have "DISTANCE_PAR" and "DISTANCE_NONPAR" relationships, and additional relationships "PARALLEL" and "NON-PARALLEL". Relationships between the nearest walls/floors and furniture are explained in the "DISTANCE_NEAREST" vertex.

The constraint creation occurs while copying the transformation from the project constraints file directly to the graph. Neo4j database is presented in Fig. 26. In the "request" field (see element 5 in Fig. 26), the user can add transformed sentences, and a new constraint will be created as shown in Fig. 26, (element 2).

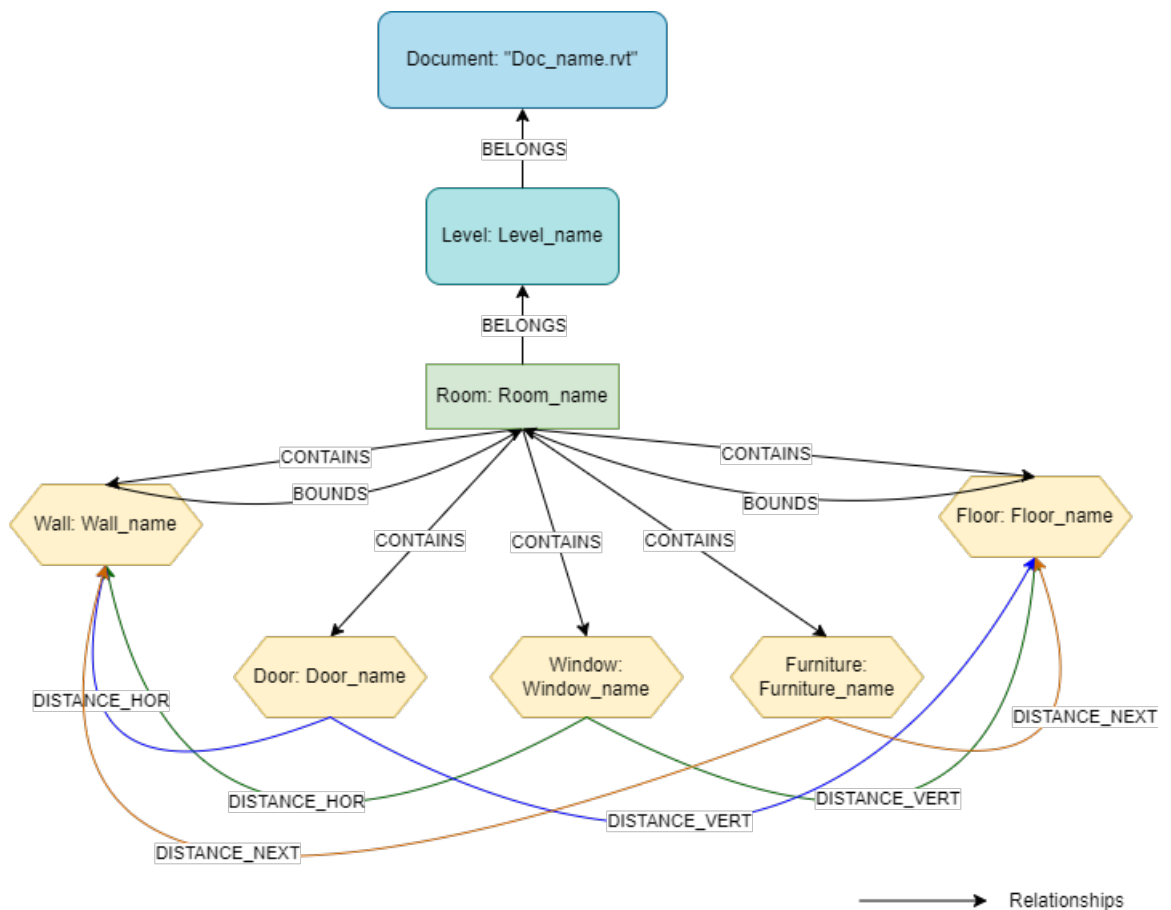


Figure 24: The elements structure from the model for representation in Neo4j graph. Arrows present different relationships between model components.

This section describes the identification and representation process of design constraints. These include analysis of geometrical data of the building model, calculation methods, and evaluation process, which is done using statistical methods and tools. The design constraint is presented as edges in a graph and can be specified and completed with properties. These are done within the automated tool (plugin) based on the Autodesk Revit 2022.

The plugin contains three panels: Collection, Constraints, and Graph. The Collection provides geometrical data calculation and aggregation. Constraints panel allows users to select potential constraints, specify them, and apply them to the list of project constraints. The Graph panel includes a graph representation of the building model, which automatically transfers building components and creates relationships within the graph. Finally, the design constraints are created by adding the transformed query to the graph.

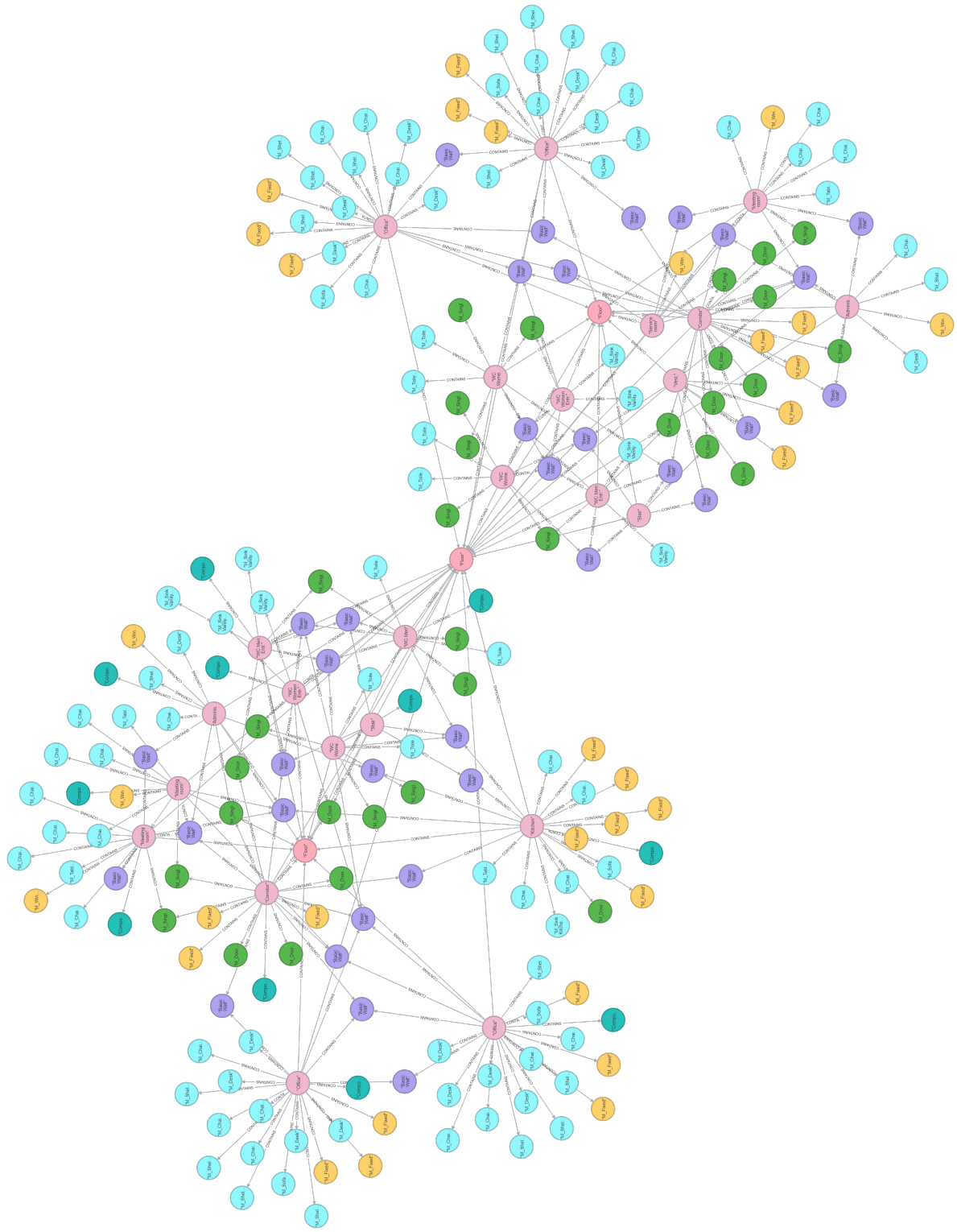


Figure 25: An example of node relationship between rooms and elements.

The screenshot displays the Neo4j browser interface. At the top, the navigation bar includes 'neo4j', 'Explore', 'Query', and 'Import'. The 'Database Information' section on the left shows 252 nodes categorized by type: Ceiling, Document, Door, Element, Floor, Furniture, Level, Room, Wall, and Window. Below this, 'Relationships (2,025)' are listed with types like BELONGS, BOUNDS, CONSTRAINT, CONTAINS, DISTANCE_HOR, DISTANCE_NEAREST, DISTANCE_NEXT, DISTANCE_PAR, DISTANCE_VERT, and PARALLEL. 'Property keys' are also listed, including category, constr_characteristics, and various distance-related properties.

The main area shows a Cypher query in the editor:


```
neo4j$ MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE n.room_name = "Office" AND m.category = 'Windows' SET m.constr_distance_horizontal_min= 0.361, m.constr_distance_horizontal_max='inf', m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.361,distance_hor_max:'inf', constraint_type: "conceptual"}]->(w) RETURN k,w,m
```

 Below the query, there are tabs for 'Graph', 'Table', and 'RAW'. The 'Graph' view shows a network of nodes and relationships. The 'Results overview' panel on the right shows:

- Nodes (18): Wall (6), Window (12)
- Relationships (24): CONSTRAINT (24)

 At the bottom, a status bar indicates 'Created 24 relationships, set 344 properties' and 'Last update: 11:38:32 AM'. Five numbered callouts (1-5) are placed over the interface:

- 1: Points to the 'Last update' timestamp.
- 2: Points to the 'Created 24 relationships, set 344 properties' message.
- 3: Points to the 'Started streaming 24 records after 319ms and completed after 533ms' message.
- 4: Points to the 'Results overview' panel.
- 5: Points to the 'Send feedback' button in the top right corner.

Figure 26: The Neo4j database, browser view. 1 - Database information, 2 - graph representation window, 3 - properties panel, 4 - field with current request, 5 - request field.

Chapter 5

Case study

The main objective of the case study is to validate the effectiveness and viability of the plugin described in Chapter 4. The building model will be analysed in detail, and corresponding algorithms, methods, and tools will be presented. Each group of supported building components will be given individually through the experiment. The proposed solution aims to fill the gap in the implementation and representation of design constraints using manual constraint handling and the automated process of constraint identification.

5.1 Experimental setup

Building model description

The experimental building model presented in Fig. 5.3 was created in Autodesk Revit 2022. The model is a two-floor office building in the workplace environment. On the ground floor are two office rooms, an administrative office, and a meeting room. The second level has the same structure as the ground floor: two office rooms, a collaborative area with three meeting rooms, and one administrative office. Additionally, the office building has a kitchen with a small wellness area for employees' well-being.

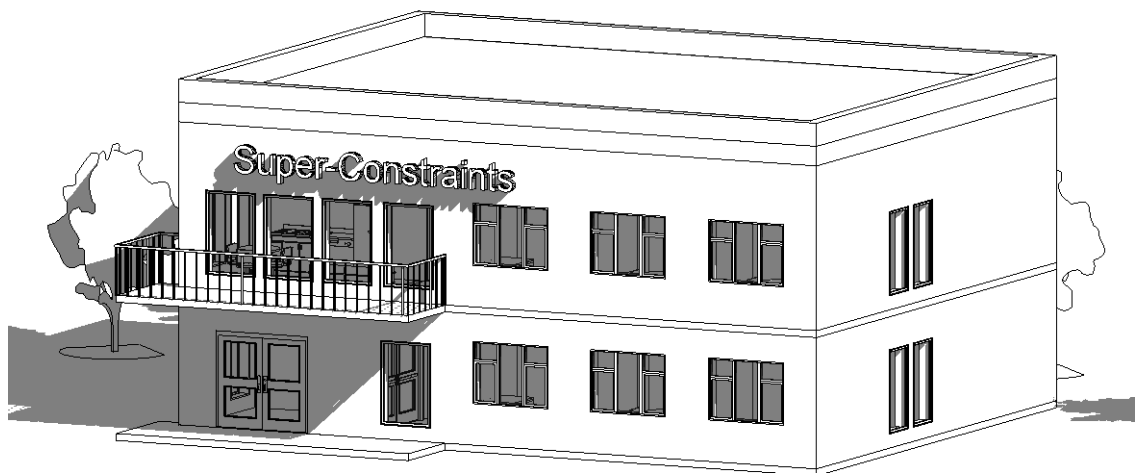


Figure 27: 3D representation of the experimental building model.

To use the plugin, several requirements are imposed on the model:

- Autodesk Revit version is 2022

- Follow the instruction to install the plugin¹
- The building model should have at least one room and one component
- Data collection takes place in the existing model and does not support collection from linked models
- Neo4j database should be created, logging data need to be placed in the [/ExportToNeo4jDirect.pushbutton/neo4j_key.txt](#)

All requirements are met by the case-study building.

5.2 Constraint identification and representation

The section describes the process of plugin application on a particular project. The process is divided into three blocks: calculation and data aggregation, constraint identification, and graph representation. Calculation and aggregation are done automatically and result in tables with calculating data. The design constraint identification (Figures 28 - 31) processes manually. Design constraint needs to be selected and specified manually. The result is saved into a text file in the form of natural language and Cypher query. The representation process is done partially within Revit. The building information, which includes supported categories and calculated geometrical information, is transferred automatically into a graph. After building the graph, the user can add design constraints manually.

Steps:

1. Select categories and mark furniture components. Default categories are marked with "Yes" in a table (see Fig. 10).
2. Start with the [CollectAll](#) button. The geometrical data will be processed only for several categories: Windows, Doors, Walls, Floors, and Furniture. After calculation, new tables are created and saved in the default destination folder [/super\ -constraints/data/tables](#).
3. Press the [ComputeTrends](#) button. The geometrical data is converted into the aggregated data
4. The [CreateConstraints](#) button allows one to select, modify, specify, and apply design constraints to a list of project constraints. The implementation steps are displayed in the Figures 28 - 31.
5. [ExportToNeo4jDirect](#) button creates new nodes and relationships from the computed model. The user has an option between deleting or updating the graph. The constraint creation comes manually by pasting the transformed sentence into the Neo4j request field, as shown in Fig. 32.

