





Transforming an Adjacency Graph into Dimensioned Floorplan Layouts

Sumit Bisht,  Krishnendra Shekhawat,  Nitant Upasani, Rahil N. Jain, Riddhesh Jayesh Tiwaskar and Chinmay Hebbar

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

Abstract

In recent times, researchers have proposed several approaches for building floorplans using parametric/generative design, shape grammars, machine learning, AI, etc. This paper aims to demonstrate a mathematical approach for the automated generation of floorplan layouts. Mathematical formulations warrant the fulfilment of all input user constraints, unlike the learning-based methods present in the literature. Moreover, the algorithms illustrated in this paper are robust, scalable and highly efficient, generating thousands of floorplans in a few milliseconds. We present G2PLAN, a software based on graph-theoretic and linear optimization techniques, that generates all topologically distinct floorplans with different boundary rooms in linear time for given adjacency and dimensional constraints. G2PLAN builds on the work of GPLAN and offers solutions to a wider range of adjacency relations (one-connected, non-triangulated graphs) and better dimensioning customizability. It also generates a catalogue of dimensionless as well as dimensioned floorplans satisfying user requirements.

Keywords: computational geometry, modeling, CAD

CCS Concepts: • Mathematics of computing → Discrete mathematics

1. Introduction

Recent trends in computational design, more specifically, architectural design and VLSI design, depict a growing attention towards the automated generation of floorplan layouts. The simplicity of defining layouts as graphs—where the vertices represent rooms and the edges represent adjacencies among them, offers exceptional visualization and allows the possibility of modifications at a very early stage in the design process. A few recent works have adhered to the use of graph theory [WYZ18], but most studies have adopted AI and machine learning techniques for automated floorplan generation [HHT*20, MSK10]; while the prior offers a robust working (but a restricted scope), the latter delivers the prospects of designing layouts with highly specific user requirements at the cost of its complexity. Studies based on machine learning techniques are often supervised or data-driven, like Wu *et al.* [WFT*19] and Merrell *et al.* [MSK10], hence, their algorithms may fail to produce unconventional, yet desirable building layouts. This paper aims at presenting an efficient approach of generating floorplans, subject to strict constraints of connectivity and size of rooms. Table 1 shows a comparison between existing floorplan generation methods and our approach.

In this work, we present a prototype G2PLAN, which combines graph theoretical algorithms and optimization techniques to construct dimensioned floorplans while satisfying given adjacency and size constraints. G2PLAN works on the foundations of creating a rectangular dual for input graphs and then augments certain rooms to produce layouts with rectilinear rooms and boundaries. The *rectangular dual* of a graph $G(V, E)$ is defined as a dimensionless rectangular boundary layout, in which each room R_1, R_2, \dots, R_n is a rectangle corresponding to every $v \in V$ and each edge $e(i, j) \in E$ represents connectivity between rooms R_i and R_j . Generating a rectangular layout for an adjacency graph is mathematically challenging and has been extensively studied in the 70s and 80s. Not all graphs can have a rectangular dual, and it is, in itself, a difficult problem to distinguish between the two classes. Kozminski and Kinnen [KK84] were the first to prove that rectangular duals exist for a certain class of graphs, i.e. the properly triangulated planar graphs (PTPGs). Kozminski and Kinnen [KK85] also presented an algorithm to generate a rectangular dual for PTPGs, but Bhasker and Sahni [BS87] were the first to give a linear time algorithm for the same. Soon in 1990, Tang and Chen [TC90] showed that multiple rectangular duals corresponding to a biconnected graph can be obtained by a set of elementary transformation rules. We

Table 1: Features of different floorplan approaches.

Features	[WWSR03]	[MSK10]	[WYZ18]	[USS20]	[WFT*19, HHT*20]	G2PLAN
Method used	Shape Grammar	Bayesian Methods	Graph Theory	Graph Theory	Deep Learning	Graph Theory
Incorporates adjacency constraints	—	X	X	X	X	X
Incorporates dimensional constraints	—	X	—	X	—	X
Generates non-rectangular floorplans	X	X	—	—	X	X
Generates multiple floorplans	—	—	X	—	X	X
2D/3D	3D	3D	2D	2D	2D	2D

identify all rectangular duals as *topological solutions* of the PTPG in consideration.

In the real world applications of design, dimensioning plays a pivotal role as the perimeter of layouts or size of circuit modules are often the governing parameters. In the methodology that we introduce through G2PLAN, we not only generate multiple topological solutions for adjacency relations, but also ensure dimensioning of these floorplans with respect to arbitrary size requirements provided by the user. Thus, G2PLAN guarantees an exhaustive search of potential floorplans satisfying adjacency and size relations, which cannot be offered by machine learning techniques.

The rest of the paper is structured as follows: first, a literature review of relevant work is discussed in the next section, followed by a few important preliminaries in Section 3. Then we present a detailed working of G2PLAN with an example in Section 4. We evaluate G2PLAN on the parameters of the number of floorplans generated and execution time and compare it with existing graph-based and data-driven methods. In Section 5, we present different features of G2PLAN with their architectural importance. Finally, we address future work in Section 6 and conclude the paper with Section 7.

2. Literature Review

Levin [Lev64] was the first to present graph theoretical applications in designing architectural layouts. Rectangular floorplans (RFPs) were considered as layouts of fundamental importance and several attempts for generating such layouts corresponding to adjacency graphs were made [Ste73, MSL76]. Roth *et al.* [RHW82] were the first to give a workflow for constructing floorplans from adjacency graphs and also for incorporating sizes of each cell simultaneously. Graph theoretical techniques established a firm position in the computational design paradigm after Bhasker and Sahni [BS87] gave a linear time algorithm to generate rectangular duals for PTPGs. Works like Kant and He [KH93] ensured easy implementation and scalability of graph theoretical approaches in the context of constructing layouts. With improvements in computational technology, a shift from traditional methods was observed. Recently, Shekhawat *et al.* [SUBJ21] presented the construction of floorplans with rectangular boundary for given planar triangulated graphs (PTGs). Vinod *et al.* [KS21] proposed a graph theoretic algorithm for transforming a RFP into another RFP having different adjacencies.

Merell *et al.* [MSK10] generated residential floorplans using Bayesian networks and were the first to introduce a supervised

learning algorithm into architectural design. Recently, Wu *et al.* [WFT*19] also used a data-driven technique to construct interior layouts with fixed outer boundaries. For this, they built their own dataset-RPLAN, which contains more than 80K distinct layouts in the form of four-channel images. Using the same dataset-RPLAN and based on graph neural network (GNN), Hu *et al.* [HHT*20] considered room adjacencies for generating floorplans. Lu *et al.* [ZWG*21] provide a dataset of residential plans in rural scenarios and also propose a deep learning method to simultaneously extract adjacency graphs and room functionalities from standard floorplan images. House-GAN++ [NHC*21] proposes using a conditional and a relational generative adversarial network (GAN) for automated floorplan generation, where a previously generated output floorplan becomes the next input constraint resulting in a constantly refined training process. Wamiq *et al.* [WPT*21] propose a new generative model for layout generation in which a transformer architecture is used to generate the layout contents in the form of nodes of a layout graph, after which constraints are set according to the edges (which represent relationships such as adjacencies or other properties such as presence of a door) present in the layout graph. The layouts are then optimised by formulating a linear programming problem. They also compare and state significant improvements in this model over the previous generative models, however, highlight a few of its shortcomings, which include generation of invalid constraints between elements (for example, doors between rooms that do not share a wall), formulation of an infeasible optimisation problem and generation of few low-quality results.

Rodrigues *et al.* [RGG13] used evolutionary techniques for generating feasible layouts for the given topological and geometric user requirements. In the same year, using simulated annealing, Bao *et al.* [BYMW13] introduced the concept of good floorplan layouts that are characterized by architectural parameters like lighting, heating, circulations, *etc.* Recently, Shi *et al.* [SSHW20] developed a heuristic search technique known as Monte Carlo Tree Search, which is based on reinforcement learning for generating a rectangular layout corresponding to any input adjacency graph while satisfies a maximum number of adjacencies.

In all of the proposed algorithms, it has been found that the time-complexity is quite high (because of the stochastic nature of these algorithms) and therefore they may not be used for generating layouts where the number of modules are quite large, for example, VLSI circuits or complex buildings.

Shape grammars can also be seen as an effective approach for generating layouts and it has been frequently used in the recent times

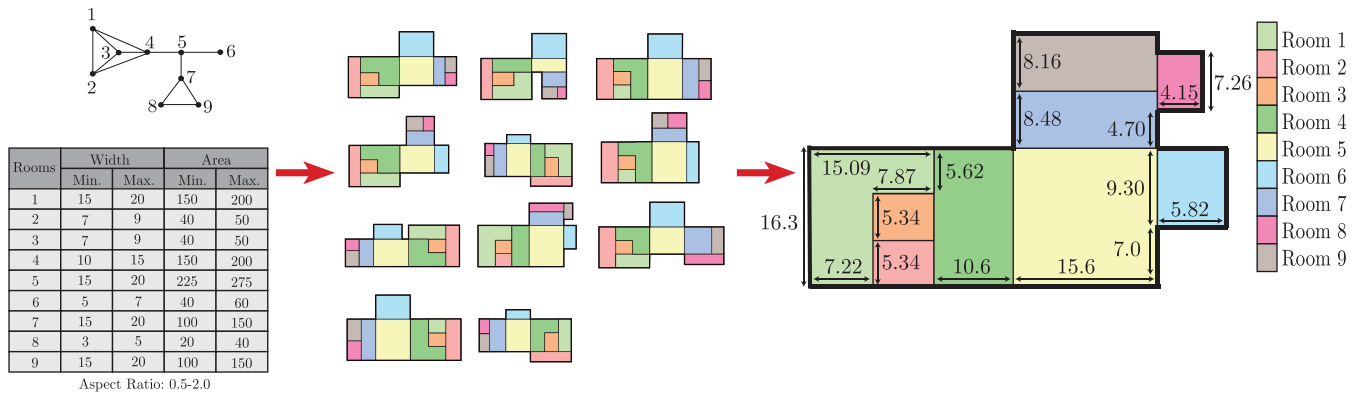


Figure 1: Our automated floorplan generator software, G2PLAN, uses graph theoretical algorithms and optimization techniques to generate all possible dimensioned floorplans corresponding to given adjacencies and dimensions: (a) an input adjacency graph and dimensional constraints, (b) a few dimensionless floorplans (among all possible floorplans) satisfying adjacency constraints, (c) a dimensioned floorplan obtained from one of the dimensionless floorplan.

in the domain of architectural design. Duarte [Dua05] described disjunctive grammar for generating layouts for the given design briefs. Muller *et al.* [MWH*06] used split grammars to produce building layouts and their 3D representations. Wang *et al.* [WLZ20] used graph grammar for designing and validating floorplans.

In the last decade, several studies were also presented for generating dimensioned floorplans. Marson *et al.* [MM10] worked on generating sliceable floorplans having their aspect ratios close to one, without considering room adjacencies. Wu *et al.* [WFLW18] used mixed integer quadratic programming for obtaining dimensioned layouts. Upasani *et al.* [USS20] used graph-theoretic tools and linear optimization for introducing dimensions to a rectangular arrangement drawn by users on a GUI.

