Artificial Intelligence for Engineering Design, Analysis and Manufacturing

www.cambridge.org/aie

Automated generation of floor plans with minimum bends

Pinki, Krishnendra Shekhawat and Akshat Lal

Department of Mathematics, BITS Pilani, Rajasthan, India

Research Article

Cite this article: P, Shekhawat K and Lal A (2025). Automated generation of floor plans with minimum bends. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **39**, e3, 1–20
https://doi.org/10.1017/S0890060424000179

Received: 02 January 2024
Revised: 08 July 2024
Accepted: 24 July 2024

Keywords:
algorithm; floor plan; graph theory; plane triangulated graph

Corresponding author:
Pinki;
Email: 456pinkiyadav@gmail.com

Abstract

The generation of floor plan layouts has been extensively studied in recent years, driven by the need for efficient and functional architectural designs. Despite significant advancements, existing methods often face limitations when dealing with specific input adjacency graphs or room shapes and boundary layouts. When adjacency graphs contain separating triangles, the floor plan must include rectilinear rooms (non-rectangular rooms with concave corners). From a design perspective, minimizing corners or bends in rooms is crucial for functionality and aesthetics. In this article, we present a Python-based application called G-Drawer for automatically generating floor plans with a minimum number of bends. G-Drawer takes any plane triangulated graph as an input and outputs a floor plan layout with minimum bends. It prioritizes generating a rectangular floor plan (RFP); if an RFP is not feasible, it then generates an orthogonal floor plan or an irregular floor plan. G-Drawer modifies orthogonal drawing techniques based on flow networks and applies them on the dual graph of a given PTG to generate the required floor plans. The results of this article demonstrate the efficacy of G-Drawer in creating efficient floor plans. However, in future, we need to work on generating multiple dimensioned floor plans having non-rectangular rooms as well as non-rectangular boundary. These enhancements will address both mathematical and architectural challenges, advancing the automated generation of floor plans toward more practical and versatile applications.

Introduction

The use of graphs for architectural design had been practiced by researchers from past many years (Levin, 1964; Cousin, 1970; Grason, 1970; Steadman, 2006). Adjacencies are very well represented by bubble diagrams, an adjacency graph directly suggests the required interior and exterior rooms, wall adjacencies can also be represented by a graph, weighted graphs can be used to input room dimensions, presence of separating triangles suggest the requirement of non-rectangular rooms, and many other FP properties can be asked by user in the form of a graph. At the same time, there is one-to-one correspondence between a dual graph and an FP. Graphs can also handle the generation of dimensioned layouts or multiple layouts. Because of all these reasons, in the recent years, many graph algorithms and theoretical results have been proposed related to FPs.

In 2006, Steadman (2006) proposed the logic of having most of the rooms and buildings being rectangular. He stated that the limitation of togetherness in rooms package and dimension flexibility permitted by rectangularity clarify the prevalence of the right edge in architectural plans. From graph theory perspective, we can easily maintain rectangularity by mainly considering the adjacency graph without separating triangles. At the same time, if the input graph has separating triangles and it is required to satisfy all the adjacencies, then rectilinear rooms need to be introduced within the FP; the rooms with at least one bend or concave corner (e.g., a rectangular room has no bends, an L-shaped room has one bend and a T-shaped room has two bends). In this article, we propose a set of graph algorithms to generate FPs with minimum number of bends.

Literature review

In general, the problem is to automate the generation of FPs for given adjacencies. To better understand the work done in the past and the proposed work, here we first present a few terminologies.

A representation P of a plane graph G (a graph without any edge crossings) is a *rectilinear representation* if every vertex v_i of G corresponds to a rectilinear polygon P_i in P. Each edge e_i of G between vertices v_r and v_s represents the wall adjacency between the corresponding polygons P_r and P_s. A representation P is called an FP for G if it is a rectilinear representation with rectilinear plan boundary (see Figure 1). The polygons within an FP are called rooms.

An FP is a *rectangular FP* (RFP) if all of its rooms are rectangular along with its boundary (see Figure 1a). An *orthogonal FP* (OFP) has rectangular boundary and at least one rectilinear room, a

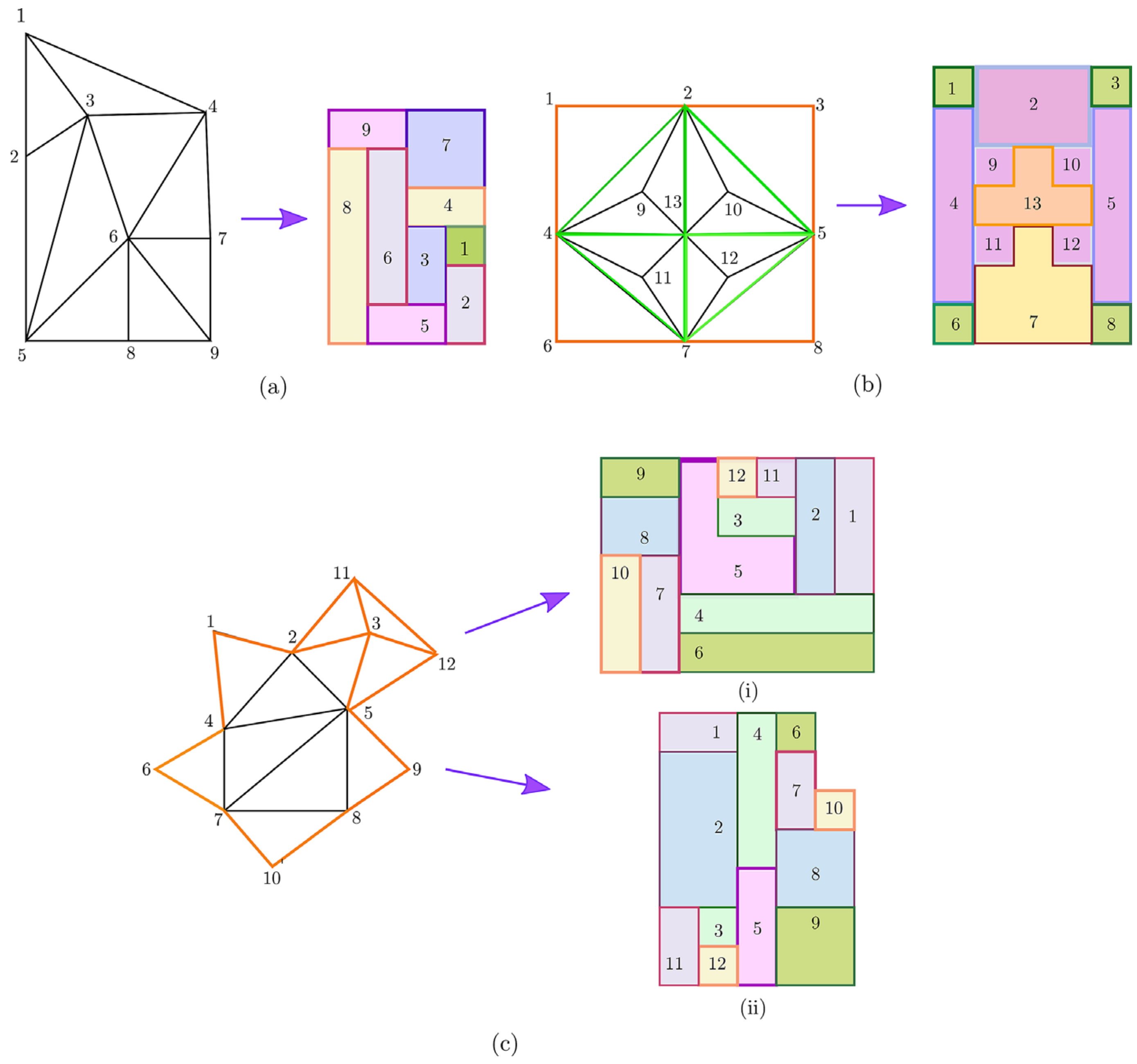


Figure 1. A plane graph representing CIPs in orange color and STs in green color, with their corresponding floor plans: (a) rectangular floor plan, (b) orthogonal floor plan, and (c) (i) orthogonal floor plan and (ii) irregular floor plan, respectively.

room with a concave corner (see Figure 1b). An *irregular FP* (IFP) has rectilinear boundary (see Figure 1c (ii)).

An *orthogonal drawing* (OD) of a plane graph G is an embedding in the plane in which all edges are drawn as an alternating sequence of horizontal and vertical line segments. The point of intersection of horizontal and vertical line segments in an OD is called *bends*. An OFP can also be defined in terms of an OD. An OFP of G is an OD of its dual G^* where every face corresponds to a vertex of G. A bend in an OFP is defined differently from a bend in an OD, that is, it represents a concave corner of a room (refer to Figure 3).

A *planar embedding* is an embedding of a graph drawn in the plane such that edges intersect only at their endpoints, that is, it has no edge crossings. There exists multiple planar embedding of a graph. Hence, the choice of an embedding has an impact on its OD as well as FP.

The automated generation of FPs using graph theory began with the generation of RFPs. In 1980s, comprehensive studies were presented for the existence and construction of RFPs for given adjacencies (Koźmiński and Kinnen, 1985; Bhasker and Sahni, 1987; Bhasker and Sahni, 1988; Rinsma, 1988). It has been proved that an RFP for a plane triangulated graph (PTG) G exists if and only if it has no more than four CIPs and no separating triangles (Koźmiński and Kinnen, 1985; Bhasker and Sahni, 1987). In recent years, a lot of work done has been published for the automated generation of RFPs (Bisht et al., 2022; Shekhawat et al., 2021; Upasani et al., 2020). In 2013, Pirouz Nourian introduced SYNTACTIC (Nourian et al., 2013):

A computational tool suite for analyzing/designing architectural configurations. This tool provides automated generation of FPs specifically for properly triangulated planar graphs. The FPs generated by this tool are restricted solely to rectangular rooms.

The work on OFPs started in 1993, when Sun and Sarrafzadeh (1993) presented the existence conditions for graphs for which RFPs do not exist and presented the construction of OFPs. In coming years, many papers have been published on OFPs which are briefed as follows:

In 1993, Yeap and Sarrafzadeh (1993) gave existence conditions for OFPs. They proposed that a room with two concave corners, that is, 2-CRMs[1] are both necessary and sufficient for the existence of an OFP for a given graph. In a subsequent study, He (1999) proposed a linear time floor planning algorithm to construct an OFP using canonical labeling. In 2003, Brandenburg et al. (2003) discussed graph theoretic open problems related to minimizing the number of bends in a 2D orthogonal representation of a graph (Problems 14 and 15). In the same year, Kurowski (2003) gave a polynomial time algorithm for constructing an OFP with restricted the number of T-shaped rooms, that is, $\frac{1}{2}(n-2)$. In 2011, Jokar and Sangchooli (2011) proposed the face area concept for the construction of OFPs. In 2012, Ueckerdt (2012) used Schnyder wood technique the generating an OFP. In 2013, Alam et al. (2013) stated that eight-sided rooms are both necessary and sufficient for building an OFP. Chang and Yen (2014) demonstrated that 12-sided rooms are required for a T-free OFP. They also presented the construction of OFPs using only monotone staircase rooms.

In 2022, Pinki and Shekhawat (2022a) presented a generalized linear-time algorithm for the construction of an FP with no restriction on the number of bends.

Due to the applications of FPs in architectural layout arrangements and circuit designing, they are not limited to rectangular boundary. A lot of research has been conducted on IFPs using different approaches such as pseudo geometric dual (Baybars and Eastman, 1980), canonical representation (Nummenmaa, 1992), perfect matching (Miura et al., 2006), and so forth. In the last few decades, researchers came up with distinct approaches for the automatic generation of FPs, that is, shape grammar (Duarte, 2001; Müller et al., 2007; Wu et al., 2013), artificial intelligence (He et al., 2022), data-driven techniques (Wu et al., 2019; Lu et al., 2021), neural networks (Hu et al., 2020), generic optimization (Laignel et al., 2021), constrained optimization (Para et al., 2021), generative adversarial layout refinement network (Nauata et al., 2021), integer linear programming (Klesen and Wolff, 2021), and so forth. Shape grammar is one of the most common approaches used for the automated generation of FPs by architects.

In the recent years, most the papers published related to the automated generation of FPs are based on machine learning techniques (Hu et al., 2020; Wu et al., 2019; Azizi et al., 2022). One of the main goals of machine learning models is to adapt to new, previously unseen data, assuming it is drawn from the same distribution. Properly trained models should capture meaningful patterns and relationships in the dataset to generalize beyond the scope of the training dataset. Therefore, while current methods excel in generating significant layouts based on available data, they need to evolve to address additional constraints and ensure adaptability to new scenarios. For example, in this article, we proposed to generate layouts with minimum rooms' corners which might be hard for data-driven approaches to handle.

We conducted a comparative analysis between the FP generated by G-Drawer and those produced by Graph2Plan (Hu et al., 2020) and House-GAN++ (Nauata et al., 2021). Our evaluation focused on several key aspects, that is, adjacencies, the shape of boundary and the shapes of rooms. The floor generated by Graph2Plan (Hu et al., 2020) and House-GAN++ (Nauata et al., 2021) either violate the specified adjacencies or include more than the required rectilinear rooms and rectilinear boundary (see Figure 2).

Gaps in the existing literature and our contributions

In 2022, Pinki and Shekhawat (2022b) presented a mathematical approach for computing the minimum number of bends required in an orthogonal FP for a given graph G. This approach relies solely on critical separating triangles and K_4's. However, the paper is restricted to a specific class of graphs and it did not include the implementation of construction algorithms. In 2023, Shekhawat et al. (2023) presented a graph-theoretic method for generating FPs with customized room shapes. It includes a Python-based application for creating FPs with specific room shapes but it generates FPs with many more bends than the minimum required bends. Also, finding a planar embedding that leads to an OD with a minimum number of bends is NP-hard (Garg and Tamassia, 2001). In this work, we are considering a fixed embedding of a dual graph G^* (triconnected cubic graph) of a given graph G and constructs an OD which may not have minimum bends for G^* but corresponds an OFP with minimum bends for G.

