MIQP-based Layout Design for Building Interiors

Wenming Wu\textsuperscript{1,2}  Lubin Fan\textsuperscript{2}  Ligang Liu\textsuperscript{1}  Peter Wonka\textsuperscript{2}

\textsuperscript{1}University of Science and Technology of China, China
\textsuperscript{2}King Abdullah University of Science & Technology, Saudi Arabia

Abstract

We propose a hierarchical framework for the generation of building interiors. Our solution is based on a mixed integer quadratic programming (MIQP) formulation. We parametrize a layout by polygons that are further decomposed into small rectangles. We identify important high-level constraints, such as room size, room position, room adjacency, and the outline of the building, and formulate them in a way that is compatible with MIQP and the problem parametrization. We also propose a hierarchical framework to improve the scalability of the approach. We demonstrate that our algorithm can be used for residential building layouts and can be scaled up to large layouts such as office buildings, shopping malls, and supermarkets. We show that our method is faster by multiple orders of magnitude than previous methods.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computing Methodologies]: Computer Graphics—Computational Geometry and Object Modeling

1. Introduction

We study the automatic computation of interior building layouts, to either more quickly explore design variations for architects or to completely replace the manual modeling process for creating large-scale virtual worlds. The design process typically starts by collecting a list of requirements (constraints) for the layout, e.g., room size and room adjacency. Then the designer uses trial-and-error to generate one or multiple layouts using a combination of intuition, prior experience, and professional knowledge. This usually takes from a couple of days to several weeks.

There are already several successful methods for generating building layouts automatically. The first aspect of layout generation is the determination of what constraints to use. Merrel et al. \cite{MSK10} sample constraints from a Bayesian network. The second aspect of layout generation is the actual optimization step that computes a layout that fulfills the given constraints. Our work focuses on this part of layout generation. The most popular approach for this step is to use stochastic sampling on a framework that evaluates constraints as a black box. For example, Metropolis-Hastings (used by Merrel et al. \cite{MSK10}), simulated annealing, or genetic algorithms are possible alternatives. The popularity of the approach...
is due to its generality as almost any type of constraints can be incorporated as a black box. The downside of this approach is that it degenerates on medium and larger-scale layouts and typically cannot find a single valid solution in a reasonable amount of time. We therefore propose to employ more powerful optimization techniques that scale better to larger layouts. For example, Peng et al. [PYW14] apply linear integer programming to generate layouts by tiling templates. Their tiling-based approach scales better, but it is less general. They can use only pre-defined room templates.

Our new framework overcomes the scalability problem of stochastic methods without all the restrictions on generality imposed by the tiling-based approach. Our solution has multiple components that are coordinated with each other. We formulate the layout optimization problem as a mixed integer quadratic programming (MIQP) problem. The integer variables are used to account for different room configurations. An integral component to this formulation is our proposed parametric layout representation. Each room is represented as a polygon that consists of a set of small rectangles. This enables us to generate rooms with various shapes in contrast to Peng et al. [PYW14]. In addition, this representation enables us to formulate important high-level constraints about room size and adjacency in a manner that is compatible with MIQP. We also propose a hierarchical framework to improve the scalability of our approach. Fig. 1 shows layouts of a residential building and an office building generated by our algorithm.

We demonstrate our method on small-scale layouts of residential buildings and large-scale layouts such as office buildings and supermarkets. Our main contributions are as follows:

- We propose a layout computation method that is faster by multiple orders of magnitude than previous work built on stochastic optimization.
- We propose a parametric representation of interior layouts that is more general than previous methods based on tiling.
- We can generate results on a larger scale than achieved by previous work.

2. Related Work

Our work relates to research in three areas: VLSI design, architectural space planning, and urban layout modeling.

