Artificial Intelligence for Engineering Design, Analysis and Manufacturing

www.cambridge.org/aie

Automated generation of circulations within a floorplan

Shiksha, Sudarshan Anand, Krishnendra Shekhawat and Karan Agrawal

Department of Mathematics, BITS Pilani, Pilani Campus, Pilani, India

Research Article

Cite this article: Shiksha, Anand S, Shekhawat K and Agrawal K (2025). Automated generation of circulations within a floorplan. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **39**, e9, 1–25
https://doi.org/10.1017/S0890060425000022

Received: 05 June 2024
Revised: 15 December 2024
Accepted: 24 January 2025

Keywords:
floorplan; adjacency; circulation; algorithm; graph theory

Corresponding author:
Krishnendra Shekhawat;
Email: krishnendra.shekhawat@pilani.bits-pilani.ac.in

Abstract

Various factors are considered when designing a floorplan layout, including the plan's outer boundary, room shape and size, adjacency, privacy, and circulation space, among others. While graph-theoretic approaches have proven effective for floorplan generation, existing algorithms generally focus on defining the boundary of the plan or different room shapes, lacking the investigation of designing circulation space within a floorplan. However, the circulation design in architectural planning is a crucial factor that affects the functionality and efficiency of areas within a building. This paper presents a graph-theoretic approach for integrating circulation within a floorplan. In this study, we use plane graphs to represent floorplans and develop graph algorithms to incorporate various types of circulation within a floorplan as follows:

i. The first phase generates a spanning circulation, that is, a corridor leading to each room using a circulation graph.
ii. Subsequently, using an approximation algorithm, the circulation space is minimized, that is, generation of minimum circulation space covering all the rooms, thereby enhancing space utilization in the floorplan.
iii. Furthermore, customized circulations are generated to cater to user preferences, distinguishing between public and private spaces within the floorplan.

In addition to the theoretical framework, we have implemented our algorithms in Python and developed a user-friendly graphical interface (GUI), enabling seamless integration of our algorithms into architectural design processes.

Introduction

The increasing demand for residential and commercial buildings has amplified the need for technological solutions that can streamline design and planning processes. Among these advancements, graph-theoretic techniques (Wang et al., 2018; Upasani et al., 2020; Shekhawat et al., 2021a; Bisht et al., 2022) for the automated generation of floorplans have gained significant attention in architectural design and space planning. This paper focuses on one critical aspect of floorplans: circulation spaces, or corridors, which are essential for ensuring efficient access to various parts of a building while maintaining privacy.

To address this need, we present an application designed to assist architects and designers by automating the construction of circulation spaces within floorplans. While it does not analyze circulation patterns, the application efficiently constructs them wherever required, significantly reducing time and eliminating the need for complex manual calculations. As a complementary tool, it aims to enhance workflows by providing fast and customizable solutions.

The application offers two primary modes for circulation construction:

1. **Spanning Circulation:** This algorithm creates a comprehensive corridor system, ensuring direct access to every room in the floorplan. Ideal for hospitals, offices, and hotels, it prioritizes simplicity, efficiency, and safety by enabling smooth movement and providing clear evacuation routes. However, it requires more floor area, potentially increasing construction and maintenance costs.
2. **Minimal Circulation:** To optimize space and reduce costs, the application also provides a minimal circulation design. This approach avoids redundancy, creating only the essential corridors required to connect every room. Some rooms may serve two functions, acting as functional spaces and pathways for circulation. For instance, common areas such as living rooms, waiting areas etc. can be designated for movement, maximizing space efficiency without compromising usability.

Additionally, for floorplans requiring a circulation layout that is more extensive than the minimal design but less comprehensive than the spanning circulation, the application includes a "Remove Corridor" feature. This allows users to selectively remove unnecessary corridors while retaining the essential ones, providing a flexible approach to meet specific design requirement.

The application also incorporates user-defined preferences to customize circulation designs, offering the flexibility to define *public* and *private* rooms. A *private* room is accessible only via a *public* space or a corridor, ensuring it cannot be crossed to access other areas of the floorplan. In contrast, *public* rooms facilitate circulation and can act as transitional spaces within the design. This capability allows tailored designs that balance functionality, privacy, and spatial optimization.

By automating the construction of circulation spaces, the application enables architects and designers to quickly generate various types of circulation patterns, eliminating manual calculations and providing visual outputs in a fraction of the time.

The structure of the paper is as follows: *Preliminaries* introduces the terminology and notation used throughout. *Literature review* reviews previous work on the automated generation of floorplans and circulation spaces, highlighting gaps in the existing literature. *Methodology* presents Algorithms 1–6 for incorporating various types of circulation into a floorplan F based on a given plane graph G. In *Generating a floorplan and circulation graph for the given PTPG*, we describe an algorithm that generates a circulation graph for the given PTPG by adding extra nodes to capture potential movement paths within the layout, serving as the foundation for various circulation spaces. *From circulation graph to floorplan with required circulation space* integrates spanning circulation into the floorplan, while *Generating minimal circulation space* introduces an approximation algorithm for generating minimal circulation. *Customization of the circulation space according to user-specified privacy constraints* details an algorithm that customizes circulation space according to user preferences, and *Algorithm validation and complexity* discusses the mathematical validation and complexity of the algorithms. Finally, *Results and discussion* discusses the results, including a comprehensive walk through of the application's features, illustrated with screenshots and examples from the graphical user interface (GUI). A functional demonstration of the GUI is accessible online and the implementation is available on GitHub via the links provided in Appendix A.

Preliminaries

This section defines the basic terminology and introduces the notations that will be used consistently throughout the paper.

1. **Floorplan (Shekhawat, 2018):** A *floorplan* (FP) can be defined as a division of a polygonal boundary into rooms using straight lines. The straight lines defining the boundary of each room are called walls. Two rooms are said to be adjacent if they share a common wall or a sub-wall.

   If all the rooms and the plan boundary are rectangles, it is referred to as a rectangular floorplan (RFP) (see Figure 1(a)). Additionally, if the outer boundary remains rectilinear while maintaining rectangular modules, the resulting floorplan is termed a non-rectangular floorplan (NRFP).

2. **Circulation (Naderpour et al., 2019):** In a floorplan, *circulation* refers to the pathway/corridor connecting various rooms of the floorplan. It facilitates movement and interaction within a building. A *spanning* circulation space represents a single interior courtyard adjacent to each room of a floorplan (see Figure 1(b)).
3. **Graph (Bondy and Murty, 1976):** A *graph* is a mathematical structure, $G = (V, E)$, consisting of a non-empty set of vertices (nodes) V and a set of edges E, which may be empty. An element (v, w) of E represents an edge joining the vertices v and w. The *degree* of a vertex v is the number of neighbors of v in G, denoted by $deg(v)$.

   Every floorplan can be envisioned as a graph, with vertices representing the rooms in the floorplan, and edges indicating the adjacency between rooms (see Figure 1(c)).

4. **Properly triangulated plane graph (PTPG) (Bhasker and Sahni, 1986):** A PTPG has the following properties (refer to Figure 1(c)):
   i. In a floorplan, the walls cannot overlap or cross; consequently, the corresponding PTPG has no edge crossings; it is a plane graph. A plane graph divides the plane into distinct regions known as faces.
   ii. Since there are no empty spaces in a floorplan (RFP or NRFP) and all rooms are rectangles, every face of the corresponding PTPG except the exterior is a triangle.
   iii. Since all the rooms in a floorplan (RFP or NRFP) are rectangles, every interior vertex v in the corresponding PTPG has at least four neighbors, i.e., $deg(v) \geq 4$.
5. **Exterior edges:** An edge $e \in E(G)$ is an *exterior edge* of a plane graph G if both the endpoints of e are exterior vertices. For example, in the graph shown in Figure 1(c), the edge (4, 5) is an exterior edge.
6. **Subdivision (Bondy and Murty, 1976):** An edge $e := (u, v) \in E(G)$ is said to be *subdivided*, if we add a new vertex w, remove the edge e and add edges (u, w) and (v, w) (see Figure 2(a)).
7. **Circulation graph:** Given a PTPG G, a modified PTPG is obtained by subdividing the necessary edges of G to integrate spanning circulation into the floorplan of G with one entry point (refer to Section title "Generating a floorplan and circulation graph for the given PTPG"). For example, Figure 2(b) shows a circulation graph of the PTPG G shown in Figure 1(c),

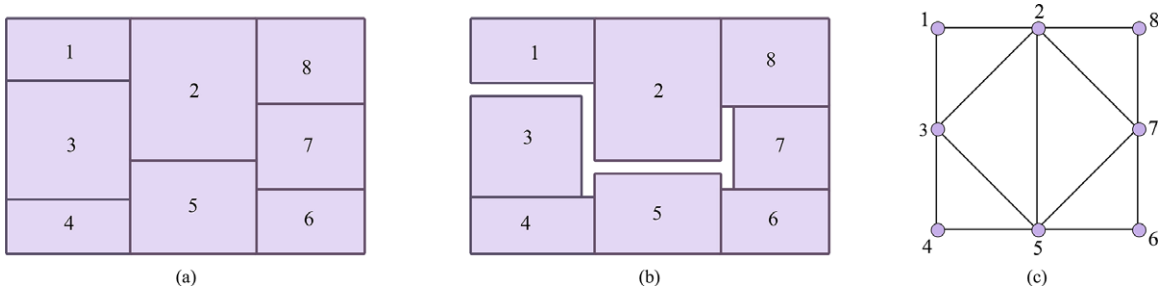


Figure 1. (a) A rectangular floorplan, (b) A rectangular floorplan featuring *spanning* circulation, and (c) Graph associated with the RFP shown in (a).

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

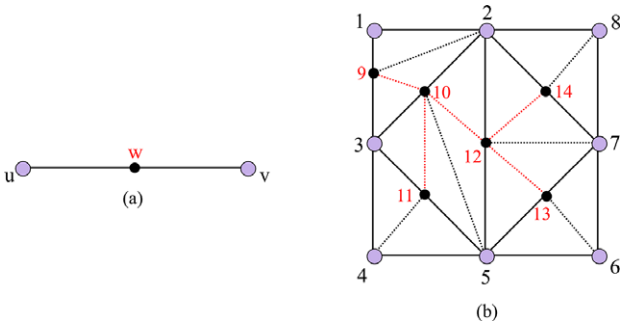


Figure 2. (a) Subdivision of edge (u,v), (b) A circulation graph of the PTPG shown in Figure 1(c) (black vertices correspond to the corridor spaces).

corresponding to which an RFP of G is obtained with circulation as shown in Figure 1(b).

8. **Minimal Circulation:** Minimal circulation space refers to the smallest set of essential corridors required to facilitate movement within a floorplan. It is achieved by adding the fewest possible corridor vertices when generating the circulation graph which covers all the vertices of the PTPG, ensuring that all rooms in the floorplan are accessible while minimizing redundant corridors (refer to Section title "Generating minimal circulation space").
9. **Dimensionless and Dimensioned Floorplans (Shekhawat et al., 2021b):** *Dimensionless floorplans* are generated based solely on the given adjacencies, without considering room dimensions as input constraints. In contrast, *Dimensioned Floorplans* incorporate both room dimensions and adjacencies as part of the input constraints.

Important Notations:
G: A given *PTPG* (user input)
n: Number of vertices in graph G (order of G)
m: Number of edges in graph G (size of G)
G_C: A circulation graph obtained from the given PTPG G
$\mathbb{F}$: A floorplan corresponding to the given PTPG G
$\mathbb{F}_C$: The floorplan featuring spanning circulation corresponding to the given PTPG G

Literature review

Graph-theoretic techniques have been integral to architecture and urban planning since the 1960s (Levin, 1964), evolving alongside computational tools in architecture. Theodora Vardouli's thesis (Vardouli, 2017) highlights how architects turned to structural abstraction to purify Modern architecture, with graphs serving as a bridge between visual depiction and mathematical analysis. Levin's pioneering work (Levin, 1964) initiated research on floorplan generation using graph theory, focusing on optimal layouts. March and Steadman's "The Geometry of Environment" (March and Steadman, 1971) in 1971 introduced a significant application of graph theory in architectural design, conceptualizing building plans as graphs with rooms as nodes and connections as edges.

The field progressed rapidly through the 1970s and 1980s. Mitchell et al. (1976) proposed a method in 1976 to produce topologically distinct floorplans from input graphs, while Bloch and Krishnamurti developed algorithms to count and classify rectangular dissections. Steadman (Steadman, 1983) introduced the concept of "morphology" in 1983, using graph theory to model architectural plans with emphasis on buildings' morphological properties. In 1984, *Hillier and Hanson (1984)* introduced Space Syntax, a methodology using graph theory to analyze spatial configurations and their influence on social behavior. By the 1990s, researchers were proposing methods to verify the existence of rectangular floorplans for given graphs with specific adjacencies and dimensions (Bhasker and Sahni, 1986; Kozminski and Kinnen, 1985; Bhasker and Sahni, 1988; Kozminski and Kinnen, 1988; He, 1993; Kant and He, 1997; He, 1999; Eppstein et al., 2009).

Advancements in design and technology have expanded floorplan generation beyond conventional rectangular floor- plans (RFPs) to include floorplans with rectilinear boundaries or rooms. These newer approaches enable architects to create flexible, customized layouts that optimize space and address diverse design challenges. Various methods – hierarchical, heuristic, algorithmic, and computational – have broadened the scope of floorplan design (Jawaherul et al., 2013; Wu et al., 2018; Wu et al., 2019; Hu et al., 2020; Rahbar et al., 2021; Wang and Zhang, 2020; Raveena and Shekhawat, 2023; Raveena et al., 2024).

Recent years have seen the rise of computer-aided approaches to floorplan generation, as summarized in Table 1. These approaches include the use of machine learning, graph theory, and generative adversarial networks (GANs), which have significantly enhanced automation in design generation. Methods such as those by Wu et al. (2018, 2019), Nauata et al. (2020; 2021), Bisht et al. (2022), and others demonstrate a strong trend towards graph-theoretic and data-driven techniques for creating highly customized, optimized floorplans based on user constraints.

While the above-discussed studies focused on room size, relations, and boundaries, they often overlooked the significance of circulation, particularly corridors. In 1997, Evans (1997) explored how corridors, doors, and spatial arrangements influence social hierarchies and interactions, tracing these shifts from the 16th century onwards. Similarly, in 2010, Jarzombek (2010) examined the social and psychological importance of corridors, highlighting their symbolic and functional evolution in shaping human experiences within architectural spaces. Together, these works emphasize the role of corridors in both mediating movement and reflecting broader societal changes.

Given the significance of circulation in floorplan designing, various approaches have emerged over the years for the analysis and representation of circulation to enhance the architectural design. As shown in Table 2, these include advanced computational methods, such as the Universal Circulation Network (UCN) and pathfinding algorithms, to optimize circulation paths within building layouts (Naderpour et al., 2019; Lee et al., 2010; K et al., 2014). Techniques like Space Syntax theory (Mustafa and Azeez, 2022; Sabir and Mustafa, 2023) have further advanced our understanding of how spatial configurations influence user behavior and movement within various building typologies.

These studies demonstrate the evolution of circulation analysis in architecture, from graph-theoretic to advanced computational methods. They show a trend towards efficient, automated processes for analyzing and optimizing building layouts, focusing on improving navigation, safety, and addressing public health concerns.