Further, in literature, a lot of approaches have been proposed and discussed for the generation of OFPs without considering the number of bends in the obtained FP. In this article, we present a Python-based application called G-Drawer developed by modifying the approach stated by Klose (2012) as per our requirements. G-Drawer considers all classes of PTGs and for a given graph G, it generates an FP with a minimum number of bends by using the following steps (see Figure 3):

(a) construct the dual G^* of G,
(b) construct an embedded preserved OD of G^* whose every face corresponds to a vertex of G (using topology shape metric approach).

More specifically, using flow networks, this article presents a user interface G-Drawer for generating FPs with minimum bends (minimum rectilinear rooms) where there is no restriction on the input graph, that is, it considers any PTG. Depending on the input graph, it generates rectangular or orthogonal or IFP.

The flow of the paper is organized as follows. Literature on FPs, and contributions based on gaps in existing literature are given in section 'Gaps in the existing literature and our contributions'. Preliminary definitions, terminologies, and some important notations are given in section 'Preliminaries', which are used throughout the paper. Section 'G-Drawer for construction of FPs for PTG with minimum bends' describes the key idea for generating FPs with minimum bends. This section also presents the working of G-Drawer with explanatory examples. Section Conclusion serves as the conclusion of the paper, while

[1]k-CRM: a room with k concave corners.

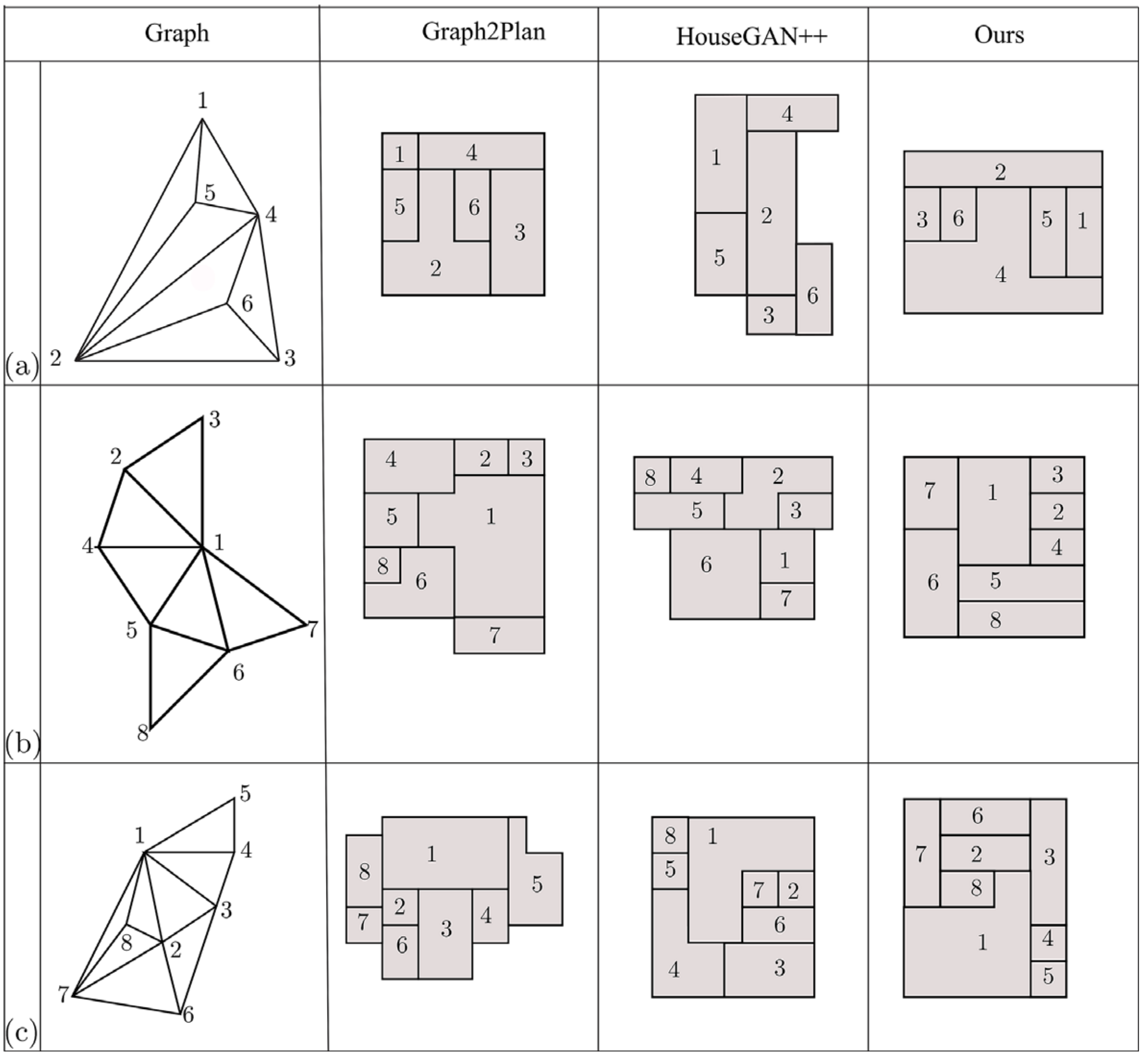


Figure 2. Comparison between G-Drawer, Graph2Plan (Hu et al., 2020), and House-GAN++ (Nauata et al., 2021) on the basis of given adjacencies, the shape of boundary and shapes of rooms.

section 'Limitations and future work' discusses the limitations of the current study and presents future directions.

Preliminaries

In this section, we provide a few definitions that will be later used in the paper.

A graph is said to be *planar* if it can be drawn on the plane without edge crossings, that is, its edges intersect only at their endpoints. Such a graph drawing with no edge crossings is called a *plane graph* or *planar embedding* of the graph. A planar embedding of a planar graph divides the plane into connected components called *faces.* The unbounded face is called the *exterior face.* All other faces except the exterior face are *internal faces.* A graph G is said to be *biconnected* if it does not have a cut-vertex (see Figure 4a). An edge of a biconnected planar graph is a *shortcut* if it is incident to two vertices on the outer boundary of G but not a part of it. In Figure 4c, edges $(4, 2)$; $(2, 5)$; $(5, 8)$; $(8, 7)$; and $(7, 4)$ are shortcuts. A *corner implying path* (CIP) of a biconnected planar graph is a path $w_1, w_2, w_3, \dots w_n$ which lies on the outer boundary of G where (w_1, w_n) is a shortcut and vertices $w_2, w_3, \dots w_n$ are not a part of any other shortcuts. In Figure 4c, there are four CIPs, namely, (1,2,4); (2,11,12,5); (5,9,8); (8,10,7); and (7,6,4).

Definition 1 [Plane triangulated graph]: A *plane triangulated graph* (PTG) is a biconnected plane graph G which satisfies the following conditions:

(i) all of its internal faces are triangular,
(ii) exterior face can or cannot be triangular.

Definition 2 [Separating triangle] (Pinki and Shekhawat, 2022b): Any triangle in a plane graph having at least one interior vertex is a *separating triangle* (ST). The complete graph of order 4, that is, K_4 is a smallest ST containing only one vertex of degree 3. An ST is *critical* if it contains a vertex of degree >3 in its interior and it is denoted as CST. A CST is an *outer* CST if it forms the boundary of

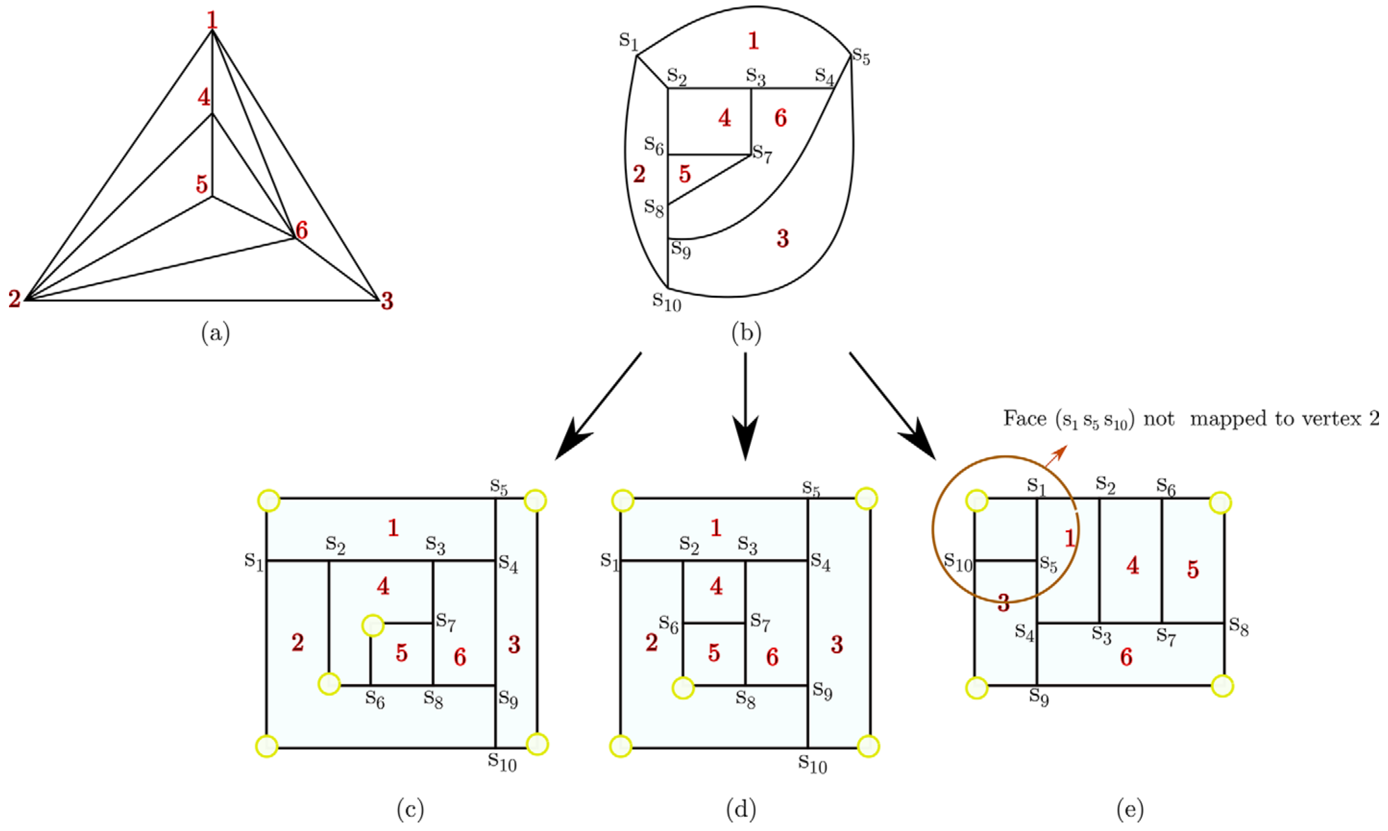


Figure 3. (a) A plane triangulated graph *G*, (b) dual graph *G** of *G*, (c) embedding preserved orthogonal drawing (without minimum bends) of *G** which is an orthogonal floor plan for *G* (without minimum bends), and (d) embedding preserved orthogonal drawing (with minimum bends) and an orthogonal floor plan with minimum bends which is obtained by G-Drawer. (e) An orthogonal drawing of *G** which is not embedding preserved but with minimum number of bends in general and it is not an orthogonal floor plan of *G*.

the external face and it is denoted as CST. A CST is categorized as follows:

i. *Simple CST*: It is a CST that does not contain any other CST or if it contains a CST then it should have either common vertices or edges with the contained CST. In Table 1 Case 1a, $\Delta 123$ represents a simple CST that contains a CST $\Delta 134$ with a shared edge (1,3).
ii. *Nested CST*: It is a CST containing a disjoint CST, that is, it does not share a vertex or an edge with the contained CST. In Table 1 Case 1c, $\Delta 456$ represents a nested CST that contains a disjoint CST $\Delta 123$.

Definition 3 [Containment tree]: A *containment tree T* represents the hierarchy of CSTs of the given graph. In this tree, each vertex represents a CST except the root of the containment tree, which may or may not be a CST. Two vertices are adjacent if one of the vertex is contained in another vertex. Its hierarchical order from root to leaves represents the containment of a vertex in its parent vertex. The containment tree is denoted as CT and its leaves are denoted as L_T. In Table 1 Case 4, $\Delta 123$ is a root vertex of a CT containing two CSTs $\Delta 124$ and $\Delta 234$. These two CSTs are leaves of the CT.

Remark. PTGs can be classified into the following categories (see Figure 4):

Type 1: PTPG: A PTG with no STs and exterior face of length at least 4 and no more than 4 CIPs. ▷ *Properly triangulated planar graph* (PTPG)

Type 2: A PTG with STs and exterior face of length 3.▷ *Maximal planar graph* (MPG)(MPG has no CIPs)

Type 3: A PTG with no STs and more than 4 CIPs.

Type 4: A PTG with STs and exterior face of length at least 4 and no more than 4 CIPs.

Type 5: A PTG with STs and more than 4 CIPs.

Notations:
FP: floor plan/s, RFP: rectangular FP, OFP: orthogonal FP, IFP: irregular FP, PTG: plane triangulated graph, PTPG: properly triangulated plane graph, MPG: maximal plane graph, CIP: corner implying path, K_4: 4-vertex complete graph, ST: separating triangle, CST: critical ST, CST°: outer CST, $B_{\min}$: minimum number of bends in an OFP, CT: containment tree, $|L_T|$: the number of leaves of a containment tree.