VLSI design. VLSI designs can be categorized as slicing structures that are presented by binary trees, and non-slicing structures that are more general. Topological representations are crucial for VLSI floorplan design. Popular VLSI floorplan representations include the following: sequence pair (SP) [SVW'11], bounded slicing grid (BSG) [NFMK97], B*-tree [MXM09], ordered tree (O-Tree) [NNA06], transitive closure graph (TCG) [LC04], corner block list (CBL) [DZH'02], and integer coding (IC) [CGC10]. Most VLSI design algorithms search for a floorplan using stochastic approaches, e.g., simulated annealing [Sec12], genetic algorithms [SDD12], particle swarm optimization [CGC10], and differential evolution [MAR07]. Simulated annealing is popular in modern design problems using the B*-tree [CC06] and sequence pair [KF00], but it requires expensive computations and it can struggle to achieve optimal or near optimal solutions. Genetic algorithms [CZ10, SDD12] are another algorithm class explored by researchers in their quest for good floorplan algorithms. However, as a generate-and-test methodology, genetic algorithms have very similar shortcomings to simulated annealing. We propose a more general representation for layout design in which each room is represented as a polygon that can be further decomposed into a set of rectangles and our constraints are different.

Architectural space planning. Architectural space planning is concerned with planning buildings of different scales. One popular approach is to apply an expert system [FUC88, KS05]. Río-Cidoncha et al. [DRCMP07] develop a model for facility layout design by combining an expert system and artificial intelligence. Another class of methods applies shape grammars that consist of a set of rules that is recursively applied to an initial assertion to produce a final statement, e.g., [RCMLS96, LHP11]. Harada et al. [HWB95] develop a system for interactive manipulation of architectural layouts by using shape grammars. Duarte [Dua05] proposes an interactive system for generating layouts on the web based on a discursive grammar that consists of a programming grammar and a designing grammar. The evolutionary approach is also used in architectural space planning [EP99, Nii06, Dou07]. Bahrehmand et al. [BBM17] present an interactive layout solver based on the evolutionary approach. It assists designers in layout planning by recommending personalized space arrangements according to architectural guidelines and user preferences. The major drawback of evolutionary algorithms is the poor performance. The constraint-based approach [BP91, LFT00, Hsu00, DB08] seems to be a promising direction for our work.
Merrel et al. [MSK10] learn attributes of rooms from a given dataset by a Bayesian network and synthesize the layout by the stochastic method. The main problem of the approach is that it is not suitable for large-scale examples due to the stochastic method. Rosser et al. [RSM17] also propose a data-driven method. They learn building measurements to produce interior plans. In [LYAM13], the authors present an interactive system to create interior room layouts subject to design constraints, user preferences, and manufacturing considerations. Hua et al. [Hu16] develop a method for automatically constructing irregular floor plans. Given image patterns, the program produces a variety of layouts that satisfy both geometric and topological requirements. Bao et al. [BYMW] propose a method to explore building layouts. To obtain variations in connectivity, they use standard simulated annealing. Liu et al. [LWK17] apply integer programming to encode high-level constraints. However, their goal is to evaluate the extracted constraints in recovering the floorplan data from a raster image.

Our problem also relates to urban layout modeling. Ali [Ali12, STBB14] review the topic in a broad context. One common approach generates large-scale layouts by first creating the road network and then by generating parcels. An interactive example-based system for synthesizing urban layouts is proposed by [AVB08]. They use the structure and image data of real-world urban areas for complex urban layout generation. Network design can be seen as a dual problem in layout design. Chen et al. [CEW08] create road networks from an existing street network by designing an underlying tensor field and editing the graph representation. Yang et al. [YWVW13] combine hierarchical splitting and global optimization to guarantee the fairness and regularity of the generated streets and parcels. Improving upon this previous work, Peng et al. [PYW14] propose a novel tiling-based approach with deformable templates. Their subsequent work [PYB16] proposes an integer-programming-based approach to generate networks starting from functional specifications. Feng et al. [FYY16] focus on mid-scale layout modeling based on human crowds. Given a layout domain, their approach synthesizes crowd-aware layouts by considering the crowd flow. Urban layout modeling has also been widely used in the design of video game environments. Ma et al. [MVL14] propose an algorithmic approach for automatically laying out complex game levels that conform to the designer's specifications.

3. Overview

Our framework consists of the following components:

- First, we introduce a parametric representation that describes a layout by a set of axis-aligned polygons. To make the model better suitable for optimization, we also present a rectangle-based layout representation (Section 4).
- Second, we describe the basic formulation of our MIQP-based method (Section 5.1).
- Third, we describe a set of high-level constraints for layout generation (Section 5.2).
- Fourth, we propose a hierarchical framework to extend the basic method (Section 5.3).