Most formulations provide only one solution corresponding to the adjacency and size relations, which may not be useful in early stages of design. Alternate solutions allow the possibility for building practitioners to analyse and compare these designs and choose the one that is most appropriate. Regateiro *et al.* [RBD12] enumerated multiple floorplans using block algebra. Eppstein *et al.* [EMSV12] illustrated simple transformation rules for constructing distinct topological solutions from a rectangular dual, corresponding to a PTPG. Zawidzki *et al.* [ZS18] used DFS backtracking algorithm for generating customized layouts for given adjacency constraints which are architecturally optimal. But the proposed algorithm requires about 8 h to generate 30 candidate solutions only. Shekhawat [She18] developed graph algorithm for enumerating all RFPs with maximum connectivity but it again requires a lot of time to generate all layouts for even a small number of rooms. Nisztuk *et al.* [NM19] used a greedy approach for generating multiple layouts but the generated GUI is very slow in enumerating solutions, which clearly shows the efficiency of graph algorithms over greedy search techniques.

In this paper, we present graph theoretical algorithms for the construction of all possible layouts for a given graph and present a linear optimisation framework for incorporating dimensional constraints, which are further implemented in Python for ensuring efficiency and

scalability of our software. The presented work is the extended version of GPLAN [SUBJ21] with the following additional features (see Figure 1):

1. GPLAN is restricted to biconnected PTGs only. G2PLAN considers 1-connected as well as non-triangulated graphs. The construction of floorplans (dimensionless or dimensioned) for 1-connected graphs has not been discussed in the literature (refer to the supplementary material for the catalogues of floorplans).
2. In GPLAN, dimensional constraints are restricted to minimum and maximum widths and heights for each room while G2PLAN also considers minimum and maximum aspect ratios for each room, which helps in avoiding very thin rooms.
3. GPLAN produces only one floorplan for given dimensional constraints while G2PLAN generates all floorplans satisfying the dimensional constraints (refer to the supplementary material for the catalogues of floorplans).
4. G2PLAN provides an option to generate the catalogue of dimensioned as well as dimensionless floorplans corresponding to a given graph, which consists of all topologically distinct floorplans. It also considers connectivity graph along with adjacency graph for generating floorplans.

3. Preliminaries

In this section, we present relevant definitions, which are used frequently throughout this paper.

1. Graph: A graph is a structure or pictorial diagram made up of two sets, i.e. vertex set and edge set. The vertex set is a finite non-empty set. The edge set may be empty, and its elements are two-element subsets of the vertex set.
2. Articulation point: A vertex in a connected graph is an articulation point if removing it disconnects the graph. For example, in Figure 2(e), vertex 3 is an articulation point.
3. Biconnected graph: A connected graph with no articulation points. For example, the graphs in Figures 2(a) – 2(d) are biconnected while the graph in Figure 2(e) is not biconnected.

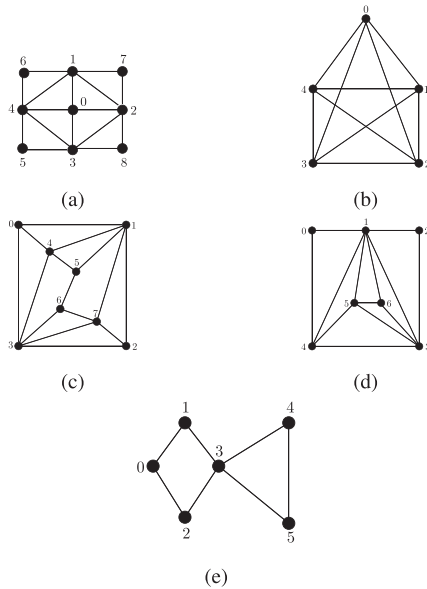


Figure 2: Adjacency graphs: (a) biconnected PTPG, (b) non-planar graph, (c) biconnected planar graph with a few non-triangular inner faces, (d) biconnected PTG with separating triangles, (e) 1-connected planar graph with a non-triangular inner face.

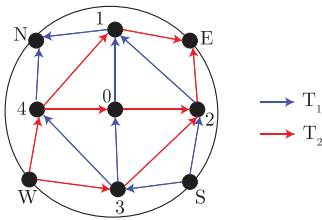


Figure 3: Regular edge labelling for a biconnected PTPG.

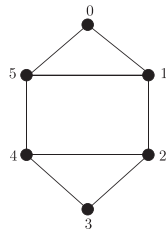


Figure 4: Shortcuts (edges 1-5 and 2-4) and corner implying paths (5-0-1 and 2-3-4).

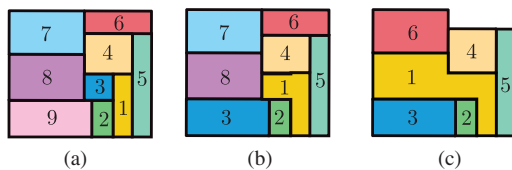


Figure 5: Characterization of floorplans: (a) rectangular floorplan, (b) orthogonal floorplan, (c) non-rectangular floorplan.

4. **Block:** A block is a maximal biconnected subgraph of a graph. For example, the subgraphs induced by vertices 0,1,2,3 and vertices 3,4,5 are blocks of the graph in Figure 2(e).
5. **PTG:** If a graph can be drawn in a plane without any crossing of edges, we call it a planar graph and its drawing without edge crossings is called plane graph. Clearly, a plane graph partitions the plane into connected regions termed as faces of the graph. A plane graph whose all interior faces are triangles is called PTG. For example, the graphs in Figures 2(a) and 2(d) are PTGs while the graphs in Figures 2(b) (non-planar) and 2(c) (not all interior faces are triangular) are not PTGs.
6. **Separating triangle (ST):** In a graph G , a ST is a cycle of length three having at least three internal faces. For example, in Figure 2(d), the $\triangle 143$ is a ST.
7. **PTPG [KK84] :** A PTPG is a connected planar graph having every interior face a triangle, its exterior face is non-triangular and it has no STs. For example, the graph shown in Figure 2(a) is a PTPG whereas the graph shown in Figures 2(c) (not all interior faces are triangular) and 2(d) ($\triangle 134$ is a ST) are not PTPGs.
8. **Regular Edge Labelling (REL) [KH93]:** A REL of a PTPG G with a quadrangular exterior face is a partition of the interior edges of G into two subsets T_1, T_2 of directed edges such that for each interior vertex u , the edges incident to u appear in a counterclockwise order around u as follows: a set of edges in T_1 leaving u , a set of edges in T_2 entering u , a set of edges in T_1 entering u and a set of edges in T_2 leaving u . Let N, E, S, W be the four exterior vertices in clockwise order. All interior edges incident to N are in T_1 and entering N . All interior edges incident to E are in T_2 and entering E . All interior edges incident to S are in T_1 and leaving S . All interior edges incident to W are in T_2 and leaving W . For example, the REL for PTPG in Figure 2(a) is shown in Figure 3.
9. **Shortcut [BS87]:** A shortcut in a planar biconnected graph G is an edge that is incident to two vertices on the outer boundary of G but is not a part of the outer boundary. For example, in Figure 4, edges (1,5) and (2,4) are shortcuts.
10. **Corner implying path (CIP) [BS87]:** A CIP in a planar biconnected graph G is a path u_1, u_2, \dots, u_n on the outer boundary of graph G with the property that (u_1, u_n) is a shortcut and u_2, u_3, \dots, u_{n-1} are not the endpoints of any shortcut. For example, in Figure 4, 5-0-1 is a CIP.
11. **st-graph [4]:** An st-graph is a directed planar graph characterized by one source node s (no incoming edges) and one target node t (no outgoing edges) on the exterior face.
12. **Floorplan [USS20]:** A floorplan can be seen as an arrangement of polygonal rooms within a polygonal boundary where empty spaces may or may not be allowed. If all rooms as well as boundary of a floorplan is rectangular, we call it RFP. When boundary is rectangular but rooms are rectilinear, floorplan is said to be orthogonal floorplan (OFP). Non-rectangular floorplans (NRFPs) have rectilinear boundary. For an illustration, refer to Figure 5.

Two floorplans F_1 and F_2 corresponding to the same adjacency graph are topologically distinct if there exists two rooms R_i and

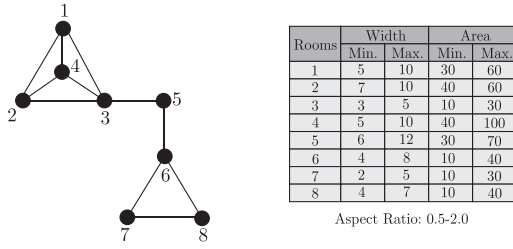


Figure 6: Input graph with size constraints.

R_j in F_1 and F_2 such that R_i and R_j are horizontally adjacent in F_1 and vertically adjacent in F_2 or vice versa.

4. Methodology

This section illustrates the working of G2PLAN—the methodology for constructing multiple floorplan layouts from the given input (planar) graph and size constraints. It is divided into three steps:

1. Biconnected PTPG formation (transforming given graph G into a biconnected PTPG G')
2. Construction of dimensioned RFPs (floorplan for G')
3. Construction of floorplans for G

The first step involves transforming an adjacency graph into a biconnected PTPG using biconnectivity augmentation and triangulation algorithms. The next step uses a modified version of the algorithm proposed by Kant and He [KH93] and modified optimization model proposed by Upasani *et al.* [USS20], to generate all possible dimensioned layouts. The final step removes extra rooms added to transform the input adjacency graph into a PTPG, resulting in floorplans for given graph.

Figure 6 shows an input adjacency graph with eight vertices and size constraints for each of the corresponding rooms. As a worked-out example, we will use this input graph to construct multiple layouts conforming to the dimensional constraints. The following subsections detail the process of floorplan generation in G2PLAN.

4.1. Biconnected PTPG formation

Kozminski *et al.* [KK85] proves that the necessary and sufficient condition for a RFP to exist is that the graph should be a PTPG with less than or equal to 4 CIPs. Hence to transform an adjacency graph into a PTPG with less than or equal to four CIPs, it is passed through the steps of biconnectivity augmentation, triangulation, STs, and CIPs elimination. The extra edges added through these steps will lead to wall adjacencies in the corresponding floorplan. Removing those adjacencies might not be a trivial task for the algorithm, hence we subdivide these edges into vertices using Algorithm 2. The extra vertices added will lead to additional rooms in the corresponding floorplan and can be easily eliminated.

Biconnectivity augmentation: The first step is to check if the input graph is biconnected. This is done using a modified version of the depth-first search proposed by Tarjan [Tar72] to find articulation points in $O(V+E)$ time. Kant *et al.* [KB91] show that making

a graph biconnected while maintaining planarity is an NP-complete problem and therefore proposes an approximation algorithm for biconnectivity augmentation. The drawback with the algorithm is that it increases a vertex's degree by n (number of vertices). Therefore we propose a modified biconnectivity augmentation algorithm (see Algorithm 1) in which every vertex receives at most two extra edges. Algorithm 1 takes the graph $G(V, E)$ and a list of articulation points $cutVertices$ (identified in depth-first search order) as the input.