G-Drawer for construction of FPs for PTG with minimum bends

This section discusses the working of G-Drawer which has been developed in Python for constructing FPs with minimum bends. The input to G-Drawer is any PTG. The working steps of G-Drawer can be summarized under the following heads:

Checking for existence of FPs

For a PTG *G*, there always exist an FP. Hence, the program takes the input graph and checks the basic criterion for the existence of an FP, which are as follows:

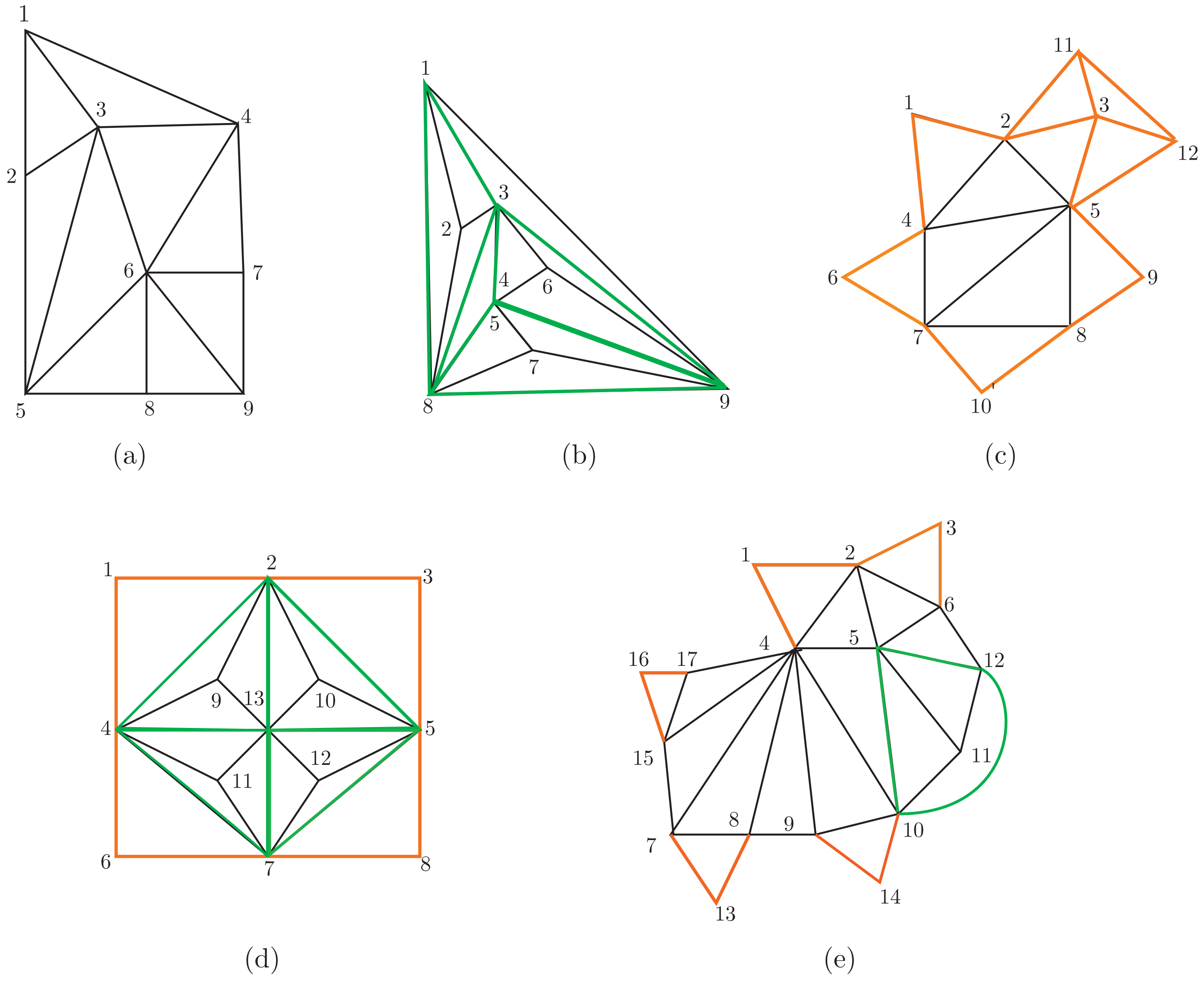


Figure 4. Types of plane triangulated graphs (PTGs) representing CIPs in orange and STs in green color: (a) properly triangulated plane graph (PTPG), (b) maximal plane graph (MPG), (c) a PTG with no STs and more than four CIPs, (d) a PTG with STs and exterior face of length at least four and no more than 4 CIPs, and (e) a PTG with STs and more than four CIPs.

1. Planarization: The task of planarization is to check if the given graph is a plane graph. We are using the Boyer–Myrvold algorithm for planarity testing (Boyer and Myrvold, 1999). The program takes the input graph via a UI that allows the user to place the vertices and edges on the canvas. If the input graph is a plane graph, the program proceeds with the dual generation steps. Otherwise, an error message is displayed at the bottom and the program execution is halted (see Figure 5a).
2. Biconnectivity: The program checks if the given input graph is biconnected (i.e., it does not have a cut-vertex). This is done by searching for the existence of a cut vertex in the graph. If there exists a cut vertex, then an error message is displayed, and the execution stops. Otherwise, the program goes to the step of dual generation (see Figure 5b).
3. Triangularity: The program checks if the given input graph is triangular, that is, every simple cycle of the graph is a triangle. If the condition is satisfied, the graph proceeds with the dual generation steps. Otherwise, the program is halted and an error message is displayed (see Figure 5c).

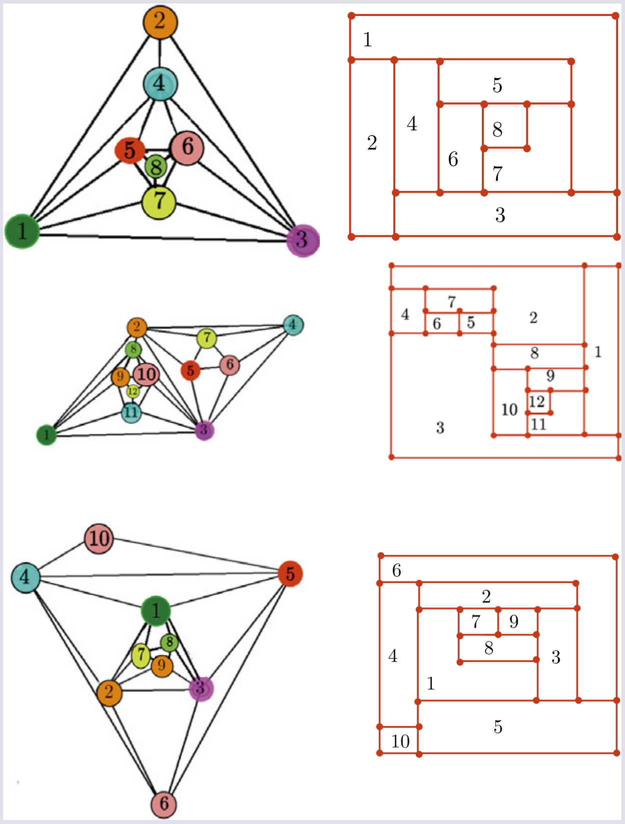
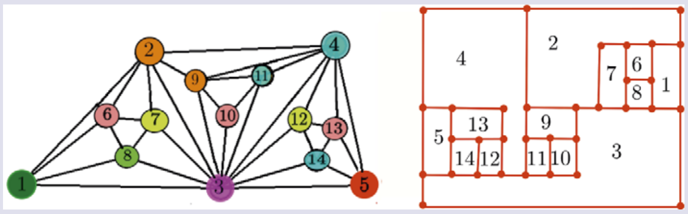
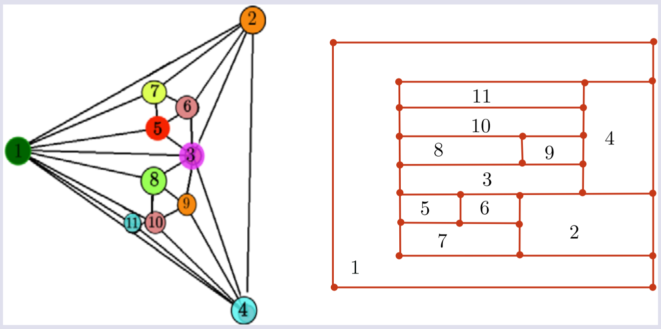
Construction of FPs

G-Drawer presents the construction of different types of FPs, that is, an RFP, an OFP, and an IFP. In 1985, Kozminski and Kinnen (1985) proposed the following theorem for the existence of an RFP.

Theorem 1. (Koźmiński and Kinnen, 1985; Bhasker and Sahni, 1987) A biconnected PTG with no STs and exterior face of length at least 4 has an RFP if and only if it has no more than four CIPs (see Figure 6).

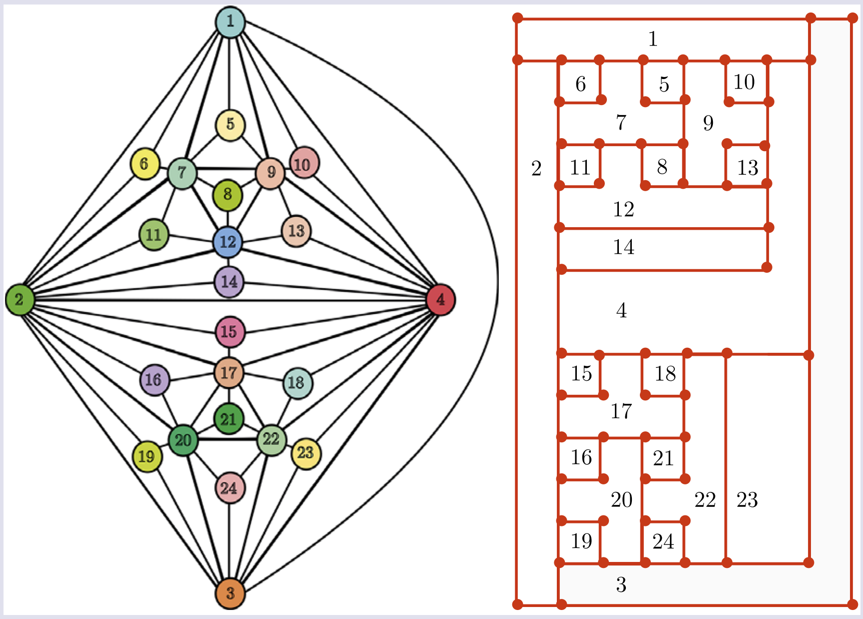
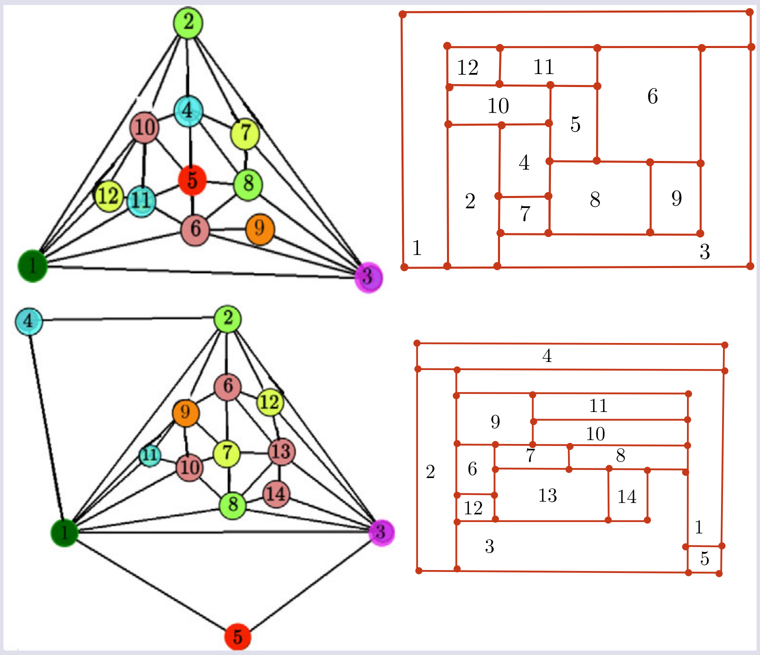
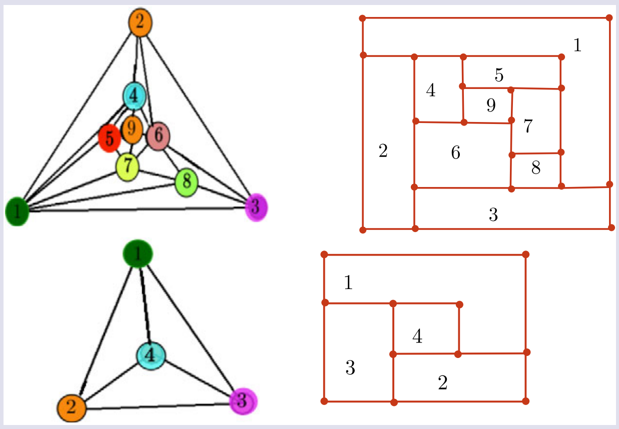
It is clear from Theorem 1 that if a PTG *G* contains either an ST or the number of CIPs >4 or the exterior face is triangular then there does not exist an RFP corresponding to it. Hence, it is required to construct an OFP for *G* where bends need to be introduced to satisfy the adjacency requirements. Bends in an FP corresponds to either an ST or a CIP, if the number of CIPs >4 (see Figure 7). When a graph includes a separating triangle, introducing bends into the FP becomes necessary to meet all adjacency requirements. Similarly,