¹<https://github.com/Elvira2227/super-constraints>

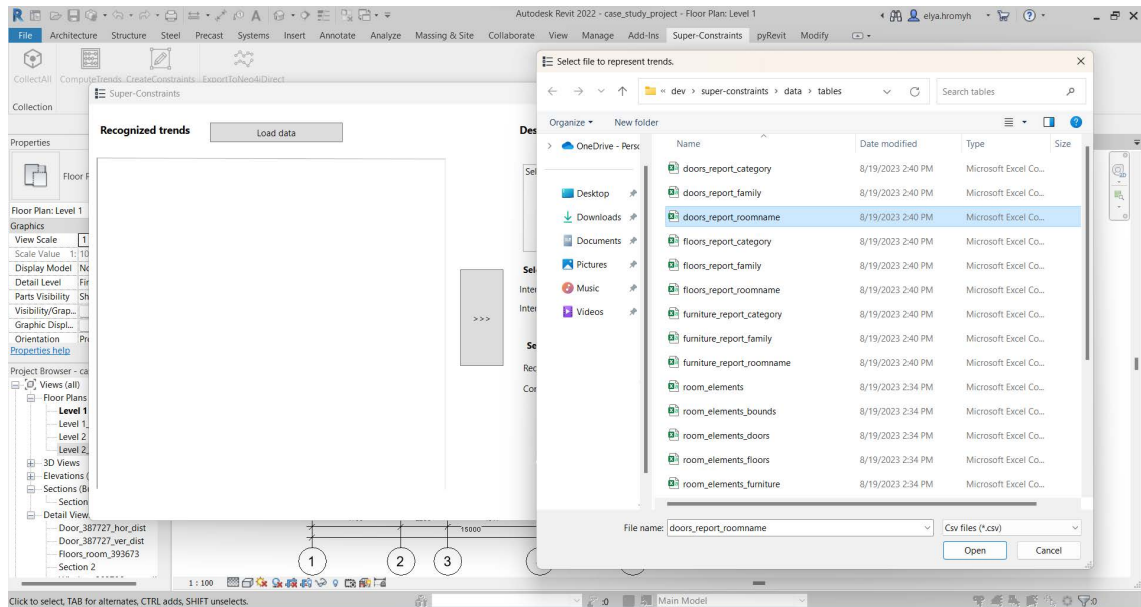


Figure 28: Selection one of the calculated tables to translate data into human language. The user must select one of the calculated in the ComputeTrends table to translate information into human-readable text. The target tables' name starts with an element category name such as "doors_", "windows_", etc.

5.3 Results and Interpretation

The design constraints were detected and represented in natural language, Cypher and graph forms in the case study. Several groups of items are tested, and for each group, different types of constraints are defined and presented as illustrated in appendix A. Throughout the validation, the geometrical information was aggregated and transformed into human-readable sentences, which allows us to alter or improve the constrained values, set the necessary attributes, such as the type and characteristics of a constraint, and apply said constraints to the list of project constraints. As a result, constraints with Cypher transformation are created. The queries are successfully integrated into a Neo4j graph and represented in a graph database.

The detailed evaluation of constraints will be described below. The assessment is done manually by comparing the dimensions of evaluated elements in Revit with the results from the calculation by finding minimum, maximum, and most common values. Below are the tests for each item group. Every group will be tested for particular types of constraints and room name, family name, or category (see sect. 4.2.2)

Windows

The constraints for Windows were tested for the following parameters:

- Distance to horizontal elements (walls)
- Distance to vertical elements (floors)
- Distance to the next window

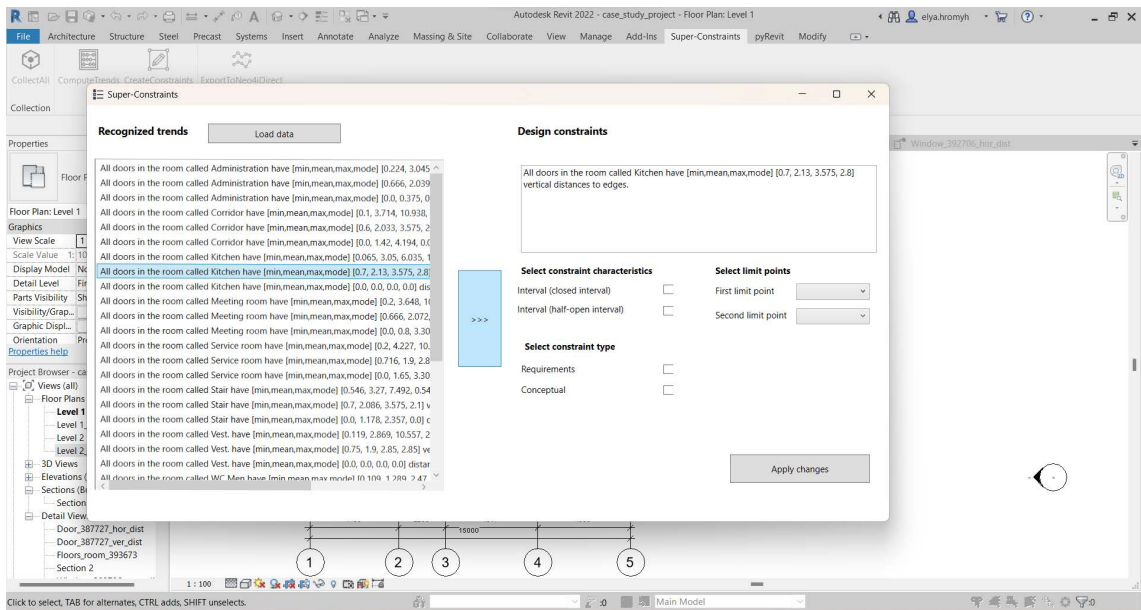


Figure 29: Selection one of the recognised trends and move into the design constraint section.

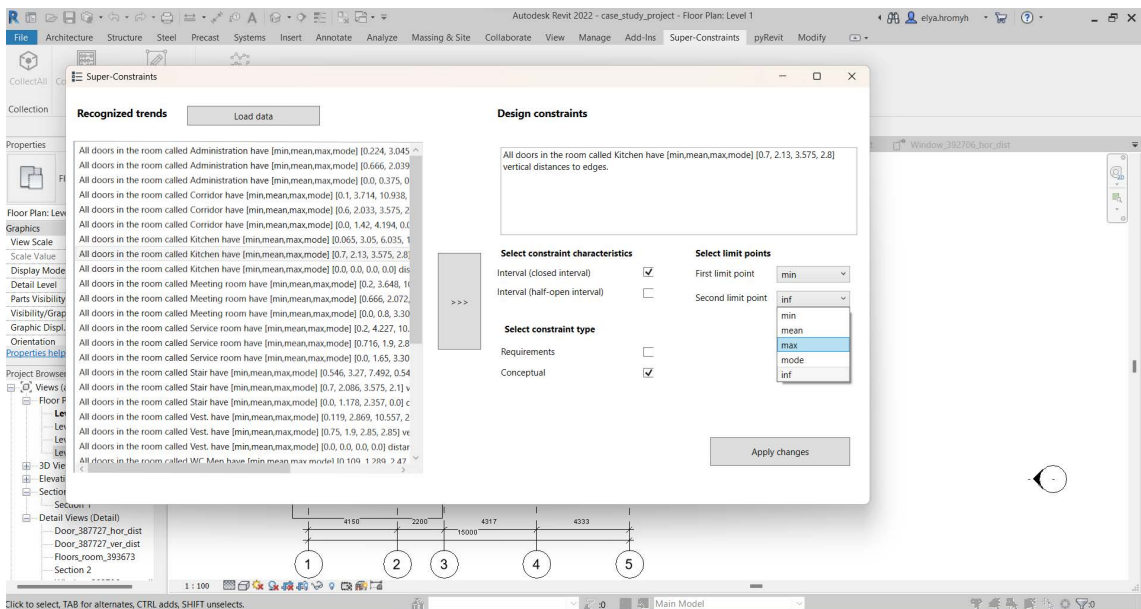


Figure 30: Design constraint specification. Design constraint needs to be specified by a user. It means that the user needs to assign interval limit points and type of constraint (requirement or conceptual). The user is allowed to edit the values if needed.

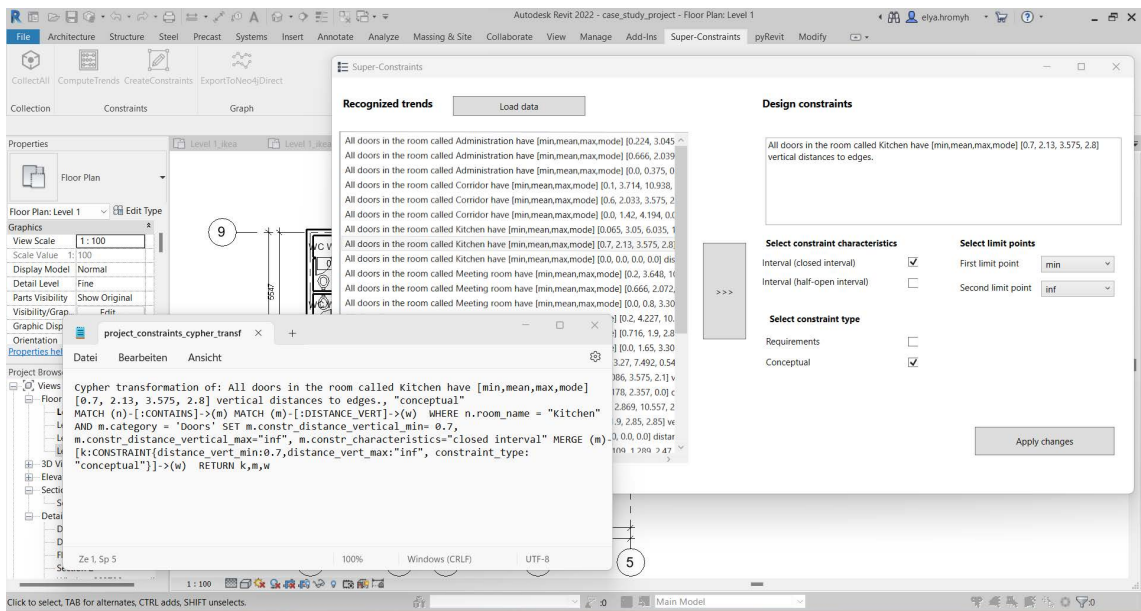


Figure 31: Design constraint is transformed into a Cypher language. After applying changes, a new design constraint is created and saved as a text file.

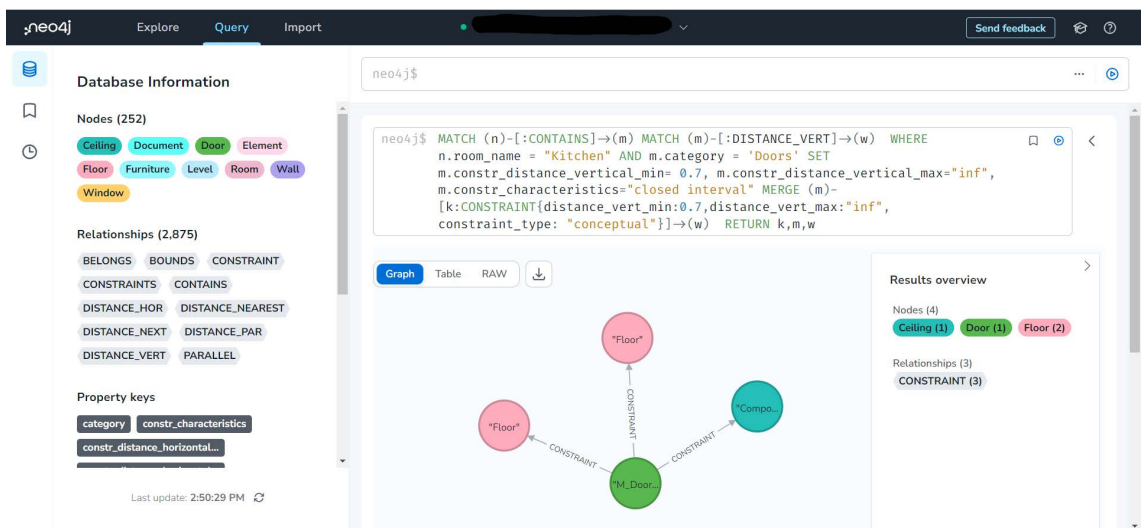


Figure 32: Design constraint representation in a graph database. The design constraint from the text file can be pasted directly into the request field of the Neo4j workspace.

The constraints are divided into:

- Constraints by room name (in this example - "Office")
- Constraints by category ("Windows")
- Constraints by family name ("M_Fixed")

The results are presented in the Appendix A. The validation of the constraints implementation will be described in detail only in the first and third cases. Note that the values for the windows were not improved or edited while applying constraints to the model.

The constraint identification and representation start with grouping all windows by the room name and selecting "Office." There are four rooms called "Office" in the building (two on level 1 (Figure 34) and two on level 2 (Fig. 35)). There are 12 windows in these rooms overall. According to the constraints description, the horizontal distances have a minimum of 0.361 m, a maximum of 3.745 m, and the most common value of 0.6 m; the vertical distances have a minimum of 0.305 m, a maximum of 3.27 m, and the most common value 0.305 m. The distances between neighbouring windows have minimum and maximum values of 0.2 m and the most common value of 0.2 m. The resulted constraints can be validated in Figures 34, 35, and 36.

In appendix A, one can find the identification and representation of the "Office" room windows constraints under the heading *"Windows: Constraints by rooms name (Office)"*. The visual table with both natural language and Cypher representation of the constraints can be found under the heading *"Windows: Constraints by rooms name (Office) - Identification"*, while the graphs are shown visually under the heading *"Windows: Constraints by rooms name (Office) – graph representation"*. An example is in Fig. 33.

In the graph representation for the horizontal distance (Fig. 33), there are 12 windows and six corresponding walls assigned. The windows and walls are divided into two separate graphs resulting from the relationships within the level (one graph corresponds to one level). The relationship is called "CONSTRAINT" and has minimum and maximum distances and types of constraint in their properties. The next constraint is created for the vertical distance parameter. Here, 12 windows with 3 floors and 2 compound ceilings are presented. On the last graph in this category, 12 windows are shown. The relationships display how the smallest distance to the room's next window combines windows.

The following example presents constraints grouped by window name "M_Fixed". There are 18 windows called "M_Fixed" (marked with a blue hexagon and number 18 in Figures 37,38,39) with dimensions 915x1830 mm and 8 windows called similarly with dimensions 1200x1830 mm (marked with blue hexagon and number 30 in Figures 37,38,39). Based on the constraint description, the horizontal distances have a minimum of 0.193, a maximum of 10.215 m, and the most common value of 0.6 m—the calculated values present distances between window edges and perpendicular wall surfaces. The dimensions are presented in Figures 37,38,39 and marked with blue ellipses.



Figure 33: The graph representation of design constraints which is created for horizontal distances between window and walls in the rooms called "Office".

Correspondingly, the identified constraint with natural language and Cypher transformation can be found in the appendix A under the heading "Windows: Constraint by family name (M_Fixed) - Identification". The visual representation of the constraints is under the heading "Windows: Constraint by family name (M_Fixed) - graph representation."

The graph representation of constraints based on horizontal distances has 26 windows called "M_Fixed" and 20 Walls. The number of relationships called "CONSTRAINT" is 59. It means one window can be constrained to more than one wall, depending on a building's geometry. The next graph presents 26 windows called "M_Fixed", three floors, and 4 compound ceilings. The graph is visually more complex than the previous one. One window is connected to at least two floors and can have connections with compound ceilings. The last graph represents distance constraints between the adjacent windows. There are 26 overall windows called "M_Fixed" in a project, but only 24 are presented in the graph. This stems from the building structure and calculation logic. Both windows on floor 1 (between axes 2-3, on axis 6) and floor 2 (between axes 2-3, on axis 9) do not have any neighbour windows in the wall they belong to.