Algorithm 1. Biconnectivity augmentation

```

1: function BCONNECT( $G(V, E), cutVertices$ )
2:   for  $v$  in  $cutVertices$  do
3:      $(u_1, \dots, u_k) \leftarrow adj(v)$   $\triangleright adj(x)$  = vertices adjacent to  $x$ 
       s.t. vertices in the same block appear consecutively
4:     for  $j \leftarrow 1$  to  $k$  do
5:       if  $u_j$  and  $u_{j+1}$  belong to different blocks then
6:          $E \leftarrow E + \{(u_j, u_{j+1})\}$ 
7:       end if
8:       if  $(v, u_j)$  was added to  $E$  earlier then
9:          $E \leftarrow E - \{(v, u_j)\}$ 
10:      end if
11:      if  $(v, u_{j+1})$  was added to  $E$  earlier then
12:         $E \leftarrow E - \{(v, u_{j+1})\}$ 
13:      end if
14:    end for
15:  end for
16: end function

```

The **bconnect** function transforms the input graph into a biconnected planar graph with a minimum number of additional edges and at most two edges are added to each vertex (refer to Figures 7a and 7b). *Appendix* provides a mathematical proof for the correctness of this algorithm.

Triangulation: Next, we triangulate the obtained biconnected graph, i.e. add edges to ensure every interior face is a triangle. Berry *et al.* [BBHP04] proposed a maximal planar algorithm for transforming any planar biconnected graph to a planar triangular graph. G2PLAN uses this algorithm to triangulate a biconnected planar graph (see Figure 7c).

Removing STs and CIPs: Kozminski *et al.* [KK85] showed that a rectangular dual for a graph G exists if and only if G is a PTPG with no more than four CIPs.

We can check if a graph has a ST by finding the number of three-vertex cycles in the graph using the algorithm defined by Johnson [Joh75]. If this count is equal to (number of edges – number of vertices + 1), then the graph has no STs. G2PLAN removes a ST from the graph by selecting an edge and adding a vertex to it as shown in Figure 7(d) [RBM01]. In case of more than four CIPs, G2PLAN eliminates extra CIPs by removing a shortcut which is done by adding a vertex to it (similar to the removal of a ST).

Transforming extra edges into vertices: The extra edges that are added in previous steps (biconnectivity augmentation or triangulation) can either lie on the interior or exterior boundary of the graph (which is now a PTPG). These edges are transformed

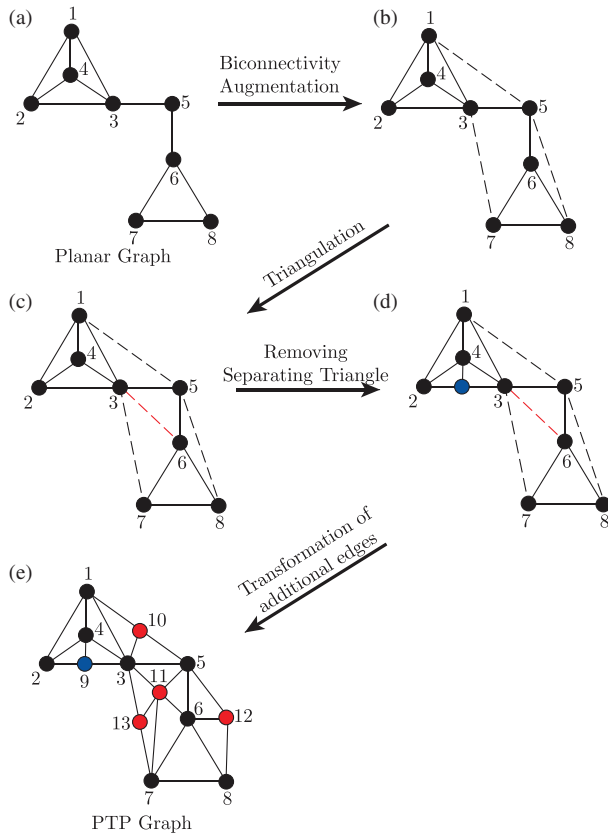


Figure 7: Transformation of a planar 1-connected graph into a biconnected PTPG.

into vertices (or rooms) so that they can easily be removed in the final step to generate the required floorplan for the given graph (see Figure 7e). Algorithm 2 reports the implementation detail for the transformation rule. The algorithm takes the graph $G(V, E)$ and the edge to be transformed (u, v) as the input.

Algorithm 2. Transformation rules

```

1: function TRANSFORM( $G(V, E), (u, v)$ )
2:   if  $(u, v)$  is an exterior edge then
3:      $x \leftarrow nbd(u) \cap nbd(v) \triangleright nbd(x) = \text{vertices adjacent to } x$ 
4:      $V \leftarrow V + \{x\}$ 
5:      $E \leftarrow E + \{(a, x), (a, u), (a, v)\}$ 
6:      $E \leftarrow E - \{(u, v)\}$ 
7:   else
8:      $(x, y) \leftarrow nbd(u) \cap nbd(v)$ 
9:      $V \leftarrow V + \{x\}$ 
10:     $E \leftarrow E + \{(a, x), (a, y), (a, u), (a, v)\}$ 
11:     $E \leftarrow E - \{(u, v)\}$ 
12:   end if
13: end function

```

These rules transform all additional edges to vertices while retaining the property of PTPG. Appendix provides a mathematical proof for the correctness of Algorithm 2.

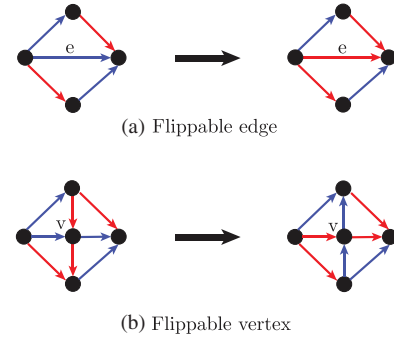


Figure 8: Transformation of a biconnected PTPG into dimensioned rectangular floorplans.

Figure 7 shows the transformation of a given adjacency graph into a biconnected PTPG using the steps discussed above.

4.2. Construction of dimensioned rectangular floorplans

This step transforms a biconnected PTPG obtained in Section 4.1 into a dimensioned RFP. In this subsection, we modify and extend the algorithm proposed by Kant and He [KH93] to generate multiple dimensioned floorplans. The first step involves finding four borders and connecting them to new vertices: north, south, east and west. These borders denote the rooms forming the north, south, east and west boundary of the floorplan. G2PLAN then transforms this new graph into a REL using the algorithm proposed by Kant and He [KH93]. In a REL, a directed edge from vertex v to u denotes the direction of adjacencies between the rooms. If the edge belongs to subset T_1 , then the room corresponding to the vertex v will lie to the north of the room corresponding to vertex u . If the edge belongs to subset T_2 , the room corresponding to vertex v will lie to the east of the room corresponding to vertex u . G2PLAN uses a linear time algorithm proposed by Kant and He [KH93] to transform a REL into rectangular dual. We use the modified version of the network flow model proposed by Upasani *et al.* [USS20] for dimensioned floorplans.

Four-completion (Figure 8b): Kant and He [KH93] proposed the addition of four new vertices n, w, s, e representing north, west, south and east, respectively, to a biconnected PTPG. This process needs four corner vertices on the outer boundary, which are obtained by using the concept of CIPs proposed by Bhasker and Sahni [BS87]. Kant and He [KH93] does not consider a particular case when the outer boundary is a triangle. Algorithm 3 details the implementation of four-completion algorithm using CIP while considering the particular case of a triangular outer boundary. The algorithm takes the PTPG $G(V, E)$, the outer boundary of the graph *outerBoundary* and a list of CIPs *cip* as the input.

G2PLAN uses Algorithm 3 to find all possible ways in which a biconnected PTPG can be transformed into a new biconnected PTPG, thus aiding in the process of multiple floorplans formation.

Algorithm 3. Four-completion

```

1: function FOURCOMPLETION( $G(V, E)$ ,  $boundary$ ,  $corner$  Implied Paths)
2:   if  $boundary$  is triangle then
3:      $\{v_1, v_2, v_3\} \leftarrow boundary$ 
4:      $cornerPoints \leftarrow \{v_1, v_2, v_3, v_1\}$ 
5:   else
6:      $u_1, u_2, \dots, u_n$  are points randomly chosen from each corner
       implying path
7:     if  $n < 4$  then
8:       for  $i \leftarrow n + 1$  to 4 do
9:          $u_i$  is a point randomly chosen from  $boundary$ 
           and different from  $u'_k$  ( $k < i$ )
10:      end for
11:    end if
12:     $cornerPoints \leftarrow \{u_1, u_2, u_3, u_4\}$ 
13:  end if
14:   $p_1 \leftarrow$  path on  $boundary$  between  $cornerPoints[0]$  and
     $cornerPoints[1]$ 
15:   $p_2 \leftarrow$  path on  $boundary$  between  $cornerPoints[1]$  and
     $cornerPoints[2]$ 
16:   $p_3 \leftarrow$  path on  $boundary$  between  $cornerPoints[2]$  and
     $cornerPoints[3]$ 
17:   $p_4 \leftarrow$  path on  $boundary$  between  $cornerPoints[3]$  and
     $cornerPoints[0]$ 
18:   $V \leftarrow V + \{n, e, s, w\}$ 
19:  add edges between  $n$  and vertices in  $p_1$  to  $E$ 
20:  add edges between  $e$  and vertices in  $p_2$  to  $E$ 
21:  add edges between  $s$  and vertices in  $p_3$  to  $E$ 
22:  add edges between  $w$  and vertices in  $p_4$  to  $E$ 
23:   $E \leftarrow E + \{(n, w), (w, s), (s, e), (e, n)\}$ 
24: end function

```

REL formation (Figure 8c): Kant and He [KH93] proposed a two-step algorithm to transform a biconnected PTPG obtained by four-completion into a REL. This algorithm first removes all the vertices in a described order and then adds the label T_1 and T_2 to the edges while adding the vertices. *Appendix* discusses this algorithm in detail.

Multiple REL formation (Figure 8d): Eppstein *et al.* [EMSV12] introduced the concept of flippable edges and vertices to transform one REL to another.

A flippable edge is an edge e , which is not adjacent to a degree-four vertex, and the four-cycle surrounding e is alternately labelled in the REL (Figure 9a). A flippable vertex v is a four-degree vertex such that the four-cycle surrounding v is alternately labelled in the REL (Figure 9b). We can obtain a new REL by changing the label of the flippable edge or the edges incident on a flippable vertex, as shown in Figure 9.

G2PLAN iterates over different possible boundary paths and enumerates all RELs possible for that boundary path using the concept of flippable item. This results in the generation of all possible topologically distinct RELs for a given biconnected PTPG. For the graph in Figure 8(a), G2PLAN generates 1280 floorplans in under 2 s.

Rectangular dualization (Figure 8e): The RELs obtained are partitioned into two st-graphs based on the labels T_1 and T_2 (see Figure 10). The T_1 graph denotes the vertical adjacencies in the floorplan and the T_2 graph denotes the horizontal adjacencies in the floorplan. The two st-graphs are merged and the extra vertices n, w, s, e are removed to obtain the floorplan for the graph.

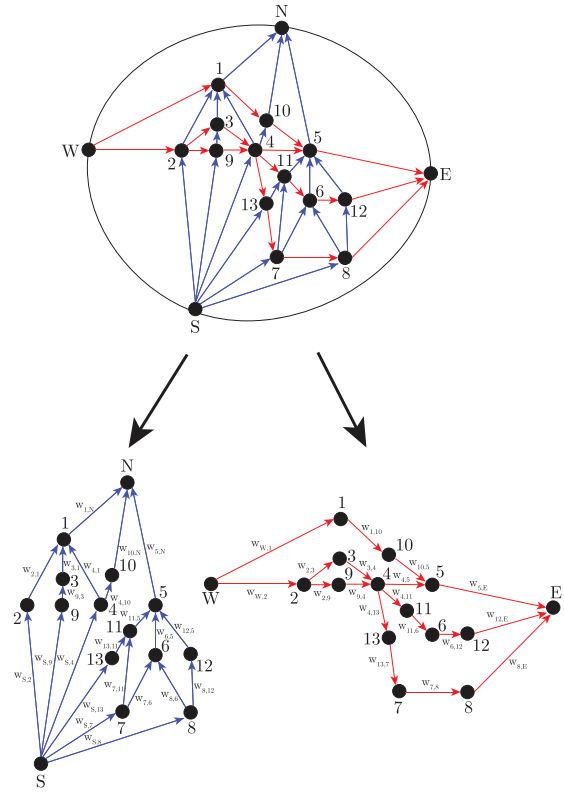


Figure 9: Flippable items.