Gaps in the existing literature and our work

Despite significant advancements in floorplan generation, there is a notable lack of studies specifically addressing the automated

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Table 1. Literature survey related to the automated generation of floor plans

Reference	Approach	Summary
(Wu et al., 2018)	Hierarchical Approach and Mixed Integer Quadratic Programming (MIQP)	Used a hierarchical approach to generate building floor plans with predefined constraints, treating the outer boundary as a polygon, subdividing it, and solving with MIQP. Displayed 3D models for visualization.
(Wang et al., 2018)	Graph-theoretic approach	Presented a graphical approach to design generation (GADG) to customize existing legacy floorplans by adding or removing a room using transformation rules.
(Wu et al., 2019)	Data-Driven Technique and Encoder-Decoder Network	Developed a technique for generating residential floor plans by determining room locations first, followed by wall placements using an encoder-decoder network while maintaining the user-specified outer boundary.
(Nauata et al., 2020)	Machine Learning and generative adversarial network (GAN)	Introduced House-GAN software, a generative adversarial network that generates housing layouts using a relational architecture to store constraints within a graph structure.
(Upasani et al., 2020)	Graph-theoretic approach	Used graph algorithms and linear optimization techniques to generate dimensioned rectangular floorplans.
(Hu et al., 2020)	Machine Learning and Deep Neural Network	Presented Graph2Plan, a method for learning floor plan generation from lay-out graphs, generating floor plans based on provided structural information.
(Wang and Zhang, 2020)	Graph-Based Approach	Demonstrated a generic approach for automated generation of floor plans with non-rectangular boundaries, using user specifications and algorithms for room placement.
(Shekhawat et al., 2021b)	Graph-theoretic approach	Introduced GPLAN, a software tool designed to create dimensioned floor plans based on user-provided adjacency graphs and dimensional constraints.
(Rahbar et al., 2021)	Hybrid Technique, Agent-Based Modeling	Demonstrated a novel hybrid technique for generating automated 2D architectural layouts using agent-based modeling and deep learning, converting bubble diagrams to heat maps for layout generation.
(Nauata et al., 2021)	Machine learning and generative adversarial network (GAN)	Discussed automated generation of building layouts using a complex generative adversarial layout refinement network and a graph-constrained relational GAN for improved floor plans.
(Wang et al., 2021)	Deep learning and generative adversarial network (GAN)	Used a generative adversarial network to automate residential floor planning, refining floor plans iteratively with a graph constrained relational GAN.
(Para et al., 2021)	Graph Theory and Deep learning techniques	Proposed a layout generation model using graph nodes for elements and constraints as edges, solving layouts through constrained optimization, enhancing lay-out generation methods.
(Sun et al., 2022)	Deep learning and Graph generation network (GraphNet)	Presented an innovative approach to floor plan creation using wall graphs and room labels, generating high-quality floor plans without the need for post-processing.
(Bisht et al., 2022)	Graph theory and Mathematical optimization	Introduced G2PLAN, a software tool that automates dimensioned floor plan creation, using adjacency graphs and dimension constraints to generate floor plans with enhanced customization.
(Xie and Ding, 2023)	Graph-theoretic approach and Deep learning techniques	Introduced methods using graph theory optimization for creating three-dimensional architectural layouts based on user input for node placements and limitations.
(Aalaei et al., 2023)	Machine Learning and generative adversarial network (GAN)	Introduced a graph-constrained conditional GAN model that iteratively creates architectural layouts by incorporating spatial relationships and limitations into the generating process.
(Wang et al., 2023)	Deep learning and graph-theoretic techniques	Introduced a deep learning and graph algorithm framework for automated building layout generation, optimizing layout selection using various metrics.
(Han et al., 2024)	Graph theoretic approach and Deep learning techniques	Presented GRAPH2PIX (G2P), a deep-learning model generating floor plans based on user constraints, ensuring architectural feasibility through a three-sub-model architecture.

generation of circulation designs within pre-defined floorplans. Much of existing research centers on plan outer boundary, optimizing room size, placement and adjacencies, with limited attention given to circulation paths (Wang et al., 2018; Shekhawat et al., 2021a; Bisht et al., 2022; Wu et al., 2018; Wang and Zhang, 2020). The previous work related to circulation mainly focused on analysis of existing circulation patterns, rather than creating new designs. Works (Naderpour et al., 2019), (Lee et al., 2010) and (Tsiamitros et al., 2023) scrutinize the movement of people within a building and identify congestion areas in the building, suggesting alterations to existing designs or proposing new ones. However, they do not provide a method for generating circulations within a floorplan. On the other hand, the method discussed in (Li et al., 2018) directly proposes circulation designs for a given floorplan. However, it is confined to commercial spaces like exhibition halls, museums, etc. While techniques like Space Syntax (Hillier and Hanson, 1984; Mustafa and Azeez, 2022; Sabir and Mustafa, 2023) analyze spatial configurations, they do not offer flexible tools for designing circulations with specific constraints (e.g., entry points, corridor width, or privacy). Limited research exists on seamlessly integrating user-defined constraints, such as entry points and corridor thickness, into circulation design. Many studies target specific building types, necessitating more versatile approaches applicable to residential, educational, healthcare, and public facilities (Mustafa and Azeez, 2022; Mustafa and Rafeeq, 2019; Rafeeq and Mustafa, 2021).

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Table 2. Overview of approaches used for circulation analysis and floor plan optimization

Reference	Approach	Summary
(Lee et al., 2010)	Universal Circulation Network (UCN), Building Information Modeling (BIM), Industry Foundation Classes (IFC)	Developed a computational method for measuring walking distances within buildings using a length-weighted graph structure. The UCN provides a new explicitly defined method for representing circulation paths on top of building models, supporting further circulation-related analysis as a network application.
(Eastman, 2009)	Building Information Modeling (BIM), Industry Foundation Classes (IFC), Rule-based systems	Surveyed rule checking systems for assessing building designs. Examined five major industrial efforts relying on IFC building models as input. Organized the functional capabilities of rule checking systems into four stages: rule interpretation, building model preparation, rule execution, and rule check reporting.
(Taneja et al., 2011)	Industry Foundation Classes (IFC), Geometric Topology Network	Developed a method for transforming IFC-based building layout information into a geometric topology network for in-door navigation assistance.
(K et al., 2014)	Graph-based representation, Universal Circulation Network (UCN), Building Information Modeling (BIM)	Extended the UCN graph structure to include the most-remote point and virtual space objects. This allowed for more accurate measurement of fire egress distances and handling of virtually subdivided spaces in open plan designs.
(Li et al., 2018)	Shape grammars, Cellular Automata (CA)	Demonstrated that shape grammars, combined with cellular automata, can be used to generate a variety of circulation designs. The system automatically generated four basic types of complex circulations for commercial spaces.
(Naderpour et al., 2019)	CAD/BIM data conversion, Theta* pathfinding algorithm, Grasshopper	Developed A2B toolkit for analyzing circulation in buildings. Created a semi-automated workflow to convert CAD/BIM floor plans into a navigation model. Used modified Theta* algorithm for efficient and realistic pathfinding. Demonstrated applications for congestion prediction and emergency egress analysis.
(Mustafa and Rafeeq, 2019)	Space Syntax Theory	Identified circulation patterns and effects of spatial configuration on user mobility in elementary schools with different floor plan typologies (L, U, O).
(Rafeeq and Mustafa, 2021)	Space Syntax Theory	Concluded that U-shaped hospitals pro-vide better accessibility for inpatient wards compared to L-shaped ones, raising further research questions on wayfinding and architectural function.
(González and Gongal, 2021)	Graph theory, Cluster Lane Method	Proposed a flexible mathematical method to turn any planar circulation network into a network of unidirectional lanes, limiting face-to-face interactions between pedestrians. Applied to informal settlements to address COVID–19 transmission in narrow public circulation spaces.
(Mustafa and Azeez, 2022)	Space Syntax Theory	Analyzed how different office layout typologies (cellular, group office, open-plan) influence user behavior and established a relationship between spatial lay-out and time spent moving within the building.
(Mustafa and Ahmed, 2023)	Space Syntax Theory	Examined the spatial configurations of outpatient clinics, measuring wayfinding, accessibility, and density; identified key factors affecting user mobility in various layout typologies.
(Tsiamitros et al., 2023)	Wi-Fi probe requests, Prophet model, ARMA model	Used AI for indoor localization, analyzing WiFi data to understand movement patterns within buildings. Compared the performance of the Prophet model and ARMA model in analyzing pedestrian movement. Performed pedestrian flow analysis to identify the most common paths in a place of interest.
(Sabir and Mustafa, 2023)	Space Syntax Theory, Axial Maps, Graph-based Techniques	Evaluated the impact of emergency department layouts on corridor circulation, utilizing axial maps to optimize spatial layouts for improved functional performance.

This paper presents an approach that employs graph algorithms to generate circulation designs for a floorplan represented by a plane graph. This approach allows users to include constraints such as entry points, corridor thickness, and privacy constraints as necessary. Our approach can be used to generate circulations in various building types, such as residential buildings, educational infrastructure, commercial spaces, and public facilities like hospitals, hotels, etc. Section title "Results and discussion" discusses several applications of our work. Furthermore, our algorithms are implemented in Python, and a user-friendly interface (GUI) is developed. This allows users to employ our technique directly and observe the results in a fraction of the time. Our previous work introduced dimensioning in rectangular arrangements (Upasani et al., 2020), developed algorithms for generating dimensioned rectangular floorplans based on adjacency graphs and user- defined constraints (Shekhawat et al., 2021b), and extended this to produce both rectangular and non-rectangular dimensioned floorplans for any given plane graph (Bisht et al., 2022). Building on these foundations, this study focuses on integrating circulation spaces into the floorplans of a given plane graph. It advances the contributions of our earlier publications by adding a new concept of circulation in the automated floorplan generation process.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Methodology

This section presents the detailed approach for integrating different circulation types into a floorplan, including spanning circulation, minimal circulation, and customized circulation to meet user preferences. The user input is taken as a PTPG, with vertices representing rooms within the floorplan and its edges indicating the connections between them. Alternatively, a floorplan can also be used as input, and the circulation spaces can be integrated into the floorplan using the algorithms outlined in this section.

For this given PTPG, we create a spanning circulation that provides space for movement around each room. If necessary, we minimize the circulation area to increase the room sizes by eliminating redundant corridors and utilizing some rooms in the floorplan for movement. Additionally, users can specify a list of public and private rooms if they require privacy constraints, such as no direct access to some specific rooms. We will generate a floorplan with circulation that includes corridors around private rooms while using public rooms for circulation.

An overview of our approach:
To illustrate the working of our algorithms, we utilize a rectangular floorplan for a given PTPG and integrate diverse circulation spaces within it. In Section title "Results and discussion," we illustrate an example where circulation spaces are integrated into a floorplan featuring non-rectangular boundary as well.

Given a PTPG G, we introduce Algorithms 1–6 to generate floorplans for G featuring spanning circulation, minimal circulation or circulation tailored to user-specific privacy constraints. Initially, we generate a floorplan corresponding to the given PTPG. Subsequently, we integrate the required circulation space into the obtained floorplan. Alternatively, a floorplan can also be taken as an input to which we add the necessary circulation space. Here, it is assumed that the floorplans have only one entrance. The procedure is divided into four steps (refer to Figure 3):

i. In the first step, a floorplan $\mathbb{F}$ is obtained for the given PTPG G using the method outlined in (Bisht et al., 2022). Simultaneously, a new graph G_C, referred to as the circulation graph, is generated from the graph G using Algorithm 1 (refer to Section title "Generating a floorplan and circulation graph for the given PTPG").
ii. In the next step, Algorithm 2 (which employs Algorithm 3 as a sub-routine) generates a floorplan $\mathbb{F}_{\mathbb{C}}$ of G with spanning circulation space C using the circulation graph G_C and the floorplan $\mathbb{F}$ as inputs (refer to Section title "From circulation graph to floorplan with required circulation space").
iii. Following that, using the well-known *Minimum set-cover problem*, Algorithm 5 (which employs Algorithm 4 as a sub-routine) reduces the circulation space C into a minimal circulation space C' (refer to Section title "Generating minimal circulation space").
iv. Furthermore, with the implementation of Algorithm 6, we incorporate customization into the floorplan $\mathbb{F}_{\mathbb{C}}$, allowing the transformation of spanning circulation into the required circulation based on user-specified privacy constraints (refer to Section title "Customization of the circulation space according to user-specified privacy constraints"). For instance, if any specific rooms are restricted from general access, users can designate them as private, while others are identified as public. Using the user-provided list, $\mathbb{F}_{\mathbb{C}}$ is transformed into a floorplan with corridors around the private rooms, and public rooms are generally used for movement in the floorplan.

Generating a floorplan and circulation graph for the given PTPG

First, we obtain a floorplan for the given PTPG G using the method described in (Bisht et al., 2022). To introduce spanning circulation into the obtained floorplan, we use Algorithm 1 to generate a circulation graph for G. Let $E = \{e_1, e_2, \ldots, e_m\}$ and $V = \{v_1, v_2, \ldots, v_n\}$ be the edge set and vertex set in G, respectively.

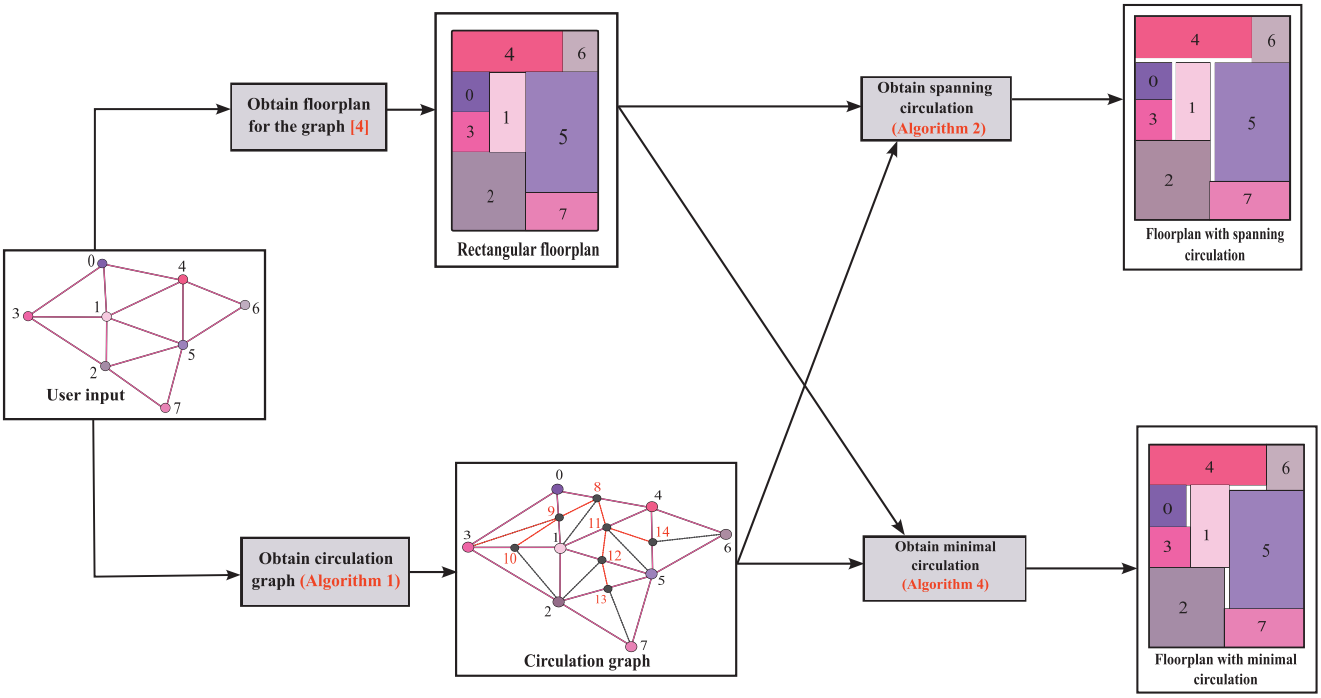


Figure 3. The relations between user input and results of application of various algorithms described in this paper.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Assume f to be the number of interior faces in the PTPG G, denoted as $F = \{F_1, F_2, \ldots, F_f\}$. By applying Algorithm 1, we obtain the circulation graph G_C of G. The graph shown in Figure 3 and its corresponding floorplan will be used as an illustrative example throughout the paper.