Doors

The constraints for doors were tested for the following parameters:

- Distance to horizontal elements (walls)

Floor plan: Level 1

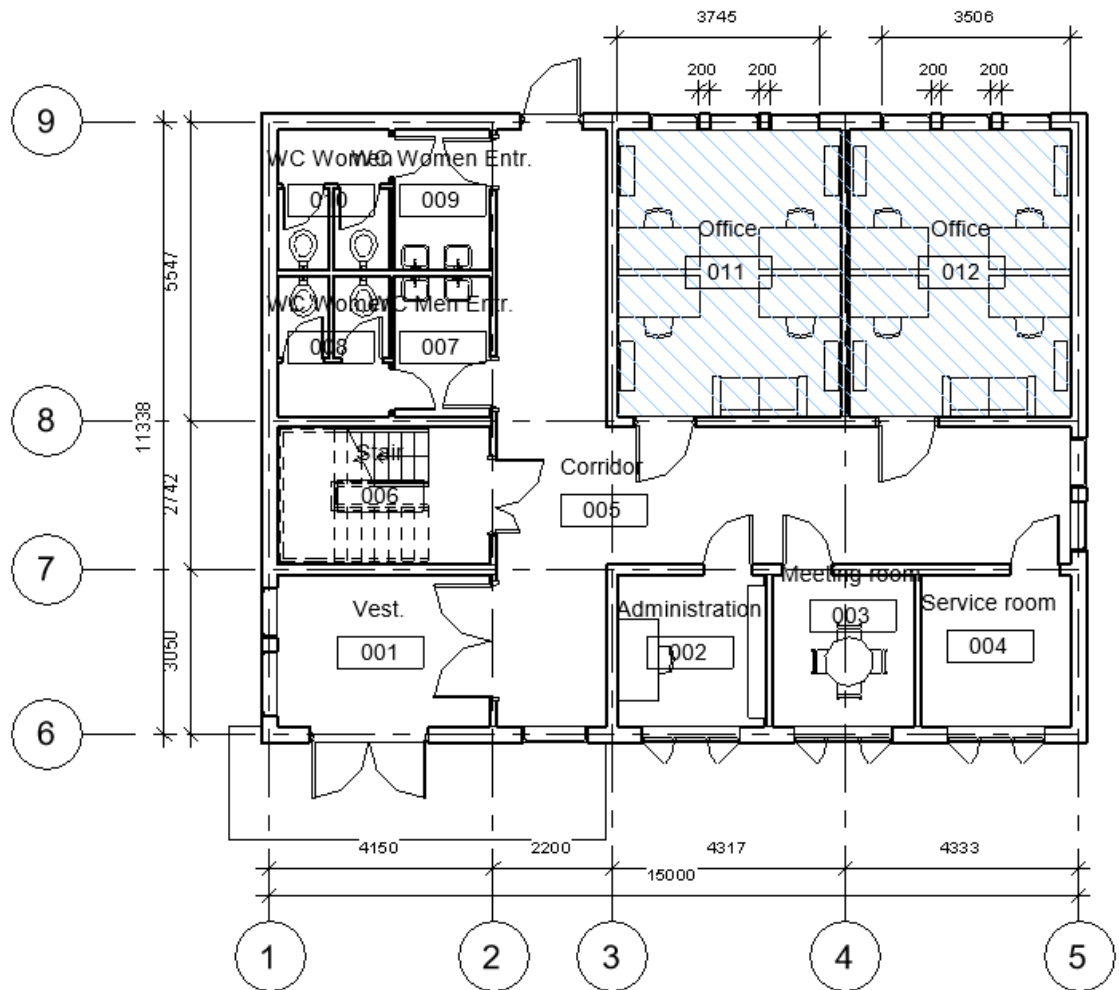


Figure 34: First level of the building. The office room is filled with a diagonal pattern. The corresponding distances are displayed. The Figure is not to scale.

Floor plan: Level 2

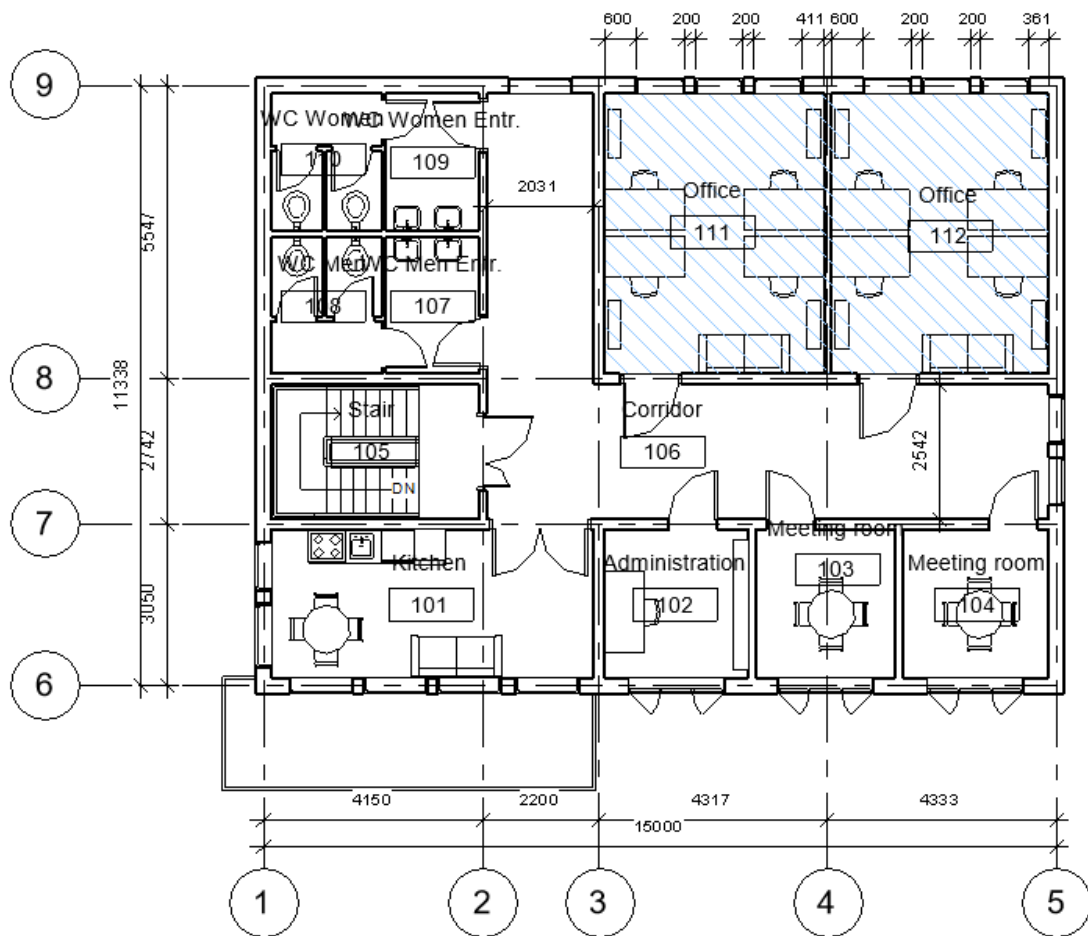


Figure 35: Second level of the building. The office room is filled with a diagonal pattern. The corresponding distances are displayed. The Figure is not to scale.

Detail 1-1

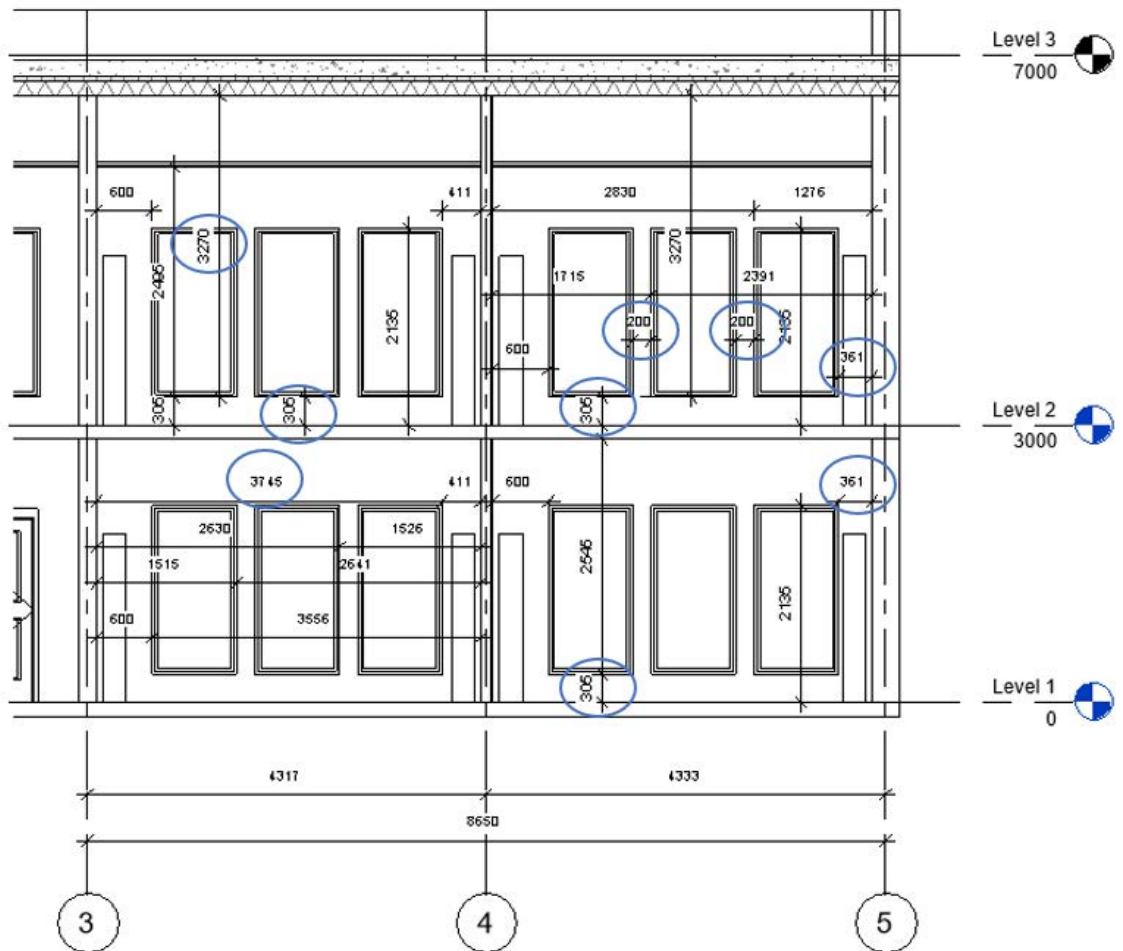


Figure 36: Detail 1 of the building. The matching distances are marked with blue ellipses. The Figure is not to scale.

Floor plan: Level 1

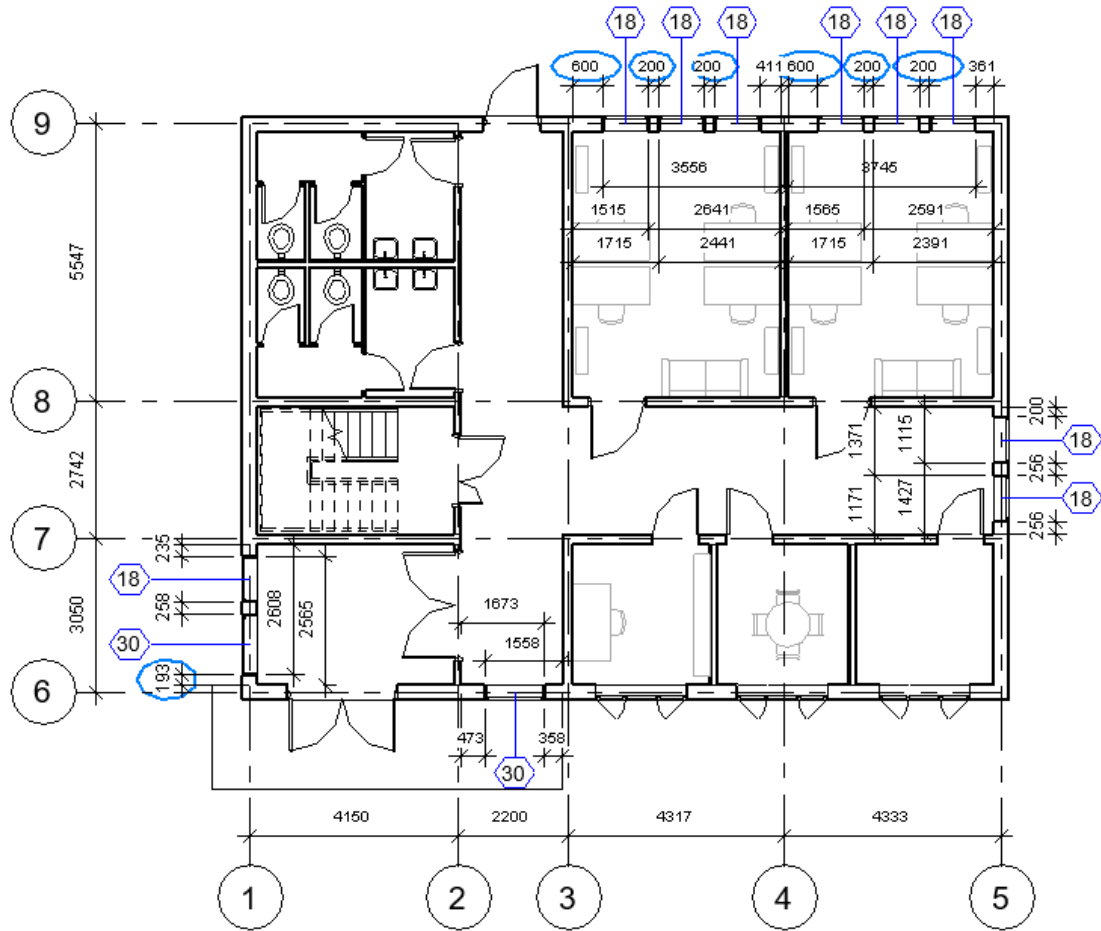


Figure 37: First level of the building. The windows "M_Fixed" are marked with a blue hexagon. Number 18 within the hexagon is the window with dimensions 915x1830 mm; number 30 is the window with dimensions 1200x1830. The matching dimensions are marked with blue ellipses. The Figure is not to scale.