Dimensioning (Figure 8f): We use the network flow model proposed by Upasani *et al.* [USS20] and define the optimisation problem for dimensioning as shown in Figure 11. T_1 and T_2 are modelled as two flow networks such that flow in the T_1 graph (vertical adjacencies) denote width and in the T_2 graph (horizontal adjacencies) denote heights. For dimensioned RFP, heights are calculated by dividing the areas with widths. While Upasani *et al.* [USS20] considered only minimum widths and aspect ratios for each room, we have also incorporated maximum widths and areas as constraints, offering users more control over dimensions of each room. The objective function and constraints for the flow networks in Figure 10 are listed in Figure 11. Here, $w_{i,j}$ denotes the weight of the edge (i, j) (flow) and d_i^{max} denotes the maximum dimension of the room i , as entered by the user (width in the case of vertical st-graph and height in the case of horizontal st-graph).

Figure 8f shows multiple dimensioned RFPs generated corresponding to PTPG and size constraints in Figure 6.

4.3. Non-rectangular floorplans

As a post-processing step, we need to merge the rectangular spaces corresponding to extra vertices (added in the step of removing STs) and eliminate the extra vertices (added for biconnectivity augmentation and triangulation) from the final layout. Algorithm 4 details the implementation of the merging and removing process for extra rooms. The algorithm takes floorplan F and a list of extra rooms $extraRooms$ as the input and generates NRFPs.

$$\text{Objective: } \min \sum_{i,j \in V(G)} (w(e_{j,i}) - d_i^{\max})$$

where

$d_i^{\max} \rightarrow$ Maximum dimension (height/width) of room i
 $e_{j,i} \rightarrow$ Directed edge in st-graph

Equality Constraints

Rooms	Vertical st-graph	Horizontal st-graph
1	$w_{2,1} + w_{3,1} + w_{4,1} = w_{1,N}$	$w_{W,1} = w_{1,10}$
2	$w_{S,2} = w_{2,1}$	$w_{W,2} = w_{2,3} + w_{2,9}$
3	$w_{9,3} = w_{3,1}$	$w_{2,3} = w_{3,4}$
4	$w_{S,4} = w_{4,1} + w_{4,10}$	$w_{3,4} + w_{9,4} = w_{4,5} + w_{4,11} + w_{4,13}$
5	$w_{11,5} + w_{6,5} + w_{12,5} = w_{5,N}$	$w_{4,5} + w_{10,5} = w_{5,E}$
6	$w_{7,6} + w_{8,6} = w_{6,5}$	$w_{11,6} = w_{6,12}$
7	$w_{S,7} = w_{7,6} + w_{7,11}$	$w_{13,7} = w_{7,8}$
8	$w_{S,8} = w_{8,6} + w_{8,12}$	$w_{7,8} = w_{8,E}$
9	$w_{S,9} = w_{9,3}$	$w_{2,9} = w_{9,4}$
10	$w_{4,10} = w_{10,N}$	$w_{1,10} = w_{10,5}$
11	$w_{7,11} + w_{13,11} = w_{11,5}$	$w_{4,11} = w_{11,6}$
12	$w_{8,12} = w_{12,5}$	$w_{6,12} = w_{12,E}$
13	$w_{S,13} = w_{13,11}$	$w_{4,13} = w_{13,7}$

Inequality Constraints

Rooms	Vertical st-graph	Horizontal st-graph
1	$5 \leq w_{2,1} + w_{3,1} + w_{4,1} \leq 10$	$3 \leq w_{W,1} \leq 12$
2	$7 \leq w_{S,2} \leq 10$	$4 \leq w_{W,2} \leq 9$
3	$3 \leq w_{9,3} \leq 5$	$2 \leq w_{2,3} \leq 10$
4	$5 \leq w_{S,4} \leq 10$	$4 \leq w_{3,4} + w_{9,4} \leq 20$
5	$6 \leq w_{11,5} + w_{6,5} + w_{12,5} \leq 12$	$3 \leq w_{4,5} + w_{10,5} \leq 11$
6	$5 \leq w_{7,6} + w_{8,6} \leq 10$	$2 \leq w_{11,6} \leq 10$
7	$2 \leq w_{S,7} \leq 5$	$2 \leq w_{13,7} \leq 15$
8	$4 \leq w_{S,8} \leq 7$	$2 \leq w_{7,8} \leq 10$

Figure 10: REL to st-graphs: network flow model for dimensioning.

Algorithm 4. Merge/remove extra rooms

```

1: function PROCESSEXTRAROOMS(floorplan, extraNodes)
2:   for  $u$  in extraNodes do
3:     if  $u$  was added for biconnectivity or triangularity then
4:        $floorplan \leftarrow floorplan - \{u\}$ 
5:     else
6:        $(x, y) \leftarrow extraRoomNeighbour(u)$  ▷
7:        $extraRoomNeighbour(u) =$  edge removed for extra room  $u$ 
8:       merge room  $u$  with room  $x$  or merge room  $u$  with
       room  $y$  ▷ Results in two different floorplans
9:     end if
10:  end for
11: end function

```

Figure 12 shows NRFPs for the adjacency graph in the Figure 6 satisfying dimensional constraints.

5. Results and Comparisons

In this section, we briefly cover different features of G2PLAN and their architectural importance.

5.1. Non-triangulated and 1-connected graphs

Existing graph algorithms related to floorplans are restricted to bi-connected PTGs only, but from designers' point, the graph may not be biconnected and triangular. One of the major strengths of G2PLAN is that it is capable of constructing layouts for any given planar graph. Also, from an architectural perspective, 1-connected graphs are useful in representing different blocks of a building (for example, the graph in Figure 13a is 1-connected and in the corresponding floorplan, there are two blocks connected through room 0) whereas non-triangulated faces in a graph represent the requirement of atriums (the graph in Figure 13a has two non-triangular faces which correspond to atriums in the required floorplan represented by white spaces in Figure 13a).

5.2. Connectivity graph

The adjacency graph describes the sharing of wall among the rooms but it does not talk about the accessibility. In other words, two adjacent rooms may not be connected through doors.

In G2PLAN, a user can input an adjacency as well as a connectivity graph simultaneously. For example, for the graph in Figure 13(b), the black edge corresponds to an adjacency between two rooms, whereas red edges suggest connectivity between them. The corresponding floorplan with doors is shown in Figure 13(b). The algorithm runs for the adjacency graph as described in Section 4. To accommodate the connectivity graph into the floorplan, a door is added to the wall adjacencies corresponding to the edges in connectivity graph, as a post processing step.

5.3. Aspect ratio and spacing for doors

From an architectural standpoint, it is crucial to ensure that the proportion of rooms in the dimensioned floorplan is not too narrow. To tackle this case, we have included a feature in G2PLAN, which allows the user to enter aspect ratio ranges permissible for each room. The default aspect ratios for rooms are kept as 0.5–2. We also allow the users to set the minimum shared wall dimension between two rooms to at least have space for installing doors connecting them. By default, it is set to 1 unit.

5.4. Comparisons and time complexity

Number of possible solutions: G2PLAN takes user-defined size constraints and processes all possible floorplans for the same. The number of possible solutions for a given adjacency graph increases exponentially. Figure 14 shows an exponential increase in floorplans (orthogonal and rectangular) as the room count increases. The

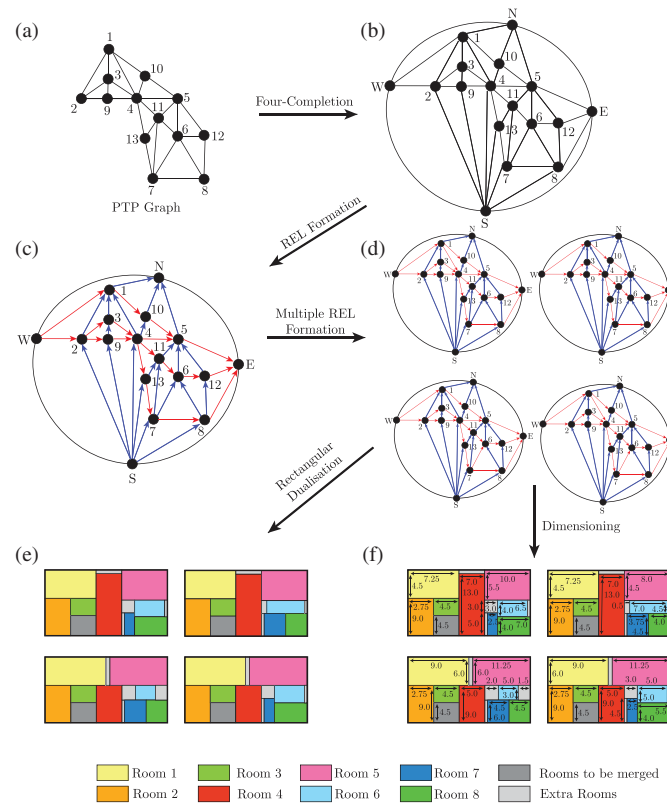


Figure 11: Optimisation: objective function and all equality and inequality constraints.

exhaustive list of floorplans that G2PLAN generates can be attributed to a loose constraint space. Since these layouts can be difficult to explore in a short time, adding dimensional constraints to further customize the design, such as minimum or maximum dimensions, aspect ratio and symmetry can significantly reduce the number of floorplans generated, by tightening the constraints (refer to the supplementary material for catalogue for dimensioned floorplans and symmetry will be introduced in the near future). However, the drawback here is that if the number of rooms is very large, better techniques would be required to filter floorplans in the generated catalogue. While learning-based algorithms that are available in the literature [6,18] are able to generate a smaller set of designs, they do not necessarily produce architecturally valid layouts and may not satisfy input constraints (Figure 15).

Time complexity: Figure 16 shows the execution time of G2PLAN for generating rectangular and OFPs. It is clearly evident that G2PLAN is scalable since, with the increase in the number of rooms, the execution time increases linearly. Also, it is interesting to note that G2PLAN can generate a floorplan having at most 10 rooms in less than 2 ms.

Comparison with Wang et al. [WYZ18]: To compare with the GADG proposed by Wang et al. [WYZ18], we consider dimensionless RFPs only. Figure 17 shows the comparison where G2PLAN is 2.5 times faster than GADG. Also, contrary to GADG, the method can generate NRFPs and can incorporate dimensional constraints.