Illustration of Algorithm 1:
Algorithm 1 produces a circulation graph G_c for the given PTPG G by introducing new vertices in G that correspond to the corridors required within the floorplan for circulation. It also returns a list of triplets (z, x, y) from G_C, where z is a corridor vertex connecting the rooms x and y. The working of Algorithm 1 is explained in the following steps with an illustrative example:

i. First, an exterior edge, say e, is chosen arbitrarily in G. The face in G to which the edge e belongs is labeled as F_1. Then, the edge e is subdivided by adding a new vertex (introducing the first corridor vertex). For the PTPG taken in Figure 4(a) the exterior edge (0, 4) is selected and subdivided by introducing the first corridor vertex labeled as 8 (see Figure 4(b)). To ensure that the graph remains a PTPG, the newly added vertex is made adjacent to the vertex of the face F_1 that is not an endpoint of the edge e prior to the subdivision (see Figure 4(b)). **This first corridor vertex on the exterior edge later translates to the entrance of the final floorplan.**
ii. In the next step, a face in G adjacent to F_1 is selected arbitrarily and labeled as F_2. The shared edge of F_1 and F_2 is subdivided by adding the new corridor vertex (see Figure 5(a)). Further, the graph is triangulated by adding an edge between the newly added corridor vertex and the vertex of F_2 that is not an endpoint of the shared edge of F_1 and F_2 prior to the subdivision. Also, an edge is added between the newly added corridor vertex and the previously added corridor vertex in F_1 to ensure that the graph remains a PTPG (see Figure 5(b)).
iii. In order to generate the complete circulation graph, the process is continued by selecting a face adjacent to one of the previously subdivided faces (a face in which at least one edge is subdivided) and subdividing the corresponding shared edge until all the faces of the input graph are subdivided. Further, the graph is triangulated by adding the new necessary edges while ensuring that the resultant circulation graph is a PTPG. Figure 6 shows the complete construction of a possible circulation graph for the PTPG given in Figure 4(a). The algorithm also returns A = {(8, 0, 4), (9, 0, 1), (10, 1, 3), (11, 1, 4), (12, 1, 5), (13, 5, 2), (14, 4, 5)}, the list containing the triplets (z, x, y) such that z is a corridor vertex connecting the rooms x and y in the circulation graph.

Now that we have grasped the algorithm's concept, let us look at an explicit definition of the algorithm.

From circulation graph to floorplan with required circulation space

To obtain the floorplan featuring spanning circulation, corridors are incorporated into the floorplan $\mathbb{F}$ with the help of the circulation graph G_C, which was derived in the previous step. Let $V' = \{c_1, c_2, \ldots, c_k\}$ be the set of corridor vertices introduced in G to form G_C. Considering each corridor vertex in G_C one at a time, Algorithm 2 generates a corridor space in $\mathbb{F}$ between the rooms associated with the corridor vertex. The working of Algorithm 2 will be explained using the example illustrated in Figure 7.

Illustration of Algorithm 2:
In the process of corridor generation within the given floorplan $\mathbb{F}$, the following sequential steps are undertaken to ensure the systematic generation of corridors:

1. **Wall Shifting:**
In the first iteration, we start with the first triplet (8, 0, 4) extracted from the list A (a list of triplets (z, x, y) returned by Algorithm 1, where z is a corridor vertex in G_C connecting rooms x (*Room* 1) and y (*Room* 2)). Initially, the common wall w shared by rooms 0 and 4 is identified within $\mathbb{F}$. The coordinates of w, its orientation (horizontal in this case), and the arrangement of rooms 0 and 4 with respect to it are recorded from $\mathbb{F}$. Following this, we refer to Table 3 to determine the walls of rooms 0 and 4 that need to be shifted to create a corridor between these rooms, along with the direction of their shifts. In this case, room 0 is positioned to the south of w, and room 4 is positioned to the north of w in $\mathbb{F}$. Therefore, Table 3 prescribes shifting the top wall (T) of room 0 to the negative y direction (south) and bottom wall (B) of room 4 to the positive y direction (north) to create a corridor between rooms 0 and 4 in $\mathbb{F}$.

[NOTE: To create a corridor of thickness t between a pair of rooms (a, b), the coincident walls of both the rooms are shifted by $t/2$ in opposite directions. In this case, the value of each shift is considered as 0.1 throughout the example, aiming to generate a circulation space of thickness 0.2 in $\mathbb{F}$.]

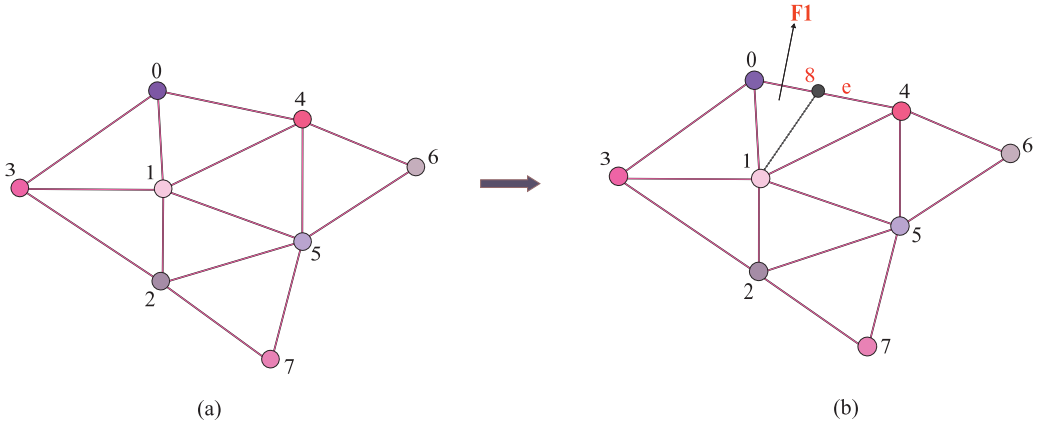


Figure 4. (a) A PTPG G, (b) Adding the first corridor vertex numbered 8 by subdividing the exterior edge (0, 4) of the face (0 −1 −4) in G.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Algorithm 1: SpanCirc(G, e)

Input: $G(V, E)$: A *properly triangulated plane graph*(PTPG)
e: Exterior edge to start the algorithm
Output: $G_C(V_C, E_C)$:A $PTPG$ with the spanning circulation space added
A: List of triples (z, x, y) such that corridor vertex z connects rooms x and y

Step 1 Choose a face F, of G having exterior edge e
Step 2 Inserting the first corridor vertex
// n: Number of vertices in the graph G (rooms in the floorplan)
// m: Number of edges in the graph G (adjacency between rooms in the floorplan)
Insert a new vertex, say V_{n+1}, on the exterior edge e (Refer Figure 4). Add new edge E_{m+1} that makes V_{n+1} adjacent to the vertex of F_1 that is not an endpoint of e. Initialize set A with the triple (V_{n+1}, u, v) where u and v are the end vertices of e.
Step 3 Growing the circulation
// f: Number of interior faces of G
for $i \leftarrow 2$ *to* f **do**
    // Face is subdivided if at least one of its edges is subdivided
    Choose a face F' that is not yet subdivided but is adjacent to a face F'' that is subdivided (break ties arbitrarily)
    // e is the common edge between the faces F' and F''
    Let $e := E(F') \cap E(F'')$. Subdivide e by inserting a new vertex, say V_k, $k \geq n+2$ and make V_k adjacent to a vertex of F' that is not an end-point of e. Add the triple (V_k, u', v') to A where u' and v' are the end vertices of e
    // Triangulation to maintain the graph as $PTPG$
    Let $V(F') = \{x, y, z\}$ and $V(F'') = \{x, y, w\}$ (xy is the common edge that is subdivided). Let a and b be the corridor vertices added to the faces F' and F'' respectively. In order to triangulate the 4-cycle $x - a - y - b$, we add the edge ab.
Define graph $G_C(V_C, E_C)$ with $V_C = V \cup \{V_{n+1}, V_{n+2}, \ldots V_{n+f}\}$ and $E_C = E \cup \{E_{m+1}, E_{m+2}, \ldots E_{m+f}\}$
return G_C, A

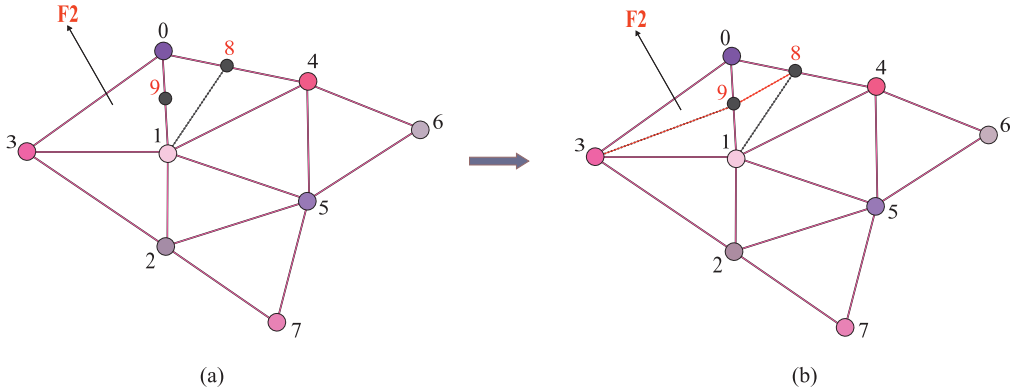


Figure 5. (a) Adding the next corridor vertex numbered 9 by selecting the face $(0 - 1 - 3)$ as F_2 and subdividing the edge (0,1), (b) Triangulating the graph shown in (a) by adding new edges (9, 3) and (9, 8) such that the resultant graph is a PTPG.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

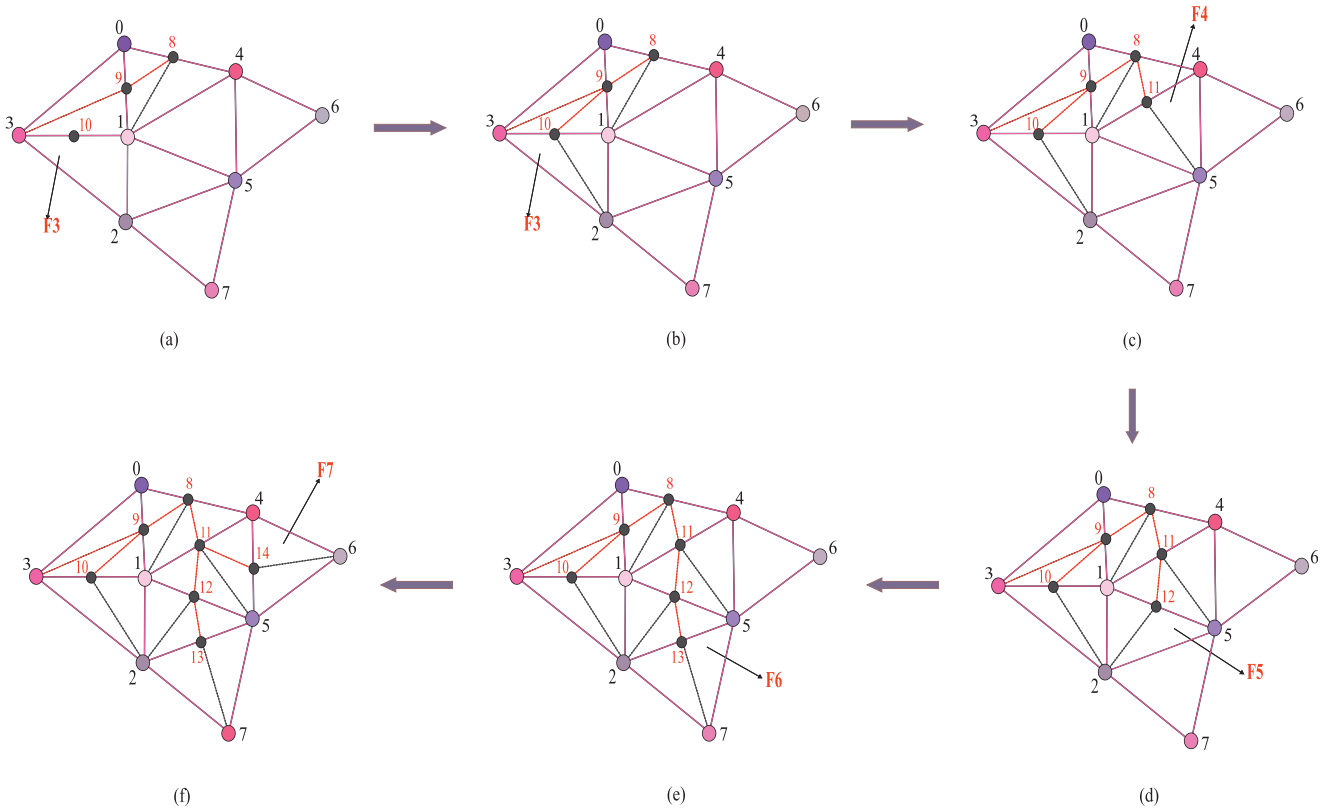


Figure 6. Choosing a neighboring face to a subdivided face at each step from (a) to (d) and introducing a new corridor vertex, ensuring that all faces in the graph undergo subdivision.

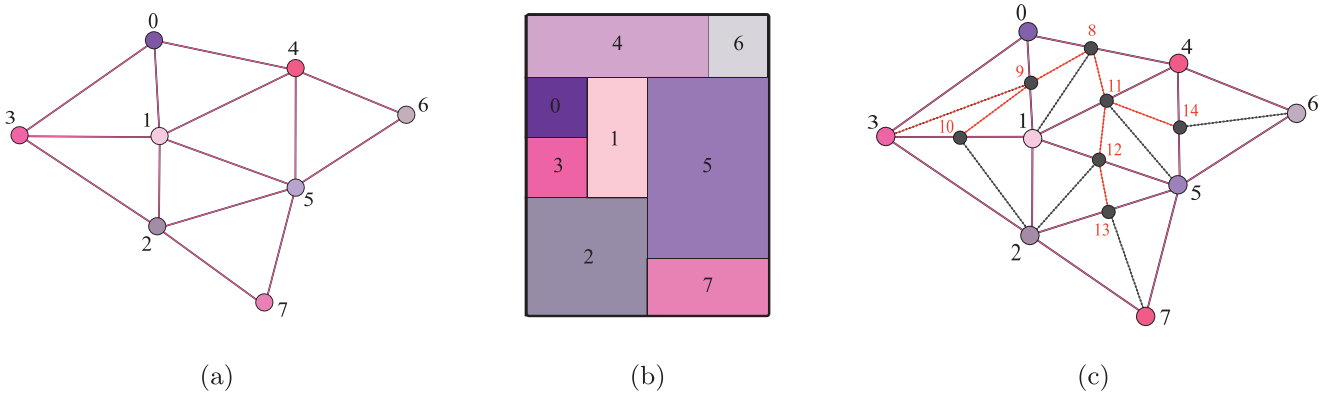


Figure 7. (a) A PTPG G, (b) an RFP $\mathbb{F}$ associated with the PTPG G, and (c) a circulation graph G_C of PTPG G.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Table 3. In this table, T=Top wall, B=Bottom wall, L=Left wall, R=Right wall, N=North, S=South, E=East, W=West. This table illustrates the orientation of the common wall *w* between two rooms and indicates the walls in both rooms that require shifting based on their arrangement around the common wall *w*.

	Position of rooms with respect to w		Modification (wall, direction)	
Orientation of *w*	*Room* 1 (*x*)	*Room* 2 (*y*)	*Room* 1 (*x*)	*Room* 2 (*y*)
Horizontal	North South	South North	(B,N) (T,S)	(T,S) (B,N)
Vertical	West East	East West	(R,W) (L,E)	(L,E) (R,W)

2. **Preventing unnecessary gaps due to wall shifting**

It can be observed if rooms 0 and 4 have other neighbors such that their common walls with rooms 0 or 4 have the same orientation as w (wall shared by rooms 0 and 4 in $\mathbb{F}$) then the shifts made to the walls of rooms 0 and 4 can cause an unnecessary gap in the floorplan. To prevent this gap, we find the neighbors of rooms 0 and 4 in $\mathbb{F}$ and determine which of their walls need to be shifted.

i. **Iterative neighbor identification**

To identify the neighbors of rooms 0 and 4, the following approach is used: Initially, the common neighbor of vertices 0 and 4 in the original graph G is identified. Vertex 1 is found as a common neighbor. Subsequently, the walls shared by room 1 with rooms 0 and 4 in F are located, say, w_1 and w_2, respectively. A pairwise comparison of the orientations of walls w, w_1, and w_2 is conducted, and the wall that aligns with the orientation of w is identified (in this case, w_2). As w_2 is the wall shared by room 1 and room 4; this implies that the gap is a result of shifting the bottom wall of room 4 to the north. All necessary shifts to eliminate this gap are then supposed to be in the north direction. To identify which wall of room 1 needs to be shifted we use Table 4 (in this case, 1 is the neighboring room referred to in Table 4).