Floor plan: Level 2

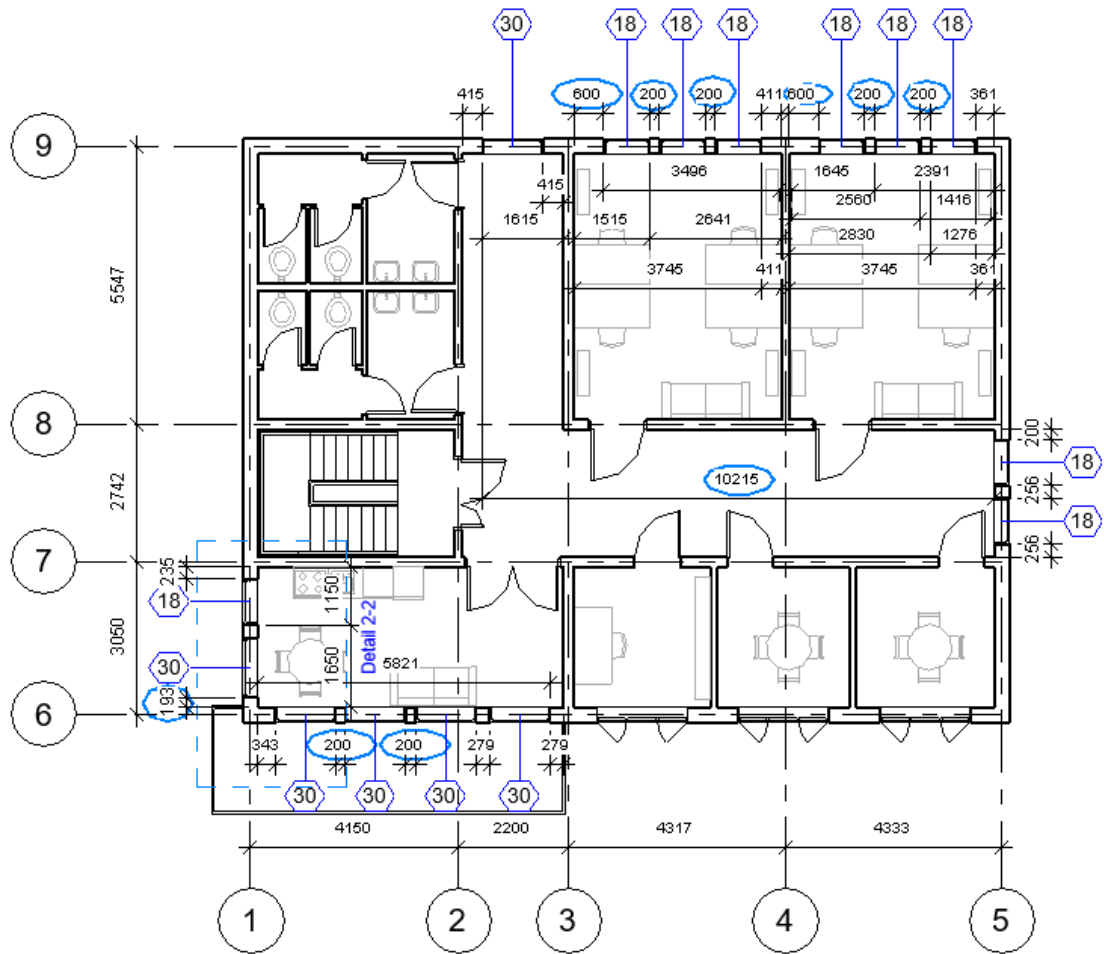


Figure 38: Second level of the building. The windows "M_Fixed" are marked with a blue hexagon. Number 18 within the hexagon is the window with dimensions 915x1830 mm; number 30 is the window with dimensions 1200x1830. The matching dimensions are marked with blue ellipses. The Figure is not to scale.

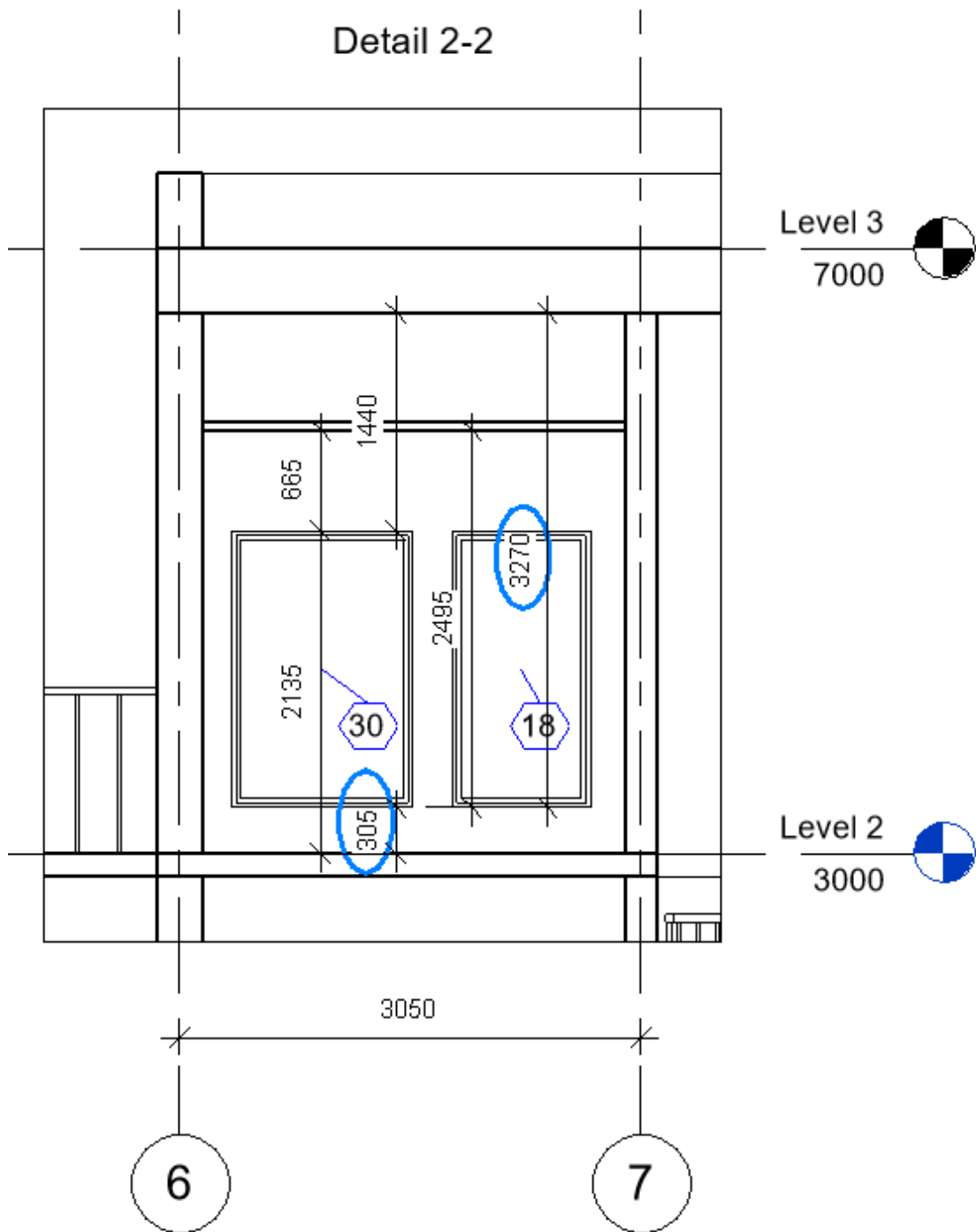


Figure 39: Detail 2-2 of the building. The windows "M_Fixed" are marked with a blue hexagon. Number 18 within the hexagon is the window with dimensions 915x1830 mm; number 30 is the window with dimensions 1200x1830. The matching dimensions are marked with blue ellipses. The Figure is not to scale.

- Distance to vertical elements (floors)
- Distance to the next door

The constraints are divided into:

- Constraints by room name ("Meeting room")
- Constraints by category ("Doors")
- Constraints by family name ("M_Window-Casement-Triple-Side-Transom")

Walls

The constraints for walls were tested for distances between parallel walls. The constraints are divided into:

- Constraints by room name ("Meeting room")
- Constraints by category ("Walls")
- Constraints by family name ("M_Window-Casement-Triple-Side-Transom")

Floors

The constraints for floors were tested for distances between parallel floors. The constraints are divided into:

- Constraints by room name ("Meeting room")
- Constraints by category ("Floors")
- Constraints by family name ("M_Window-Casement-Triple-Side-Transom")

Furniture

The constraints for floors were tested for distances to the nearest element (floor, wall). The constraints are divided into:

- Constraints by room name ("Meeting room")
- Constraints by category ("Furniture")
- Constraints by family name ("M_Window-Casement-Triple-Side-Transom")

The natural language, Cypher and graph representation of all the above constraints can be found in the appendix [A](#) under the appropriate heading. The validation results are similar to what was explained in the case of Windows and discussed below.

Several groups of items are tested, and for each group, different types of constraints are defined and presented as illustrated in appendix A. Throughout the validation, the geometrical information was aggregated and transformed into human-readable sentences, which allows us to alter or improve the constrained values, set the necessary attributes, such as the type and characteristics of a constraint, and apply said constraints to the list of project constraints. As a result, constraints with Cypher transformation are created. The queries are successfully integrated into a Neo4j graph and represented in a graph database.

The information from the graph can be used in several applications. For instance, the data obtained from the plugin's calculations can be a project version control. Once new objects are created, and the graph is updated, the information about current geometrical data can be gathered by making a simple request directly to a graph. The resulting graph can hint at how many objects match the project constraints and how many do not meet the requirements. Another application is to store the geometrical information in a graph format. The resulting graph representation and corresponding queries can be exported as table files for further research.

Chapter 6

Discussion

This thesis aims to identify and represent design constraints from the building model. As described in the Sect. 4.2, the automated solution enables the calculation and aggregation of geometrical information. It allows users to select and apply a recognised trend to the building graph model. A graph approach is utilised to represent the constraints. The graph enables not only to represent the relationships among building elements but also the constraints. In the case study, the automated tool was tested on the experimental office building model. The results of constraint identification and representation were presented for the particular type of constraints in detail and compared with the actual model dimensions.

This thesis, however, is a subject of three limitations. The primary limitations result in design constraint identification. Identifying constraints provides only for several geometrical constraints (distances, angles) and not for topological or parametric constraints. Some spatial constraints with semantic meaning, such as design ideas, cannot be collected and calculated with statistics and need more advanced algorithms.

Another limitation includes the restricted number of supported building components. The calculation is done only for specific categories, such as windows, doors, walls, floors, and furniture. For the windows and doors elements, design constraints are implemented for vertical and horizontal distances to the perpendicular object surfaces (floors and walls) and horizontal distances to the closest window/ door. The constraints for the walls are presented in the distance to the parallel wall and the angle between nonparallel walls. Constraints for the walls include distance between parallel floors and ceilings and nonparallel elements such as floor and sloping roofs. Finally, the constraint for the furniture takes into account only the distance between furniture and neighbouring elements such as walls, floors, and ceilings.

Considering the geometrical data analysis, the limitation lies in the scarce functionality of the applied statistical description. This thesis uses only min, mean, max, and mode functions. For more complicated project tasks, the method may include other statistical measurements, such as standard deviation, standard error, etc. In some cases, the constraints will be none of the abovementioned functions.

The automated solution provides the geometrical data collection directly from the Revit authoring tool. Data analysis is done using statistical methods and tools. The analysis does not need complex calculations, algorithms, or datasets (unlike Machine Learning techniques). The resulting output provides necessary information about the dimensions of the building component. The calculated data can be used as a basis for the machine Learning (ML) data sets and Natural Language (NLP).

The design constraint identification is done automatically. The user decides the identified design constraints and selects the appropriate ones to apply to the project constraints. The design constraint can be specified and limited by the provided values. The list with the project constraints can be used during the project. It allows one to manage changes, get an overview of the building's geometrical information, and control defined constraints.

In this thesis, design constraint representation is presented in various ways. First, the representation with a deterministic parser was done. The parser translates aggregated data into human-readable text, which makes information more understandable for the user (architect). Second, the representation of constraint in the form of query language to translate it into a graph database (DB). Finally, design constraint representation with a graph. Representation of design constraints with a graph provides visual information about connected building components. Design constraints in a graph are relationships with properties. Searching, filtering, and querying data in a graph is very effective. Based on the graph representation, the method opens a way to use graph databases in various ways, such as storing, updating, rewriting, and managing building information efficiently.

Chapter 7

Conclusion & outlook

The building industry has focused on transferring building projects into digital form. The building model can contain numerous details and data. The 3D representation allows architects and engineers to better understand the semantics of objects and the relationships between building components. However, the solution for managing those relationships and providing an overview of the designer's ideas is currently not included in the ordinary Building Information Modelling (BIM) authoring tools as a standard solution. The thesis introduces an identification and representation of design constraints in a building model. The approach analyses building geometry and translates this information in a human-readable way. The translated text can be manually selected and defined as a design constraint. A design constraint is translated into a query language ready to be applied to the graph database (DB). The graph approach enables efficient design constraint representation and specification. Due to the potential problems in data quality and consistency, the proposed tool is an appropriate solution to collect geometrical information and summarise it in the form of tables, reports, and graph representation.

The proposed automated solution (plugin) is tested for different design constraints in the case study. The approach can handle multiple elements and constraints from the model and select one by one to apply in a graph. The results are successfully integrated into a graph DB and represented in the Appendix A. However, due to several limitations discussed in Chapter 6, the automated tool supports only a limited number of elements and design constraints. Currently, the tool does not provide semantic information from the design and is limited to geometrical information.

The automated tool can successfully work with complex data structures of building models and identify relationships between main building components (e.g., walls, windows, floors). With the ability to translate the building data to the graph database, the user can overview the building structure and add design constraints to a graph. The solution opens a space for further research. The main topics for future research are:

- Application of design constraints to the model. This allows one to improve the model according to the selected design constraints.
- Using ML techniques to capture semantic information automatically. ML techniques retrieve design ideas more accurately.

Appendix A

Design constraints implementation and representation

This appendix shows the graph, natural language and Cypher representation of constraints found during the case study.