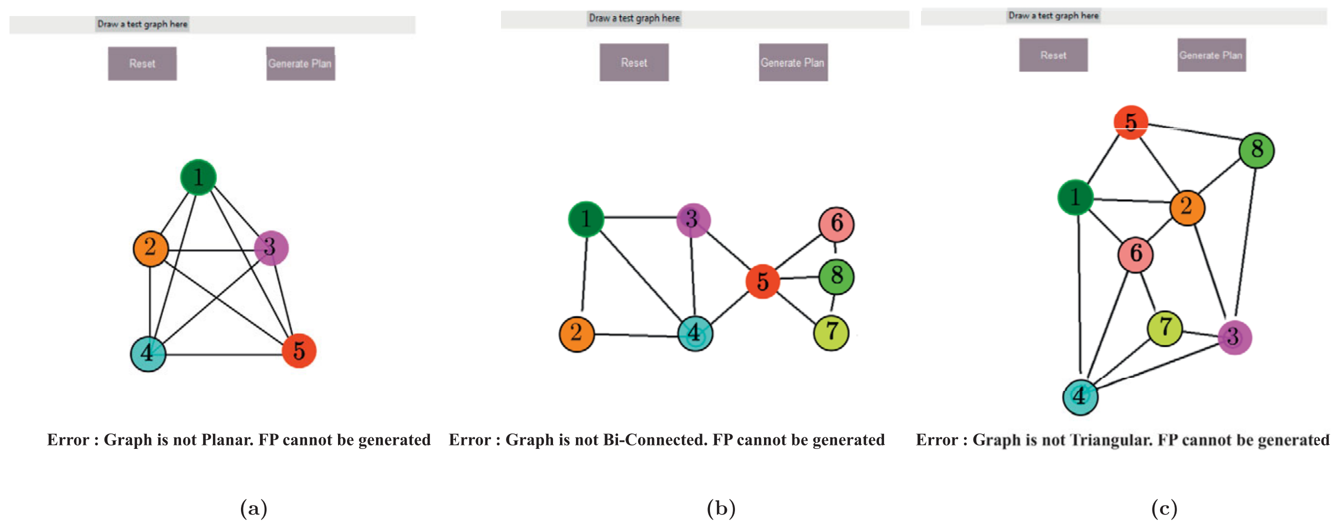
Table 1. Illustration of cases of Theorem 2 with explanatory examples

Cases	Bend	Graph and its corresponding floor plan
If |CST| ≥ 2 and one of the conditions holds which are as follows: (a) Each CST is a leaf of a containment tree except root of containment tree which is a CST (that is CST°) (b) Each CST is a leaf of containment tree. (c) Each CST is either a leaf of a containment tree or an intermediate CST	$B_{\min} = \rho$	
If |CST| ≥ 2 and there exists a $K_4 \subseteq$ CST which do not share an edge with any of the CSTs	$B_{\min} = \rho$	
If |CST| ≥ 2 and there exists at least one CST containing K_4 and sharing an edge with K_4	$B_{\min} = \rho - 1$	

(*Continued*)

Table 1. (*Continued*)

Cases	Bend	Graph and its corresponding floor plan
If $\|CST\| \geq 2$ and there exists at least two CSTs containing K_4 and sharing an edge with K_4. If the shared edges have a vertex in common	$B_{min} = \rho - 2$	
If $\|CST\| = 1$ and $\|K_4\| \geq 1$ and there exists a $K_4 \subseteq CST$ sharing an edge or a vertex with the CST	$B_{min} = \rho$	
If none of the above cases is applicable	$B_{min} = \rho + 1$	



Error : Graph is not Planar. FP cannot be generated

Error : Graph is not Bi-Connected. FP cannot be generated

Error : Graph is not Triangular. FP cannot be generated

Figure 5. G-Drawer gave an error if the input graph is (a) not a planar graph, (b) not bi-connected, and (c) not triangular.

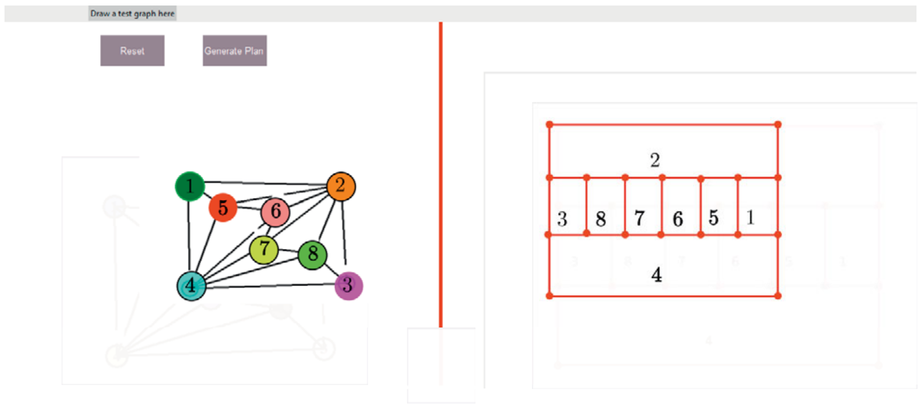


Figure 6. A PTPG and its corresponding RFP obtained using G-Drawer.

when there are four CIPs, their adjacency corresponds to the four corners of an RFP. However, if the count exceeds four, it leads to more than four corners, resulting in a rectilinear representation with a rectilinear boundary.

G-Drawer presents the automated generation of an RFP for PTPG, an OFP for PTGs (except for PTPG) with the minimum number of bends. An IFP is also generated if number of CIPs >4 (see Figure 8).

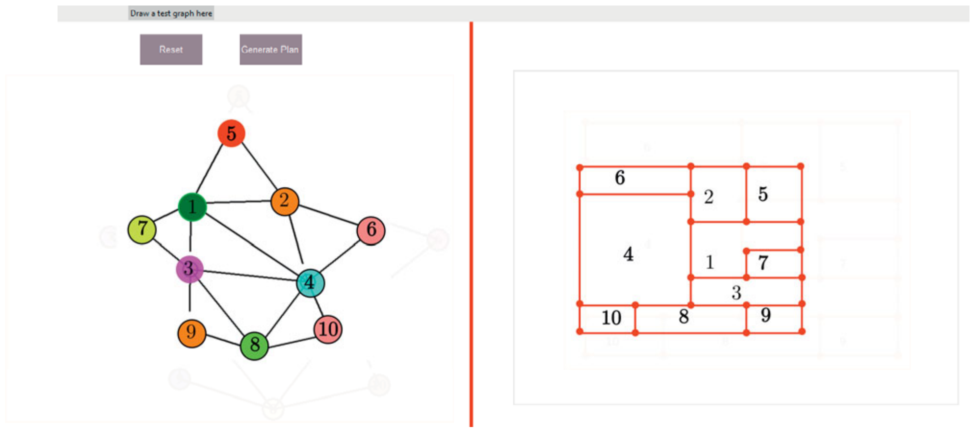
Theorem 2. Bends in an OFP corresponding to a PTG G is $\rho - 2 \le B_{\min} \le \rho + 1$ where

$$\rho = \begin{cases} |L_T(G)| + |K_4(G)| + |\text{nested CST}(G)| & \text{if} |\text{CIPs}| \le 4; \\ |L_T(G)| + |K_4(G)| + |\text{nested CST}(G)| + |\text{CIPs}| - 4 & \text{if} |\text{CIPs}| > 4. \end{cases}$$

Proof.

The existence of bends in a PTG G depends on triangles that are not a face, that is, on CST or K_4 and CIPs. This implies that contributors to the bends in an OFP for G are:

1. CSTs
   (a) Root of a containment tree can either be a CST or not. If it is a CST then it is denoted by CST° and CST° has three vertices on the boundary but an OFP has four sides at the boundary. The rooms corresponding to three boundary vertices of CST° can form the boundary of a required OFP if any of them has a bend (refer to Case 4 of Table 1 where a bend can be seen on room M_3 of the OFP corresponding to CST° ($\Delta 123$) of the given graph). Otherwise, the root of a containment tree does not contribute to a bend in an OFP.
   (b) CSTs that are leaves of a containment tree: Each CST is represented by a triangular face in its dual graph and a triangular face needs to be made rectangular which generates a room with a bend in the required OFP (see Case 4 of Table 1 where bends can be seen on rooms M_3 and M_4 corresponding to leaves CSTs [$\Delta 124$] and [$\Delta 234$]). Hence, all CSTs introduce $|L_T|$ bends in an OFP.
   (c) Intermediate CSTs of a containment tree: Intermediate CSTs are either simple CST or nested CST.
      (i) If an intermediate CST is a simple CST then it has a shared edge with its descendant CSTs or its child CSTs in CT. Because of the shared edge, the bend required for an intermediate CST is already introduced by the CST with which it is sharing an edge. Hence, intermediate simple CSTs do not contribute to the bends.



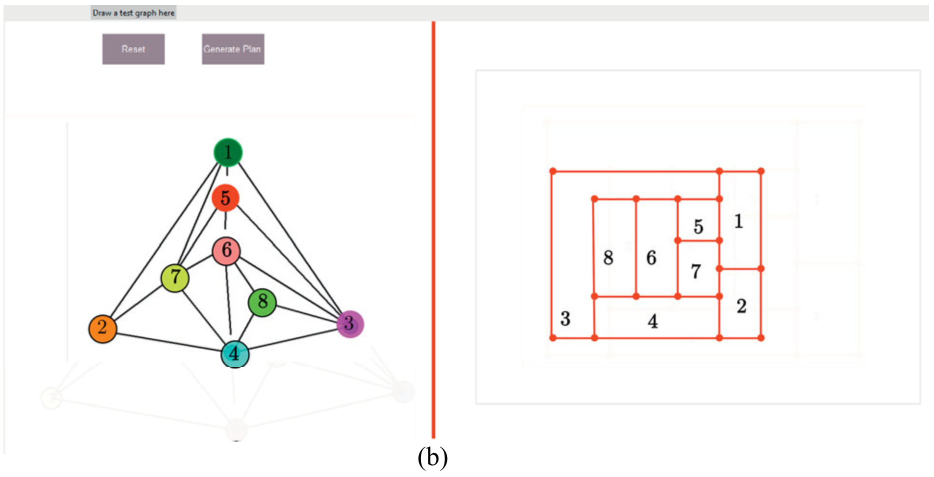


Figure 7. A PTG and its corresponding OFP with one bend (obtained using G-Drawer) due to (a) CIPs (# CIPs = 5) and (b) STs (# STs = 1).

(ii) If an intermediate CST is a nested CST and the nested CST is disjoint from other CSTs then a bend is required to make the triangular face rectangular. Hence, all intermediate nested CSTs introduce |nested CST| bends in an OFP.

2. K_4s: Every K_4 is a triangular face in its dual graph and to make it rectangular, $|K_4|$ bends are required in an OFP.
3. CIPs: Every CIP is a path which lies on the outer boundary and has a shortcut which is an edge. A shortcut disconnects the graph into subgraphs where each subgraph corresponds to a rectangular plan/block. In an OFP, the rectangular block corresponding to a CIP is covered from two sides by the room corresponding to either starting vertex or ending vertex of the CIP, which introduces a bend (refer to OFP in Figure 1c where the rectangular block with rooms M_3, M_{11}, and M_{12} is covered from two sides by room M_5 which introduces a bend). If $|\text{CIP}| < 4$ then the rectangular block obtained by these subgraphs lies on any of the four corners of a rectangular boundary which requires no bend.

It can be seen that the contributors for bends are: root of the containment tree (if it is CST°), leaves CSTs of a containment tree, intermediate CSTs which are nested CSTs, K_4s and CIPs. Therefore the number of bends required to construct an OFP is at most $\rho + 1$. Hence, $B_{\min}$ is at most $\rho + 1$, which can be reduced further in the following cases:

Case 1. If $|\text{CST}| \geq 2$ and any of the following holds:

(a) each CST is a leaf of a containment tree except the root of containment tree which is a CST (that is CST°) then $B_{\min} = \rho$ (refer to Case 1a of Table 1).
(b) each CST is a leaf of a containment tree then $B_{\min} = \rho$ (refer to Case 1b of Table 1).
(c) each CST is either an intermediate CST or leaf of a containment tree then $B_{\min} = \rho$ (refer to Case 1c of Table 1).

Explanation |:CST| ≥ 2 implies that either root of the containment tree is a CST (which is denoted by CST°) or not. If it is a CST, that is, CST° then it contains one or more CSTs and they further do not contain any other CST. If CST° is simple CST, that is, it always has a shared edge with the contained CSTs. Hence, a bend corresponding to any CST is also a bend for CST°, that is, a bend corresponding to CST° is not required and |nested CST| = 0. Hence, $B_{\min} = |L_T| = \rho$. Else if CST° is a nested CST then the bend required for CST° and nested CST is the same. Here, nested CST is independent and disjoint. Hence, the bend reduction in case of sharing of a vertex or an edge of CST° with other CSTs is not applicable, and therefore, $B_{\min} = \rho$.

Else if the root of a containment tree is not a CST and there are no intermediate CSTs, that is, |nested CST| = 0. $|K_4| = 0$ and a bend corresponding to the root is not required. Therefore, $B_{\min} = |L_T| = \rho$.