Following this, the wall w_2 of room 1 is relabeled as w, and the common neighbor of vertices 1 and 4 is identified in graph G (vertices 1 and 0 are not considered here, since the gap in $\mathbb{F}$ is caused by shifting the bottom wall of 4 to the north). Here, although 0 is a common neighbor, it is disregarded as it has already been visited once. Now, the common walls w_1 and w_2 of the common neighbor of 1 and 4 are located, and their orientations are compared with that of w, as similar to the previous step. This process is repeated until one of the two conditions, C_1 or C_2, is met:

- The orientation of common wall w does not match with those of walls w_1 and w_2. (*C1*)
- There are no more common neighbors that are not visited. (*C2*)

Table 4. This table outlines the necessary wall shift in the neighboring room when the orientation of either wall w_1 or w_2 aligns with the orientation of *w*.

Orientation of *w*	Orientation of $w_1/w_2 = w$ Position of neighboring room w.r.t. to w_1/w_2	Modification of wall of neighboring room
Horizontal	North	Bottom (B)
	South	Top (T)
Vertical	West	Right (R)
	East	Left (L)

Upon identification of neighboring rooms, Table 4 is utilized to determine the walls necessitating adjustments and the results are recorded as pairs (r, w). In this instance, three such pairs were identified: (1, *T*), (5, *T*), (6, *B*). This indicates that, to avoid the gap resulting from shifting the bottom wall of room 4 northward, the top walls of rooms 1 and 5 must be shifted northward. Shifting the top wall of room 5 northward would lead to an overlap between rooms 5 and 6, which is prevented by shifting the bottom wall of room 6 northward.

Nevertheless, employing the iterative method to identify neighbors for any room pairs (a, b) yields all pairs (r, w) that prevent the gap caused by the creation of a corridor between a and b and also addresses any potential overlap resulting from avoiding unnecessary gaps. Moreover, when rooms a and b have two common neighbors, we initiate the process by selecting any one of the common neighbors first. After identifying all neighbors on one side until one of the two conditions, C_1 or C_2 is met, we proceed to the other common neighbor on the opposite side.

To track the shifts, two tables, Shift Table 1 and Shift Table 2, are maintained (see Figure 8). Both of these tables are updated at the end of every iteration. Shift Table 1 records the shift values of walls, while Shift Table 2 stores triplets in the form of (r, w, c), indicating that the wall w of room r has been shifted during the generation of the corridor corresponding to a corridor vertex c.

Initially, the shift value for every wall is set to 0 in Shift Table 1. As corridors are generated based on triplets (z, x, y) from the list A, the updates to the shift values of (r, w), for the wall w of any room r, requiring adjustments is determined as $\max\{current, 0.1\}$ or $\min\{current, -0.1\}$ (depending on the direction of shift) at the conclusion of each iteration. Following the final iteration, Shift Table 1 is utilized to shift the walls and generate a spanning circulation within the floorplan.

ii. **Preventing redundant shifts:**

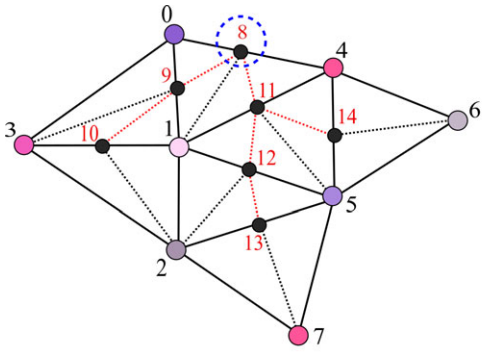
After identifying the pairs (r, w) to address potential gaps or overlaps, the shift value for wall w from this pair will be updated in Shift Table 1, but only if Shift Table 2 does not already include the triplet (r, w, c). In other words, the revision occurs if the wall w of room r has not been previously shifted during the creation of a corridor associated with a corridor vertex c.

This 2-step process is followed in each iteration while creating a corridor corresponding to the triplet (z, x, y) from the list A. The following Figures 9, 10, and 11 illustrates several successive iterations. Figure 12 shows the results of final iteration for the input given in Figure 7 and a spanning circulation in the given floorplan $\mathbb{F}$.

Now, that we have gained an intuitive idea of the algorithm, let us look at it in a formal manner with all the required details.

The above Algorithm 3 is used as a sub-routine in Algorithm 2 in Line 12 to check, which other rooms' walls have to be shifted to accommodate for the creation of corridor space between rooms x and y.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press



(a)

Shift Table 1

r \ w	T	B	L	R
0	-0.1	0	0	0
1	0.1	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0.1	0	0
5	0.1	0	0	0
6	0	0.1	0	0
7	0	0	0	0

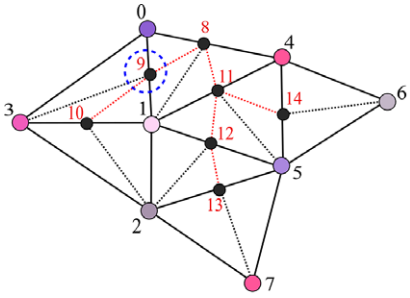
(b)

Shift Table 2

(0, T, 8)
(4, B, 8)

(c)

Figure 8. Iteration 1: (a) Considering the first triplet (8, 0, 4) with corridor vertex 8 from G_C, (b) Shift Table 1 at the end of iteration 1, and (c) Shift Table 2 at the end of iteration 1.



(a)

Shift Table 1

r \ w	T	B	L	R
0	-0.1	0	0	-0.1
1	0.1	0	0.1	0
2	0	0	0	0
3	0	0	0	0.1
4	0	0.1	0	0
5	0.1	0	0	0
6	0	0.1	0	0
7	0	0	0	0

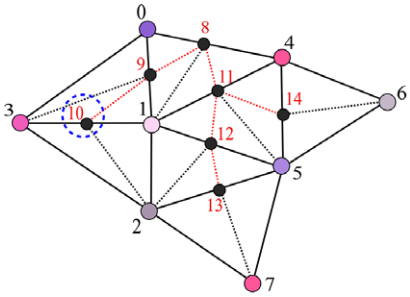
(b)

Shift Table 2

(0, T, 8)
(4, B, 8)
(1, L, 9)
(0, R, 9)

(c)

Figure 9. Iteration 2: (a) Considering the triplet (9, 0, 1) with corridor vertex 9 from G_C, (b) Shift Table 1 at the end of iteration 2, and (c) Shift Table 2 at the end of iteration 2.



(a)

Shift Table 1

r \ w	T	B	L	R
0	-0.1	0	0	-0.1
1	0.1	0	0.1	0
2	0	0	0	0
3	0	0	0	-0.1
4	0	0.1	0	0
5	0.1	0	0	0
6	0	0.1	0	0
7	0	0	0	0

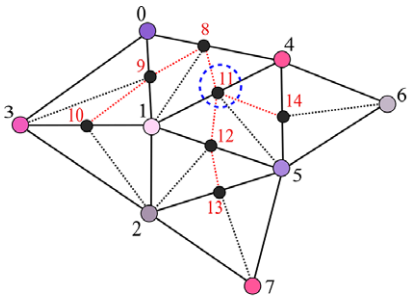
(b)

Shift Table 2

(0, T, 8)
(4, B, 8)
(1, L, 9)
(0, R, 9)
(3, R, 10)
(1, L, 10)

(c)

Figure 10. Iteration 3: (a) Considering the triplet (10, 3, 1) with corridor vertex 8 from G_C, (b) Shift Table 1 at the end of iteration 3, and (c) Shift Table 2 at the end of iteration 3.



(a)

Shift Table 1

r \ w	T	B	L	R
0	-0.1	0	0	-0.1
1	-0.1	0	0.1	0
2	0	0	0	0
3	0	0	0	-0.1
4	0	0.1	0	0
5	0.1	0	0	0
6	0	0.1	0	0
7	0	0	0	0

(b)

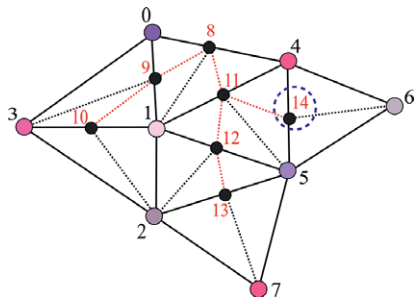
Shift Table 2

(0, T, 8)
(4, B, 8)
(1, L, 9)
(0, R, 9)
(3, R, 10)
(1, L, 10)
(4, B, 11)
(1, T, 11)

(c)

Figure 11. Iteration 4: (a) Considering the triplet (11, 1, 4) with corridor vertex 11 from G_C, (b) Shift Table 1 at the end of iteration 4, and (c) Shift Table 2 at the end of iteration 4.

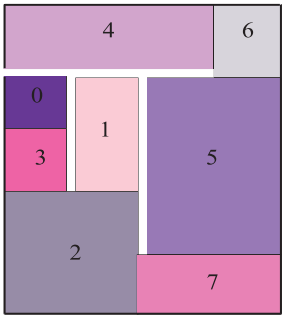
https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press



(a)

Shift Table 1				
r \ w	T	B	L	R
0	-0.1	0	0	-0.1
1	-0.1	0	0.1	-0.1
2	0	0	0	-0.1
3	0	0	0	-0.1
4	0	0.1	0	0
5	-0.1	0	0.1	0
6	0	-0.1	0	0
7	0	0		0

(b)



(c)

Figure 12. Final Iteration: (a) Considering the triplet (14, 4, 5) with the last corridor vertex 14, (b) Shift Table 1 at the final iteration, and (c) An RFP of *G* featuring spanning circulation obtained by using the Shift Table 1 shown in (b).

Algorithm 2: FpCirc(G_c, F, A, (p, q), t)

Input: G_C: The circulation graph corresponding to G
F: The FP corresponding to the graph G (to be modified)
A: Dictionary with elements of the form $(v, (x, y))$ where corridor v connects rooms x and y
(p, q): Exterior edge of G chosen by Algorithm 1 to have the first corridor
t: Corridor thickness
Output: F_C: The floorplan with the required spanning circulation added

Step 1 Preprocessing to get corridor vertices:
// V: Vertex set of graph G ; V_C: Vertex set of circulation graph G_C
Define $V' := V_C - V$ // V' is the set of corridor vertices
Sort V' in ascending order of their subscripts.

Step 2(a) Get the first uncovered corridor vertex in $V' \cap A.keys$, and find rooms connecting this vertex
$v \leftarrow pop(V' \cap A.keys)$
$(x, y) \leftarrow$ Pair of rooms that is connected by corridor v // $(v, (x, y)) \in A$
Step 2(b) Find the shift values for corresponding and related rooms
$w \leftarrow CommonWall(x, y)$ // $CommonWall(a, b)$ finds shared wall between a and b
Based on position and orientation of w with respect to x and y, the shift values and directions of walls of rooms x and y are assigned based on Tables 5 and 6
$CorridorLoop(x, y)$ // Calculates shift values for other related rooms that are affected
Step 3 Control step
// If all corridors have been covered
if $V' = \phi$ **then**
  (Goto Step 2(a))
Step 4 Adjust coordinates
$F_C \longleftarrow F$ // Take a copy of floorplan F
For each room in F_C, update the room coordinates based on the wall to be shifted and the shift values to create the corridor spaces
return F_C

Generating minimal circulation space

In this stage, we produce minimal circulation within the floorplan, precisely the most essential corridors needed to access all areas. This optimization is crucial for addressing corridor redundancy, preventing certain rooms from needlessly losing space and shrinking in size, especially after multiple coordinate shifts outlined in Algorithm 2.

Notably, the display of the initial corridor, recognized as the door, can be excluded since it is already known and thus can be ignored in the representation.

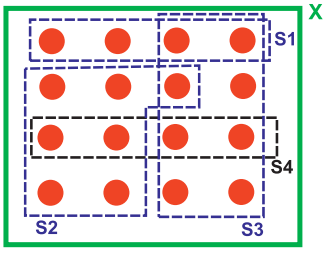
We generate minimal circulation with the help of the well-known problem of the *Minimum set-cover* (Cormen et al., 2001).

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Algorithm 3: CorridorLoop(a, b)

Input: a, b : Rooms' whose common neighbors have to be tested if their wall shifts have to be calculated

```
w ← CommonWall(a, b)                    // Common wall between rooms a and b
N ⟵ CommonNeighbors(a, b)               // Common neighbors of vertices a and b in G
if |N| > 0 then
    for i ← 1 to |N| do
        if N[i] was not the last visited then
            c ⟵ N[i]
            w1 ⟵ CommonWall(c, a)       // Common wall between c and a
            w2 ⟵ CommonWall(c, b)       // Common wall between c and b
            Calculate shift values of respective walls of u and w based on Tables 5 and 6
            // Ort(w) gives the orientation of edge w (horizontal or vertical)
            if Ort(w1) == Ort(w) then
                CorridorLoop(a, c)
            else if Ort(w2) == Ort(w) then
                CorridorLoop(c, b)
            else
                return                  // Control goes back to Algorithm 2
        else
            continue
return                                  // Control goes back to Algorithm 2
```



	Iteration 1	Iteration 2	Iterarion3	
d(1)	4	2	2	
d(2)	7	6	-	
d(3)	8	-	-	
d(4)	4	2	0	

Figure 13. An instance of minimum set cover to illustrate Algorithm 4.

Problem statement Given a set $X = \{e_1, e_2, \ldots e_n\}$ and a set $S = \{S_1, S_2, \ldots, S_m\}$, where each $S_k \subseteq X$ $\forall k \in \{1, 2, \ldots, m\}$ and $\forall i \in \{1, 2, \ldots, n\}$ $\exists j \in \{1, 2, \ldots, m\}$ such that $e_i \in S_j$. The objective of this problem is to find a set $Y \subseteq S$ such that

$$\bigcup_{A \in Y} A = X \tag{1}$$

The set-covering problem is classified as NP-hard, indicating that only sub-optimal solutions can be obtained using approximation algorithms. The following algorithm is one of several greedy approximation algorithms proposed to address this problem.

Working of Algorithm 4:
First, let us intuitively understand the algorithm before delving into its details. Initially, the set $Y = \phi$, and at each step of the algorithm, we add a subset S_i ($i \in \{1, 2, \ldots, m\}$) to Y, covering the maximum number of uncovered elements in X, with ties being broken by taking the lower i. Let us look at an example to understand this algorithm further.

The sets X and $S = \{S_1, S_2, S_3, S_4\}$ are defined as shown in Figure 13 (each element of X is represented by a red dot, and the boundaries of the sets S_i are marked by dotted-lines). Let U be the set of uncovered elements in X and $d_j = |S_j \cap U|$ for every $S_j \in S$. In the first iteration of the algorithm, we observe that $U = X$ and the value of $d(j)$ is the maximum for $j = 3$, with a value of 8; hence, we include S_3 into the set cover Y. In the next step, we update set U and the values of $d(j)$. In the second iteration, $d(j)$ is the maximum for $j = 2$, with a value of 6; hence, we include S_2 into Y. Next, we update set U and include S_1 based on the updated $d(j)$ values. When we recompute U, we note that it is empty (meaning all elements of set X are covered), so we stop the algorithm. Hence, the required Minimum set-cover for this problem instance is $Y = \{S_1, S_2, S_3\}$ (marked by blue dotted-lines). Note that in this case, the algorithm coincidentally gives an optimal set cover.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Algorithm 4: GREEDY-SET-COVER(X, S) (Cormen et al., 2001)

Input: X : The set of elements $\{e_1, e_2, \ldots, e_n\}$ that are to be covered
S : The set $\{S_1, S_2, \ldots, S_m\}$ of subsets of X from which we get the minimum cover X
Output: Y : The smallest set containing elements from the set S, whose union will cover X

$Y \leftarrow \phi$ // The set that accumulates the cover
$U \leftarrow X$ // The set of vertices yet to be covered
// Greedy choice: Pick the set having maximum overlap with the set U
while $U \neq \phi$ **do**
  Let $d_j = |S_j \cap U| \; \forall S_j \in S$
  Let $i_0 = \underset{j \in \{1,2,\ldots,m\}}{\operatorname{argmax}} \; d(j)$ // Break ties by taking lower j
  $Y \leftarrow Y \cup \{i_0\}$ // Include S_{i_0} into the set cover
  $U \leftarrow U \setminus S_{i_0}$ // Remove S_{i_0} from set X
return Y

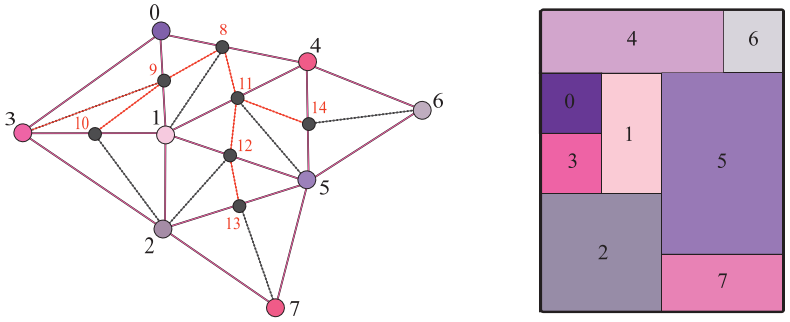


Figure 14. The circulation graph G_C (from Algorithm 1) and the *RFP* $\mathbb{F}$ (from (Shekhawat, 2018)).