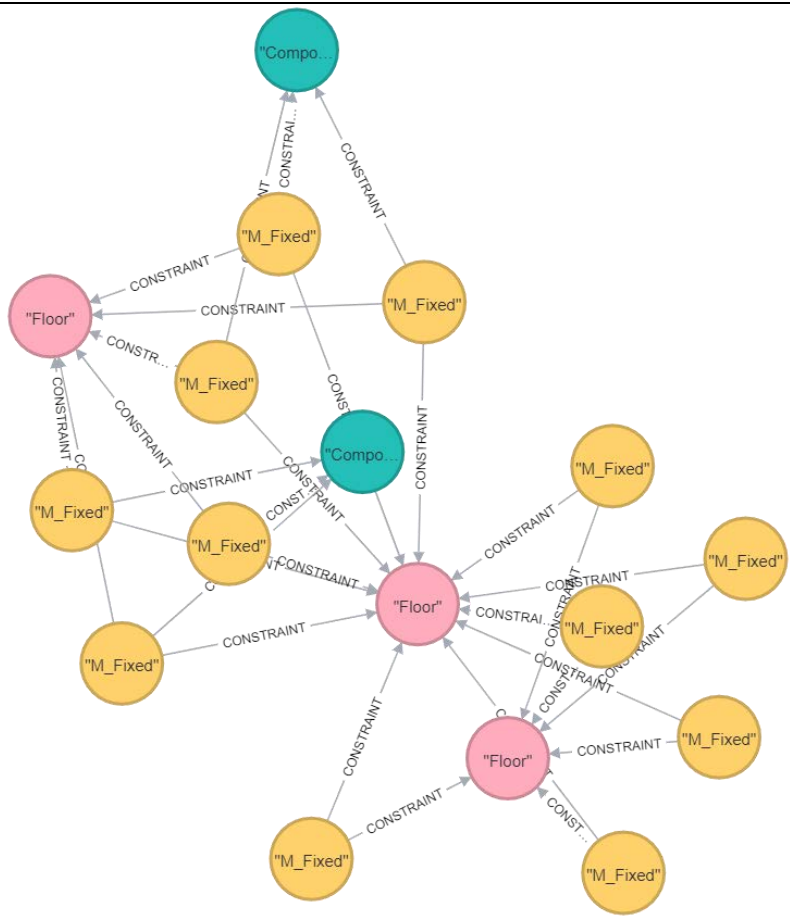
Windows: Constraints by rooms name (Office) - implementation

Constraint	Cypher query
All windows in the room called Office have [min, mean, max, mode] [0.361, 2.065, 3.745, 0.6] horizontal distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE n.room_name = "Office" AND m.category = 'Windows' SET m.constr_distance_horizontal_min= 0.361, m.constr_distance_horizontal_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.361,distance_hor_max:"inf",constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
All windows in the room called Office have [min, mean, max, mode] [0.305, 1.601, 3.27, 0.305] vertical distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE n.room_name = "Office" AND m.category = 'Windows' SET m.constr_distance_vertical_min= 0.305, m.constr_distance_vertical_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.305,distance_vert_max:"inf",constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
All windows in the room called Office have [min, mean, max, mode] [0.2, 0.2, 0.2, 0.2] distance to next windows., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE n.room_name = "Office" AND m.category = 'Windows' SET m.constr_distance_next_min= 0.2, m.constr_distance_next_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_next_min:0.2,distance_next_max:"inf",constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

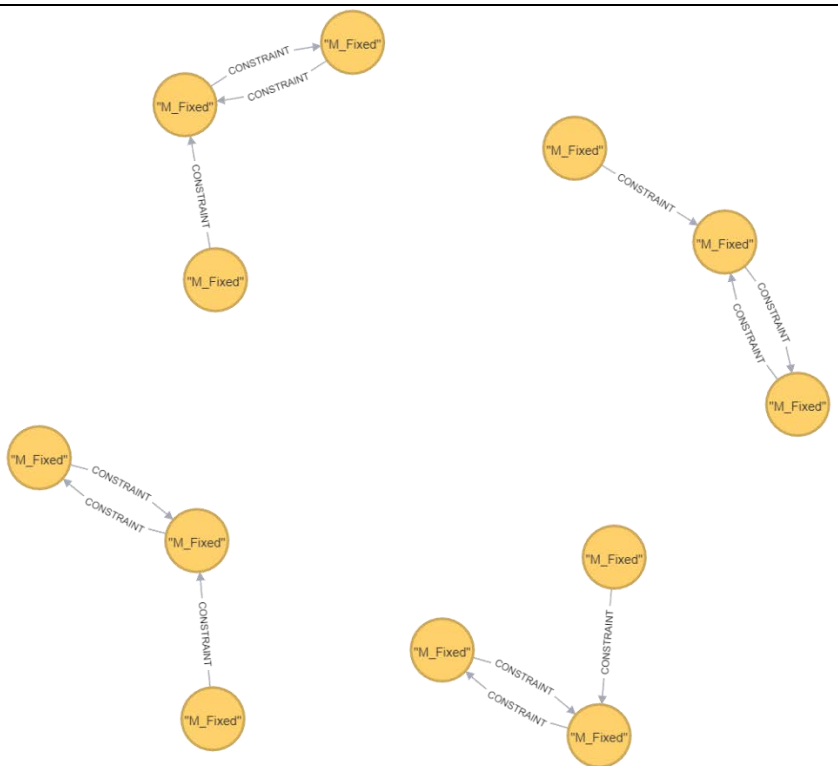
Windows: Constraints by rooms name (Office) – graph representation

Constraint	Property graph
All windows in the room called Office have [min, mean, max, mode] [0.361, 2.065, 3.745, 0.6] horizontal distances to edges., "conceptual"	

All windows in the room called Office have [min, mean, max, mode] [0.305, 1.601, 3.27, 0.305] vertical distances to edges., "conceptual"



All windows in the room called Office have [min, mean, max, mode] [0.2, 0.2, 0.2, 0.2] distance to next windows., "conceptual"



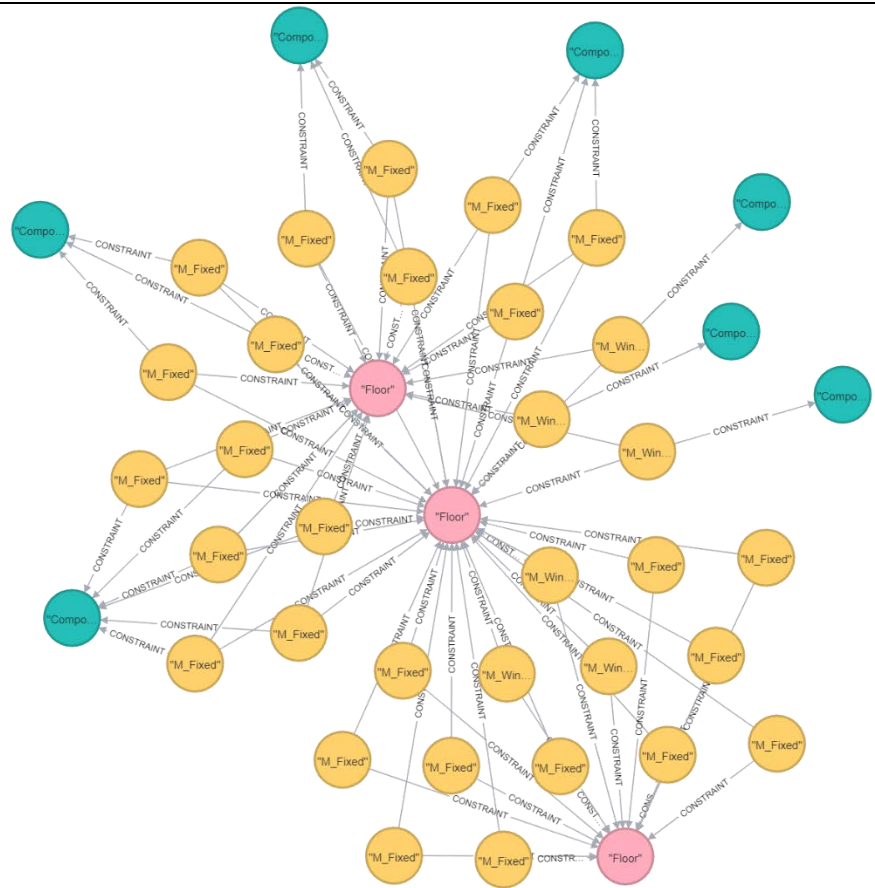
Windows: Constraints by category (Windows) - implementation

Constraint	Cypher query
All windows in the category Windows have [min,mean,max,mode] [0.193, 2.481, 10.215, 0.6] horizontal distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE m.category = 'Windows' SET m.constr_distance_horizontal_min= 0.193, m.constr_distance_horizontal_max=10.215, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.193,distance_hor_max:10.215, constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
All windows in the category Windows have [min,mean,max,mode] [0.305, 1.605, 3.27, 0.305] vertical distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE m.category = 'Windows' SET m.constr_distance_vertical_min= 0.305, m.constr_distance_vertical_max=3.27, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.305,distance_vert_max:3.27, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
All windows in the category Windows have [min,mean,max,mode] [0.0, 0.499, 1.687, 0.2] distance to next windows., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE m.category = 'Windows' SET m.constr_distance_next_min= 0.2, m.constr_distance_next_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_next_min:0.2,distance_next_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

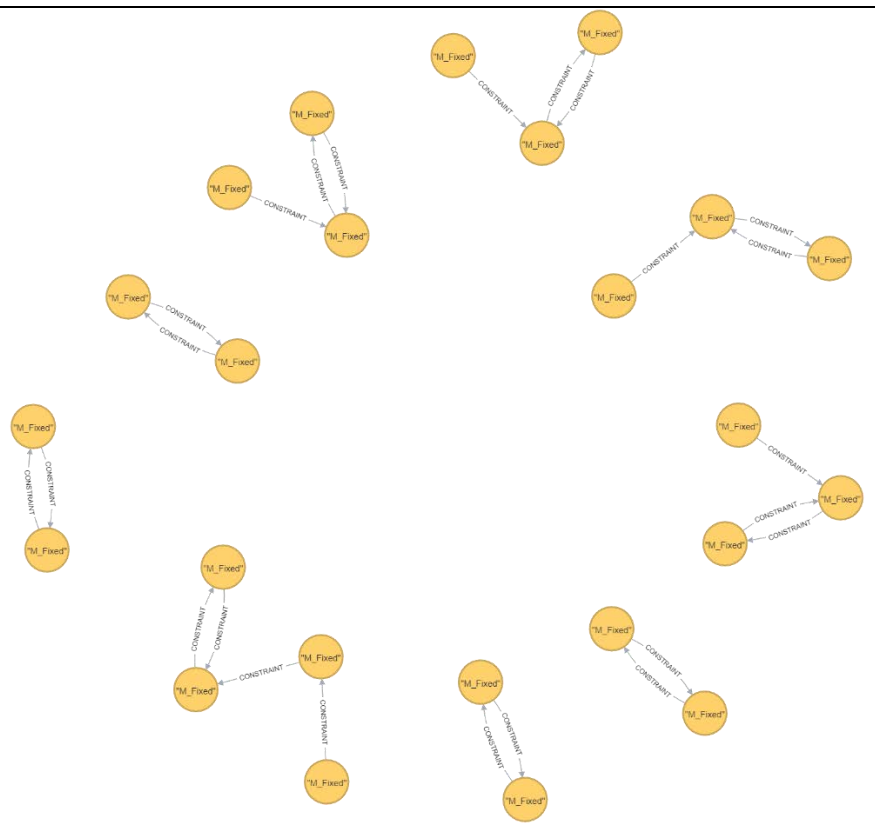
Windows: Constraints by category (Windows) – graph representation

Constraint	Property graph
All windows in the category Windows have [min,mean,max,mode] [0.193, 2.481, 10.215, 0.6] horizontal distances to edges., "conceptual"	

All windows in the category Windows have [min,mean,max,mode] [0.305, 1.605, 3.27, 0.305] vertical distances to edges., "conceptual"



All windows in the category Windows have [min,mean,max,mode] [0.0, 0.499, 1.687, 0.2] distance to next windows., "conceptual"



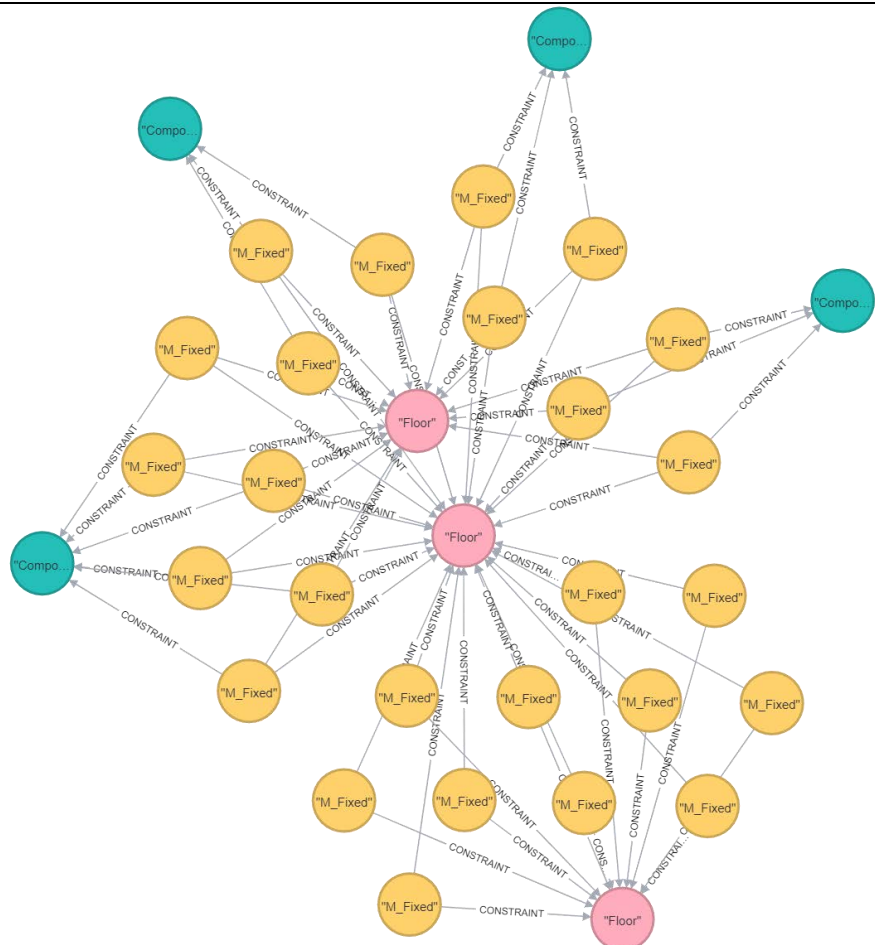
Windows: Constraints by family name (M_Fixed) - implementation

Constraint	Cypher query
All windows of the family M_Fixed have [min,mean,max,mode] [0.193, 2.595, 10.215, 0.6] horizontal distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE m.family_name = "M_Fixed" SET m.constr_distance_horizontal_min= 0.193, m.constr_distance_horizontal_max = "inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.193,distance_hor_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
All windows of the family M_Fixed have [min,mean,max,mode] [0.305, 1.622, 3.27, 0.305] vertical distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE m.family_name = "M_Fixed" SET m.constr_distance_vertical_min= 0.305, m.constr_distance_vertical_max=3.27, m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.305,distance_vert_max:3.27, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
All windows of the family M_Fixed have [min,mean,max,mode] [0.0, 0.614, 1.687, 0.2] distance to next windows., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE m.family_name = "M_Fixed" SET m.constr_distance_next_min= 0.2, m.constr_distance_next_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_next_min:0.2,distance_next_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