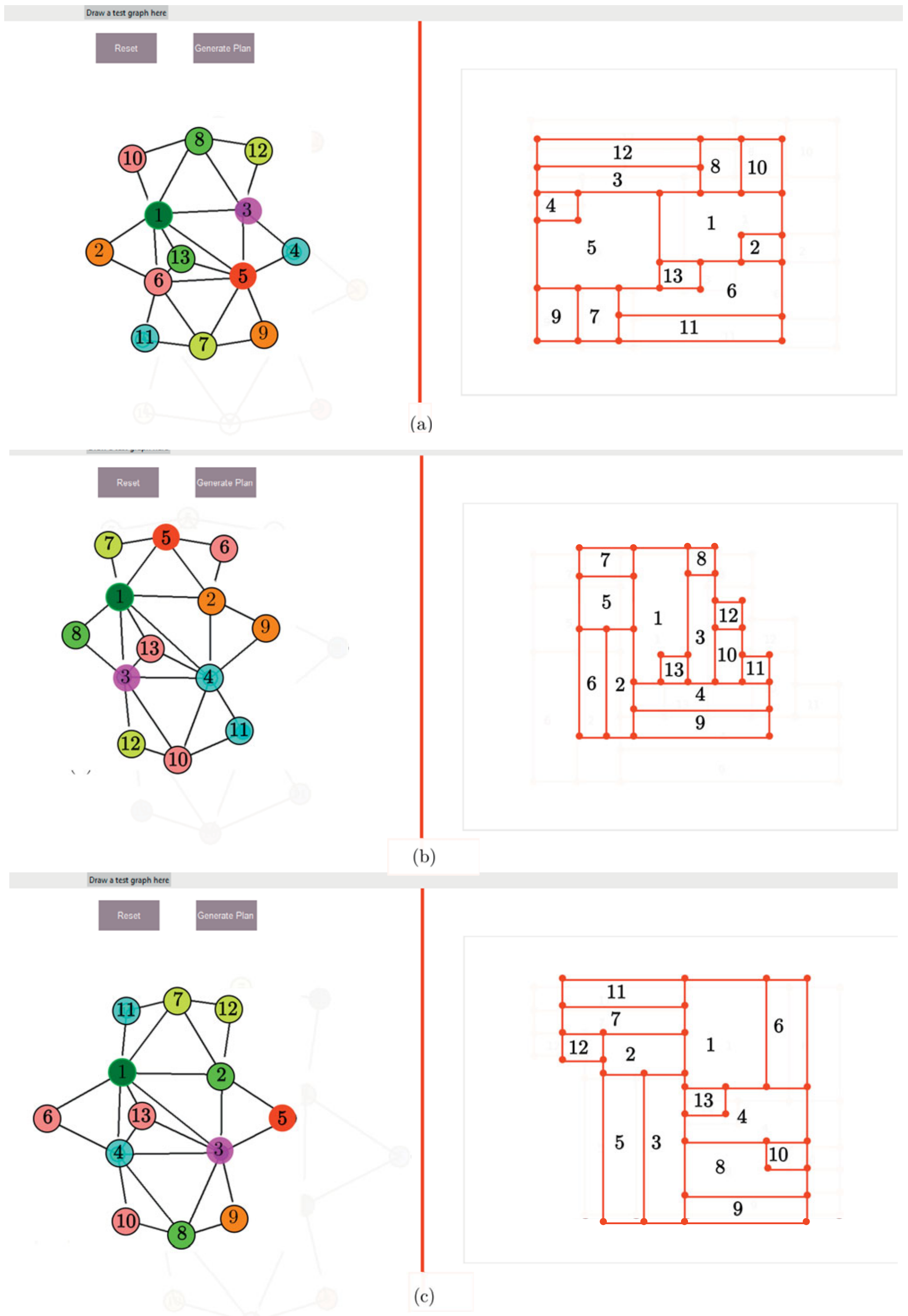


Figure 8. A PTG with different solutions (a) an OFP, (b) an IFP (staircase FP), and (c) an IFP (L-shaped FP).

Else if the root of a containment tree is not a CST, it does not contribute to a bend. There are intermediate CSTs which can be either simple CST or nested CST and intermediate simple CSTs do not contribute to bends. $|K_4| = 0$ and a bend corresponding to the root is not required. Therefore, $B_{\min} = |L_T| + |\text{nested CST}| = \rho$.

Case 2. If $|\text{CST}| \geq 2$ and there exists a $K_4 \subseteq \text{CST}$ which do not share an edge with any of the CSTs, then $B_{\min} = \rho$ (refer to Case 2 of Table 1).

Explanation: In this case, only one bend gets reduced because of CST° as discussed in Case 1. Hence, $B_{\min} = |L_T| + |K_4| + |\text{nested CST}| = \rho$.

Case 3. If $|\text{CST}| \geq 2$ and there exists at least one CST containing K_4 and sharing an edge with K_4 then $B_{\min} = \rho - 1$ (refer to Case 3 of Table 1).

Explanation: If a CST and a K_4 are sharing an edge, then only one bend is needed for both of them (bend for either a CST or a

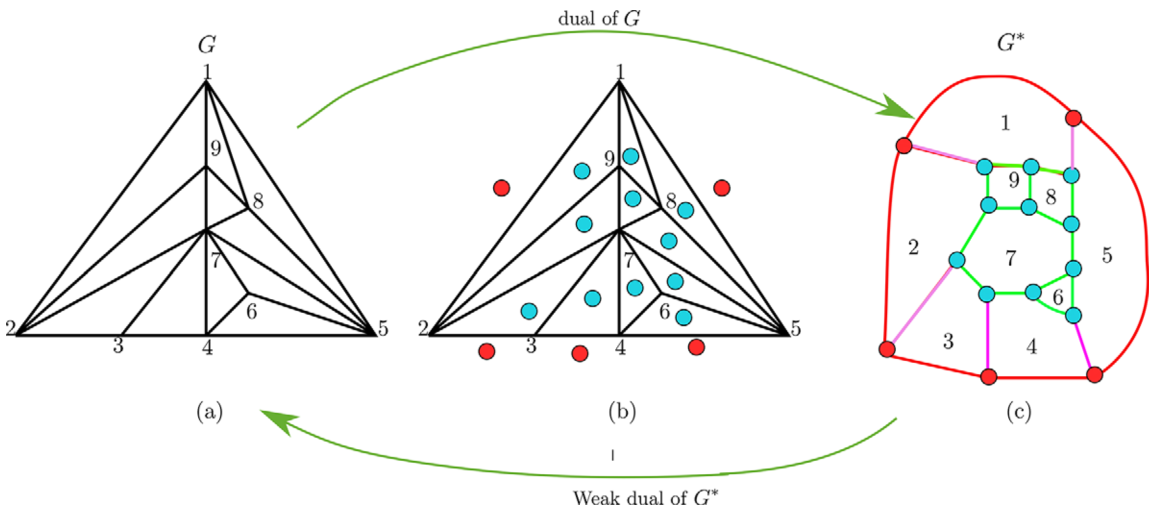


Figure 9. A PTG G and its relation with dual graph G^* where the blue vertices correspond to the interior faces, while the red vertices correspond to each edge of the exterior face. The green edges represent adjacencies between the interior faces, the pink edges represent adjacencies between the interior faces and the exterior face, and the red edges signify adjacencies within the exterior face.

K_4 satisfies the adjacency requirements for one another as they have a shared edge). The bend for CST° is also not required, as explained in Case 1. Hence, $B_{\min} = |L_T| + |K_4| + |\text{nested CST}| - 1 = \rho - 1$.

Case 4. If $|\text{CST}| \geq 2$ and there exists at least two CSTs containing K_4 and sharing an edge with K_4. If the shared edges have a vertex in common then $B_{\min} = \rho - 2$ (refer to Case 4 of Table 1).

Explanation: Since a CST shares an edge with a K_4, from Case 2, $B_{\min} = \rho - 1$. Also, if more than one CSTs are sharing an edge with K_4 and shared edges have a common vertex, then the bend at the common vertex compensate for the bends required for these CSTs and K_4s. Hence, $B_{\min} = |L_T| + |K_4| + |\text{nested CST}| - 2 = \rho - 2$.

Case 5. If $|\text{CST}| = 1$ and $|K_4| \geq 1$ and there exists a $K_4 \subseteq \text{CST}$ sharing an edge or a vertex with the CST then $B_{\min} = \rho$ (refer to Case 5 of Table 1).

Explanation: Since a CST and a K_4 have a common edge or a common vertex, only one bend is required for both of them. Hence, $B_{\min} = \rho$, that is, $|K_4|$.|

Three major steps are required to construct an FP for a PTG which are as follows:

Step 1: Dual generation This step creates the dual of a given PTG G. The planar embedding of G is taken as an input for this step. A graph G^* is a dual of G if the weak dual of G^* is G (see Figure 9). In the dual of G, every vertex of G corresponds to an interior face of G^* and every edge of G corresponds to common line segment of the adjacent faces of G^*. Dual of G is always triconnected cubic graph (G is biconnected and triangulated).

For a PTG G, F_{int} denotes the set of all the interior faces of G and F_{ext} denotes the exterior face of G.

Algorithm 1 : Dual Generation.

```
Input: A biconnected PTG $G$.
Output: Dual of $G$ ($G^*$)
1 Place a vertex inside every $f_i \in F_{int}$ of $G$, call such vertices $r_1, r_2, \dots r_k$ (refer to blue vertices in Figure 9b).
2 Place a vertex next to the midpoint of every edge of $F_{ext}$, call such vertices $b_1, b_2, \dots b_l$ (refer to red vertices in Figure 9b).
3 If $f_i$ and $f_j \in f_{int}$ are adjacent then join the corresponding vertices with an edge such that it intersects a common edge between them (refer to green edges in Figure 9c).
4 If $f_i \in F_{int}$ shares an edge with $F_{ext}$ then join the corresponding $r_i$'s and $b_j$'s with an edge such that it only intersects the common edge between them (refer to pink edges in Figure 9c).
5 Join $b_j$'s if their corresponding edges have a vertex in common (refer to red edges in Figure 9c).
```

For the implementation of dual generation in G-Drawer, refer to Figure 10.

Step 2: Orthogonalization This step is based on Tamassia's approach (refer to Algorithm 2 of (Di Battista and Eades, 1999)) which begins with the construction of the flow network of G^*. Then minimum cost flow is computed for the constructed flow network using Successive Shortest Path (SSP) algorithm (Ahuja et al., 1989). The obtained minimum cost flow is used to determine the angle and bend-data to generate an OFP. We have modified and then implemented this approach to obtain an OFP.

Algorithm 2 : Steps of Tamassia's approach.

```
Input: A Triconnected Cubic Graph $G^*$.
Output: An Orthogonal Floor Plan
1 Construct flow network $N$.
2 Compute minimum cost flow of $N$ and call SSP minimizer.
```

3 Construct an orthogonal representation using data from minimum cost flow.
4 Exit

Step 2.1: Construction of flow network To obtain a flow network from a dual graph G^*, refer to the following steps and Figure 11:

1. Addition of vertices:
   (i) Create a square vertex w_i in N for every face of G^*.
   (ii) Create a round vertex v_j in N for every vertex of G^*.
2. Addition of arcs:
   (i) Connect a vertex v_j of N to a vertex w_i of N by a red arc, if face w_i is adjacent with vertex v_j in G^*.
   (ii) Connect each vertex w_i to another vertex w_j of N by a blue arc, if the corresponding faces of G^* are adjacent.

A flow network is a directed graph where each edge has a capacity, and each edge receives a flow. The edges can be mono-directional or bidirectional. There is a cost associated with each edge, which can be 0 and 1. The round vertices acts as source and squared vertices acts as sink. The cost of a flow network is the net cost of an edge which is equal to the edge cost multiplied by the amount of flow at that edge. This net valued summed over all the edges gives the total cost of the flow network. The flow value of an edge represents the angle between the round vertex (original vertex) and the square vertex (face of G^*) that each vertex shares with a face. Capacity is defined as the maximum value that can flow in network which lies between $\pi/2$ and 2π.

Relation between min cost flow and minimum bend To understand how the solution of flow problem is linked to orthogonal representation, some essentials are required which are as follows:

Vertex Angle: A vertex-angle is defined as the counterclockwise angle between two consecutive edges adjacent to a vertex in the orthogonal representation. The angle which is formed by a bend in the orthogonal representation is called bend-angle. In Figure 12b, angle between edges (8, 2) and (8, 9) is $\pi/2$ which is the vertex angle whereas b_0 is the bend angle. For a better understanding, the counterclockwise angle directions are defined as:

- A 90° angle is called left.
- A 180° angle is called straight.
- A 270° angle is called right.
- A 360° angle is called full.

In a flow network N, every round vertex acts as a source while every face vertex acts as a sink. There are two types of arcs in N. First type arc is from the round vertices to square vertices (refer to Figure 11b). For a round vertex, the outgoing flow is always 4. This is because each unit flow denotes an angle of $\pi/2$ from the vertex to the corresponding face in the orthogonal representation (refer to Figure 12a, where vertex 8 to vertex a has flow value 1 which corresponds to the $\pi/2$ angle between vertex 8 and the adjacent face A). The second type of arcs is bidirectional between faces. The flow in such arcs represents that there exists a bend vertex between the faces. Initially, the cost of flow in the bidirectional arcs is 1 while the cost of flow in first type of arc is 0 and the problem is to minimize the cost.

In minimization algorithm over the flow network, the flow in the bidirectional arcs gets minimized, while the flow in the mono directional arcs is unaffected. This flow minimization indirectly minimizes the number of bends in the FP by minimizing the total cost of the flow network, since there is a one to one correspondence between the flow in the bidirectional arcs and the number of bends in the OFP.