- Fifth, we evaluate our method and perform comparisons with previous methods on examples of different scales (Section 6).

Fig. 2 shows an example of generating a layout of an apartment. Given the outline of the apartment and a set of user-specified high-level constraints, our method generates the layout in a coarse-to-fine manner, i.e., two levels in this example. Fig. 2 right shows the final result.

4. A Parametric Layout Representation

The layout L of a building consists of its domain D as an axis-aligned polygon and rooms $R = \{r_i\}^N_{i=1}$, where $N$ is the number of rooms in the layout (see Fig. 3 left). The shape of the domain is an axis-aligned polygon that is defined by an array of vertices, $D = \{d_i\}$. Each room is also an axis-aligned polygon defined by vertices, $r_i = \{r'_i\}$, and a label, $l_i$, to describe its functionality, e.g., kitchen, living room, etc. Given the layout domain, the goal of the layout design problem is to arrange a set of rooms inside the domain according to some constraints. A valid layout should satisfy two basic constraints: first, all rooms are inside the domain; second, there is no overlap between rooms. A layout should also satisfy a set of high-level constraints describing how a synthesized layout should behave, e.g., room size and adjacency.

There are two challenges of this parametric representation. First, the number of vertices of each polygonal room is not known in advance. This means that the number of variables has to change during the optimization, even with a fixed number of rooms. Second, it is not easy to formulate high-level constraints for arbitrary polygons. For example, expressing constraints about the adjacency between general polygonal rooms is very difficult. We therefore introduce the computational object, room rectangle, that is defined as the basic element of a layout (see Fig. 3 right). A room rectangle is described by a tuple $(x_c, y_c, w_i, d_i, l_i)$ where $(x_c, y_c)$ denotes the position of the bottom-left corner of the rectangle, $(w_i, d_i)$ denotes the...
width and depth, and \( l_i \) denotes its label. Then, a polygonal room, \( r_i \), can be presented as the union of a set of room rectangles with the same label. The boundary of the room polygon is the outline of the union of rectangles in the polygon. The layout domain can also be presented by rectangles (see Fig. 3). The polygonal domain can be described as its bounding box minus a set of special rectangles labeled as \( \text{obstacle} \). Such obstacle rectangles cannot be covered by any room. By introducing the room rectangle, the layout, \( L \), can be presented by a set of rectangular rooms. Given the number of room rectangles of the layout, \( N \), the number of parameters is fixed, i.e., \( 4 \times N \). Moreover, the relationship between rectangles is easy to evaluate. According to the definitions of the interior layout and the room rectangle, our problem is to find the optimal set of rectangles, i.e., \( \{ (x_i, y_i, w_i, d_i, l_i) \} \).

Given the layout domain and the number of rooms, the interior layout design task changes to generate a valid layout in which room rectangles are tiled in the domain with optimal parameters. We apply mixed integer quadratic programming (MIQP) optimization to solve the tiling problem. Then, we obtain the final layout by merging connected room rectangles with the same label.

5. MIQP-based Hierarchical Layout Design

We introduce our algorithm starting from the simplest case in which both the layout domain and the rooms are rectangles (Fig. 4 left). Next, we introduce five high-level constraints abstracted from real floorplans to generate more realistic layouts. Finally, we introduce a hierarchical framework to generate more complicated layout generation in which the layout domain and rooms are polygons (Fig. 4 right).

5.1. Basic Formulation

Our goal is to create a valid layout that covers the whole domain. We consider the layout to be valid if it satisfies two basic constraints: the inside constraint, which ensures that all rooms are inside the boundary, and the non-overlap constraint, which ensures that there is no overlap between rooms.

**Inside constraint, \( C_{\text{inside}} \)**. In a valid layout, we require that all rooms are inside the layout domain. Since the domain and rooms are all rectangles, the constraint requires that four corners of each rectangle are inside of the bounding box of the domain. The constraint is expressed as

\[
\begin{align*}
0 \leq x_i & \leq w \\
0 \leq y_i & \leq d \\
x_i + w_i & \leq w \\
y_i + d_i & \leq d,
\end{align*}
\]

(1)

where \( 1 \leq i \leq N \), \( N \) is the number of rooms, and \( w \) and \( d \) are the width and depth of the bounding box of the domain, respectively. For the case of a polygonal domain, we update the inside constraint in Section 5.3.