Comparison with data-driven methods: With recent improvements in deep learning, several data-driven approaches for floorplan generation have been introduced. Due to the stochastic nature of these algorithms, the adjacency constraints specified by the users are not necessarily satisfied. House-GAN++ [NHC*21] proposes compatibility as a metric to evaluate the floorplans. It is computed by calculating the graph-edit distance between the input graph and the estimated graph from the generated floorplan. Hence, the lower the compatibility score, the better is the generated floorplan. Due to the deterministic nature of our algorithm, G2PLAN ensures that user adjacency is maintained, thus achieving a compatibility score of 0. In contrast, on 1000 samples divided into four groups on basis of number of rooms (5,6,7,8), state-of-the-art methods like House-GAN++ achieve a compatibility score of 1.9 [NHC*21]. Figure 15 shows input graphs and the floorplans achieved using state-of-the-art data-driven methods Graph2Plan [HHT*20] and House-GAN++, and G2PLAN. Figure 15 shows that the estimated graphs from generated floorplans are different from the input graph for Graph2Plan and House-GAN++, but it remains the same for G2PLAN.

6. Future Work

At this stage, we have not incorporated many architectural features into G2PLAN, which would like to add in the near future. Our primary focus would be to introduce rooms functionality based on the

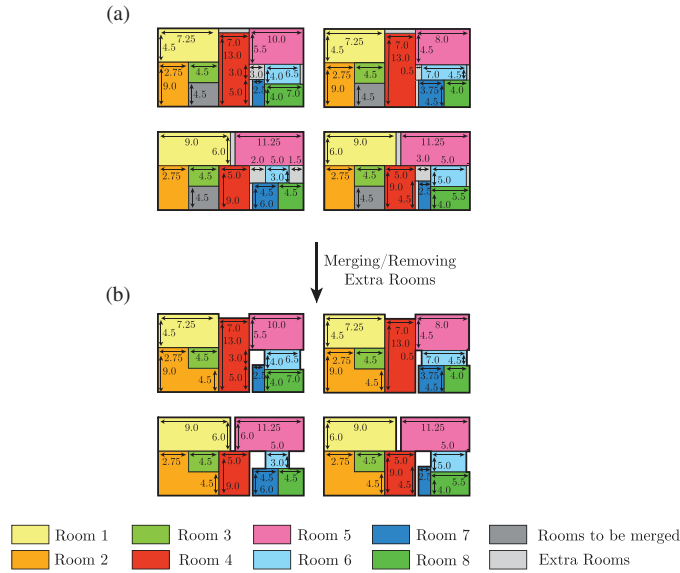


Figure 12: Transformation of rectangular floorplans into non-rectangular floorplans by deleting extra rooms.

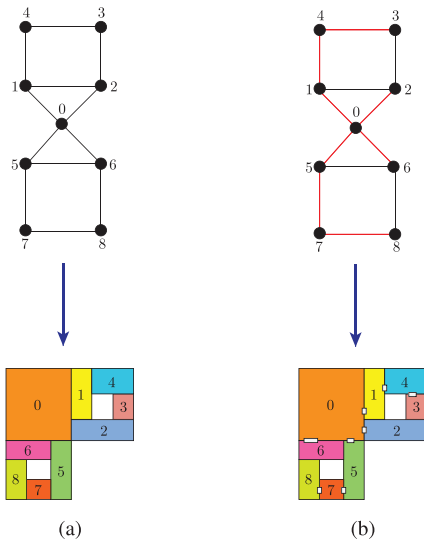


Figure 13: Floorplan with atriums and doors.

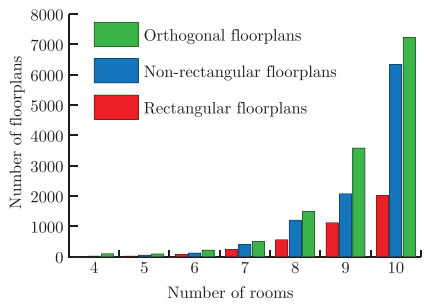


Figure 14: Number of different floorplans generated by G2PLAN (rectangular, non-rectangular and orthogonal floorplans).

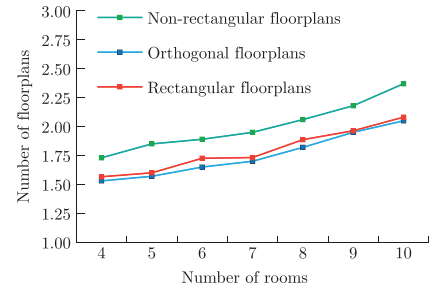


Figure 15: G2PLAN versus data-driven methods (HouseGAN++ and Graph2Plan).

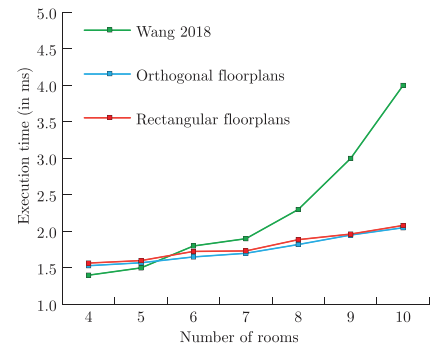


Figure 16: Average execution time of G2PLAN for rectangular, orthogonal and non-rectangular floorplans.

existing data, to consider boundary layout as input and to insert circulations within the obtained floorplans. For further details, refer to Appendix Section A.4.

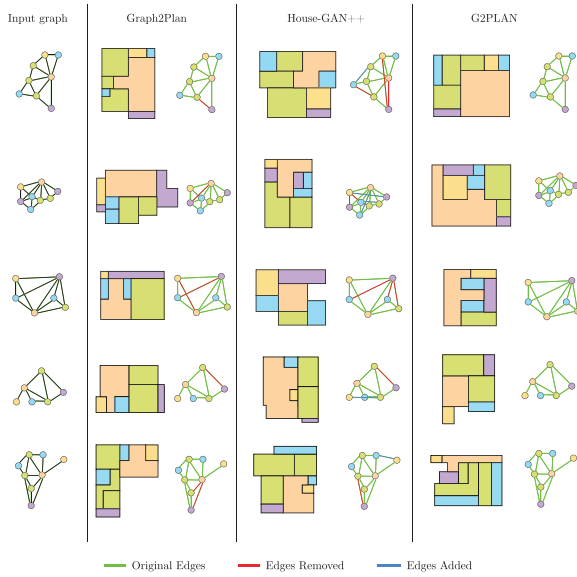


Figure 17: Average execution time (G2PLAN vs. Wang et al. [WYZ18]).

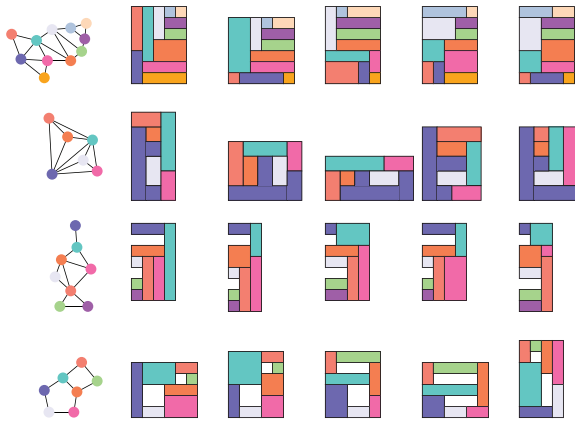


Figure 18: Illustrating a few layouts generating using G2PLAN.

7. Conclusion

In this paper, we presented the working of G2PLAN; a software developed for designing dimensioned floorplan layouts corresponding to given adjacency and size constraints (see Figure 18). The utilization of graph theoretical tools in G2PLAN ensures time efficiency and robustness of this approach supported by a strong mathematical foundation, whereas the algorithmic approach makes it highly customizable. In future, G2PLAN shall have a designer-friendly interface, capable of incorporating specific user requirements for generating customized building designs. Presently, the software does not allow designers to include the shape of exterior boundary as a parameter, for choosing appropriate designs from the catalogue, which could be introduced in a future version of the software. Since the number of possible feasible solutions is large, data-driven approaches can be used to refine design searches in favour of architectural layouts or other design problems.

Acknowledgements

We are thankful to BITS Pilani for supporting this research. We also thank all reviewers for providing relevant and insightful reviews.

Appendix A

In this section, we will further detail the following aspects which we omitted in the main paper:

1. Correctness of Biconnectivity Augmentation (Algorithm 1)
2. Correctness of Transformation rules (Algorithm 2)
3. REL Formation (Algorithm 3)
4. Detailed future work

A.1. Biconnectivity augmentation

In the paper we have proposed a biconnectivity augmentation algorithm (Algorithm 1) such that no more than two edges are added to a vertex. In this section, we prove its correctness.

First we prove that Algorithm 1 gives a biconnected graph. Let v be an articulation point in a given graph G . Let B_1, B_2, \dots, B_k be the blocks in clockwise order around vertex v and u_1, u_2, \dots, u_k be the neighbours of v such that $u_1 \in B_1, u_2 \in B_2, \dots, u_k \in B_k$. Lines 5-7 of the algorithm adds an edge between B_j and B_{j+1} for $1 \leq j < k$ and hence all neighbours of v belong to the one common block. Thus v is not a cut-vertex hence resulting in a biconnected graph.

We will try to prove that Algorithm 1 adds no more than 2 edges to a vertex by induction.

Let $P(n)$ define the statement that in a graph G , for a vertex v adjacent to n articulation points, at most 2 more edges are added to the vertex by the algorithm, to make the graph biconnected.

We will first prove this for the base cases, $n = 0, 1$ and 2 .

Base cases:

1. $P(0)$: In the graph G , if vertex v is not adjacent to any articulation point, no augmenting edge will be added to the vertex by Algorithm 1. Hence the statement $P(0)$ is true.
2. $P(1)$: Let w_1 be an articulation point adjacent to vertex v . Let the neighbours of w_1 identified in such a way that the neighbours in the same block appear consecutively be $x_1, x_2, \dots, x_i, v, x_{i+1}, \dots, x_n$. Now line 5-7 of the algorithm will add at most 2 edges (x_i, v) and (v, x_{i+1}) to the graph. Hence the statement $P(1)$ is true.
3. $P(2)$: Let w_1 and w_2 be two articulation points adjacent to the vertex v . Let the neighbours of w_1 identified in such a way that the neighbours in the same block appear consecutively be $x_1, x_2, \dots, x_i, v, x_{i+1}, \dots, x_n$. There can be two cases:
 1. v is an articulation point:
Without loss of generality, w_1, v, w_2 are identified as articulation points, arranged in depth-first-search order. While visiting vertex w_1 , vertex v receives at most two augmenting edges (x_i, v) and (v, x_{i+1}) . After visiting vertex v , we will get two neighbouring vertices of v (say a and b) such that a, w_2, b will be in the same block. The edges (a, w_2) and (w_2, b) will be added either by lines 5-7 of the algorithm or

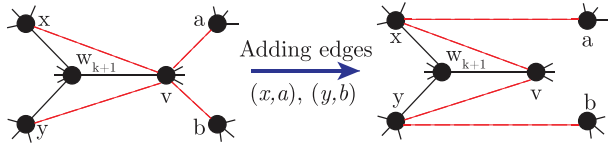


Figure A.1: Adding edges (x, a) and (y, b) removes edges (a, v) and (v, b) .

they are part of the original graph G . Now while iterating through vertex w_2 , the neighbours obtained will be of order a, v, b . Since a, v and b will be in the same block; hence no extra edges will be added to vertex v . So, at most 2 edges (x_i, v) and (v, x_{i+1}) are added to vertex v .

b. v is not an articulation point:

As v is not an articulation point, there exists a vertex (say a) such that vertices a and v are in the same block. So without loss of generality, the neighbours of w_1 identified in such a way that the neighbours in the same block appear consecutively are $x_1, x_2, \dots, a, v, x_{i+1}, \dots, x_n$. Hence while iterating through w_1 , at most one edge i.e. (v, x_{i+1}) will be added by lines 5-7 of the algorithm. Similarly while iterating through vertex w_2 , at most one edge will be added. So at most two extra edges will be added to vertex v .