Step 2.2: Finding min cost flow The next step is to find the minimum cost flow of N which can be done using SSP algorithm

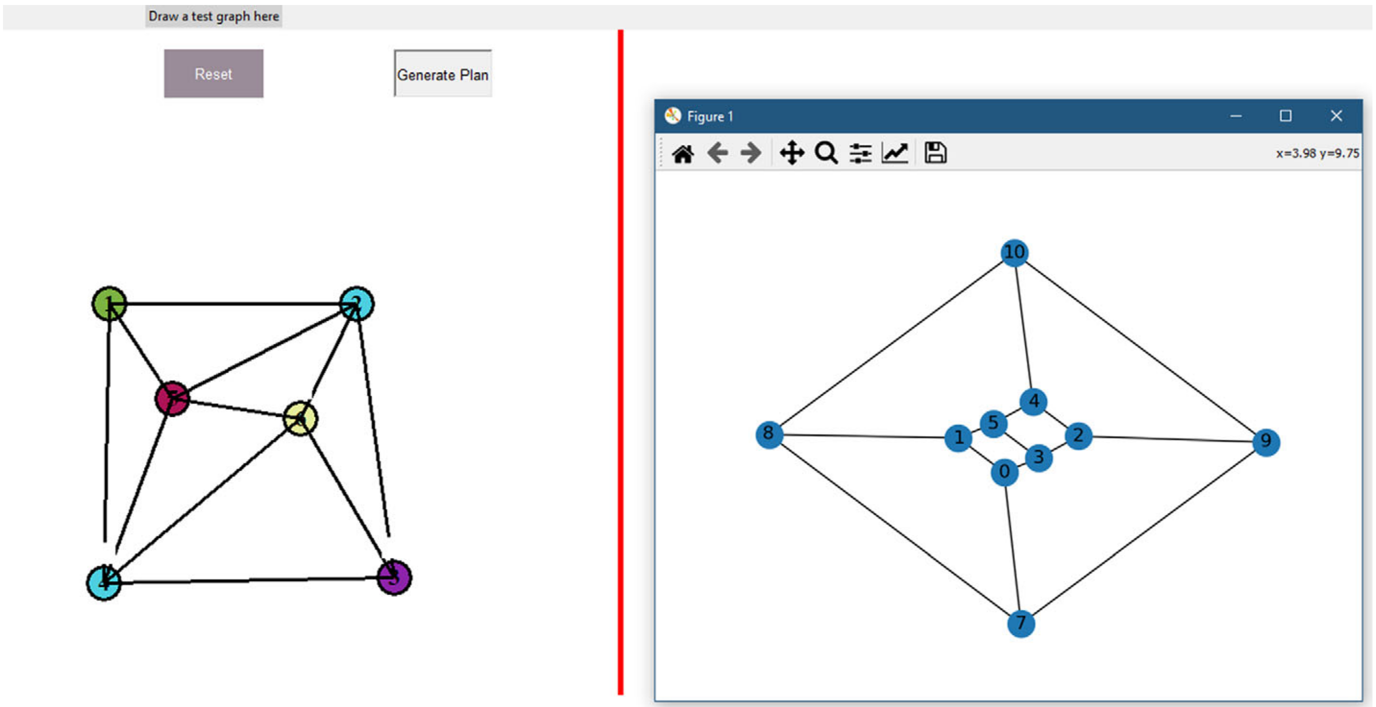


Figure10. A dual graph G^* for PTPG G.

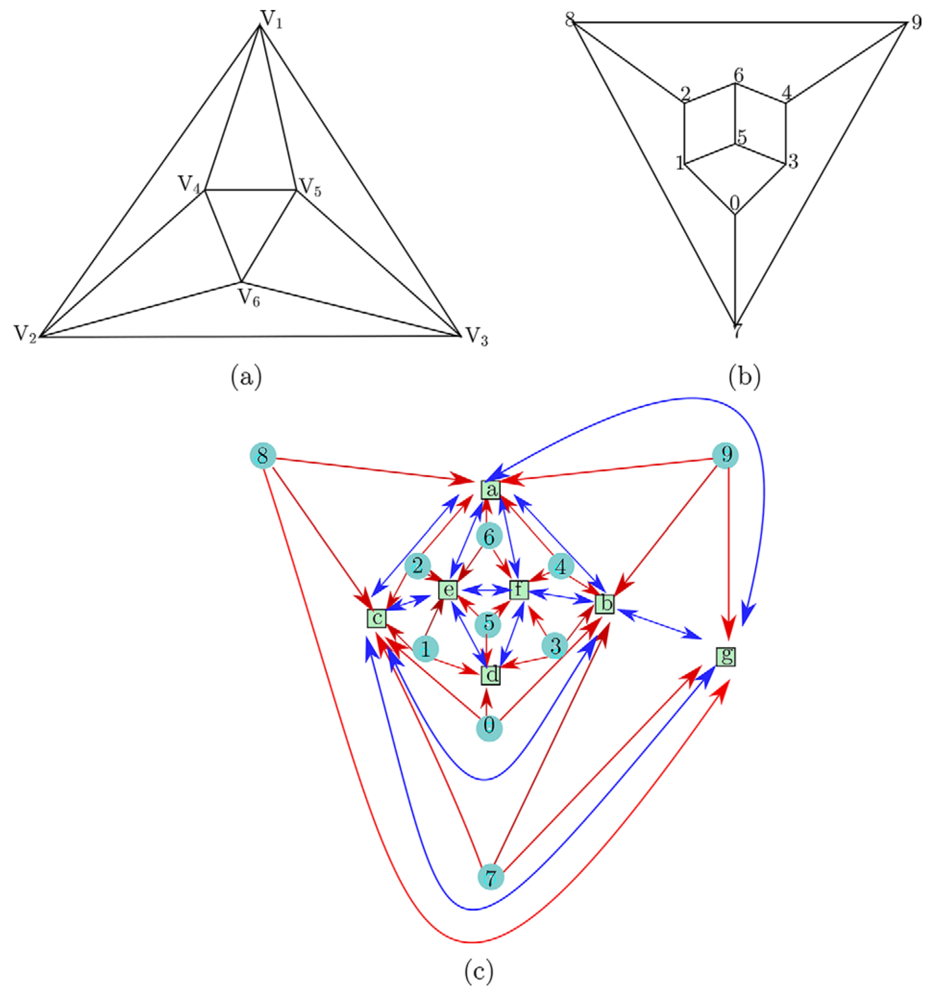


Figure 11. (a) A PTG G, (b) a dual graph G^* of G, and (c) flow network N of G^* where the round blue vertices represent each face of G, the square green vertices represent each vertex of G, the red arcs denote adjacencies between faces and vertices of G, and the blue arcs indicate adjacencies between faces of G.

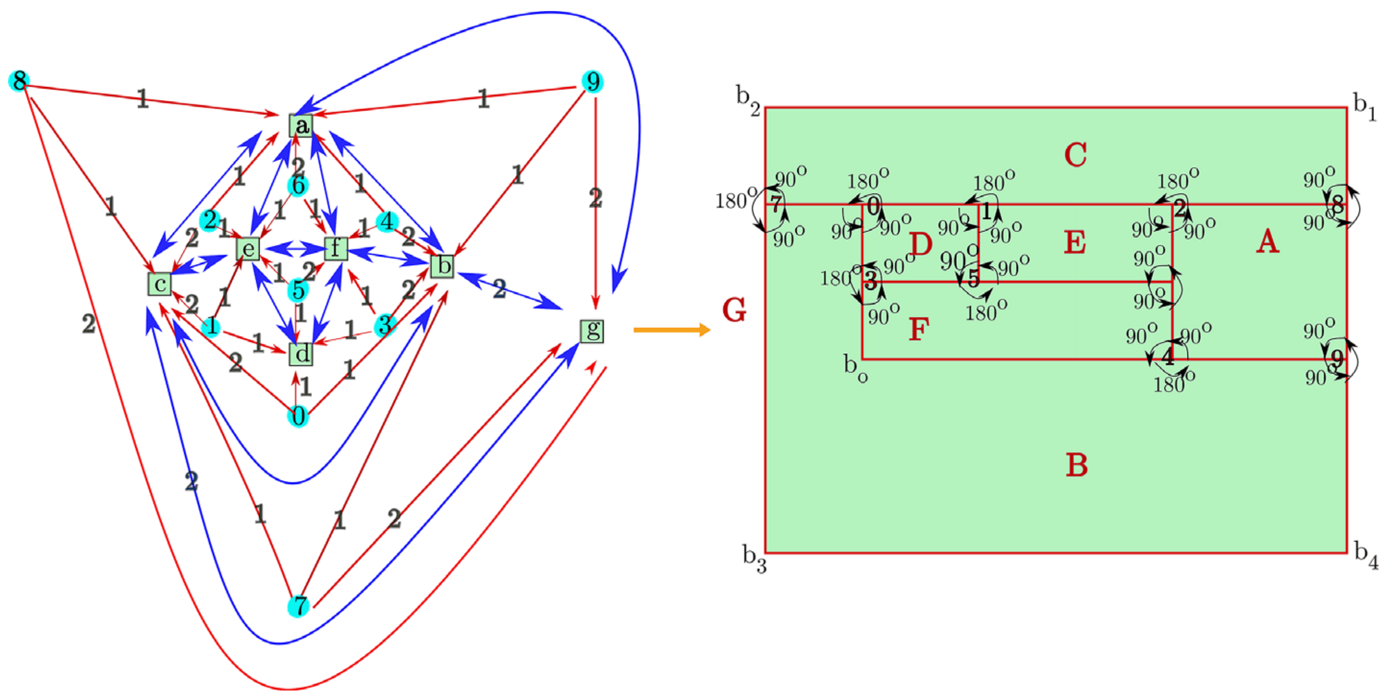


Figure 12. Mapping of flow value to angle between the adjacent faces.

which is a generalization of Ford Fulkerson algorithm. This solver uses the residual network to find the shortest path and outputs the flow value as well as cost value which are vertex and bend angles (refer to Figure 12a).

Step 2.3: Mapping vertex angles and bend angles to obtain an orthogonal representation To obtain an orthogonal representation from minimum cost flow solver, we need to map the flow of a vertex arc to its corresponding angle in the orthogonal representation. The flow value is directly mapped to the angle between the adjacent faces (see Figure 12), where

- 1 forms a left angle
- 2 forms a straight angle
- 3 forms a right angle
- 4 forms a full angle

In Figure 12, the flow value of 2 for arc (0,C), 1 for (0,D), and 1 for (0,B), that is, vertex 0 is adjacent to face *C* with 180°, 90° with D and 90° with B.

The cost of flow in a face arc represents the number of bends between the corresponding faces. Hence, to add bends in the orthogonal representation, we need to iterate over all face arcs depending on the left and right face of an edge. The bends are added with either a left angle or a right angle. Mapping of flow and cost to vertex angles and bend angles results in an orthogonal representation which is given in Figure 12b.

Step 3: Compaction The task here is to assign minimum lengths to the segments of the edges of the obtained orthogonal representation with the condition that there are no edge crossings and no vertex overlaps. Tamassia (Tamassia et al., 1991) investigated a technique that uses a flow network to minimize the edge lengths.

Step 3.1: Flow networks for vertical and horizontal edge minimization The flow network for the horizontal segments is denoted N_{hor}, and the network for the vertical segments is denoted N_{ver}, respectively (see Figure 13b,c). Network N_{hor} consists of vertices that represent the internal faces of G^* and it has two new vertices representing the lower and upper regions on the external face for source and sink which are denoted by s and t (see Figure 13a). It has an arc for every adjacent pair of faces f and g, that is, if they share a common horizontal segment e. Similarly, Network N_{ver} consists of vertices which represent the internal faces of G^* and it has two new vertices representing the left and right region on the external face for source and sink which are denoted by s and t (see Figure 13c). It has an arc for every adjacent pair of faces f and g, that is, if they share a common vertical segment e. The arcs of the flow network N_{hor} and N_{ver} in the context of the compaction have the following properties:

(i) A lower bound for the flow is 1.
(ii) A upper bound or a capacity is ∞.
(iii) Cost is 1.

While solving networks N_{hor} and N_{ver}, we assign initial flow value as 1 on every arc. Every vertex except source and sink is a transport vertex with the property that the incoming flow at a vertex is equal to the outgoing flow at the vertex. First, each arc gets a flow of 1 then Breadth.

First Search from source to sink has been processed and each found vertex is added to a list of vertices such that the list is ordered by the BFS. Then, for each transport, vertex v of the list it is checked whether the incoming flow and outgoing flow at v is equal or not. There can be three cases:

Case 1: The incoming and outgoing flow at v is equal, that is, the flow of that arc is feasible.

Case 2: The outgoing flow at v is more than the incoming flow at v. In such a case, the incoming flow to v must be increased. A vertex can be incident to various incoming arcs and outgoing arcs, and we need to identify which arc flow should be increased to keep the flow minimal. This is done by finding the shortest path from the source s to v, and the deficit value is added to all the arcs of the shortest path. For example, consider the vertex a in Figure 13b. Initially, all the arcs were assigned a flow of 1. This means that the outgoing flow from a is 3 while the incoming flow is 1, which leads to a deficit of 2. To overcome this deficit, we increase the flow in arc $s-a$ from 1 to 3, thus mass balancing the flow. Thus, whenever the outflow is more than inflow, we find the shortest path from the source to the given vertex and increase the flow in the path. Figure 13a,c shows the modified network after mass balancing has been done for all the vertices.

Case 3: The outgoing flow from v is less than the incoming flow at v then the shortest path from v to the sink t is calculated and the flow of each arc is adjusted.

Once the mass balancing is done at all the vertices, the solutions of networks N_{hor} and N_{ver} outputs the minimum edge lengths in an orthogonal representation (see Figure 13a,c). Combining data obtained from N_{hor} and N_{ver} to the orthogonal representation, an OFP is obtained (refer to Figure 14a,b). Since in the OFP, the bends corresponding to the minimum cost, therefore OFP has minimum bends.

Theorem 3. G-Drawer requires polynomial time to generate an FP with minimum bends for a given PTG.