**Non-overlap constraint, \( C_{\text{overlap}} \)**. We require that there is no overlap between any pair of rooms. There are four kinds of relationship between two rectangular rooms, i.e., **room i** is on the front/back/left/right side of **room j**. The difficulty of setting this constraint is that we have no prior knowledge of which case it is. We introduce an auxiliary binary variable, \( \sigma_{i,j}^d \), into our problem for each pair of rooms \( (i,j) \) to select the relationship automatically, where \( d \in \{ \text{front, back, left, right} \} \), and \( \sigma_{i,j}^d = 1 \) means that the relationship \( d \) of **room i** and **room j** is selected. The non-overlap constraint is then defined as

\[
\begin{align*}
x_i - w_j & \geq x_j - M \cdot (1 - \sigma_{i,j}^{\text{left}}) \\
x_i + w_i & \leq x_j + M \cdot (1 - \sigma_{i,j}^{\text{right}}) \\
y_i - d_j & \geq y_j - M \cdot (1 - \sigma_{i,j}^{\text{front}}) \\
y_i + d_i & \leq y_j + M \cdot (1 - \sigma_{i,j}^{\text{back}}) \\
\sum_{d=1}^4 \sigma_{i,j}^d & \geq 1,
\end{align*}
\]

(2)

where \( M \) is a large constant to ensure that there is no overlap between **room i** and **room j** in direction \( d \) when \( \sigma_{i,j}^d = 1 \), and the inequality is always established when \( \sigma_{i,j}^d = 0 \). In our implementation, we set \( M = w + d \). The last inequality ensures that at least one of the above cases should be satisfied. Note that the introduced auxiliary binary variables, \( \sigma_{i,j}^d \), encode the relationship between **room i** and **room j** in the constraint, and they are assigned automatically during the optimization. This method is also used in the following constraints.

**Objective function.** We evaluate valid layouts that satisfy the two basic constraints, \( C_{\text{inside}} \) and \( C_{\text{overlap}} \), based on an energy function that prefers layouts that cover the domain as much as possible. The energy function is defined as

\[
E_{\text{cover}}(L) = \text{Area}(L) - \sum_{i} w_i \times d_i,
\]

(3)

where \( \text{Area}(\cdot) \) is the area operator. Minimizing Eq. 3 under the two basic constraints, \( C_{\text{inside}} \) and \( C_{\text{overlap}} \), yields a mixed integer program with a quadratic objective function (MIQP) and linear constraints,

\[
\min_L E_{\text{cover}}(L).
\]

(4)

Fig. 4 left shows a layout generated by the basic formulation.

5.2. Optional High-level Constraints

We model five optional high-level constraints that are essential to interior building layout generation.

**Size control.** The size of each room can be controlled by two
approaches: adding size range constraints for rooms and specifying target sizes for rooms. The range of the room size can be constrained by providing the minimum and maximum values of \( w_i \) and \( d_i \). Then, the size range constraint, \( C_{\text{size}} \), is defined as

\[
\begin{cases}
    w_i \leq w_i' \leq \bar{w}_i, \\
    d_i \leq d_i' \leq \bar{d}_i,
\end{cases}
\]

where \((w_i', d_i')\) and \((\bar{w}_i, \bar{d}_i)\) are the minimum and maximum sizes of room \( i \), respectively. The user can also specify target sizes for rooms. Then, the size error of rooms can be evaluated by a size error term, which is defined as

\[
E_{\text{size}} = \sum_i (w_i - w_i')^2 + (d_i - d_i')^2,
\]

where \((w_i', d_i')\) is the target size of room \( i \). We can encourage room sizes to become close to the user-specified values by adding \( E_{\text{size}} \) to the objective function.