By virtue of being a Greedy Algorithm, we must prove that the Algorithm 4 indeed returns a set-cover that is not significantly larger than the optimal set cover. A comprehensive proof of this algorithm can be found in Appendix title "Greedy set cover".

Conversion to minimum set-cover problem:
In this step, we transform our minimization problem into an instance of the Minimum set-cover problem. Given G_C as the circulation graph and V_C as the set of corridor vertices, V represents the set of rooms in $\mathbb{F}$. As earlier mentioned in this section, we can omit the first corridor vertex as it is identified as a door. Consequently, the updated set of corridor vertices is defined as:

$$V'_C = V_C \setminus min\{j | j \in V_C\} \tag{2}$$

For every corridor vertex j in V'_C and room v in V, we define:

$$P_j = \{v \in V | v \text{ belongs to same face as } j \text{ in } G_C\} \text{ and } Q_v = \{j \in V_C | (j,v) \in E(G_C)\} \tag{3}$$

$$P = \{P_j | j \in V'_C\} \text{ and } Q = \{Q_v | v \in V\} \tag{4}$$

By considering the set of rooms V as the set X and the set P as the set S, we do some pre-processing and then use Algorithm 4 as a sub-routine in the following algorithm.

Working of Algorithm 5:
With the aid of an example, we will explore the process of transforming our problem instance into a minimum set-cover problem and utilizing Algorithm 4 as a sub-routine to achieve our objective (refer to Figure 14).

Our objective is to connect (cover) all rooms using corridors, the set of rooms V can be seen as the set X of the Minimum Set-Cover problem. To determine the coverage of each corridor (set of rooms it connects in F_C), we defined the set P_j for each corridor vertex j (refer to Equation 3). Since $P_j \subseteq V \; \forall j \in V'_C$ (refer to Equation 2), we can deduce that these P_j's serve as the subsets S_i in the Minimum Set-Cover problem. Additionally, to ensure no room is left uncovered, we introduced an auxiliary set Q_v for every room vertex $v \in V$, which contains the corridor vertices it is adjacent to in G_C (refer to Equation 3).

In our example, we have the following sets:

$$X = V = \{0,1,2,3,4,5,6,7\} \tag{5}$$

$$V'_C = \{9,10,11,12,13,14\} \tag{6}$$

$$P_9 = \{0,1,3,4\}, P_{10} = \{0,1,3,2\}, P_{11} = \{0,1,4,5\}, \tag{7}$$

$$P_{12} = \{1,2,4,5\}, P_{13} = \{1,2,5,7\}, P_{14} = \{1,4,5,6\} \tag{8}$$

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

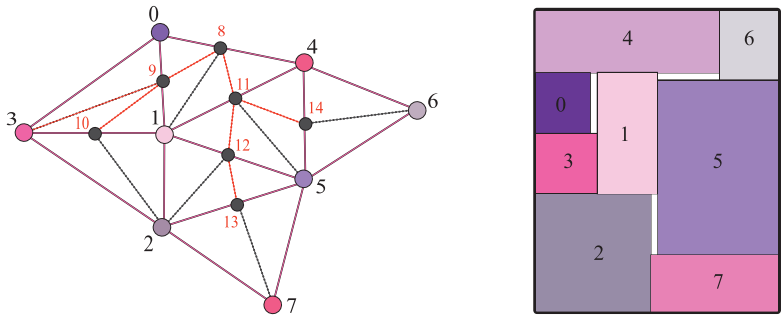


Figure 15. The circulation graph G_c (from Algorithm 1) with the set of non-redundant corridors marked in red and the floorplan $\mathbb{F}$ (from (Shekhawat, 2018)) featuring minimal circulation.

Algorithm 5: MinCirc(P, Q, V)

Input: $P := \{P_j |\ j \in V_C\}$ (where $P_j = \{v \in V |\ (j, v) \in E(G_C)\}$, V_C is defined in equation 2)
$Q := \{Q_v |\ v \in V\}$ (where $Q_v = \{j \in V'_C |\ (j, v) \in E(G_C)\}$, V'_C is defined in equation 2)
V : The set of room vertices
Output: C' : The minimized set of corridor vertices

$C' \leftarrow \phi$ // Initialize C' to be empty
$M \leftarrow \phi$ // Initialize minimum set cover as empty
$U \leftarrow V$
for $i \leftarrow 1$ *to* $|U|$ **do**
  // If there is only one corridor connecting the room i then add it to C'
  if $|Q_i| == 1$ **then**
    $C' \leftarrow C' \cup \{i\}$
    $U \leftarrow U \setminus \{i\}$ // We remove this since this room is already covered
if $U \neq \phi$ **then**
  $M \leftarrow$ Greedy-Set-Cover(U, P) // Calling the minimum set cover subroutine
$C' \leftarrow C' \cup M$ // Final minimized list of corridors is obtained
return C'

$$Q_0 = \{9\}, Q_1 = \{9, 10, 11, 12\}, Q_2 = \{10, 12, 13\}, Q_3 = \{9, 10\} \quad (9)$$

$$Q_4 = \{11, 14\}, Q_5 = \{11, 12, 13, 14\}, Q_6 = \{14\}, Q_7 = \{13\} \quad (10)$$

$$P = \{P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}\}, Q = \{Q_0, Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7\} \quad (11)$$

From the sets defined in the equations 5 to 11, it can be observed that the rooms 0, 6 and 7 are made accessible by only one corridor each as $|Q_0| = |Q_6| = |Q_7| = 1$. Therefore, the corresponding corridor vertices 9, 14 and 13 will definitely be there in the minimum set of non-redundant corridors. We observe that $P_9 \cup P_{13} \cup P_{14} = V = X$, i.e., the corridor vertices 9, 13, and 14 covers the whole set X, and thus directly gives us the minimum set of non-redundant corridors to be $C' = \{9, 13, 14\}$. In the event that some rooms in X were left uncovered, {9, 13, 14} would not be the minimum set of non-redundant corridors. To find the minimum set of non-redundant corridors, we would have run the subroutine 4 on the remaining uncovered rooms and P.

Once the set C' of non-redundant corridors is obtained, Algorithm 2 is used to calculate shift values for the rooms only to insert the corridors corresponding to the corridor vertices in set C'. This process results in a floorplan with the minimal set of corridors, ensuring accessibility to all rooms (refer to Figure 15).

In Figure 15, we can observe that there are only 3 corridor spaces. To ensure accessibility throughout the floorplan, certain rooms must be designated as public. This is achieved by designing rooms with entrances on walls facing a corridor, thereby enabling seamless navigation. Rooms with walls facing multiple corridors include more than one door, serving as transition points to maintain connectivity across the floorplan. For instance, in the corridor surrounded by rooms 0, 1, 3, and 4, all four rooms feature a door on the wall facing this corridor. To facilitate movement within this floorplan, room 5 will have two doors: one on the top wall (facing the corridor surrounded by 1, 4, 6, and 5) and another on the left wall (facing the corridor surrounded by 1, 2, 5, and 7). Similarly, room 4 will have two entrances on the bottom wall, with one opening in the corridor surrounded by the rooms 0, 1, 3, and 4, and the other in the corridor surrounded by the rooms 1, 4, 6, and 5. Alternatively, for seamless movement throughout this

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

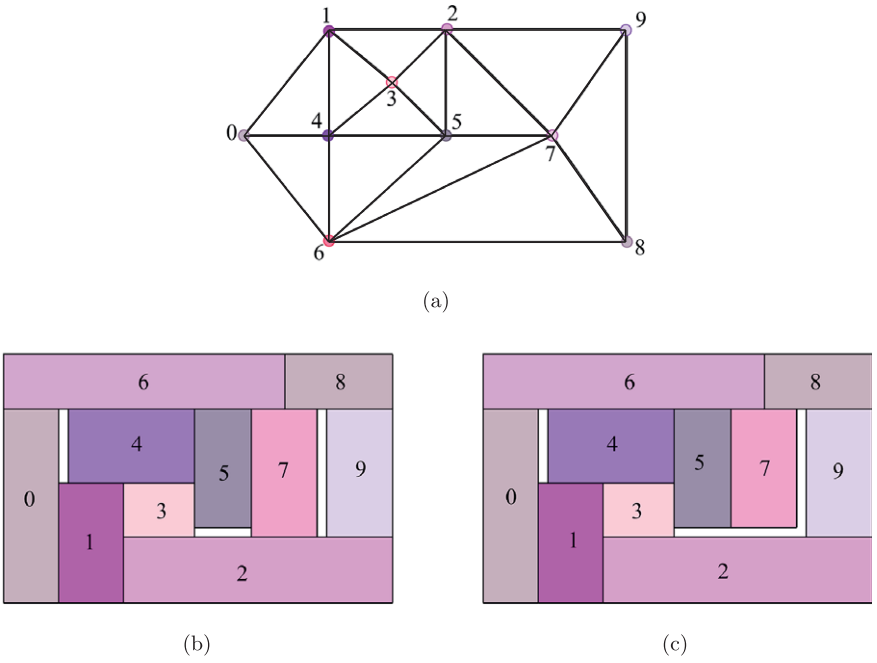


Figure 16. (a) The input to the GUI for which we want the minimal set of corridors with entry between rooms 2 & 9 (b) The expected minimal corridor space (c) The sub-optimal minimized set of corridors given by Algorithm 4 (redundant corridor between rooms 2 & 7).

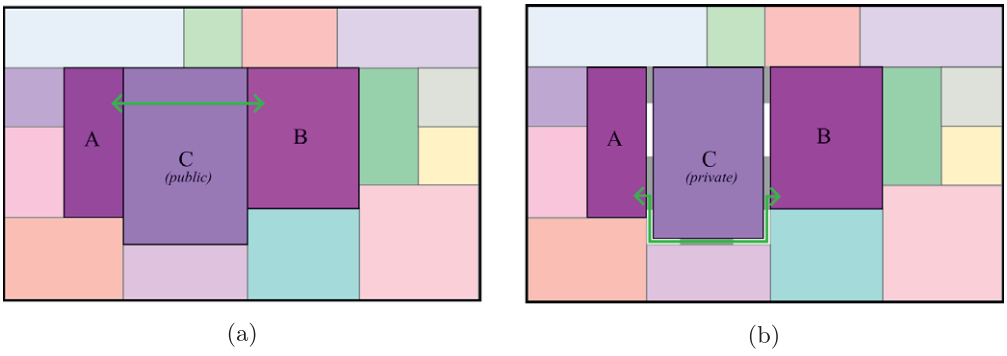


Figure 17. (a) If Room *C* is *public*, we can move from *A* to *B* through *C* without a corridor (b) If Room *C* is *private*, we have to create a corridor similar to the one marked by the grey stripes and use it or other adjacent public rooms to move between *A* and *B*.

floorplan, room 1 can have three doors, one on each wall facing a corridor, while other rooms have exactly one door opening into any one of the connected corridors.

Note: The minimal circulation space is generated independently of room designation. Once the circulation layout is established, some rooms are made 'public' to ensure accessibility for all other rooms. This is achieved by designing entrances on every wall that faces a corridor.

Limitations of the approximation algorithm

As previously explained, the minimization of the corridor space is a problem equivalent to the *Minimum set-cover* problem. Based on the understanding of complexity classes, we know that this problem belongs to the NP-hard class of problems, implying that no polynomial time algorithm exists to solve this problem. Due to the inherent complexity of NP-hard problems, we could only approximate the solution using approximation algorithms such as Algorithm 4. As a result, there may be instances where the algorithm returns a sub-optimal solution to the optimization problem.

As shown in Figure 16, there could be cases where the obtained corridor space might not be the optimal minimized corridor space but a sub-optimal one. This limitation is an intrinsic characteristic of the minimization problem. Nevertheless, we are actively exploring various alternatives to reduce such sub-optimal cases. One such alternative is the application of the above-mentioned greedy algorithm (Algorithm 4) consecutively, aiming to further eliminate redundant corridors.

Customization of the circulation space according to user-specified privacy constraints

In any building, it is common for the users to designate certain rooms as *private*, accessible only to a specific group of individuals, while the others are considered as *public*. Typically, in residences, bedrooms, storage rooms, and occasionally kitchens are kept

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

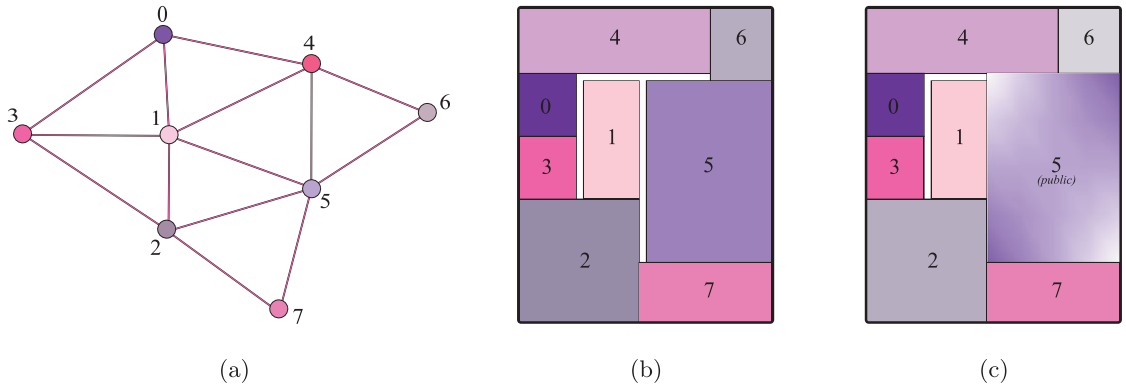


Figure 18. (a) The input graph. (b) The spanning circulation (like mentioned in previous algorithm the entry corridor, here $0-1$, is not explicitly shown since it is known to be a door). (c) The corridor placement if the room 5 alone is made *public*.

private, whereas other rooms made open for public access. In the case of office buildings or factories, areas like server rooms, storage spaces, and specific security zones are restricted to authorized personnel. Consequently, these restrictions influence the layout of corridors within such buildings.

Consider the following scenario (see Figure 17): Rooms *A* and *B* each share a wall with Room *C*. If Room *C* is designated as *public* space, one can move from *A* to *B* (or vice versa) through Room *C*. Conversely, if Room *C* is *private*, the only route to move between *A* and *B* would be through a corridor around Room *C*.

Building on this fundamental idea, next, we recall the definition of *private* and *public* rooms. Following this, we describe the algorithm designed to handle these constraints while generating the floorplans with the required circulation space. In addition to other inputs, the user also provides a list of *private* and *public* rooms.

Private Room: A *private room* is the one that is only accessible via a public space or a corridor; it cannot be crossed to get access to other areas of the floorplan.

In other words, when going from Room *A* to *B*, one will not visit any *private rooms* unless Room *A* or *B* is private. There will always exist a path from Room A to Room B via corridors and other *public rooms*. There is no restriction on the number of entry doors to any arbitrary room, but it can be assumed that a *public room* can have more than one door.

Illustration of Algorithm 6:
To understand the working of Algorithm 6, we will look at the approach used to address privacy constraints with the help of an example (refer to Figure 18).

Our central idea is to position corridors only around private rooms, ensuring that movement from one room to another does not necessitate passing through a private room, as defined. Additionally, given that free-movement is permitted in public rooms, there is no need for a corridor around them. In fact, a public room can be regarded as a corridor in itself. This approach helps prevent a reduction in room area, as constructing corridors would otherwise result in a smaller room area for a given plot size.