Proof. Construction of FPs requires three major operations:

(a) Dual generation: In this step, we iterates over all the inner triangular faces and over all the external edges and place the vertices corresponding to them. Vertices are joined using a face to vertex map. Hence, overall complexity for this step is $O(T + P)$, where T is the number of inner faces and P is the number of exterior edges. It can be seen that $O(T + P)$ is less than $O(n)$, where n is the number of vertices.
(b) Orthogonalization: This step includes three operations:
    (i) Generating flow network: This step is similar to dual generation and hence, the implementation of this step requires $O(R)$, where R is the total number of faces of G^*.
    (ii) Solving flow network: This step uses the SSP algorithm, which is generalization of Ford Fulkerson algorithm and requires $O(n \times m)$ time, where n is the number of arcs and m is the number of nodes. The number of iterations is at most $n \times U$, where U is the largest supply, and Dijkstra's shortest path finder needs $O(n^2)$. Summing up $O(n^3 \times U)$ complexity is needed for the SSP algorithm.
    (iii) Computing orthogonal representation: This is done by using the solution of the flow network to map the angles. The number of angles $\leq 4n$ where n is the number of

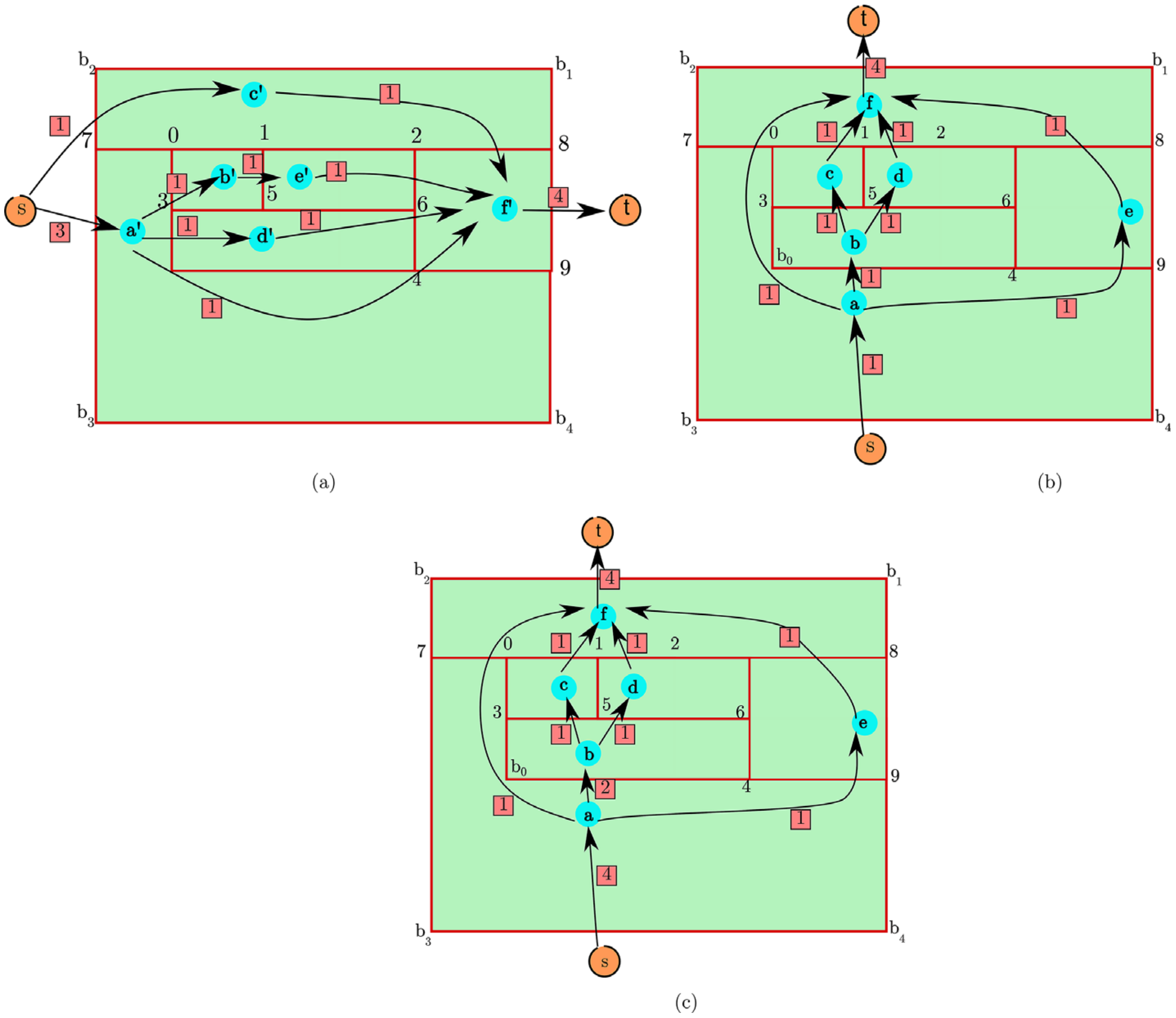


Figure 13. Flow networks. (a) Min cost flow solution for N_{hor}. (b) Non feasible solution for N_{ver}. (c) Min cost flow solution for N_{ver}.

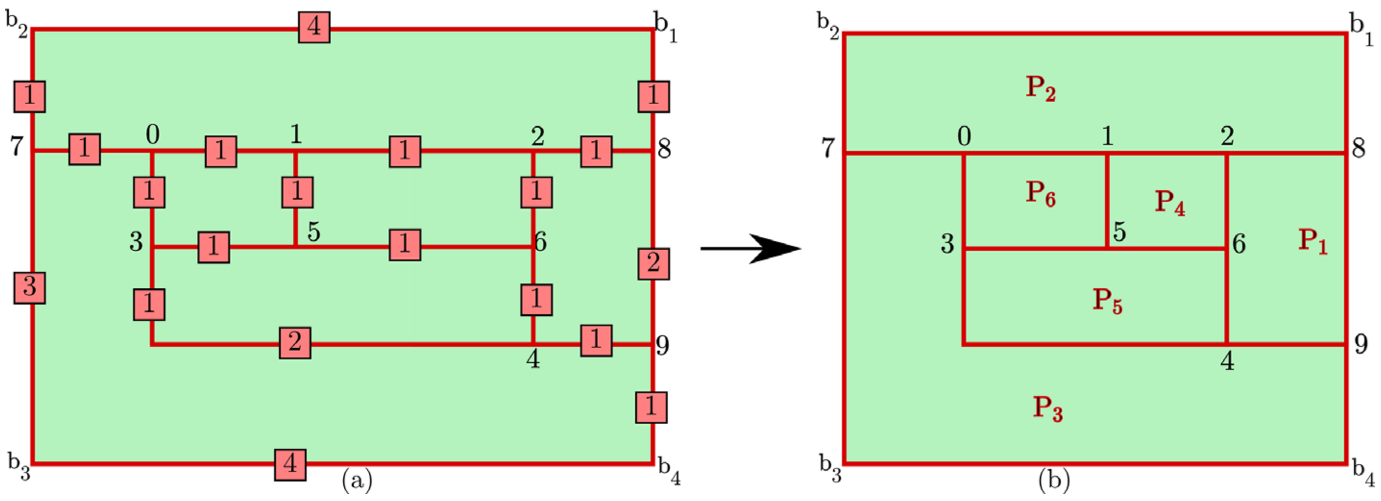


Figure 14. Solution obtained from compaction, that is, an OFP corresponding to a PTG given in Figure 11a.

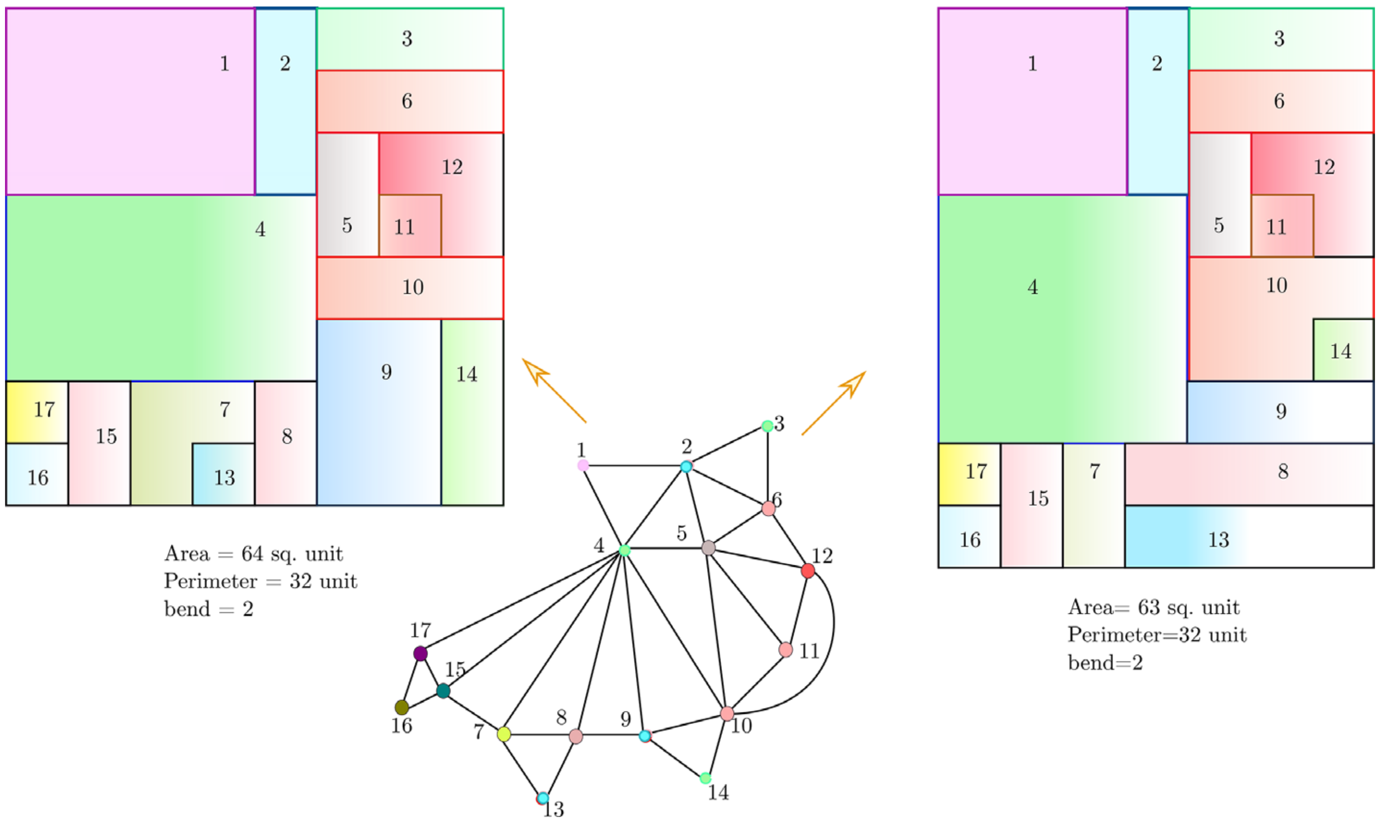


Figure 15. Two floor plans corresponding to the same input graph having the same number of bends and different grid size (area and perimeter).

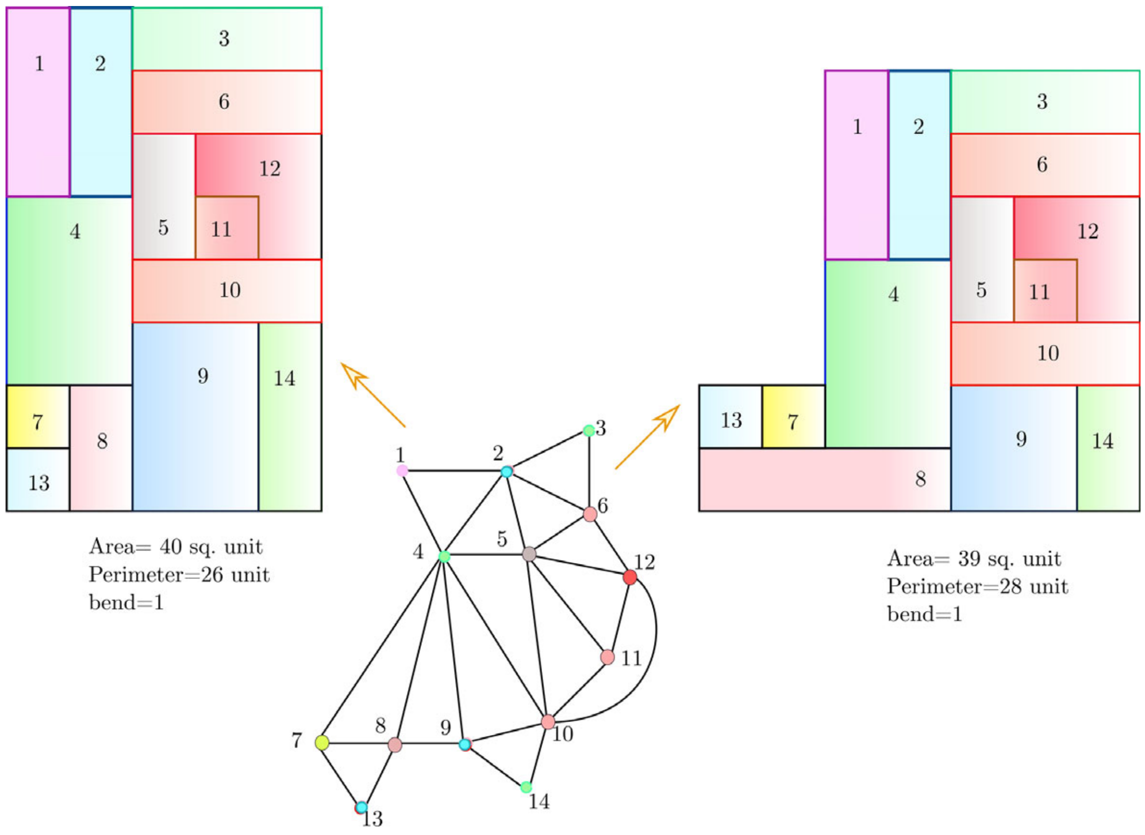


Figure 16. Two floor plans corresponding to the same input graph having the same number of bends and different grid size.

vertices, since every vertex can have a maximum angle of 4×90. Overall complexity for this step is $O(n)$.

(c) Compaction: This step includes three operations:

(i) Generating the flow network: This can be done in $O(E)$ time, since we iterate over all edges and over every face that an edge shares and an edge can be a part of a maximum of 2 faces. Joining the two faces by creating a vertex inside them and adding an edge takes $O(1)$ time. Overall complexity is $O(E)$.

(ii) Solving network flow: For solving the flow network, we use a simple flow solving algorithm where complexity is stated as follows. Adding initial flow to arcs and performing the BFS which is done only once and has negligible complexity. The used implementation of Dijkstra's algorithm needs $O(n^2)$ in rare cases. This is done for every node with gap not equal to 0. Assuming the rare case that all non-source and non-sink nodes have such a gap, Dijkstra's algorithm is processed for $n - 2$ nodes. Increasing the nodes arcs is only linear and is negligible, too. Summing up a complexity of $O((n-2) \times (n^2))$. We do this two times for the horizontal and vertical flow networks.