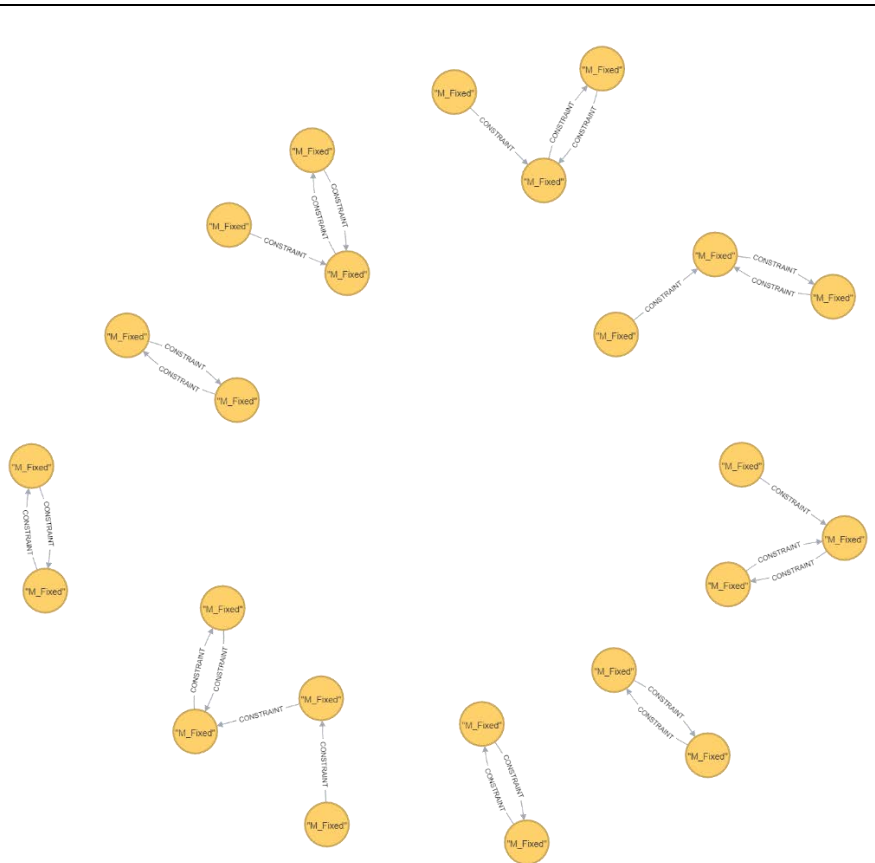
Windows: Constraints by category (M_Fixed) – graph representation

Constraint	Property graph
All windows of the family M_Fixed have [min,mean,max,mode] [0.0, 0.614, 1.687, 0.2] distance to next windows., "conceptual"	

All windows of the family *M_Fixed* have [min,mean,max,mode] [0.193, 2.595, 10.215, 0.6] horizontal distances to edges., "conceptual"



All windows of the family *M_Fixed* have [min,mean,max,mode] [0.0, 0.614, 1.687, 0.2] distance to next windows., "conceptual"



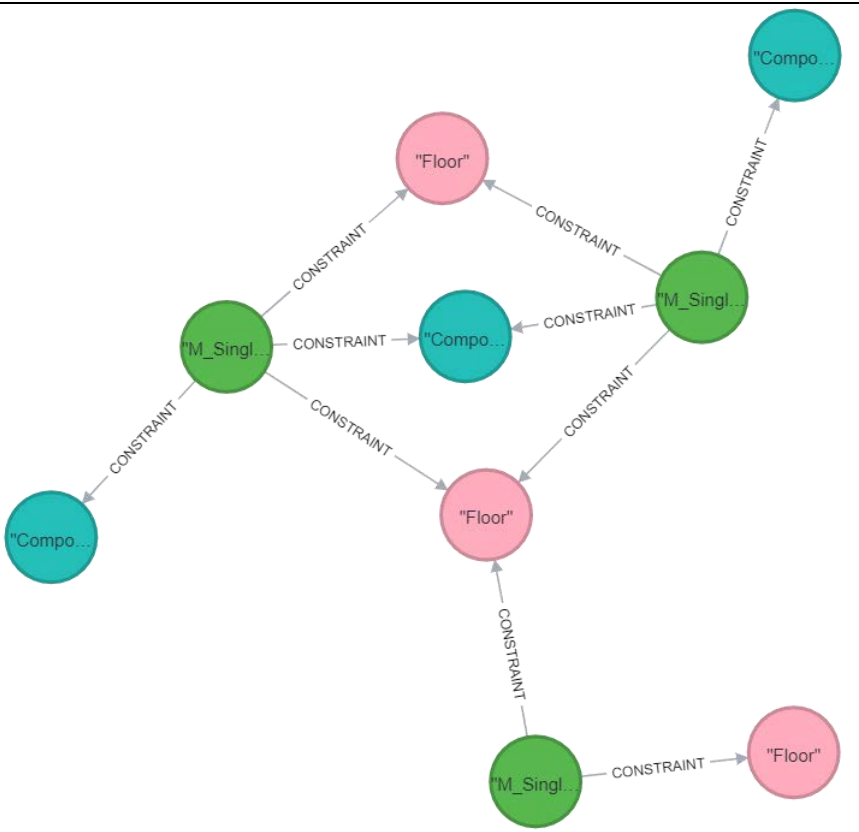
Doors: Constraints by room name (Meeting room) - implementation

Constraint	Cypher query
<p>All doors in the room called Meeting room have</p> <p>[min,mean,max,mode] [0.2, 3.648, 10.431, 0.2] horizontal distances to edges., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE n.room_name = "Meeting room" AND m.category = 'Doors' SET m.constr_distance_horizontal_min=0.2, m.constr_distance_horizontal_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.2,distance_hor_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
<p>All doors in the room called Meeting room have</p> <p>[min,mean,max,mode] [0.666, 2.072, 3.575, 2.134] vertical distances to edges., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE n.room_name = "Meeting room" AND m.category = 'Doors' SET m.constr_distance_vertical_min= 0.666, m.constr_distance_vertical_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.666,distance_vert_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
<p>All doors in the room called Meeting room have</p> <p>[min,mean,max,mode] [0.0, 0.8, 3.301, 0.0] distance to next doors., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE n.room_name = "Meeting room" AND m.category = 'Doors' SET m.constr_distance_next_min= 0.8, m.constr_distance_next_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_next_max:"inf",distance_next_min:0.8, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

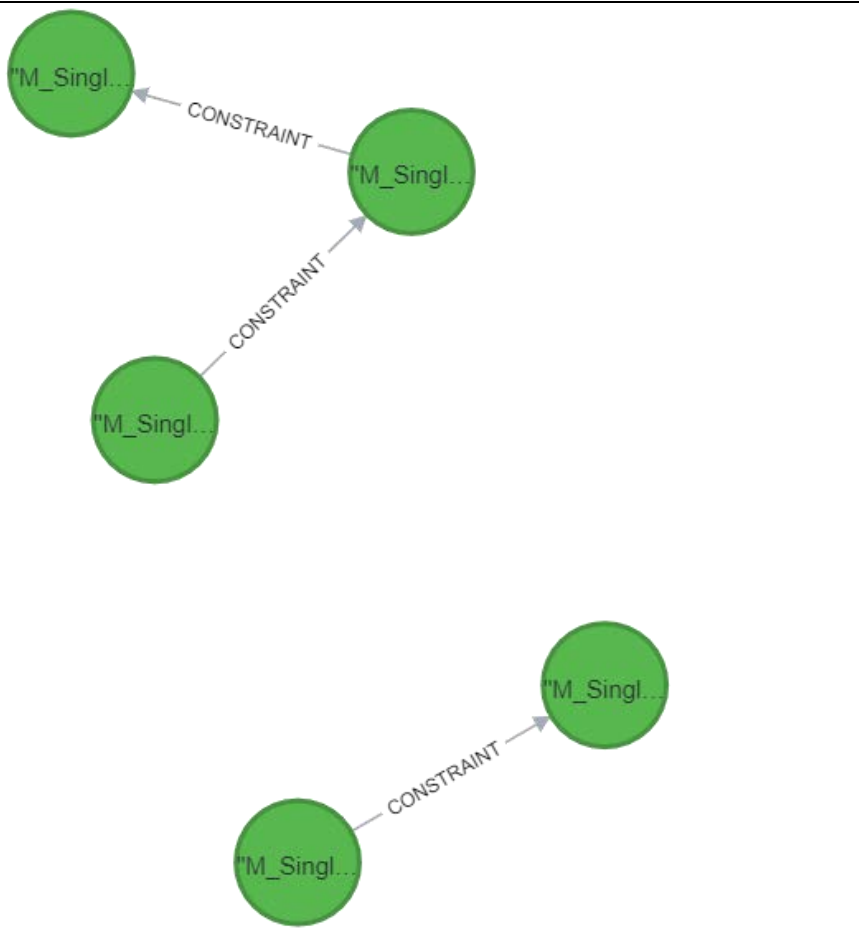
Doors: Constraints by room name (Meeting room) – graph representation

Constraint	Property graph
<p>All doors in the room called Meeting room have</p> <p>[min,mean,max,mode] [0.2, 3.648, 10.431, 0.2] horizontal distances to edges., "conceptual"</p>	

All doors in the room called Meeting room have
 [min,mean,max,mode]
 [0.666, 2.072, 3.575, 2.134] vertical distances to edges., "conceptual"



All doors in the room called Meeting room have
 [min,mean,max,mode]
 [0.0, 0.8, 3.301, 0.0] distance to next doors., "conceptual"



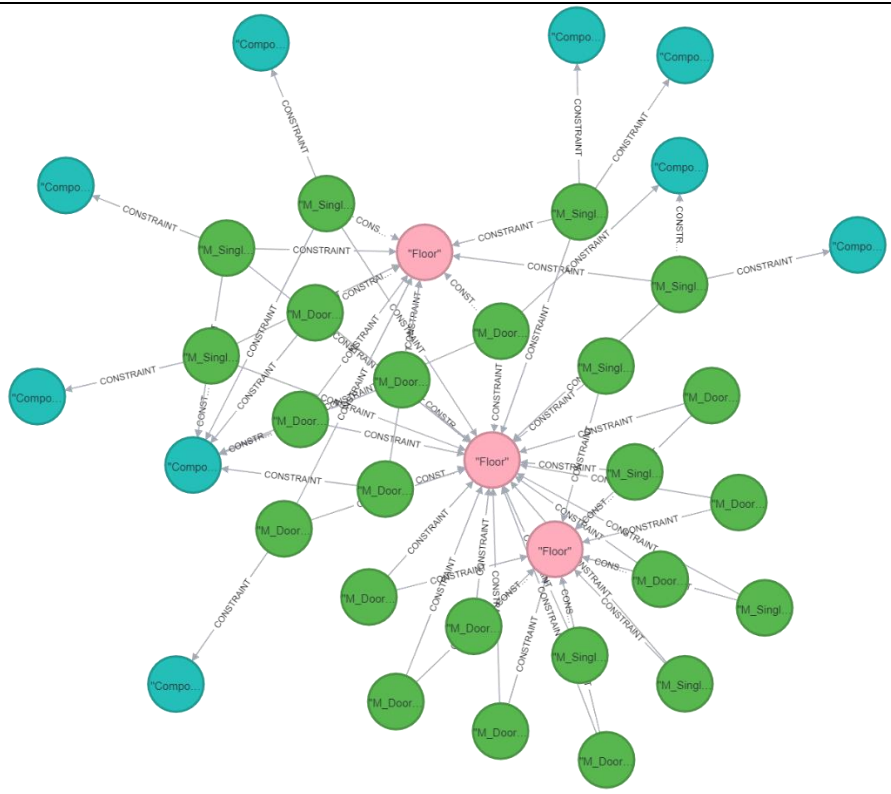
Doors: Constraints by category (Doors) - implementation

Constraint	Cypher query
All doors in the category Doors have [min,mean,max,mode] [0.065, 3.255, 10.938, 0.1] horizontal distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE m.category = 'Doors' SET m.constr_distance_horizontal_min= 0.065, m.constr_distance_horizontal_max=10.938, m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.065,distance_hor_max:10.938, constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
All doors in the category Doors have [min,mean,max,mode] [0.6, 2.038, 3.575, 2.85] vertical distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE m.category = 'Doors' SET m.constr_distance_vertical_min= 0.6, m.constr_distance_vertical_max="inf", m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.6,distance_vert_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
All doors in the category Doors have [min,mean,max,mode] [0.0, 1.056, 4.194, 0.0] distance to next doors., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE m.category = 'Doors' SET m.constr_distance_next_min= 0.0, m.constr_distance_next_max=4.194, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_next_min:0.0,distance_next_max:4.194, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

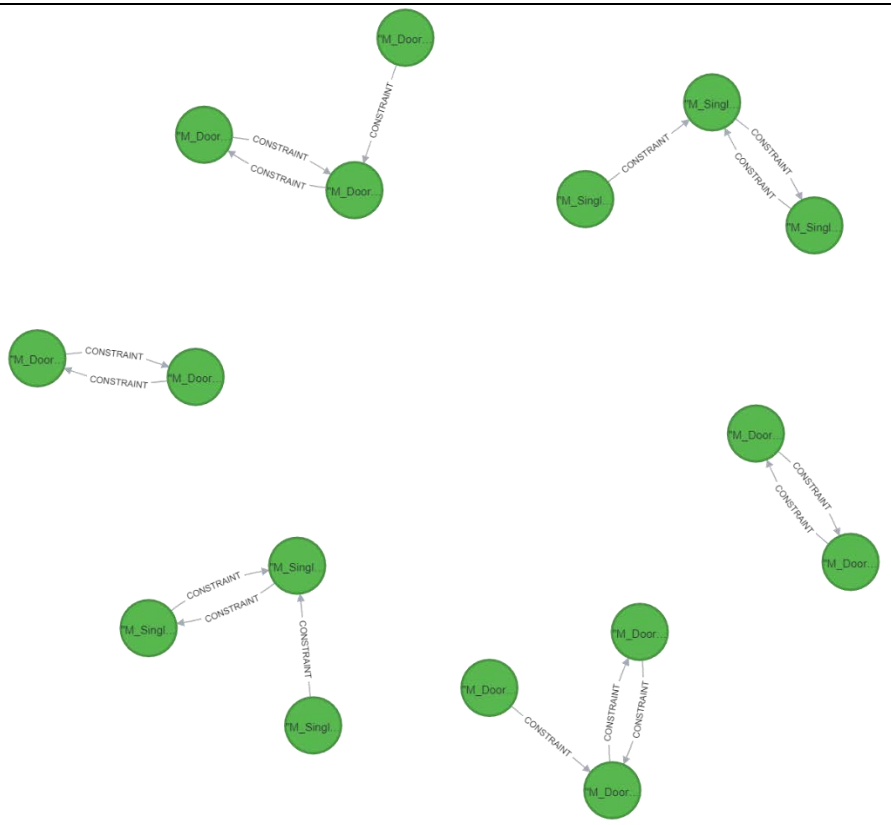
Doors: Constraints by category (Doors) – graph representation

Constraint	Property graph
All doors in the category Doors have [min,mean,max] [0.065, 2.88425641025641, 10.938] horizontal distances to edges., "conceptual"	

All doors in the category Doors have [min,mean,max] [0.6, 2.011461538461539 , 3.575] vertical distances to edges., "conceptual"



All doors in the category Doors have [min,mean,max] [0.3, 1.055641025641026 , 4.194] distance to next door., "conceptual"