For both the cases, at most two edges are added to vertex v . Hence, the statement $P(2)$ is true.

Induction hypothesis: Let $P(1), P(2), \dots, P(k)$ be true i.e. in a graph G , for a vertex v , adjacent to less than or equal to k articulation points, at most 2 more edges are added to the vertex by the algorithm, to make the graph biconnected.

Now we need to prove that $P(k+1)$ is true. Let $w_1, w_2, \dots, w_k, w_{k+1}$ be the articulation points adjacent to the vertex v , in the graph G . Let G' be a new graph obtained by removing block to which vertex w_{k+1} belong to. So, the new graph G' is a graph where vertex v is adjacent to k articulation points. Using induction hypothesis, the algorithm can biconnect this graph G' by adding at most two edges (say (a, v) and (v, b)) to vertex v . Let the graph obtained by biconnecting graph G' be G'' . Now let us add the block of vertex w_{k+1} to the graph G' to obtain a new graph (say H). In graph H , the vertex v is adjacent to only one articulation point w_{k+1} . The vertex v is also an articulation point as removing it will disconnect the vertices belonging to block of vertex w_{k+1} .

Let us assume w_{k+1}, v are identified as articulation points, arranged in depth-first-search order. While iterating through vertex w_{k+1} , at most 2 edges will be added to vertex v (say (x, v) and (y, v)). Now while iterating through vertex v , as shown in Figure A.1, line 5-7 will add edges (x, a) and (y, b) . Line 8-13 will remove edges (a, v) and (v, b) . Hence, at most 2 edges will be added to the vertex v .

If v, w_{k+1} are identified as articulation points, arranged in depth-first-search order. While iterating through vertex v , 2 edges will be added to vertex w_{k+1} i.e. (a, w_{k+1}) and (b, w_{k+1}) . Now while iterating through vertex w_{k+1} , as shown in Figure A.2, the neighbours obtained will be of order a, v, b . Since a, v and b are in the same

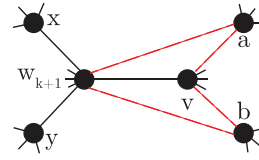


Figure A.2: While iterating through w_{k+1} , the neighbours will be of order a, v, b .

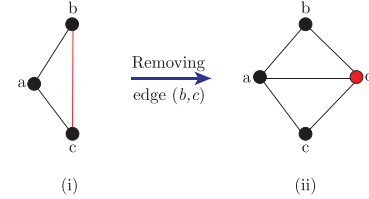


Figure A.3: Removing the exterior edge (b, c) by introducing a new vertex.

block, no extra edges will be added to vertex v , i.e., at most 2 edges (a, v) and (v, b) are added to the vertex v .

So, at most 2 edges are added to vertex v to make graph G biconnected. Hence, the statement $P(k+1)$ is true and by the principle of strong mathematical induction, statement $P(n)$ is true, i.e., in a graph G , for a vertex v adjacent to n articulation points, at most 2 more edges are added to the vertex by the algorithm, to make the graph biconnected.

A.2. Transformation rules

In the paper, we have proposed an algorithm to transform the additional edges into extra rooms (Algorithm 2). Here, we prove that the transformation rules doesn't violate the property of PTPG.

Let (b, c) be the edge that we want to remove. There can be 2 cases:

1. (b, c) is an exterior edge:

If (b, c) is an exterior edge in the PTPG, it will be a part of only one triangular face. Let $\{a, b, c\}$ be the triangular face containing (b, c) . We remove edge (b, c) by adding vertex d and connecting it to a, b and c as shown in Figure A.3. The resulting graph after transformation (Figure A.3(ii)) is triangulated and does not form a separating triangle, hence the properties of PTPG are retained. So we just need to prove that no separating triangle is formed in order to show that the graph remains a PTPG after addition of vertex d .

If addition of vertex d forms a separating triangle, then vertex d will be a part of the separating triangle. Hence, two edges of that separating triangle will be incident on vertex d . Since d has 3 edges incident on it, we can have 3 different pairs of edges incident on d which can be part of the separating triangle viz. $\{(a, d), (b, d)\}$, $\{(a, d), (c, d)\}$ and $\{(b, d), (c, d)\}$.

If $\{(a, d), (b, d)\}$ is a part of separating triangle then $\{a, b, d\}$ forms a separating triangle which is not possible as $\{a, b, c\}$ was

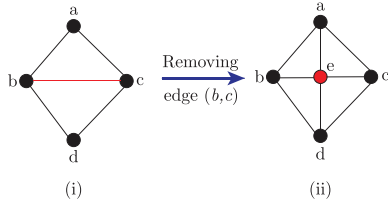


Figure A.4: Removing the interior edge (b, c) by introducing a new vertex.

not a separating triangle (graph was biconnected PTPG). Similarly, $\{(a, d), (c, d)\}$ can not be a part of separating triangle. If $\{(b, d), (c, d)\}$ is a part of separating triangle, then $\{b, c, d\}$ forms a separating triangle which means edge (b, c) is a part of separating triangle but this edge was removed while adding vertex d .

Hence, if edge (b, c) is an exterior edge then sub-dividing it by adding additional vertex does not lead to formation of separating triangle and the properties of a PTP Graph are not violated.

2. (b, c) is an interior edge:

If (b, c) is an interior edge in a PTPG, it will be a part of two triangular faces. Let $\{a, b, c\}$ and $\{b, c, d\}$ be those two triangular faces containing (b, c) . We can remove edge (b, c) by adding vertex e and connecting it to a, b, c and d as shown in Figure A.4. The resulting graph transformation (Figure A.4(ii)) is triangulated and does not form a separating triangle, hence the properties of PTPG are retained in this case as well. Now we only need to prove that no separating triangle is formed in order to show that the graph remains a PTPG after addition of vertex e .

If addition of vertex e forms a separating triangle, then vertex e will be a part of the separating triangle. We can not choose $\{(a, e), (c, e)\}$, $\{(a, e), (b, e)\}$, $\{(b, e), (c, e)\}$ and $\{(c, e), (d, e)\}$ as pair of edges incident on e which can be a part of separating triangle as our graph before addition of vertex e is a biconnected PTPG (similar to Case 1).

Now the remaining pair of edges are $\{(a, e), (d, e)\}$ and $\{(b, e), (c, e)\}$. If $\{(b, e), (c, e)\}$ is a part of separating triangle, then $\{b, c, e\}$ forms a separating triangle which means edge (b, c) is a part of separating triangle but this edge was removed while adding vertex e . If $\{(a, e), (d, e)\}$ is part of separating triangle, then $\{a, d, e\}$ forms a separating triangle. This is not possible because edge (a, d) in Figure A.4(i) will form separating triangle $\triangle acd$ but our graph is a biconnected PTPG.

Hence, if edge (b, c) is an interior edge then breaking it by adding additional vertex does not lead to formation of separating triangle and properties of a PTP graph are not violated.

A.3. REL formation

G2PLAN uses the REL formation algorithm defined by [KH93] to transform a 4-connected PTPG into a REL. In this subsection, we will detail the algorithm used by G2PLAN.

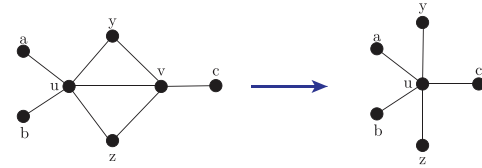


Figure A.5: Contraction of the edge (u, v) .

Edge contraction. Edge contraction for an interior edge $e_{v,u}$, connecting internal vertices v, u of a PTPG G , is defined as the elimination of the edge $e_{v,u}$ and the vertex v from G followed by the connection of all neighbouring vertices of v to u , as shown in Figure A.5. The resulting graph after edge contraction is denoted by G/e . The objective of this procedure is to trivialise REL by reducing the inner vertices to one, after which, the rules of expansion defined in section A.3.2 would be used to obtain REL for G .

Edge contraction is defined only for PTPGs and thus it is essential to ensure that G/e does not lose the characteristics of a PTPG. [KH93] introduced the following definitions.

1. **Light and Heavy vertices:** Vertices which have degree < 20 are called light vertices and those having degree ≥ 20 are called heavy vertices.
2. **Good vertex:** A good vertex in a PTPG is an inner vertex with the following properties:
 1. Has degree 4 and (has at most 1 heavy neighbour or has 2 non adjacent heavy neighbours)
 2. Has degree 5 and has at most 1 heavy neighbour
3. **Contractible neighbour** (of a good vertex): Let u be a neighbouring vertex of a good vertex v , and y and z be their two common neighbours. If $x (\neq y, z)$ is any other neighbour of v , then only mutual neighbour(s) of u and x must be v (or either of y or z) for u to be a contractible neighbour of v .

To begin edge contraction, a list of all good vertices of the PTPG is made. A contractible neighbour to one of these good vertices is found and the edge connecting them is contracted. After each edge contraction, the list of good vertices is updated. Contraction is continued until only one inner vertex is remaining, as shown in Figure A.6. REL is trivial for this graph and is shown in Figure A.7. For the next section of expansion, knowing the order and details of each contraction is necessary. Hence each contraction is defined with the following information, and is stored in a list in the order they were performed:

1. u, v - as defined in the definition of edge contraction.
2. Neighbouring vertices of v .
3. y, z - as defined in the definition of edge contraction.

With the trivial REL and the list of contractions, we now proceed to the process of expansion.

Edge expansion. Edge expansion for a contractible edge $e_{v,u}$ is defined as adding the edge $e_{v,u}$ and the vertex v (eliminated in the step of edge contraction) into the graph G/e . The objective of this procedure is to obtain REL for the graph G by adding interior vertices

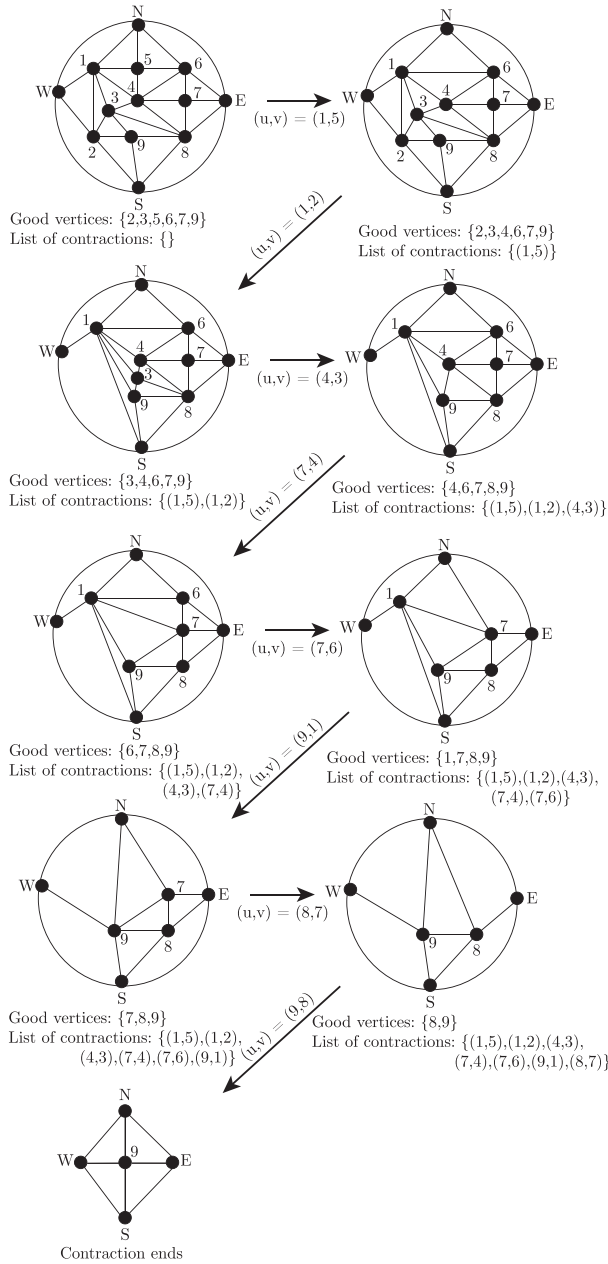


Figure A.6: Illustrating all edge contractions for a 4-completion of a given PTPG.