The time complexity of solving the flow network to find the angles around each vertex takes the maximum amount of time. So, the overall complexity of the FP generating algorithm is $O(n^3 \times U)$, where n is the number of vertices and U is the largest flow. For the worst case, the complexity is $O(n^3\ (n-1))$ where, the largest flow U is $n - 1$.

Conclusion

In this article, we presented a software G-Drawer which automates the generation of FPs based on the specific class of PTGs (refer to 'Supplementary Material'). For a PTG G, there always exist an FP and depending on a PTG, there exists many algorithms for generating an RFP or an OFP, whenever an RFP does not exist. G-Drawer takes a PTG as an input and generates a corresponding FP with a

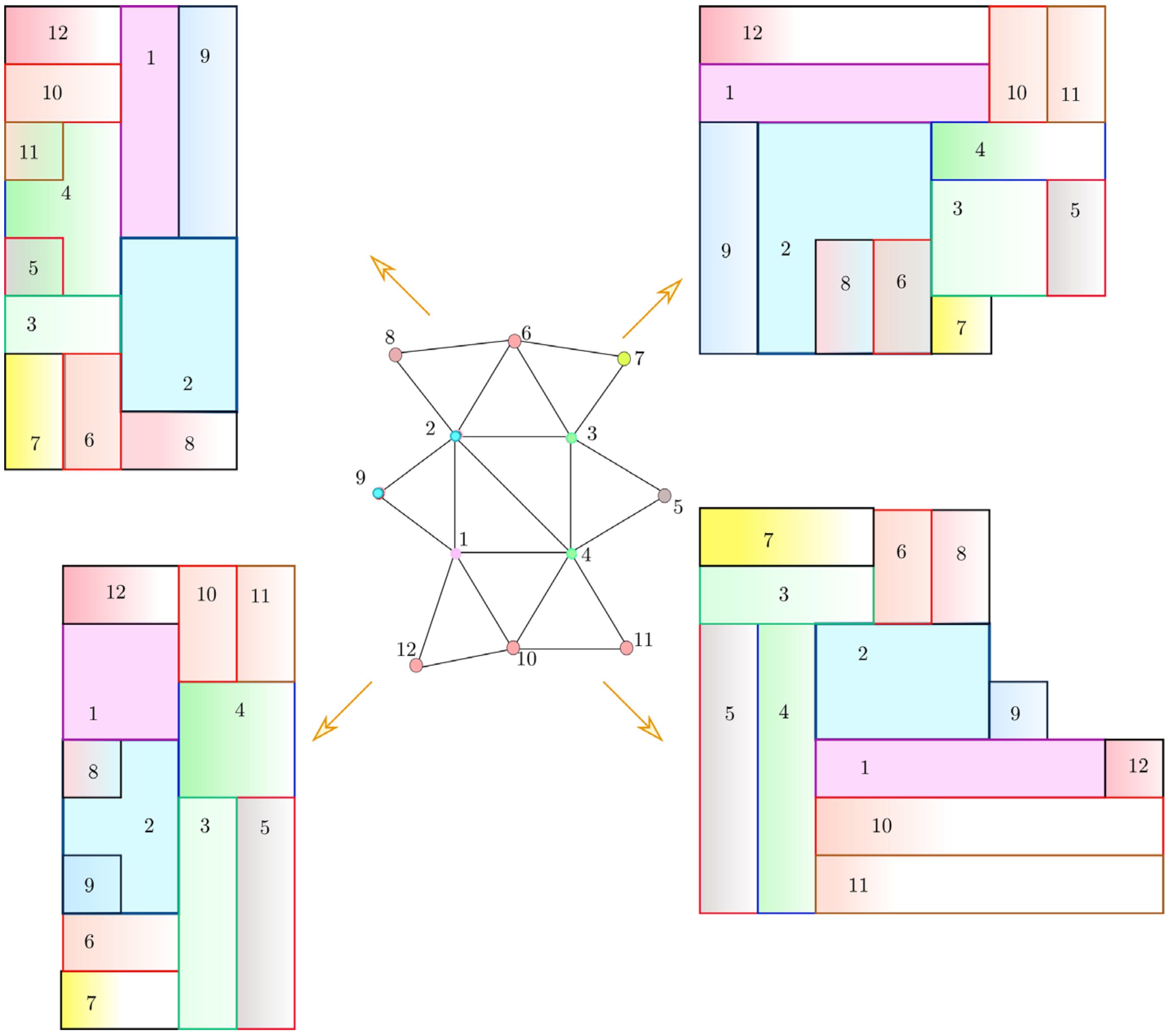


Figure 17. Different solutions for a PTG (with six CIPs) with specific location of bends.

minimum number of bends, that is, it first prefers to generate an RFP (FP with 0 bends if exist), otherwise an OFP (FPs having rectangular boundary, that is, bends are present at the rooms but not at the boundary) or an IFP (FPs with non-rectangular boundary, where if possible, the bends at the rooms are shifted at the boundary). For details, see Figure 8, where for the same input graph, an OFP and two different IFPs have been constructed using G-Drawer.

Limitations and future work

The focus of this study is on the automated generation of FPs with minimum bends. From here on, there are many mathematical as well as architectural problems which we are planning to address in the near future, a few of them are as follows:

1. G-Drawer generates dimensionless FPs with minimum bends, but from application perspective, it is required to generate dimensioned layouts. Also, G-Drawer covers FPs having non-rectangular rooms. There exists a few approaches to generate dimensioned FPs having rectangular rooms (Bisht et al., 2022; Upasani et al., 2020) but it is more difficult to generate dimensioned FPs with non-rectangular rooms and non-rectangular boundary. In future, we are working on ways to incorporate dimensions of rooms as well as boundary to have dimensioned FPs.
2. G-Drawer generates FPs with minimum bends without considering the grid size, that is, the generated FPs may not correspond to minimum grid size (see Figure 15 where for the same input graph, G-Drawer generates two FPs, each having a different grid size). In future, we would like to develop an algorithm for generating FPs with minimum grid size. Furthermore, it is interesting to note that the minimum bend FP may not corresponds to a minimum grid size FP (see Figure 16). Hence, mathematically, it is more challenging to obtain minimum bend FP with minimum grid size.
3. Sometimes, user may require multiple FPs corresponding to the same adjacencies, which is difficult to achieve manually. At this stage, G-Drawer generates only one FP corresponding to the given requirements. In future, my making a few changes in G-Drawer, multiple layouts can be generated. For illustrations, refer to Figure 17).
4. To ensure that G-Drawer can be reached to wide range of audience, in particular architects and computational designers, in future, we aim to develop different plugins in Grasshopper, Revit, and so forth.

Supplementary material. 1. A video illustrating the implementation of G-Drawer which is done in python is available at https://www.dropbox.com/s/hsdbp1bx30agn9x/Demo.mp4?dl=0.

2. The Implementation of the code for G-Drawer is available at https://github.com/GPlanTeam/GDrawer-Code.

References

Ahuja RK, Magnanti TL and Orlin JB (1989) Chapter iv network flows. *Handbooks in operations Research and Management Science* **1**, 211–369.

Alam MJ, Biedl T, Felsner S, Kaufmann M, Kobourov SG and Ueckerdt T (2013) Computing cartograms with optimal complexity. *Discrete & Computational Geometry* **50**(3), 784–810.

Azizi Vd, Usman M, Zhou H, Faloutsos P and Kapadia M (2022) Graph-based generative representation learning of semantically and behaviorally augmented floorplans. *The Visual Computer* **38**(8), 2785–2800.

Baybars I and Eastman CM (1980) Enumerating architectural arrangements by generating their underlying graphs. *Environment and Planning B: Planning and Design* **7**(3), 289–310.

Bhasker J and Sahni S (1987) A linear time algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks* **17**(3), 307–317.

Bhasker J and Sahni S (1988) A linear algorithm to find a rectangular dual of a planar triangulated graph. *Algorithmica* **3**(2), 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**.

Boyer JM and Myrvold WJ (1999) Stop minding your p's and q's: A simplified o (n) planar embedding algorithm. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, ACM, pp. 140–146.

Brandenburg F, Eppstein D, Goodrich MT, Kobourov S, Liotta G and Mutzel P (2003) Selected open problems in graph drawing. In *International Symposium on Graph Drawing*. Springer, pp. 515–539..

Chang Y-J and Yen H-C (2014) Rectilinear duals using monotone staircase polygons. In Zhang Z, Wu L, Xu W, Du DZ (eds), *Combinatorial Optimization and Applications. COCOA 2014. Lecture Notes in Computer Science*, volume **8881**, Cham: Springer, pp. 86–100. doi: 10.1007/978-3-319-12691-3_8.

Cousin J (1970) Topological organization of architectural spaces. *Architectural Design* **40**, 491–493.

Di Battista G, Eades P (1999) Roberto Tamassia, and Ioannis G Tollis. In *Graph Drawing*, volume **357**. Upper Saddle River, NJ: Prentice Hall.

Duarte JP (2001) Customizing mass housing: A discursive grammar for siza's malagueira houses.

Garg A and Tamassia R (2001) On the computational complexity of upward and rectilinear planarity testing. *SIAM Journal on Computing* **31**(2), 601–625.

Grason J (1970) A dual linear graph representation for space-filling location problems of the floor plan type. *Emerging Methods in Environmental Design and Planning, Proceedings of The Design Methods Group, 1st International Conference*, Cambridge, pp. 170–178.

He F, Huang Y and Wang H (2022) iplan: Interactive and procedural layout planning. *arXiv preprint arXiv:2203.14412*.

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

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.

Jokar MRA and Sangchooli AS (2011) Constructing a block layout by face area. *The International Journal of Advanced Manufacturing Technology* **54**(5–8), 801–809.

Klawitter J, Klesen F and Wolff A (2021) Algorithms for floor planning with proximity requirements. *arXiv preprint arXiv:2107.05036*.

Klose P (2012) *A Generic Framework for the Topology-Shapemetrics Based Layout*. Christian-Albrechts-Universität zu Kiel.

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

Kurowski M (2003) Simple and efficient floor-planning. *Information Processing Letters* **86**(3), 113–119.

Laignel G, Pozin N, Geffrier X, Delevaux L, Brun F and Dolla B (2021) Floor plan generation through a mixed constraint programming-genetic optimization approach. *Automation in Construction* **123**, 103491.

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

Lu Z, Wang T, Guo J, Meng W, Xiao J, Zhang W and Zhang X (2021) Data-driven floor plan understanding in rural residential buildings via deep recognition. *Information Sciences* **567**, 58–74.

Miura K, Haga H and Nishizeki T (2006) Inner rectangular drawing of plane graphs. *International Journal of Computational Geometry & Applications* **16**, 249–270.

Müller P, Zeng G, Wonka P and Van Gool L (2007) Image-based procedural modeling of facades. *ACM Transactions on Graphics* **26**(3), 85.

Nauata N, Hosseini S, Chang K-H, Chu H, Cheng C-i, and Furukawa Y (2021) House-gan++: Generative adversarial layout refinement networks. *arXiv preprint arXiv:2103.02574*.

Nourian P, Rezvani S and Sariyildiz S (2013) A syntactic architectural design methodology: Integrating real-time space syntax analysis in a configurative architectural design process. In *9th International Space Syntax Symposium*, *SSS 2013*. Sejong University,. https://sites.google.com/site/pirouznourian/syntactic-design.

Nummenmaa J (1992) Constructing compact rectilinear planar layouts using canonical representation of planar graphs. *Theoretical Computer Science* **99**, 213–230.

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*, IEEE, pp. 6690–6700.

Pinki P and Shekhawat K (2022a) Linear-time construction of floor plans for plane triangulations. *Communications in Combinatorics and Optimization*, **8**(4), 673–692.

Pinki P and Shekhawat K (2022b) Characterization of graphs based on number of bends in corresponding floor plans. In *Proceedings of the 6th International Conference on Algorithms, Computing and Systems*, ACM, pp. 1–8.

Rinsma I (1988) Rectangular and orthogonal floor plans with required room areas and tree adjacency. *Environment and Planning B: Planning and Design* **15**(1), 111–118.

Shekhawat K, **Lohani R**, **Dasannacharya C**, **Bisht S and Rastogi S** (2023) Automated generation of floorplans with non-rectangular rooms. *Graphical Models* **127**, 101175.

Shekhawat K, **Upasani N**, **Bisht S and Jain RN** (2021) A tool for computer-generated dimensioned floorplans based on given adjacencies. *Automation in Construction* **127**, 103718.

Steadman P (2006) Why are most buildings rectangular? *Arq: Architectural Research Quarterly* **10**(2), 119–130.

Sun Y and Sarrafzadeh M (1993) Floor planning by graph dualization: L-shaped modules. *Algorithmica* **10**(6), 429–456.

Tamassia R, **Tollis IG and Vitter JS** (1991) Lower bounds for planar orthogonal drawings of graphs. *Information Processing Letters* **39**(1), 35–40.

Ueckerdt T (2012) Geometric representations of graphs with low polygonal complexity. Phd thesis, Department of Mathematics at TU Berlin. https://i11www.iti.kit.edu/_media/members/torsten_ueckerdt/torsten_ueckerdt-phd_thesis.pdf/.

Upasani N, **Shekhawat K and Sachdeva G** (2020) Automated generation of dimensioned rectangular floorplans. *Automation in Construction* **113**, 103149.

Wu F, **Yan D-M**, **Dong W**, **Zhang X and Wonka P** (2013) Inverse procedural modeling of facade layouts. *arXiv preprint arXiv:1308.0419*.

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.

Yeap K-H and Sarrafzadeh M. (1993) Floor-planning by graph dualization: 2-concave rectilinear modules. *SIAM Journal on Computing* **22**(3), 500–526.