Aspect ratio constraint, \( C_{\text{ratio}} \). We provide a room aspect ratio constraint, \( C_{\text{ratio}} \), to avoid generating rooms that are too wide or too narrow. Similar to \( C_{\text{overlap}} \), we add an auxiliary binary variable for each room to adjust the orientation (i.e., horizontal and vertical) of the room rectangle. The aspect ratio constraint is defined as

\[
\begin{cases}
    \ell_i \cdot d_i \leq w_i + M \cdot \sigma_i, \\
    \ll_i \cdot w_i \leq d_i - M \cdot (1 - \sigma_i),
\end{cases}
\]

where \((\ell_i, \ell)\) are the minimum and maximum aspect ratios between \( w_i \) and \( d_i \) of room \( i \), and \( \ell \geq 1 \). \( \sigma_i \) is 0 means that room \( i \) is a horizontal room and its aspect ratio is in the range of \([\ell_i, \ell] \). \( \sigma_i = 1 \) means that room \( i \) is a vertical room.

Position constraint, \( C_{\text{pos}} \). During the design process, sometimes the designer has to specify the approximate position for a room, e.g., the foyer should connect to the main door. A guide position for a room is presented as a single point \((x^*, y^*)\) or two points. For each point, the position constraint requires the room to cover the point:

\[
\begin{cases}
    x_i \leq x^* \leq x_i + w_i, \\
    y_i \leq y^* \leq y_i + d_i.
\end{cases}
\]

Boundary constraint, \( C_{\text{boundary}} \). Sun exposure is an important planning criterion. For example, people typically have a preference between the master bedroom receiving light in the morning (facing east) or not receiving light in the morning (facing west). To fulfill these types of constraints, we add a boundary constraint that requires the room to be adjacent to at least one of a set of user-specified edges of the domain polygon. We introduce a binary variable, \( \rho_k (k = 1, 2, \ldots, n) \), for edge \( k \) where \( n \) is the number of edges of the domain polygon, and \( \rho_k = 1 \) indicates that the room is adjacent to edge \( k \). For each candidate edge in the given set, we formulate the constraint to check if the edge covers the corresponding edge of the room. Suppose that the edge is a bottom edge. It is then defined as \([(x_1, y), (x_2, y)]\) (i.e., a horizontal edge on the bottom side of the domain polygon), Fig. 5 left illustrates the constraint for room \( i \), and it is expressed as

\[
\begin{cases}
    y_i \leq y + M \cdot (1 - \rho_k), \\
    x_i \leq x_2 - w_i + M \cdot (1 - \rho_k), \\
    x_i \geq x_1 - M \cdot (1 - \rho_k), \\
    \sum_{k=1}^n \rho_k \geq 1,
\end{cases}
\]

where the last inequality signifies that at least one of the candidate edges should be selected. There are two differences between \( C_{\text{boundary}} \) and \( C_{\text{pos}} \). First, \( C_{\text{pos}} \) requires the room to cover the specified positions, while \( C_{\text{boundary}} \) requires the room to select one edge from a candidate set. Second, the candidate edge is always on the boundary of the layout domain, but users can specify any position for constraint, \( C_{\text{pos}} \).

Adjacency constraint, \( C_{\text{adj}} \). The user can also specify two rooms to be adjacent, e.g., the master bedroom is connected to the living room. Suppose that room \( i \) \((x_i, y_i, w_i, d_i)\) and room \( j \) \((x_j, y_j, w_j, d_j)\) should be connected. The idea for modeling this constraint is to force the two rooms to overlap. In combination with the non-overlap constraint, \( C_{\text{overlap}} \), which forbids the inside of the rooms to overlap, this will force the overlap to happen only at the boundary of the rooms. This is not sufficient, however, because we would also like to ensure that there is a minimum overlap, \( c \), called the contact length, of the common edge between the two connected rooms. The contact length is important to ensure that there is enough space for adding a door between the two rooms. The constraint can be written as

\[
\begin{cases}
    x_i \leq x_j + w_i - c \cdot \theta_{ij}, \\
    x_j + w_i \geq x_i + c \cdot \theta_{ij}, \\
    y_i \leq y_j + d_i - c \cdot (1 - \theta_{ij}), \\
    y_i + d_i \geq y_j + c \cdot (1 - \theta_{ij}),
\end{cases}
\]

where \( \theta_{ij} \) is a binary variable that determines the connection direction automatically. \( \theta_{ij} = 1 \) denotes a vertical connection and otherwise a horizontal connection. Fig. 5 right illustrates the constraint for a vertical connection. We extend the constraint to allow room \( i \) to connect either room \( j \), room \( k \), or both. Then the constraint
The sub-domain is selected automatically by the following strategies. First, if the layout domain is not completely covered by rooms generated in the previous layout, then we select the polygon with rectangular rooms that surround the empty region. Second, if the size error of a rectangular room is larger than a threshold, then we select the polygon with that room and its neighboring rooms as the sub-domain. Moreover, the sub-domain can also be specified by the user.