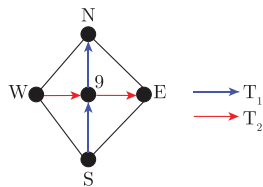


Figure A.7: Trivial REL.

one-by-one into the trivial REL while maintaining the characteristics of a REL and adjacency relations between the vertices.

Suppose we have a REL $\{T_1, T_2\}$ for graph G/e and we want to expand edge $e_{v,u}$ to obtain the REL $\{T'_1, T'_2\}$. Let u and v be the vertices containing edge $e_{v,u}$ before edge contraction, v be the vertex which will be added after edge expansion and y and z be two common neighbours of u and v as defined in edge contraction. The non-residue edges (edges other than (u, y) and (u, z)) will have same label with respect to $\{T'_1, T'_2\}$ as their label with respect to $\{T_1, T_2\}$. So, to ensure that G does not lose the characteristics of a REL, we need to assign a proper label to edge $e_{v,u}$ and edges (u, y) , (u, z) , (v, y) and (v, z) .

The edge expansion hence depends upon the edges (u, y) and (u, z) . In edges (u, y) and (u, z) , there are two parameters: direction of the edge and the vertices y and z , which leads to 4 different cases. For each case the edges (u, y) and (u, z) can have one of the labels $\{T'_1, T'_2\}$, hence each case has 4 sub cases. The cases are illustrated below:

- Case 1: $u \rightarrow y, u \rightarrow z$:**
For $(u, y) \in T_1$, $(u, z) \in T_2$ the expansion is illustrated in Figure A.8(a) and A.8(b). $(u, y) \in T_2$ and $(u, z) \in T_1$ can be expanded as in Figure A.8(a) and A.8(b) by exchanging vertices y and z . For $(u, y) \in T_1$, $(u, z) \in T_1$, the expansion is shown in Figure A.8(c). Expansion for $(u, y) \in T_2$, $(u, z) \in T_2$ can be obtained by label change ($T_1 \rightarrow T_2$, $T_2 \rightarrow T_1$ direction reversed) in Figure A.8(c).
- Case 2: $u \rightarrow y, z \rightarrow u$:**
For $(u, y) \in T_1$, $(z, u) \in T_1$ the expansion rules are defined in Figure A.8(e) and A.8(f). In $(u, y) \in T_2$, $(z, u) \in T_1$, the label change ($T_1 \rightarrow T_2$, $T_2 \rightarrow T_1$ direction reversed) in Figure A.8(a) and label change ($T_1 \rightarrow T_2$ direction reversed, $T_2 \rightarrow T_1$) in Figure A.8(b) defines expansion rules. For $(u, y) \in T_1$, $(z, u) \in T_1$, the expansion is shown in Figure A.8(d). The label change ($T_1 \rightarrow T_2$, $T_2 \rightarrow T_1$ direction reversed) in Figure A.8(d) gives the expansion rule for $(u, y) \in T_2$, $(z, u) \in T_2$.
- Case 3: $y \rightarrow u, u \rightarrow z$:**
This case is similar to Case 2 which can be seen by interchanging vertices y and z .
- Case 4: $y \rightarrow u, z \rightarrow u$:**
 $(y, u) \in T_1$, $(z, u) \in T_2$ can be expanded as in Figure A.8(a) and (b) by label change (T_1 direction reversed, T_2 direction reversed). Exchanging the vertices y and z and label change (T_1 direction reversed, T_2 direction reversed) in Figure A.8(a) and (b) defines expansion rules for $(y, u) \in T_2$, $(z, u) \in T_1$. In Figure A.8(c) the label change (T_1 direction reversed, T_2 direction reversed) and ($T_1 \rightarrow T_2$ direction reversed, $T_2 \rightarrow T_1$) gives expansion rules for $(y, u) \in T_1$, $(z, u) \in T_1$ and $(y, u) \in T_2$, $(z, u) \in T_2$ respectively.

To begin edge expansion, we start with the trivial REL and list of contractions (obtained in section A.3.1) in reverse order, i.e., taking the last contraction first. For each contraction, the expansion of edge $e_{v,u}$ is performed according to expansion case identified on basis of direction and labels of edges (u, y) and (u, z) . The adjacency of the vertex v is maintained by using the neighbouring vertices of v stored in the edge contraction. After each expansion, the subsequent contraction is removed from the list of contractions. Edge

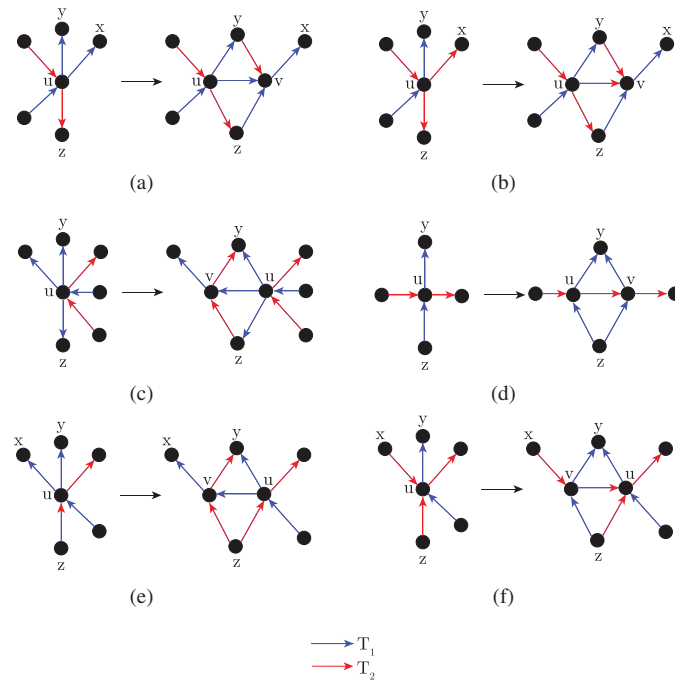


Figure A.8: Illustrating all cases for edge expansion.

expansion is continued until the list of contractions is empty as shown in Figure A.9.

After the process of expansion is over, we obtain a REL for the 4-completion PTPG.

A.4. Future work

In this section, we present features which we are planning to add to G2PLAN in the near future.

Rooms with functionality. At this stage, G2PLAN does not take into account the functionality of a room but it can easily be considered in near future by using the existing data sets of floorplans. For a trial, we used the data set [WFT*19] consisting of more than 80 thousand floorplans. The workflow of the software in this use case would be as follows:

- i. The user selects the rooms with dimensional constraints.
- ii. G2PLAN showcases all the graphs from the data set with the selected rooms.
- iii. The user selects one of the adjacency graphs and G2PLAN generates all topologically distinct floor plans to the user.

For example, assume that we require a living room, a master room, a kitchen, two bathrooms, a dining room and a balcony. Based on the proposed requirements, the adjacency graphs in Figure A.10 are derived [WFT*19] and floorplans are constructed using G2PLAN.

Future work in this direction would be to add some architectural rules to the rooms and rank the floor plans accordingly.

Circulations. A *circulation* refers to the spaces people move through and interact within a building. A *spanning circulation space* stands for a single interior passage adjacent to each of the rooms of a floorplan. In the near future, we are planning to integrate spanning circulation within a floorplan for a given entry point which can be further modified by removing some part of the circulation. For example, refer to Figure A.11 where for a given graph, floorplans with spanning circulations are shown having different entry points.

Floorplans with given boundary. Mathematically, it is challenging to construct floorplans with arbitrary input non-rectangular boundaries. In literature also, there do not exist graph algorithms for constructing such floorplans. In future, we aim to generalize the proposed approach in this paper for constructing floorplans with given boundary layouts, which can be done by first deriving the condition for the existence of such layouts. For example, consider the graph G in Figure A.12a for which there exists rectangular, L-shaped and U-shaped floorplans (see Figures A.12b - A.12d) but there does not exist T-shaped, Z-shaped and plus-shaped floorplans. Here it is interesting to note that the proposed floorplans are not trivial, i.e., it is not possible to transform one floorplan (say L-shaped) into another floorplan (say rectangular) by expanding or shrinking exterior walls while preserving adjacencies.

Game layouts. In 2014, Chongyang et al. [CNSA14] presented the construction of game level layouts for a given graph where the shape of blocks has also been considered. At this stage, G2PLAN is not considering the shape of rooms as input and will be considered in near future. For an illustration, corresponding to the graph in Figure A.13 taken from [CNSA14], two layouts are shown which are produced using G2PLAN.

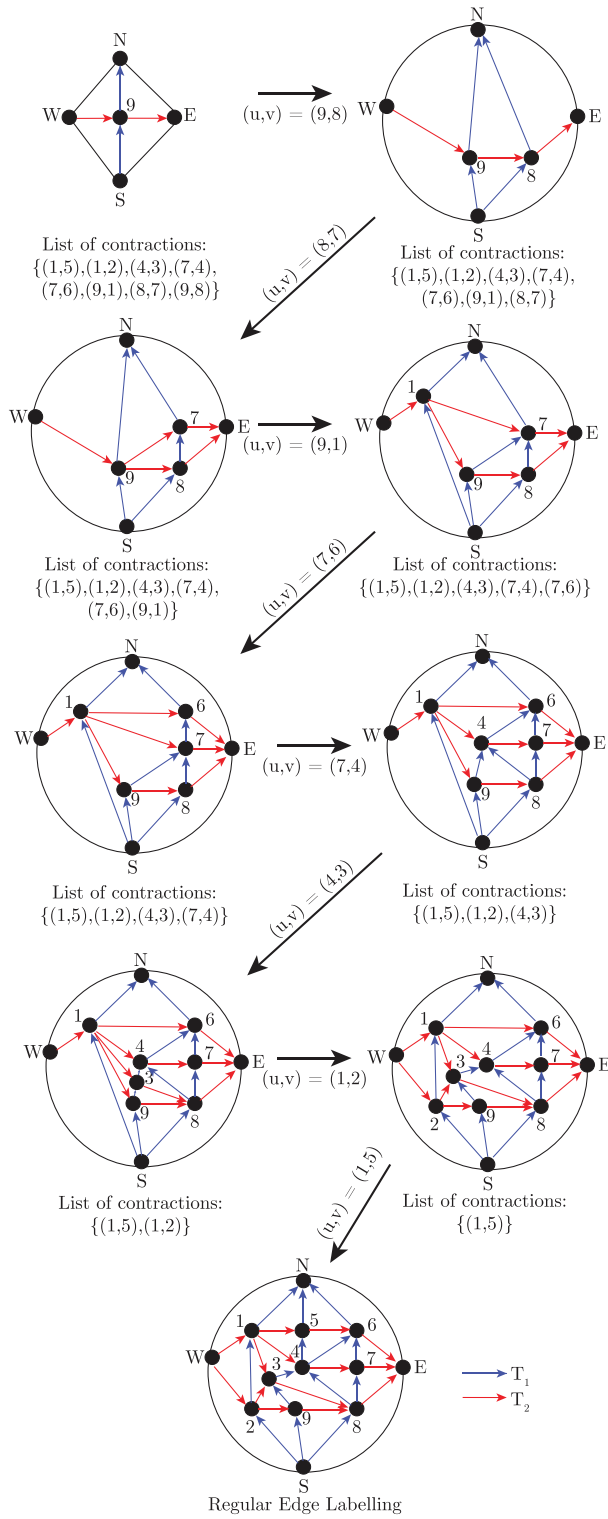


Figure A.9: Transforming a trivial REL into a non-trivial REL by edge expansion.