To obtain the required circulation space, first, we obtain the spanning circulation for the given floorplan using Algorithm 2. Next, we get the list of *public* and *private* rooms from the user. By analyzing the circulation graph of the spanning circulation, we obtain the set C' of corridors that do not surround any *public* rooms. Subsequently, we use Algorithm 2 to calculate shift values for the rooms to insert the corridors corresponding to the corridor vertices in set C'. The resulting floorplan satisfies the privacy constraints defined by the user (refer to Figure 19).

To demonstrate the use of public rooms, here are a few paths facilitating movement between different rooms in the floorplan shown in Figure 19:

$$2 \rightarrow 5 \rightarrow 4$$
$$7 \rightarrow 5 \xrightarrow{\text{via corridor}} 3$$
$$3 \xrightarrow{\text{via corridor}} 5 \rightarrow 6$$
$$0 \xrightarrow{\text{via corridor}} 5$$

It can be observed that in all of the paths, the private rooms serve as either the source or destination but never as an intermediate room. Now that we have a general understanding of Algorithm 6, let us formally define the Algorithm.

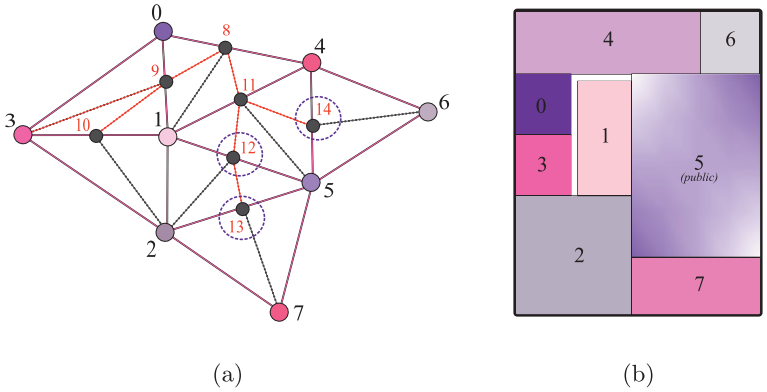


Figure 19. (a) The circulation graph corresponding to the spanning circulation. The corridor vertices excluding the encircled ones are the ones that will be considered for the final floorplan. (b) The final floorplan with the corridor placement according to the constraint that only room 5 is *public* and the rest are *private*.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Algorithm 6: PublicPrivate(G, (p, q))

Input: G: The input graph by user
(p, q): Exterior eedge corresponding the entry door
Output: F_c' : The floorplan with corridors satifying privacy constraints

Step 1 Get the spanning circulation corresponding to G and (p, q)
$G_c, A \longleftarrow$ SpanCirc$(G, (p, q))$ `// Spanning circulation using Algorithm 1`
`// F: floorplan corresponding to G, t: corridor thickness given by user`
$F_c \longleftarrow$ FpCirc$(G_c, F, A, (p, q), t)$ `// Modified floorplan with corridors using Algorithm 2`
Step 2 Privacy constraints
$R \longleftarrow$ User input of *public* rooms
$C' \longleftarrow$ List of corridors vertices not adjacent to any of vertices in R `// Obtained using` G_c, A
$A' \longleftarrow \{(v, (x, y)) \in A | v \in C'\}$ `// Update the adjacencies`
Step 3 Final required floorplan
$F_c' \longleftarrow$ FpCirc$(G_c, F, A', (p, q), t)$ `// Final floorplan with updated` A' `using Algorithm 2`
return F_c'

Algorithm validation and complexity

Mathematical Validation: Algorithm 1:

To mathematically validate Algorithm 1, we confirm it preserves the properties of a PTPG while generating a circulation graph.

i. **Input Validity:** The input graph $G(V, E)$ is a valid PTPG with n vertices, m edges, and f faces satisfying Euler's formula $n - m + f = 2$.
ii. **Planarity and Triangulation:** The algorithm maintains planarity by adding vertices through edge subdivision and forming new triangular faces. Each new vertex is connected to adjacent vertices, ensuring the resulting graph G_C remains planar and fully triangulated.
iii. **Connectivity and Spanning:** Every new corridor vertex connects to at least two existing vertices, forming a connected subgraph. The algorithm ensures that every room in the original graph is adjacent to at least one corridor vertex, providing a complete spanning circulation.
iv. **Termination:** The algorithm processes each face once, and since the number of faces f is finite, it always terminates.

Time Complexity Analysis: Algorithm 1:

i. **Input Initialization:** The input graph $G(V, E)$ contains n vertices and m edges. Initializing the graph takes linear time in terms of the vertices and edges. Time Complexity: $O(n + m)$.
ii. **Processing the First Corridor (Step 2):** In this step, one vertex is added by subdividing an edge, and the adjacency between this new corridor vertex and the surrounding vertices is updated. Since this involves a constant number of operations: Time Complexity: O(1).
iii. **Main Loop: Growing the Circulation (Step 3):** The algorithm processes each face adjacent to previously subdivided faces. For each face, a new vertex is inserted, and edges are added to maintain adjacency and triangulation.
   1. Subdividing a face (inserting a vertex and adding edges) takes constant time, $O(1)$.
   2. This loop runs once for each face; hence the total time depends on the number of faces, f. Since a PTPG graph has a linear relationship between its faces and vertices ($f = O(n)$), the total time for this step is proportional to the number of faces: Time Complexity: $O(f) = O(n)$.
iv. **Triangulation (Maintaining PTPG):** The graph remains a triangulated PTPG throughout the algorithm by adding edges to maintain triangular faces. Each edge might be subdivided once, but since subdivision is constant-time per face, the overall triangulation step takes linear time in the number of edges. Time Complexity: $O(m)$.
v. **Adjacency Matrix Calculation:** Building the adjacency matrix to represent the modified graph structure takes $O(n^2)$ time in the worst case, as each vertex may be adjacent to any other vertex.

Overall Time Complexity: $O(n^2)$ (since $m = O(n)$) + $O(n)$ (since $f = O(n)$) + $O(n^2) = O(n^2)$).

Mathematical Validation: Algorithms 2 and 3

1. **Correctness:**
   i. Algorithm 2 iterates through all corridor vertices, ensuring that each corridor is processed.
   ii. Algorithm 3 recursively explores common neighbors, ensuring all affected rooms are considered.
   iii. The use of Shift Tables ensures that wall shifts are correctly tracked and applied.
2. **Termination:**
   i. Algorithm 2 terminates when all corridor vertices in V' are processed.
   ii. Algorithm 3 recursively explores common neighbors, ensuring all affected rooms are considered.
   iii. The use of Shift Tables ensures that wall shifts are correctly tracked and applied.
3. **Invariants:**
   i. The planarity of the floorplan is maintained throughout the process.
   ii. The connectivity of rooms is preserved while adding corridors.

Complexity Analysis: Algorithms 2 and 3
Algorithm 2:

i. Let n be the number of rooms and m be the number of corridors.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

ii. Sorting V' takes $O(mlogm)$ time.
iii. The main loop iterates m times, once for each corridor.
iv. For each iteration: Finding common walls and shift values: $O(1)$
v. Updating room coordinates: $O(n)$

Overall time complexity: $O(mlogm + m * T(Algorithm3) + n)$, where $T(Algorithm3)$ is the time complexity of Algorithm 3.

Algorithm 3:

i. Let d be the maximum degree of a vertex in the graph.
ii. Finding common neighbors: $O(d)$
iii. The loop iterates at most d times. For each iteration: Finding common walls: $O(1)$.
iv. The recurrence relation: $T(d) = d * (O(1) + T(d — 1))$. Solving this, we get: $T(d) = O(d!)$.

However, in practice, the depth of recursion is limited by the planarity of the graph and the orientation condition, so the actual runtime is likely to be closer to $O(d^2)$ or $O(d^3)$.

The mathematical proof for Algorithm 4, along with the time complexity analysis for both Algorithms 4 and 5, is provided in Appendix title "Greedy set cover".

The validation and complexity analysis for Algorithm 6 follow a similar approach to Algorithm 1, as the circulation graph is constructed in the same manner, with the exception that corridor vertices corresponding to public rooms are removed.

Results and discussion

In this study, we have developed an innovative approach to automate the generation of circulation designs within floorplans. Our method successfully integrates circulation spaces within the floorplans derived from a given PTPG. The process is divided into four key steps, as detailed in Section titles "Generating a floorplan and circulation graph for the given PTPG" to "Customization of the circulation space according to user-specified privacy constraints". These sections demonstrate the working of our algorithms by incorporating various circulation types in a rectangular floorplan corresponding to any given PTPG. However, it's important to note that our algorithm is not limited to rectangular shapes. It can be easily adapted to floorplans with non-rectangular outer boundaries as well (see Figures 27 and 28).

The initial stage involves generating a floorplan from the PTPG and creating a circulation graph using Algorithm 1. This circulation graph effectively captures potential movement paths within the layout, serving as the foundation for adding various circulation spaces. Subsequently, Algorithm 2 is employed to integrate spanning circulation, ensuring all rooms are connected for full accessibility.

Recognizing that spanning circulation can create redundant corridors leading to inefficient space use, we developed Algorithm 5 to optimize the design. This algorithm transforms the spanning circulation into a minimal circulation, significantly reducing redundant corridors while maintaining connectivity. This optimization is particularly crucial in designs where space-saving is a priority. To enhance the flexibility and applicability of our approach, we introduced Algorithm 6, which allows for customization based on privacy constraints. Users can designate certain rooms as private and others as public, enabling the creation of tailored designs that balance accessibility with privacy. This feature makes our approach adaptable to various architectural needs, such as office or residential layouts.

Note: Minimal circulation and public-private circulation approaches offer distinct benefits in floorplan design. Minimal circulation optimizes space efficiency by reducing redundant corridors, potentially lowering costs while ensuring connectivity. It simplifies navigation and maximizes area for primary functions. Public-private circulation provides greater control over privacy and security, offering flexibility in creating distinct functional zones and enabling public rooms to serve dual purposes. The choice between these approaches depends on specific building requirements. Minimal circulation is ideal for scenarios without strict privacy restrictions, suitable for open-plan offices or public buildings where flexibility is desired. Public-private circulation is more appropriate when greater control over space designation is required, well-suited for buildings with distinct privacy needs such as healthcare facilities or mixed-use complexes. This approach balances accessibility and privacy, crucial in designs where controlling access to certain areas is important.

We have implemented all our algorithms in Python with a user-friendly graphical interface (GUI), which improves the accessibility of our tool. This interface allows architects and designers to easily input constraints, visualize results, and iterate designs rapidly. The ability to observe results "in a fraction of the time" compared to traditional methods represents a significant improvement in the efficiency of design workflow.

Limitations

While our approach addresses many challenges in automated circulation design, there are areas for future development.

i. Incorporating more complex constraints, such as building codes and accessibility requirements, could further improve the real-world applicability of the generated designs.
ii. The approach currently focuses on 2D floorplans and does not automatically generate 3D representations or multi-story circulation, which could be studied further.
iii. One limitation of the *Minimal Circulation* methodology is that the designation of public rooms to ensure accessibility is performed after the circulation space is generated. Designating rooms as 'public' with multiple doors can affect privacy in environments where it is important, often requiring additional adjustments or re-designations to address this issue effectively. This approach requires users to manually make adjustments to refine the circulation design and ensure that it meets specific accessibility and functional requirements. However, this limitation can be partially addressed using the public-private circulation customization algorithm.

Walkthrough: using our approach for automated circulation design

Here we provide a step-by-step guide on how to use our approach for generating floorplans with optimized circulation. The steps are demonstrated in the examples shown in Figures 20–26 to illustrate the process. Additionally, a working demonstration of the graphic user interface (GUI) is available online on the link provided in Appendix A:

1. **Initial input**
   The user begins by preparing a PTPG representing the desired room adjacencies. This graph serves as the foundation for the floorplan generation.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

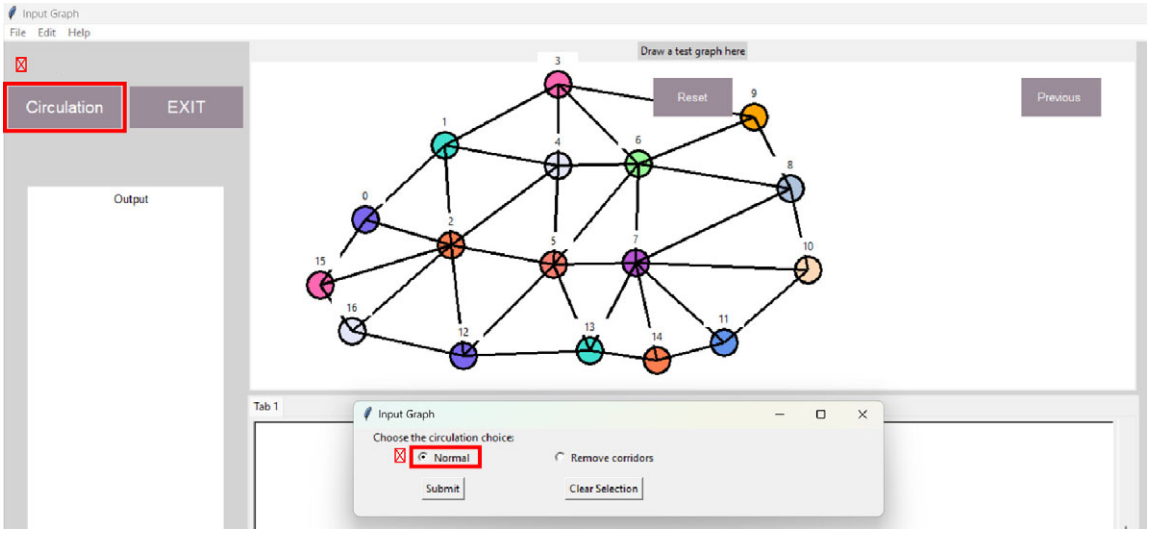


Figure 20. User draw a PTPG as input graph and chooses the option of circulation and then "*normal*" when the pop-up appears.

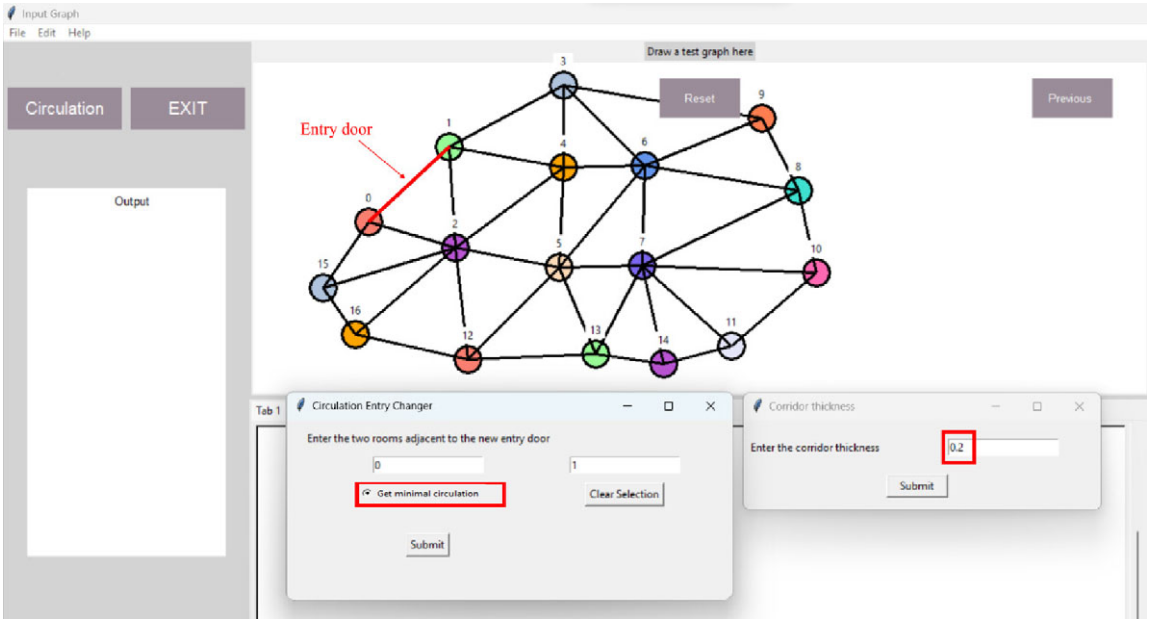


Figure 21. For the input graph in Figure 20, the user selects the Circulation option, specifies the entry door, sets the corridor thickness, and chooses either 'Get minimal circulation' for optimized space or 'Submit' for spanning circulation.