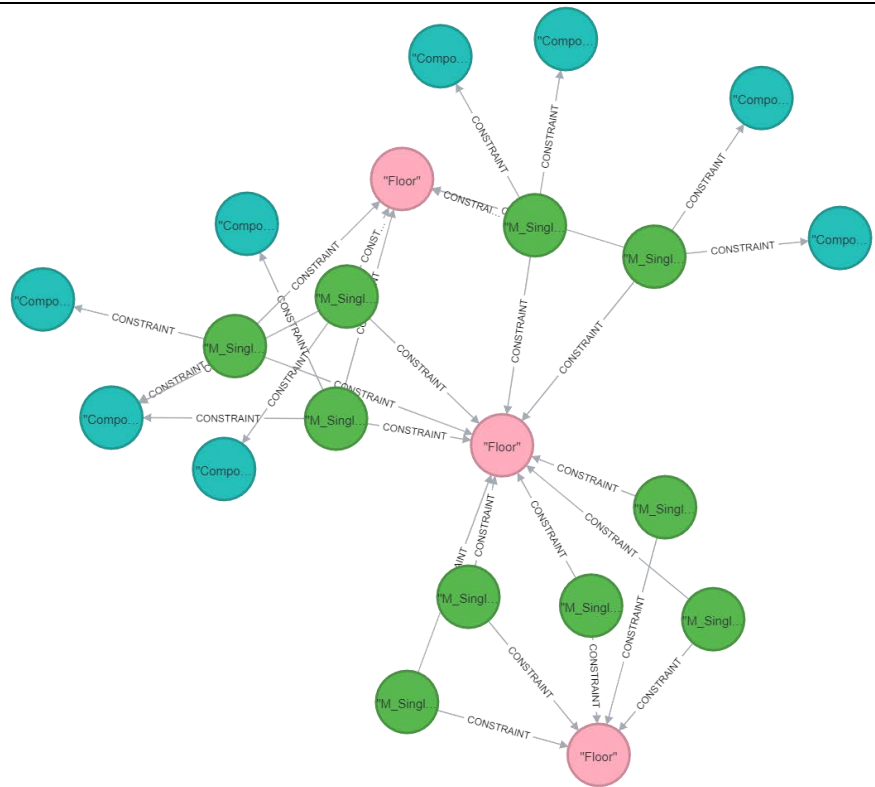
Doors: Constraints by family name (M_Single-Flush) - implementation

Constraint	Cypher query
All doors of the family M_Single-Flush have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] horizontal distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_HOR]->(w) WHERE m.family_name = "M_Single-Flush" SET m.constr_distance_horizontal_min= 0.109, m.constr_distance_horizontal_max=10.431, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_hor_min:0.109,distance_hor_max:10.431, constraint_type: "conceptual"}]->(w) RETURN k,w,m</pre>
All doors of the family M_Single-Flush have [min,mean,max,mode] [0.666, 2.045, 3.575, 2.134] vertical distances to edges., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_VERT]->(w) WHERE m.family_name = "M_Single-Flush" SET m.constr_distance_vertical_min= 0.666, m.constr_distance_vertical_max=3.575, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_vert_min:0.666,distance_vert_max:3.575, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>
All doors of the family M_Single-Flush have [min,mean,max,mode] [0.0, 0.48, 3.301, 0.0] distance to next doors., "conceptual"	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEXT]->(w) WHERE m.family_name = "M_Single-Flush" SET m.constr_distance_next_min= 0.0, m.constr_distance_next_max=3.301, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_next_min:0.0,distance_next_max:3.301, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

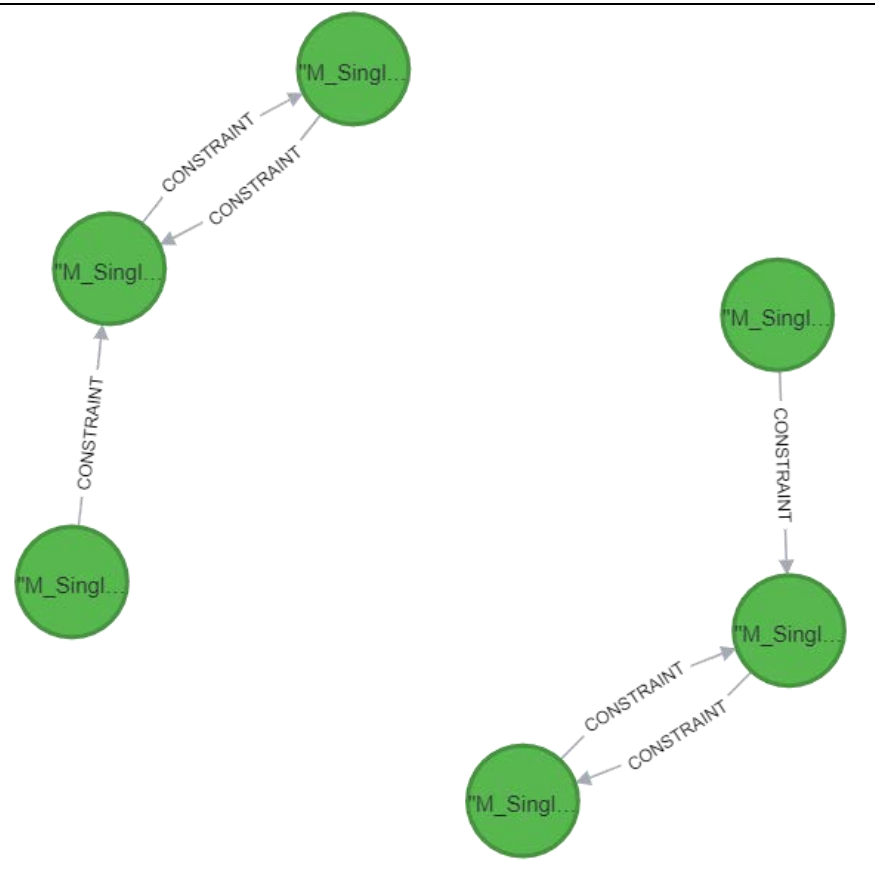
Doors: Constraints by family name (M_Single-Flush) – graph representation

Constraint	Property graph
All doors of the family M_Single-Flush have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] horizontal distances to edges., "conceptual"	<p>The graph shows nodes for 'Basic Wall' (purple) and 'M_Sing' (green). Edges are labeled 'CONSTRAINT' and 'INFLUENCED'. The graph illustrates the relationships between these nodes based on the horizontal distance constraint.</p>

All doors of the family *M_Single-Flush* have $[min, mean, max, mode]$ $[0.666, 2.045, 3.575, 2.134]$ vertical distances to edges., "conceptual"



All doors of the family *M_Single-Flush* have $[min, mean, max, mode]$ $[0.0, 0.48, 3.301, 0.0]$ distance to next doors., "conceptual"



Walls: Constraints by room name (Corridor) - implementation

Constraint	Cypher query
<p>All walls in the room called Corridor have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE n.room_name = "Corridor" AND m.category = 'Walls' SET m.constr_distance_parall_min= 0.109, m.constr_distance_parall_max=10.431, m.constr_characteristics="half-open interval" MERGE (m)-[:CONSTRAINT{distance_parall_min:0.109,distance_parall_max:10.431, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

Walls: Constraints by room name (Corridor) -graph representation

Constraint	Cypher query
<p>All walls in the room called Corridor have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	

Walls: Constraints by category (Walls) - implementation

Constraint	Cypher query
<p>All walls in the category Walls have</p> <p>[min,mean,max,mode]</p> <p>[0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE m.category = 'Walls' SET m.constr_distance_parallel_min= 0.109, m.constr_distance_parallel_max=10.431, m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_parallel_min:0.109,distance_parallel_max:10.431, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

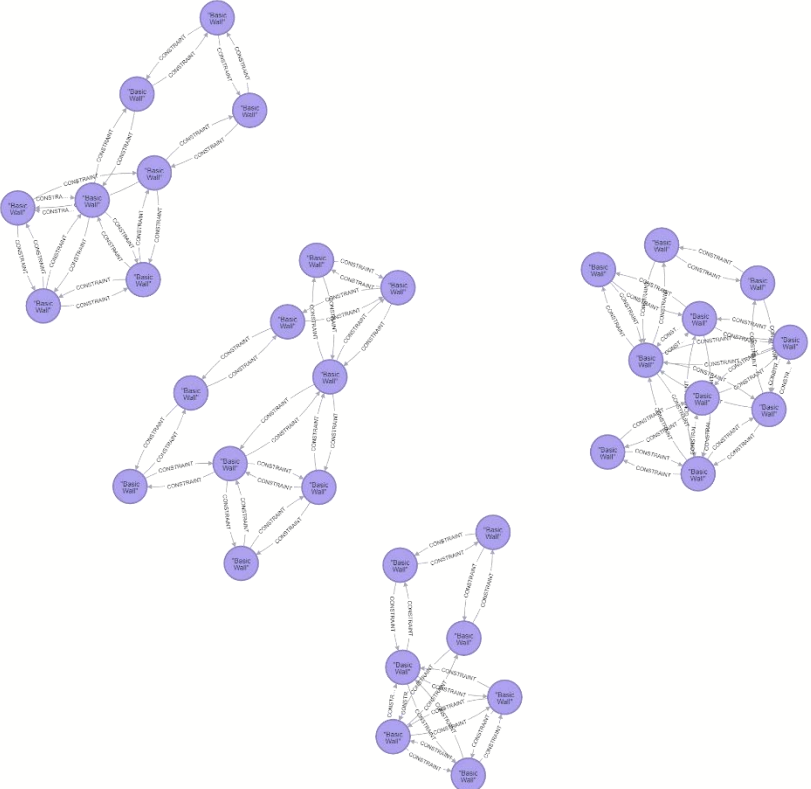
Walls: Constraints by category (Walls) -graph representation

Constraint	Cypher query
<p>All walls in the category Walls have</p> <p>[min,mean,max,mode]</p> <p>[0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	

Walls: Constraints by family name (Basic Wall) - implementation

Constraint	Cypher query
<p>All walls of the family Basic Wall have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE m.family_name = "Basic Wall" SET m.constr_distance_parall_min= 0.109, m.constr_distance_parall_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_parall_min:0.109,distance_parall_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

Walls: Constraints by family name (Walls) -graph representation

Constraint	Cypher query
<p>All walls of the family Basic Wall have [min,mean,max,mode] [0.109, 2.759, 10.431, 0.2] distance to parallel walls., "conceptual"</p>	

Floors: Constraints by room name (Corridor) - implementation

Constraint	Cypher query
All floors in the room called Corridor have [min,mean,max] [0.718, 2.501, 3.575] distance to parallel floors., "requirements"	<pre> MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE n.room_name = "Corridor" AND m.category = 'Floors' SET m.constr_distance_parall_min= 0.718, m.constr_distance_parall_max=3.575, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_parall_max:3.575,distance_parall_min:0.718, constraint_type: "requirements"}]->(w) RETURN k,m,w </pre>

Floors: Constraints by room name (Corridor) -graph representation

Constraint	Cypher query
All floors in the room called Corridor have [min,mean,max] [0.718, 2.501, 3.575] distance to parallel floors., "requirements"	<p>The graph shows two central pink nodes labeled "Floor". Each "Floor" node is connected to several teal nodes labeled "Compo...". The connections are labeled "CONSTRAINT". The graph illustrates the relationships between floors and their components, with constraints applied to these relationships.</p>

Floors: Constraints by category(Floors) - implementation

Constraint	Cypher query
<p>All floors in the category Floors have [min,mean,max,mode] [0.718, 2.756, 3.575, 2.85] distance to parallel floors., "requirements"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE m.category = 'Floors' SET m.constr_distance_parallel_min= 0.718, m.constr_distance_parallel_max=3.575, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_parallel_min:0.718,distance_parallel_max:3.575, constraint_type: "requirements"}]->(w) RETURN k,m,w</pre>

Floors: Constraints by category (Floors) -graph representation

Constraint	Cypher query
<p>All floors in the category Floors have [min,mean,max,mode] [0.718, 2.756, 3.575, 2.85] distance to parallel floors., "requirements"</p>	<p>The graph illustrates the relationships between floor nodes and component nodes. There are two pink nodes labeled "Floor" and several teal nodes labeled "Compo...". Each teal node is connected to one or more pink nodes via edges labeled "CONSTRAINT". The connections form a complex network where components are associated with specific floor nodes and also have relationships between themselves.</p>

Floors: Constraints by family name(Compound Ceiling) - implementation

Constraint	Cypher query
<p>All floors of the family Compound Ceiling have [min,mean,max,mode] [0.718, 1.759, 2.8, 0.718] distance to parallel floors., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_PAR]->(w) WHERE m.family_name = "Compound Ceiling" SET m.constr_distance_parall_min= 0.718, m.constr_distance_parall_max=2.8, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_parall_min:0.718,distance_parall_max:2.8, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

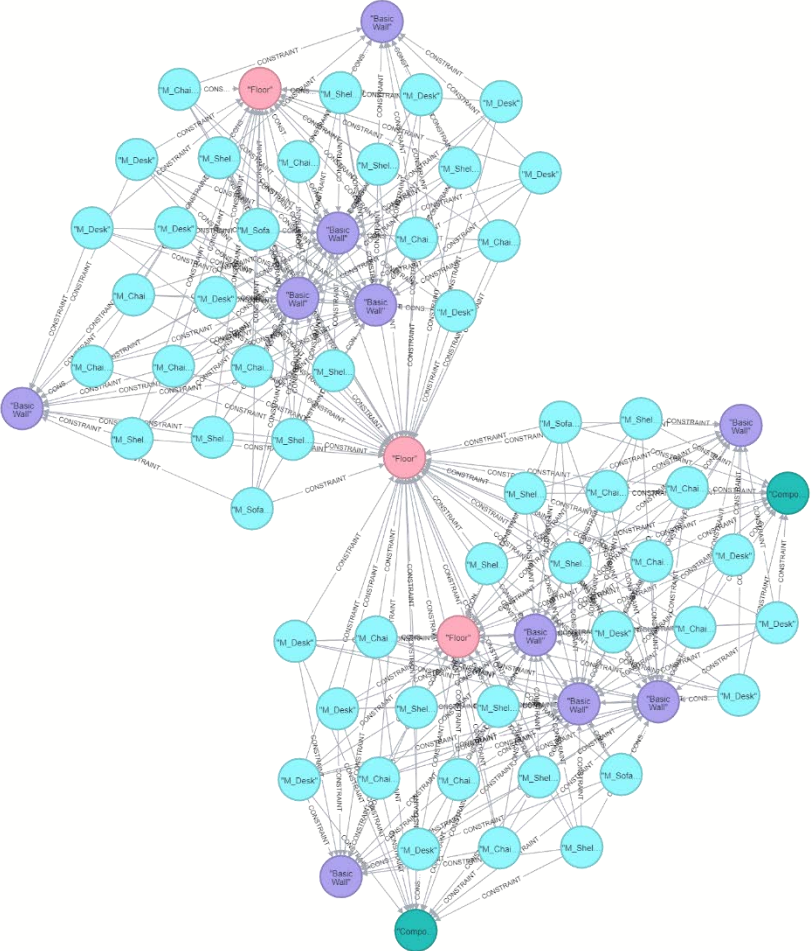
Floors: Constraints by family name (Floor) -graph representation

Constraint	Cypher query
<p>All floors of the family Floor have [min,mean,max] [0.718, 2.517, 3.575] distance to parallel floors., "requirements"</p>	<p>The diagram illustrates a graph structure where a central pink node labeled "Floor" is connected to approximately 15 teal nodes labeled "Compo...". Each connection is labeled "CONSTRAINT". The connections are bidirectional, indicating a mutual relationship between the central "Floor" node and each "Compo..." node.</p>