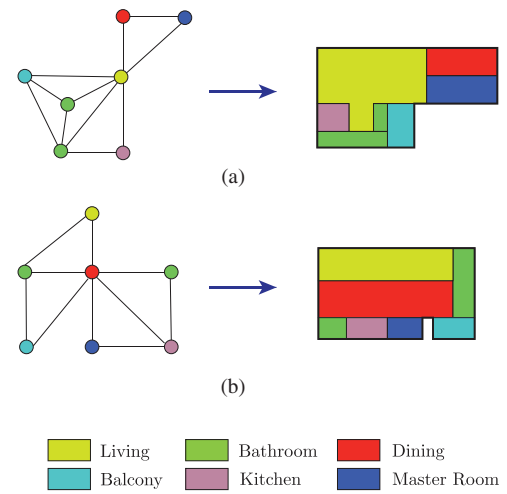


Figure A.10: Data driven floorplans.

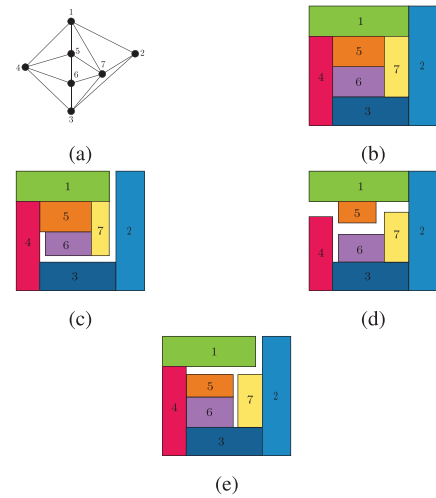


Figure A.11: Floorplans with spanning circulations corresponding to a given PTPG.

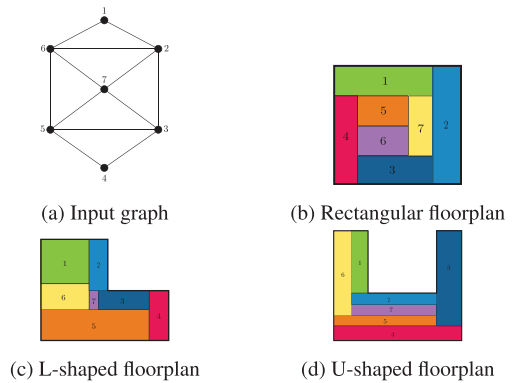


Figure A.12: Different shaped floorplans for a given graph.

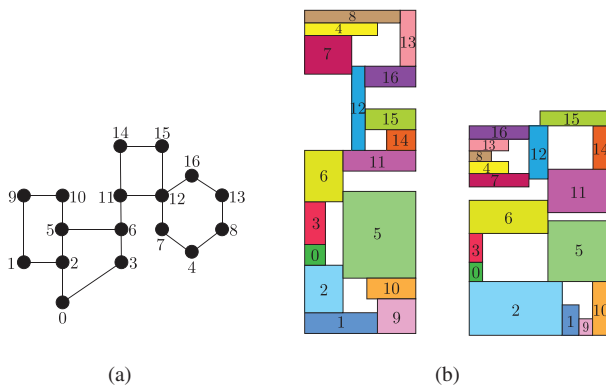


Figure A.13: Game layouts corresponding to a given graph.

References

- [BBHP04] BERRY A., BLAIR J. R., HEGGERNES P., PEYTON B. W.: Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica* 39, 4 (2004), 287–298.
- [BS87] BHASKER J., SAHNI S.: A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks* 17, 3 (1987), 307–317.
- [BYMW13] BAO F., YAN D.-M., MITRA N. J., WONKA P.: Generating and exploring good building layouts. *ACM Transactions on Graphics* 32, 4 (2013), 1–10.
- [CNSA14] CHONGYANG M., NICHOLAS V., SYLVAIN L., ALLA: Game level layout from design specification. *Computer Graphics Forum* 33, 2 (2014), 95–104.
- [Dua05] DUARTE J. P.: A discursive grammar for customizing mass housing: the case of siza's houses at malagueira. *Automation in Construction* 92 (2005), 151–165.
- [EMSV12] EPPSTEIN D., MUMFORD E., SPECKMANN B., VERBEEK K.: Area-universal and constrained rectangular layouts. *SIAM Journal on Computing* 41, 3 (2012), 537–564.
- [HHT*20] HU R., HUANG Z., TANG Y., VAN KAICK O., ZHANG H., HUANG H.: Graph2plan: Learning floorplan generation from layout graphs. *ACM Transactions on Graphics* 39, 4 (2020), 1–14.
- [Joh75] JOHNSON D. B.: Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing* 4, 1 (1975), 77–84.
- [KB91] KANT G., BODLAENDER H. L.: Planar graph augmentation problems. In *Proceedings of the Workshop on Algorithms and Data Structures* (1991), Springer, pp. 286–298.
- [KH93] KANT G., HE X.: Two algorithms for finding rectangular duals of planar graphs. In *Proceedings of the International Workshop on Graph-Theoretic Concepts in Computer Science* (1993), Springer, pp. 396–410.
- [KK84] KOZMINSKI K., KINNEN E.: An algorithm for finding a rectangular dual of a planar graph for use in area planning for vlsi integrated circuits. In *Proceedings of the 21st Design Automation Conference Proceedings* (1984), IEEE, pp. 655–656.
- [KK85] KOZMIŃSKI K., KINNEN E.: Rectangular duals of planar graphs. *Networks* 15, 2 (1985), 145–157.
- [KS21] KUMAR V., SHEKHAWAT K.: A transformation algorithm to construct a rectangular floorplan. *Theoretical Computer Science* 871 (2021), 1–13.
- [Lev64] LEVIN P. H.: Use of graphs to decide the optimum layout of buildings. *The Architects' Journal* 7 (1964), 809–815.
- [MM10] MARSON F., MUSSE S. R.: Automatic real-time generation of floor plans based on squarified treemaps algorithm. *International Journal of Computer Games Technology 2010* (2010), 1–11.
- [MSK10] MERRELL P., SCHKUFZA E., KOLTUN V.: Computer-generated residential building layouts. *ACM Transactions on Graphics* 29, 6 (2010), 1–12.
- [MSL76] MITCHELL W. J., STEADMAN J. P., LIGGETT R. S.: Synthesis and optimization of small rectangular floor plans. *Environment and Planning B: Planning and Design* 3, 1 (1976), 37–70.
- [MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM Transactions on Graphics* 25, 3 (2006), 614–623.
- [NHC*21] NAUATA N., HOSSEINI S., CHANG K.-H., CHU H., CHENG C.-Y., FURUKAWA Y.: House-gan++: 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* (2021), pp. 13632–13641.
- [NM19] NISZTUK M., MYSZKOWSKI P. B.: Hybrid evolutionary algorithm applied to automated floor plan generation. *International Journal of Architectural Computing* 17, 3 (2019), 260–283.
- [RBD12] REGATEIRO F., BENTO J., DIAS J.: Floor plan design using block algebra and constraint satisfaction. *Advanced Engineering Informatics* 26, 2 (2012), 361–382.
- [RBM01] ROY S., BANDYOPADHYAY S., MAULIK U.: Proof regarding the np-completeness of the unweighted complex-triangle elimination (CTE) problem for general adjacency graphs. *IEEE Proceedings-Computers and Digital Techniques* 148, 6 (2001), 238–244.
- [RGG13] RODRIGUES E., GASPAR A. R., GOMES Á.: An evolutionary strategy enhanced with a local search technique for the space allocation problem in architecture, part 2: Validation and performance tests. *Computer-Aided Design* 45, 5 (2013), 898–910.
- [RHW82] ROTH J., HASHIMSHONY R., WACHMAN A.: Turning a graph into a rectangular floor plan. *Building and Environment* 17, 3 (1982), 163–173.

- [She18] SHEKHAWAT K.: Enumerating generic rectangular floor plans. *Automation in Construction* 92 (2018), 151–165.
- [SSHW20] SHI F., SOMAN R. K., HAN J., WHYTE J. K.: Addressing adjacency constraints in rectangular floor plans using monte-carlo tree search. *Automation in Construction* 115 (2020), 1–14.
- [Ste73] STEADMAN P.: Graph theoretic representation of architectural arrangement. *Architectural Research and Teaching* (1973), 161–172.
- [SUBJ21] SHEKHAWAT K., UPASANI N., BISHT S., JAIN R. N.: A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction* 127 (2021), 1–21.
- [Tar72] TARJAN R.: Depth-first search and linear graph algorithms. *SIAM Journal on Computing* 1, 2 (1972), 146–160.
- [TC90] TANG H., CHEN W.-K.: Generation of rectangular duals of a planar triangulated graph by elementary transformations. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (1990), IEEE, pp. 2857–2860.
- [USS20] UPASANI N., SHEKHAWAT K., SACHDEVA G.: Automated generation of dimensioned rectangular floorplans. *Automation in Construction* 113 (2020), 1–12.
- [WFLW18] WU W., FAN L., LIU L., WONKA P.: Miqu-based layout design for building interiors. *Computer Graphics Forum* 37, 2 (2018), 511–521.
- [WFT*19] WU W., FU X.-M., TANG R., WANG Y., QI Y.-H., LIU L.: Data-driven interior plan generation for residential buildings. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–12.
- [WLZ20] WANG X.-Y., LIU Y.-F., ZHANG K.: A graph grammar approach to the design and validation of floor plans. *The Computer Journal* 63, 1 (2020), 1–14.
- [WPT*21] WAMIQ P., PAUL G., TOM K., LEONIDAS J. G., PETER W.: Generative layout modeling using constraint graphs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 6690–6700.
- [WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Transactions on Graphics* 22, 3 (2003), 669–677.
- [WYZ18] WANG X.-Y., YANG Y., ZHANG K.: Customization and generation of floor plans based on graph transformations. *Automation in Construction* 94 (2018), 405–416.
- [ZS18] ZAWIDZKI M., SZKLARSKI J.: Effective multi-objective discrete optimization of truss-z layouts using a GPU. *Applied Soft Computing* 70 (2018), 501–512.
- [ZWG*21] ZHENGDA L., WANG T., GUO J., MENG W., XIAO J., ZHANG W., XIAOPENG Z.: Data-driven floor plan understanding in rural residential buildings via deep recognition. *Information Sciences* 567 (2021), 58–74.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Supporting Information