2. **Generating spanning Circulation:** (refer to Figures 20, 21, and 25)
   i. Click on the 'Circulation' button in the user interface.
   ii. Choose 'Normal' as the circulation option.
   iii. Input the entry door location and desired corridor thickness.
   iv. Click 'Submit' to generate the floorplan with spanning circulation (see Figure 25 for spanning circulation).
3. **Optimizing Circulation Space:** To minimize circulation space and optimize room sizes (refer Figures 21 and 22).
   i. Choose 'Get Minimal Circulation' from the options.
   ii. Input the entry door location and corridor thickness.
   iii. Submit to generate a floorplan with minimized circulation space.
4. **Customizing Circulation (Optional):** If the user wishes to eliminate specific corridors (refer Figures 23–26).
   i. Click the 'Circulation' button again.
   ii. Select the 'Remove Corridor' option.
   iii. For each pair of rooms, input '0' to remove the corridor or '1' to keep it.
   iv. Submit to update the circulation design.

Note: This option may suggest using certain rooms for transit as well as their primary purpose.

5. **Implementing Privacy Constraints:** For more control over public and private spaces.
   i. Provide a list specifying which rooms should be private or public.
   ii. Submit to generate a circulation design that includes corridors around private rooms while allowing public rooms to serve dual purposes (transit and primary use).

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

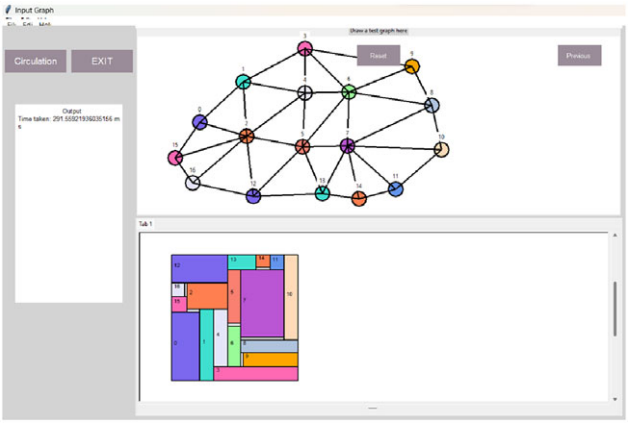


Figure 22. The GUI displays the floorplan with minimal circulation. Note that every room has at least one of its walls facing a shared corridor. The rooms which share a wall instead of a corridor can have an internal connection or no connection as per user preferences.

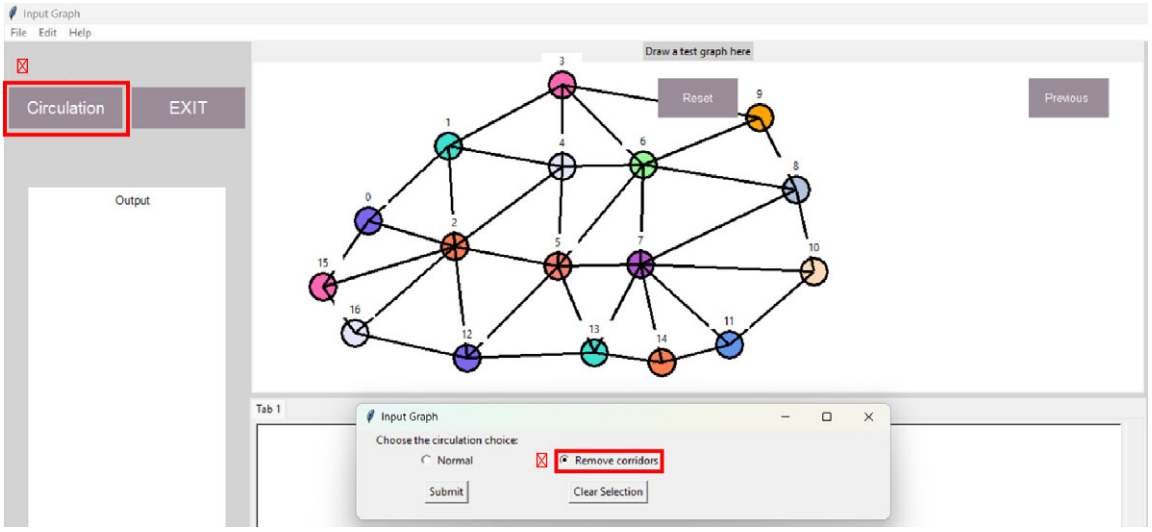


Figure 23. For the same input graph. user checks “Remove circulation” and chooses the option of circulation.

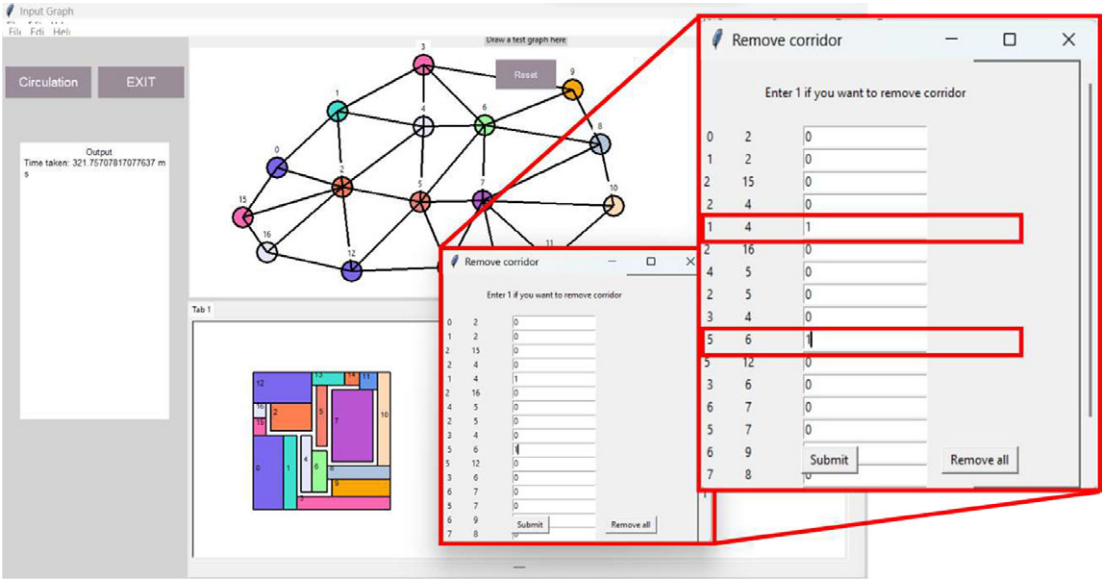


Figure 24. The GUI shows a possible spanning circulation and displays a window where user can remove any set of corridors at once. The pair of numbers displayed to the left of a text box denotes the pair of rooms on either side of that corridor. For example, as per image on right, if user wants to remove corridor between rooms 1 and 4, then the user enters “1” in the text box beside the pair “1 4”. There is also a button which allows user to remove all corridors and give back the corresponding floorplan.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

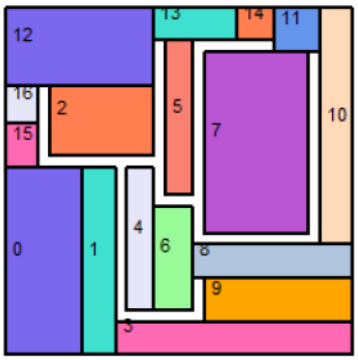


Figure 25. The initial spanning circulation.

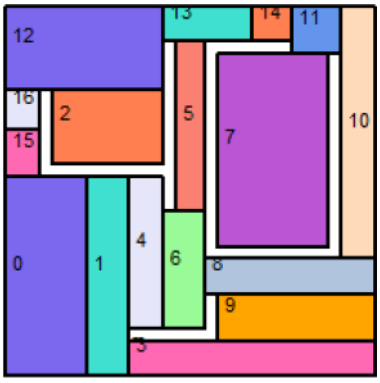


Figure 26. The Spanning circulation has been updated, removing corridors between room pairs (1,4) and (5,6).

Suppose there's a need for additional corridors beyond the minimal circulation but lesser than the spanning circulation. In that case, users can remove specific corridors after generating the spanning circulation, if necessary. Figures 23–26 illustrate this option for removing corridors.

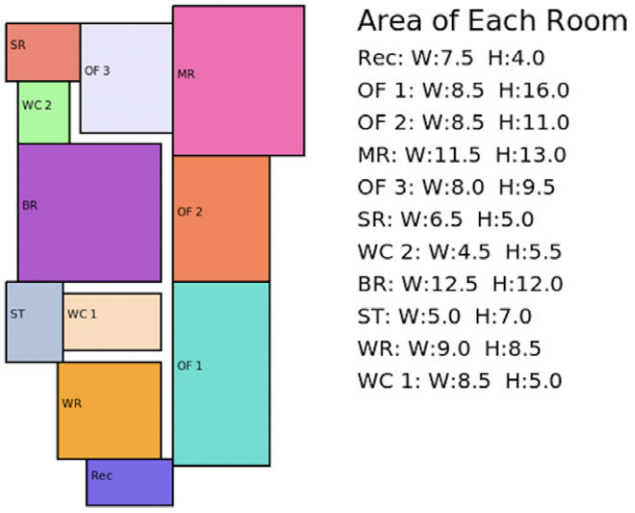


Figure 28. A floorplan for the given input graph featuring spanning circulation.

Circulation designs for a small office layout

To explore the practical applicability of our software, we evaluate its functionality in generating circulation designs for a small office layout. Figures 27 and 28 illustrate a floorplan of the office, demonstrating how the approach addresses non-rectangular boundaries and generates a spanning circulation layout. This layout is further refined using minimal circulation while designating certain rooms as public (see Figure 29). The configuration of the office layout is represented using the following permutation: ['Reception (Rec)', 'Office1 (OF1)', 'Office2 (OF2)', 'Meeting Room (MR)', 'Office3 (OF3)', 'Server Room (SR)', 'WC2', 'Break Room (BR)', 'Store (ST)', 'Waiting Room (WR)', 'WC1'].

The floorplan shown in Figure 28 includes a spanning circulation that can be optimized using the minimal circulation feature. Corridors 1, 2, and 3, as depicted in Figure 29(a), are redundant and can be eliminated by designating the Break Room

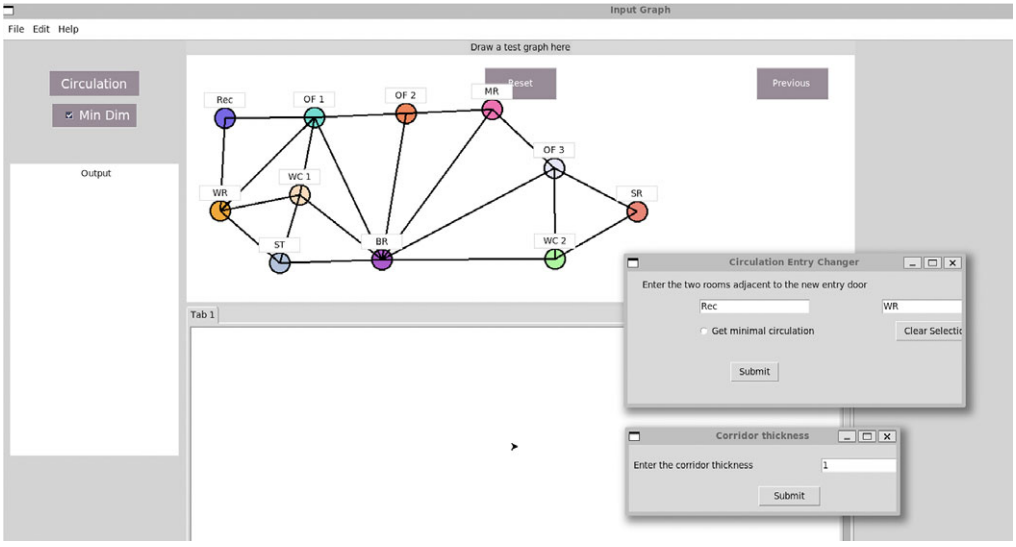


Figure 27. An input graph corresponding to the room list for a small office, selecting entrance of the layout and thickness of the required corridor.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

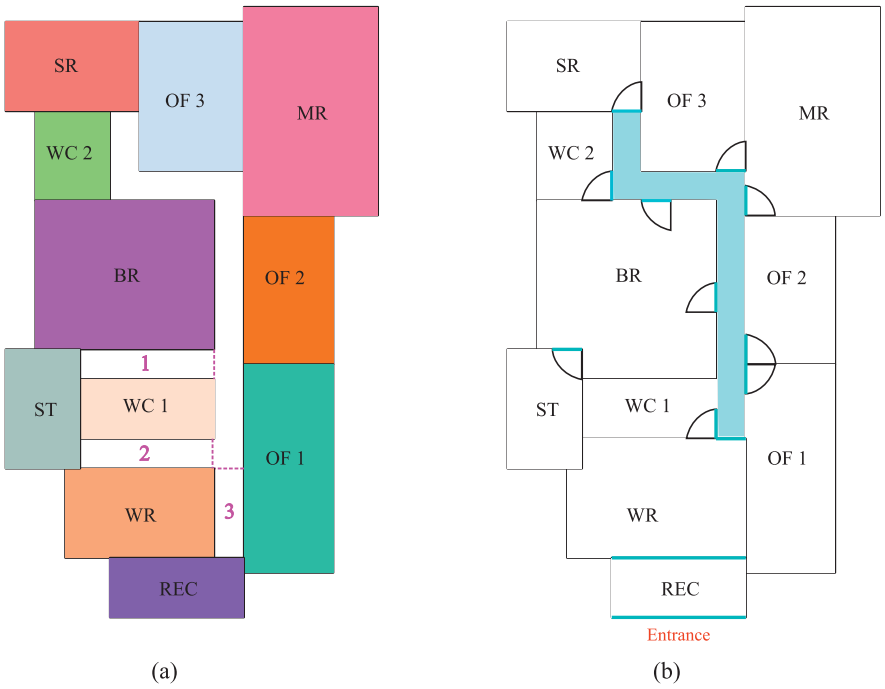


Figure 29. Optimizing the circulation space: (a) Initial floorplan with redundant corridors (1, 2, and 3), (b) Optimized floorplan after designating the Break Room (BR) and Waiting Room (WR) as public rooms.

(BR) and Waiting Room (WR) as public rooms. The resulting optimized floorplan is presented in Figure 29(b).

Conclusion and future enhancements

In this paper, we addressed the challenge of integrating corridors into a floorplan using a graph-theoretic technique. We presented a series of algorithms designed to systematically insert corridors, starting with the modification of the input graph to translating the changes in the floorplan by modifying the room coordinates. Additionally, we delved into the optimization aspect, aiming to enhance cost-efficiency in construction by removing redundant corridors. To achieve this optimization, we utilized the renowned Minimum set-cover problem. Based on our findings, we developed a Python-based application to automate circulation within a floorplan and presented example from the GUI.

Building upon this foundation, our future work aims to enhance the system's flexibility, usability, and functionality, as outlined below:

i. **Enhanced Input Methods:** At present, our algorithm accepts input in the form of a PTPG, necessitating users to be familiar with its associated terms. In our future development, our goal is to allow users to input data using various parameters, such as the number of rooms, desired relations (both adjacency and non-adjacency) between rooms, and dimensional constraints. Users will be able to provide this input through a connectivity graph, specifying room adjacency in terms of wall and door connections. With this information, we will construct the necessary graph and generate a corresponding floorplan, offering a more flexible and user-friendly input method.

ii. **Integration with existing designs:** To support remodeling and existing floorplans, the system will be enhanced to analyze architectural elements such as rooms, doors, and openings from pre-existing designs. From this analysis, an adjacency graph will be automatically generated, representing rooms as nodes and connections between them (via doors or openings) as edges, with additional attributes such as room types and dimensions encoded.

iii. **Refined spatial relationships:** Future models will include multiple connectivity types, such as door and wall connectivity, represented by distinct edge sets to better capture spatial relationships. In addition to the adjacency graph, a door connectivity graph will be created to capture how rooms are linked through doors and openings. These graphs will be analyzed to extract the layout's topological structure and spatial relationships. Building on the extracted data, our system will apply the algorithms detailed in this paper to integrate various circulation types within the existing floorplans, ensuring the generated designs maintain core functional and spatial relationships.

iv. **Directional and functional constraints:** Cardinal direction constraints will be introduced to position rooms based on requirements like daylight access or ventilation. For instance, rooms needing northern exposure will be placed along the northern boundary.

v. **Plot boundaries:** An algorithm to accommodate non-rectangular plot boundaries as an input constraint is currently under development. This functionality will also be extended to support non-rectangular room shapes, improving the system's adaptability.