Furniture: Constraints by room name(Office) - implementation

Constraint	Cypher query
<p>All furniture in the room called Office have [min,mean,max,mode] [0.0, 2.2, 4.934, 0.0] distances to nearest wall or floor., "conceptual"</p>	<pre>MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEAREST]->(w) WHERE n.room_name = "Office" AND m.category = 'Furniture' SET m.constr_distance_parall_min= 0.0, m.constr_distance_parall_max=4.934, m.constr_characteristics="closed interval" MERGE (m)-[k:CONSTRAINT{distance_parall_min:0.0,distance_parall_max:4.934, constraint_type: "conceptual"}]->(w) RETURN k,m,w</pre>

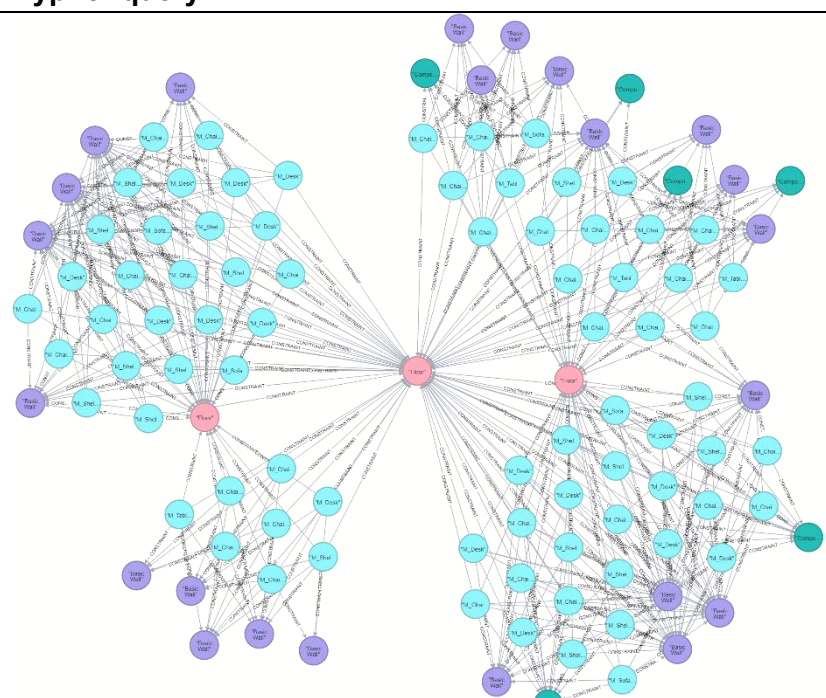
Furniture: Constraints by family name (Office) -graph representation

Constraint	Cypher query
<p>All furniture in the room called Office have [min,mean,max,mode] [0.0, 2.2, 4.934, 0.0] distances to nearest wall or floor., "conceptual"</p>	

Furniture: Constraints by category (Furniture) - implementation

Constraint	Cypher query
All furniture in the category Furniture have [min,mean,max,mode] [0.0, 1.986, 5.516, 2.782] distances to nearest wall or floor., "conceptual"	<pre> MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEAREST]->(w) WHERE m.category = 'Furniture' SET m.constr_distance_parall_min= -0.069, m.constr_distance_parall_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[k:CONSTRAINT{distance_parall_min:-0.069,distance_parall_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w </pre>

Furniture: Constraints by category (Furniture) -graph representation

Constraint	Cypher query
All furniture in the category Furniture have [min,mean,max,mode] [0.0, 1.986, 5.516, 2.782] distances to nearest wall or floor., "conceptual"	

Furniture: Constraints by family name (M_Toilet-Commercial-Wall-3D) - implementation

Constraint	Cypher query
<p><i>All furniture of the family M_Toilet-Commercial-Wall-3D have [min,mean,max,mode] [0.0, 1.51, 3.476, 0.076] distances to nearest wall or floor., "conceptual"</i></p>	<pre> MATCH (n)-[:CONTAINS]->(m) MATCH (m)-[:DISTANCE_NEAREST]->(w) WHERE m.family_name = "M_Toilet-Commercial-Wall-3D" SET m.constr_distance_parall_min= 0.0, m.constr_distance_parall_max="inf", m.constr_characteristics="half-open interval" MERGE (m)-[:CONSTRAINT{distance_parall_min:0.0,distance_parall_max:"inf", constraint_type: "conceptual"}]->(w) RETURN k,m,w </pre>

Furniture: Constraints by family name (M_Toilet-Commercial-Wall-3D) -graph representation

Constraint	Cypher query
<p><i>All furniture of the family M_Toilet-Commercial-Wall-3D have [min,mean,max,mode] [0.0, 1.51, 3.476, 0.076] distances to nearest wall or floor., "conceptual"</i></p>	

Bibliography

- ABUALDENIEN, J., & BORRMANN, A. (2021). Pbg: A parametric building graph capturing and transferring detailing patterns of building models.
- ARORA, H., BIELSKI, J., EISENSTADT, V., LANGENHAN, C., ZIEGLER, C., ALTHOFF, K.-d., DENGEL, A., & FORSCHUNGSZEN-, D. (2021). Consistency checker an automatic. *2*, 351–358.
- BHATT, M., LEE, J. H., & SCHULTZ, C. (2011). Spatial information theory. In M. EGENHOFER, N. GIUDICE, R. MORATZ, & M. WORBOYS (Eds.). Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-23196-4>
- BISKJAER, M. M., DALSGAARD, P., & HALSKOV, K. (2014). A constraint-based understanding of design spaces. *Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, DIS*, 453–462. <https://doi.org/10.1145/2598510.2598533>
- BORRMANN, A., & BEETZ, J. (2010). Towards spatial reasoning on building information models. *eWork and eBusiness in Architecture, Engineering and Construction - Proceedings of the European Conference on Product and Process Modelling 2010*, 61–67. <https://doi.org/10.1201/b10527-13>
- BORRMANN, A., HYVÄRINEN, J., & RANK, E. (2009). Spatial constraints in collaborative design processes. *Proceedings of the International Conference on Intelligent Computing in Engineering (ICE'09)*, 1–8. http://www.andre-borrmann.de/docs/paper_ICE09.pdf
- BORRMANN, A., & RANK, E. (2010). Query support for bims using semantic and spatial conditions, 405–450. <https://doi.org/10.4018/978-1-60566-928-1.ch018>
- BORRMANN, A., SCHRAUFSTETTER, S., & RANK, E. (2009). Implementing metric operators of a spatial query language for 3d building models: Octree and b-rep approaches [mediatitle: Journal of Computing in Civil Engineering
number: 1
]. *Journal of Computing in Civil Engineering*, *23*(1), 34–46.
- BRAUN, A., TUTTAS, S., & BORRMANN, A. (2015). A concept for automated construction progress monitoring using bim-based geometric constraints and photogrammetric point clouds. *Journal of Information Technology in Construction (ITcon)*, *20*, 68–79. <http://www.itcon.org/2015/5>
- DAUM, S., & BORRMANN, A. (2014). Processing of topological bim queries using boundary representation based methods. *Advanced Engineering Informatics*, *28*, 272–286. <https://doi.org/10.1016/j.aei.2014.06.001>
- DUBEY, R. K., KHOO, W. P., MORAD, M. G., HÖLSCHER, C., & KAPADIA, M. (2020). Autosign: A multi-criteria optimization approach to computer aided design of signage layouts in complex buildings. *Computers Graphics*, *88*, 13–23. <https://doi.org/10.1016/j.cag.2020.02.007>

- EASTMAN, C., LEE, J.-m., JEONG, Y.-s., & LEE, J.-k. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, 18, 1011–1033. <https://doi.org/10.1016/j.autcon.2009.07.002>
- EASTMAN, C., TEICHOLZ, P., SACKS, R., & LISTON, K. (2008). *Bim handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. Wiley Publishing.
- EISENSTADT, V., ARORA, H., ZIEGLER, C., BIELSKI, J., LANGENHAN, C., ALTHOFF, K.-D., & DENGEL, A. (2021). Exploring optimal ways to represent topological and spatial features of building designs in deep learning methods and applications for architecture.
- EMUNDS, C., PAUEN, N., RICHTER, V., FRISCH, J., & van TREECK, C. (2022). Sparse-bim: Classification of ifc-based geometry via sparse convolutional neural networks. *Advanced Engineering Informatics*, 53, 101641. <https://doi.org/10.1016/j.aei.2022.101641>
- EXNER, H., ABUALDENIEN, J., KÖNIG, M., & BORRMANN, A. (2019). Managing building design variants at multiple development levels. *Proc. of the 36th International Council for Research and Innovation in Building and Construction (CIB W78)*.
- FONSECA, B. B., AGUILERA, M. V. C., & VIDAL, M. C. R. (2012). Conceptual design pattern for ergonomic workplaces. *Work*, 41, 797–803. <https://doi.org/10.3233/WOR-2012-0243-797>
- ISMAIL, A., NAHAR, A., & SCHERER, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of bim models based on ifc standard.
- JUNG, N., & LEE, G. (2019). Automated classification of building information modeling (bim) case studies by bim use based on natural language processing (nlp) and unsupervised learning. *Advanced Engineering Informatics*, 41, 100917. <https://doi.org/10.1016/j.aei.2019.04.007>
- KIRCHNER, J., & HUHNT, W. (2018). Benefits and limits of interval constraints in acyclic constraint graphs for geometric modeling. *Proc. of the 17th International Conference on Computing in Civil and Building Engineering*.
- KNEIDL, A., BORRMANN, A., & HARTMANN, D. (2012). Generation and use of sparse navigation graphs for microscopic pedestrian simulation models. *Advanced Engineering Informatics*, 26, 669–680. <https://doi.org/10.1016/j.aei.2012.03.006>
- KOTIRANTA, P., JUNKKARI, M., & NUMMENMAA, J. (2022). Performance of graph and relational databases in complex queries. *Applied Sciences*, 12, 6490. <https://doi.org/10.3390/app12136490>
- KRAFT, B., & WILHELMS, N. (2005). Visual knowledge specification for conceptual design. *Computing in Civil Engineering (2005)*, 1–14. [https://doi.org/10.1061/40794\(179\)16](https://doi.org/10.1061/40794(179)16)
- LANGENHAN, C., WEBER, M., LIWICKI, M., PETZOLD, F., & DENGEL, A. (2013). Graph-based retrieval of building information models for supporting the early design stages. *Advanced Engineering Informatics*, 27, 413–426. <https://doi.org/10.1016/j.aei.2013.04.005>
- LAU, E., & KONG, J. J. (2022). *Identification of constraints in construction projects to improve performance*.

- LEE, G., SACKS, R., & EASTMAN, C. M. (2006). Specifying parametric building object behavior (bob) for a building information modeling system. *Automation in Construction*, 15, 758–776. <https://doi.org/10.1016/j.autcon.2005.09.009>
- LEE, Y.-C., EASTMAN, C. M., & SOLIHIN, W. (2016). An ontology-based approach for developing data exchange requirements and model views of building information modeling. *Advanced Engineering Informatics*, 30, 354–367. <https://doi.org/10.1016/j.aei.2016.04.008>
- LÓPEZ, F. M. S., & CRUZ, E. G. S. D. L. (2015). Literature review about neo4j graph database as a feasible alternative for replacing rdbms. *Industrial Data*, 18, 135. <https://doi.org/10.15381/idata.v18i2.12106>
- LUO, H., GAO, G., HUANG, H., KE, Z., PENG, C., & GU, M. (2022). A geometric-relational deep learning framework for bim object classification.
- MARCOS ALBA, C. L. (2017). Bim implications in the design process and project-based learning: Comprehensive integration of bim in architecture.
- NIEMEIJER, R. A., VRIES, B. D., & BEETZ, J. (2010). Designing with constraints towards mass customization in the housing industry. *10th International Conference on Design and Decision Support Systems, DDSS 2010*.
- NIEMEIJER, R., de B. VRIES, & J., B. (2009). Check-mate: Automatic constraint checking of ifc models.
- NIEMEIJER, R., de VRIES, B., & BEETZ, J. (2014). Freedom through constraints: User-oriented architectural design. *Advanced Engineering Informatics*, 28, 28–36. <https://doi.org/10.1016/j.aei.2013.11.003>
- POKORNÝ, J. (2015). Graph databases: Their power and limitations, 58–69. https://doi.org/10.1007/978-3-319-24369-6_5
- SACKS, R., WANG, Z., OUYANG, B., UTKUCU, D., & CHEN, S. (2022). Toward artificially intelligent cloud-based building information modelling for collaborative multidisciplinary design. *Advanced Engineering Informatics*, 53, 101711. <https://doi.org/10.1016/j.aei.2022.101711>
- SASAKI, B. M., CHAO, J., & HOWARD, R. (2018). *Graph db for beginner*. https://neo4j.com/wp-content/themes/neo4jweb/assets/images/Graph_Databases_for_Beginners.pdf
- SCHULTZ, C., BHATT, M., & BORRMANN, A. (2017). Bridging qualitative spatial constraints and feature-based parametric modelling: Expressing visibility and movement constraints. *Advanced Engineering Informatics*, 31, 2–17. <https://doi.org/10.1016/j.aei.2015.10.004>
- SOLIHIN, W., EASTMAN, C., LEE, Y.-C., & YANG, D.-H. (2017). A simplified relational database schema for transformation of bim data into a query-efficient and spatially enabled database. *Automation in Construction*, 84, 367–383. <https://doi.org/10.1016/j.autcon.2017.10.002>
- TAKIM, R., HARRIS, M., & NAWAWI, A. H. (2013). Building information modeling (bim): A new paradigm for quality of life within architectural, engineering and construction (aec) industry [AMER (ABRA malaysia) International Conference on Quality of Life,

- AicQoL2013Langkawij]. *Procedia - Social and Behavioral Sciences*, 101, 23–32. <https://doi.org/https://doi.org/10.1016/j.sbspro.2013.07.175>
- TANG, Z., ZOU, Q., FENG, H.-Y., GAO, S., ZHOU, C., & LIU, Y. (2022). A review on geometric constraint solving, 1–15. <http://arxiv.org/abs/2202.13795>
- VILGERTSHOFER, S., & BORRMANN, A. (2017). Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models. *Advanced Engineering Informatics*, 33, 502–515. <https://doi.org/10.1016/J.AEI.2017.07.003>
- YIN, M., TANG, L., WEBSTER, C., XU, S., LI, X., & YING, H. (2023). An ontology-aided, natural language-based approach for multi-constraint bim model querying.
- ZABIN, A., GONZÁLEZ, V. A., ZOU, Y., & AMOR, R. (2022). Applications of machine learning to bim: A systematic literature review. *Advanced Engineering Informatics*, 51, 101474. <https://doi.org/10.1016/j.aei.2021.101474>
- ZAHEDI, A., ABUALDENIEN, J., PETZOLD, F., & BORRMANN, A. (2022). *Bim-based design decisions documentation using design episodes, explanation tags, and constraints* (Vol. 27). <https://doi.org/10.36680/j.itcon.2022.037>
- ZHOU, Y. W., HU, Z. Z., LIN, J. R., & ZHANG, J. P. (2020). A review on 3d spatial data analytics for building information models. *Archives of Computational Methods in Engineering*, 27, 1449–1463. <https://doi.org/10.1007/s11831-019-09356-6>