Competing interest. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

References

Aalaei M, Saadi M, Rahbar M and Ekhlassi A (2023) Architectural layout generation using a graph-constrained conditional generative adversarial network (GAN). *Automation in Construction* **155**, 1–17.

Bhasker J and Sahni S (1986) A linear algorithm to find a rectangular dual of a planar triangulated graph. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference, DAC '86*. IEEE Press, pp. 108–114.

Bhasker J and Sahni S (1988) A linear algorithm to find a rectangular dual of a planar triangulated graph. *Algorithmica* **3**(1), 247–278

Bisht S, Shekhawat K, Upasani N, Jain RN, Tiwaskar RJ and Hebbar C (2022) Transforming an adjacency graph into dimensioned floorplan layouts. *Computer Graphics Forum* **41**(6), 5–22.

Bondy JA and Murty USR (1976) *Graph Theory with Applications*. Elsevier, Vanderbilt Avenue, New York, N.Y.

Cormen TH, Leiserson CE, Rivest RL and Stein C (2001) *Introduction to Algorithms*, 2 Edn. The MIT Press.

Eastman C (2009) Automated assessment of early concept designs. *Architectural Design* **79**(2), 52–57.

Eppstein D, Mumford E, Speckmann B and Verbeek K (2009) Area-universal rectangular layouts. *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry* **41**(3), 267–276.

Evans R (1997) Figures, doors and passages. In *Translations from Drawing to Building and Other Essays*. Architectural Association, pp. 55–91.

González JF and Gongal A (2021) Unidirectional pedestrian circulation: physical distancing in informal settlements. *Buildings and Cities*. **2** (1), 655–665

Han Z, Xiaoqian L, Yuan Y and Stouffs R (2024) Graph2pix: A generative model for converting room adjacency relationships into layout IM-ages. In *Proceedings of the 29th International Conference of the Association for Computer-Aided Architectural Design Research in Asia (CAADRIA) 2024*, pp. 139–148.

He X (1993) On finding the rectangular duals of planar triangular graphs. *SIAM Journal on Computing* **22**(6), 1218–1226.

He X (1999) On floor-plan of plane graphs. *SIAM Journal on Computing* **28**(6), 2150–2167.

Hillier B. and Hanson J. (1984) *The Social Logic of Space*. Cambridge, Cambridge University Press.

Hu R, Huang Z, Tang Y, Van Kaick O, Zhang H and Huang H (2020) Graph2plan: learning floorplan generation from layout graphs. *ACM Transactions on Graphics (TOG)* **39**(4), 1–118.

Jarzombek M (2010) Corridor spaces. *Critical Inquiry* **36**(4), 724–744.

Jawaherul AM, Therese B, Stefan F, Michael K, Kobourov SG and Torsten U (2013) Computing cartograms with optimal complexity. *Discrete Computational Geometry* **50**, 784–810.

Jisoo K, Hyunsoo L, Minkyu S, Choib JW and Lee J-K (2014) Graph-based representation of building circulation with the most-remote points and virtual space objects. In *Proceedings of the 31st International Symposium on Automation and Robotics in Construction and Mining (ISARC)*. International Association for Automation and Robotics in Construction (IAARC), pp. 210–216.

Kant G and He X (1997) Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science* **172**(1), 175–193.

Kozminski K and Kinnen E (1985) Rectangular duals of planar graphs. *Networks* **15**(2), 145–157.

Kozminski K and Kinnen E (1988) Rectangular dualization and rectangular dissections. *IEEE Transactions on Circuits and Systems* **35**(11), 1401–1416.

Lee J-K, Eastman CM, Lee J, Kannala M and Jeong Y-S (2010) Computing walking distances within buildings using the universal circulation network. *Environment and Planning B: Planning and Design* **37**(4), 628–645.

Levin PH (1964) Use of graphs to decide the optimum layout of buildings. *The Architects' Journal* **7**, 809–815.

Li C, Jiang L, Sun F and Zhang K (2018) Generating circulation designs using shape grammars. *Tsinghua Science and Technology* **23**(6), 680–689.

March L and Steadman P (1971) *Geometry of the Environment*. London: RIBA Publications.

Mitchell WJ, Steadman JP and Liggett RS (1976) Synthesis and optimization of small rectangular floor plans. *Environment and Planning B* **3**(1), 37–70.

Mustafa FA and Ahmed SS (2023) The role of waiting area typology in limiting the spread of covid-19: Outpatient clinics of erbil hospitals as a case study. *Indoor and Built Environment* **32**, 1914–1928.

Mustafa FA and Azeez SA (2022) Role of office layout typology in saving time and distance spent by users: case of office buildings in erbil city. *Ain Shams Engineering Journal* **13**, 1–14.

Mustafa FA and Rafeeq DA (2019) Assessment of elementary school buildings in Erbil city using space syntax analysis and school teachers feedback. *Alexandria Engineering Journal* **58**, 1039–1052.

Naderpour A, Johnson B and Anderson A (2019) A2b: A toolkit for computing circulation metrics in buildings. *Proceedings of the 16th IBPSA Conference* **16**, 2576–2583.

Nauata N, Chang K-H, Cheng C-Y, Mori G and Furukawa Y (2020) House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. *In Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 162–177. https://doi.org/10.1007/978-3-030-58452-8_10

Nauata N, Hosseini S, Chang K-H, Chu H, Cheng C-Y and Furukawa Y (2021) Housegan++: Generative adversarial layout refinement network towards intelligent computational agent for professional architects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13632–13641.

Para W, Guerrero P, Kelly T, Guibas LJ and Wonka P (2021) Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6690–6700.

Rafeeq DA and Mustafa FA (2021) Evidence-based design: The role of inpatient typology in creating healing environment, hospitals in Erbil city as a case study. *Ain Shams Engineering Journal* **12**, 1073–1087.

Rahbar M, Mahdavinejad M, Markazi AD and Bemanian M (2021) Architectural layout design through deep learning and agent-based modeling: A hybrid approach. *Journal of Building Engineering* **47**, 103–122.

Raveena and Shekhawat K (2023) A theory of l-shaped floor-plans. *Theoretical Computer Science* **942**, 57–92.

Raveena, Shekhawat K and Shekhawat R (2024) A graph theoretic approach for generating t-shaped floor plans. *Theoretical Computer Science* **1011**, 1–29.

Sabir BM and Mustafa FA (2023) Performance-based building design: impact of emergency department layout on its functional performance efficiency - the case of erbil hospitals. *Open House International* **48**, 840–862.

Shekhawat K (2018) Enumerating generic rectangular floor plans. *Automation in Construction* **92**, 151–165.

Shekhawat K, Jain RN, Bisht S, Kondaveeti A and Goswami D () Graph-based approach for enumerating floorplans based on users specifications. *AI EDAM* **35**(4), 438–459.

Shekhawat K, Upasani N, Bisht S and Jain RN () A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction* **127**, 1–21.

Steadman P (1983) *Architectural Morphology: An Introduction to the Geometry of Building Plans*. London: Pion.

Sun J, Wu W, Liu L, Min We, Zhang G and Zheng L (2022) Wallplan: synthesizing floorplans by learning to generate wall graphs. *ACM Transactions on Graphics (TOG)* **41**(4), 1–14.

Taneja S, Akinci B, Garrett JH, Soibelman L and East B (2011) Transforming IFC-based building layout information into a geometric topology network for indoor navigation assistance. *Computing in Civil Engineering*, **2011** 315–322.

Tsiamitros N, Mahapatra T, Passalidis I, Kailashnath K and Pipelidis G (2023) Pedestrian flow identification and occupancy prediction for indoor areas. *Sensors* **23**(9), 1–26.

Upasani N, Shekhawat K and Sachdeva G (2020) Automated generation of dimensioned rectangular floorplans. *Automation in Construction* **113**, 103–149.

Vardouli T (2017) Thesis: Ph.D. in Architecture: Design and Computation, Massachusetts Institute of Technology, Department of Architecture, http://hdl.handle.net/1721.1/113917, Massachusetts Institute of Technology.

Wang L, Liu J, Zeng Y, Cheng G, Hu H, Hu J and Huang X (2023) Automated building layout generation using deep learning and graph algorithms. *Automation in Construction* **154**, 1–21.

Wang S, Zeng W, Chen X, Ye Y, Qiao Y and Fu C-W (2021) Actfloor-gan: activity-guided adversarial networks for human-centric floorplan design. *IEEE Transactions on Visualization and Computer Graphics* **29**(3), 1610–1624.

Wang X-Y, Yang Y and Zhang K (2018) Customization and generation of floor plans based on graph transformations. *Automation in Construction* **94**, 405–416.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press

Wang X-Y and Zhang K (2020) Generating layout designs from high-level specifications. *Automation in Construction* **119**, 1–16.
Wu W, **Fan L**, **Liu L and Wonka P** (2018) Miqp-based layout design for building interiors. *Computer Graphics Forum* **37**(2), 511–521.
Wu W, **Fu X-M**, **Tang R**, **Wang Y**, **Qi Y-H and Liu L** (2019) Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)* **38**(6), 1–12.
Xie X and Ding W (2023) An interactive approach for generating spatial architecture layout based on graph theory. *Frontiers of Architectural Research* **12**(4), 630–650.

Appendix A. Supplementary Material

An online demonstration of the Graphic User Interface, showcasing various types of circulation in floorplans is accessible at: Demonstration video (https://www.dropbox.com/scl/fi/ucwy1uc7cz9iahm6gsmf0/Circulation.mp4?rlkey=fg6ws4qkcyo1mkz4qek6o2cno&dl=0).

The implementation is available on GitHub at the following link: GitHub link (https://github.com/GPLAN-team/Circulation_Paper).

Appendix B. Analysis of algorithms

Greedy set cover

In this proof, we denote the k^{th} Harmonic number: $H_k = \sum_{j=1}^{k} 1/j$, as $H(k)$, with the boundary condition $H(0) = 0$.

Theorem 1. (Cormen et al., 2001) GREEDY-SET-COVER *is a* $O(log_e m)$*–approximation algorithm, where*

$$m = max\{|S_i|: S_i \in S\})$$

(i.e., If OPT is the optimal set-cover for the given problem instance (X, S) *and G is the set-cover by* GREEDY-SET-COVER, *then* $|G| \le O(log_e m)|OPT|$ *where* $|X| = n$*)*

Proof. (**NOTE:** Here, we prove this by utilizing the k^{th} Harmonic number: $H_k = \sum_{j=1}^{k} 1/j$, as $H(k)$, with the boundary condition $H(0) = 0$. So, we first prove that $|G| \le H(m)|OPT|$, and utilize the fact that $H_k = \sum_{j=1}^{k} 1/j \approx log_e k$)

Our idea is to first assign each subset $S_i \in S$ selected by the greedy algorithm a cost of 1 and distribute it across the elements $x \in X$ covered for the first time by S_i. Accumulating these costs will get us the required relationship between the greedy solution G and the optimal solution OPT. Let S_j be the subset added at the j^th iteration of the algorithm. Let c_x be the cost allocated to element $x \in X$, which is added only when x is covered by one of the subsets for the first time. If S_j covers element x for the first time:

$$c_x = \frac{1}{|S_j - (S_1 \cup S_2 \cup \cdots \cup S_{j-1})|}$$

Note that each step of the algorithm assigns a cost of 1 and hence:

$$|G| = \sum_{x \in X} c_x \tag{12}$$

Since each element $x \in X$ is at least in one of the subsets in OPT, we have

$$\sum_{B \in OPT} \sum_{x \in B} c_x \ge \sum_{x \in X} c_x \tag{13}$$

Now, combining equation 12 and inequality 13, we get

$$|G| \le \sum_{B \in OPT} \sum_{x \in B} c_x \tag{14}$$

Next, we focus on proving $\sum_{x \in B} c_x \le H(|B|)$ so that from inequality 14, we get

$$|G| \le \sum_{B \in OPT} H(|B|)$$
$$\Rightarrow |G| \le |OPT|.H(max\{|S_i|: S_i \in S\}) = |OPT|.H(m)$$

Consider an arbitrary subset $S_j \in S$ and for all $i = 1, 2, \ldots, |G|$ let

$$u_i = |S_j - (S_1 \cup S_2 \cup \cdots \cup S_i)|$$

be the number of elements of S_j that remain uncovered after the algorithm has chosen the subsets $S_1, S_2, \ldots S_i$. Since initially none of the elements are covered, we can fix $u_0 = |S_j|$. Let r be the least index such that all elements of S_j are covered by one of $S_1, S_2, \ldots S_r$ and at least one element of S_j is uncovered by $S_1 \cup S_2 \cup \cdots \cup S_{r-1}$. Consequently, $u_{i-1} \ge u_i$ and $u_{i-1} - u_i$ elements are covered for the first time by S_i for $i = 1, 2, \ldots, r$. So, we have

$$\sum_{x \in X} c_x = \sum_{i=1}^{r} (u_{i-1} - u_i) \cdot \frac{1}{|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})|}$$

Since S_i is a greedy choice, it covers more new elements than S_j (else algorithm will choose S_j over S*i*). Consequently,

$$|S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \ge |S_j - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})$$
$$\Rightarrow |S_i - (S_1 \cup S_2 \cup \cdots \cup S_{i-1})| \ge u_{i-1}$$
$$\Rightarrow \sum_{x \in S_j} c_x \le \sum_{i=1}^{r} (u_{i-1} - u_i).\frac{1}{u_{i-1}}$$
$$\Rightarrow \sum_{x \in S_j} c_x \le \sum_{i=1}^{r} \sum_{p=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}}$$
$$\Rightarrow \sum_{x \in S_j} c_x \le \sum_{i=1}^{r} \sum_{p=u_i+1}^{u_{i-1}} \frac{1}{p} \quad (\text{since } p \le u_{i-1})$$
$$\Rightarrow \sum_{x \in S_j} c_x \le \sum_{i=1}^{r} \left(\sum_{p=1}^{u_{i-1}} \frac{1}{p} - \sum_{p=1}^{u_i} \frac{1}{p} \right)$$
$$\Rightarrow \sum_{x \in S_j} c_x \le \sum_{i=1}^{r} (H(u_{i-1}) - H(u_i))$$
$$\Rightarrow \sum_{x \in S_j} c_x \le (H(u_0) - H(u_r)) \quad (\text{since the sum telescopes})$$
$$\Rightarrow \sum_{x \in S_j} c_x \le (H(u_0) - H(0))$$
$$\Rightarrow \sum_{x \in S_j} c_x \le H(u_0) \quad (\text{since } H(0) = 0)$$
$$\Rightarrow \sum_{x \in S_j} c_x \le H(|S_j|) \tag{15}$$

So, applying inequality 15 in the inequality 14, we get the result that

$$|G| \le \sum_{B \in OPT} H(B)$$
$$\Rightarrow |G| \le |OPT|.H(max\{|S_i|: S_i \in S\})$$
$$\Rightarrow |G| \le |OPT|.H(m) \quad (\text{Since } m = max\{|S_i|: S_i \in S\})$$
$$\Rightarrow |G| \le (\log_e m).|OPT| \quad (\text{Since } H(m) \approx \log_e m)$$

Therefore, we have shown that the size of the greedy set cover is at most $log_e m$ times the optimal set cover (where m is the size of the largest chosen subset in the greedy algorithm). Thus, the algorithm is a $O(log_e m)$-approximation algorithm.

https://doi.org/10.1017/S0890060425000022 Published online by Cambridge University Press