**Initialization by room decomposition.** We initialize the layout optimization by decomposing rooms in the sub-domain. Given a rectangular room in the sub-domain, we decompose it into two small rectangular rooms of the same size in a random direction (see Fig. 6). Each small rectangular room inherits the label from its parent.

**Constraints update.** Given the new layout domain and initialized rooms inside the domain, we set the constraints for the new MIQP optimization as follows. First, we update the inside constraint, $C_{\text{inside}}$, and the non-overlap constraints, $C_{\text{overlap}}$, using the new layout domain and rectangular rooms. For a polygonal domain, the inside constraint for each room rectangle is presented by two parts: first, the rectangle should be inside the bounding box of the domain; second, the rectangle cannot cover any obstacle. Second, for the original room with position constraint, $C_{\text{pos}}$, we set the new position constraint for its child, which has to cover the position guidance and leave the position constrain free for the other child. We update the boundary constraint, $C_{\text{boundary}}$, in the same manner. Third, an additional room adjacency constraint for two child rooms is added to keep them connected in the final layout. Fourth, we add a size refinement constraint, $C_{\text{refine}}$, for each child room (i.e., $r_{i1}$ and $r_{i2}$) to replace the size constraints on their parent room, $r_i$. The idea of $C_{\text{refine}}$ is that we can restrict size changes of each child room in a small refinement range, $\delta$, to avoid significant changes in the final layout. Suppose a vertical split is applied in the previous step (see Fig. 6 middle). Then, the refinement constraint is defined as

$$\begin{align*}
x_{i1} & \geq x_i - \delta \\
x_{i1} & \leq x_i \\
y_{i1} & \geq y_i - \delta \\
y_{i1} & \leq y_i \\
y_{i1} + d_{i1} & \geq y_{i1} + d_{i1} \\
y_{i1} + d_{i1} & \leq y_{i1} + d_{i1} + \delta \\
x_{i2} + w_{i2} & \geq x_i + w_i \\
x_{i2} + w_{i2} & \leq x_i + w_i + \delta \\
y_{i2} & \geq y_i - \delta \\
y_{i2} & \leq y_i \\
y_{i2} + d_{i2} & \geq y_{i2} + d_{i2} \\
y_{i2} + d_{i2} & \leq y_{i2} + d_{i2} + \delta,
\end{align*}$$

where $(x_{i1}, y_{i1}, w_{i1}, d_{i1})$ are parameters of child room $i_1$, $(x_{i2}, y_{i2}, w_{i2}, d_{i2})$ are parameters of child room $i_2$, and $\delta$ is the refinement range. The refinement constraint for the horizontal split is presented in the supplementary material.

**Optimization.** We formulate the objective function as Eq. 12 subject to the updated constraints for the sub-domain. The generated layout will replace the original rectangular rooms in the sub-domain. If there are several sub-domains, we apply the same procedure to all of them sequentially. We move to the next level when
Initialization & Dining
2.25 x 2.25
Kitchen
2.00 x 2.75
M. Bedroom
2.25 x 3.25

Figure 7: Pipeline of the hierarchical framework. (a) Starting from the empty layout domain with the door position and two obstacles, our method generates (b) an interior layout on the first level according to the user-specified high-level constraints. There is a region (shown in red) that is not covered by any room. (c) Our algorithm automatically selects the sub-domain (highlighted as the red polygon) for the optimization on the next level. (d) Then, the sub-domain is initialized with small rectangles and constraints are updated. (e) The new layout is generated in the sub-domain (i.e., red polygon). The user specifies the study room (blue rectangle) for further improvement. (f) The final result.

all sub-domains are optimized. Starting from an empty layout domain, we generate the layout design in a coarse-to-fine manner by the proposed hierarchical framework. We terminate the hierarchical optimization when the layout domain is completely covered and the size error of each room is smaller than the threshold. To obtain the final interior layout, which is represented by polygonal rooms, we merge rectangular rooms with the same label.

Large-scale layout generation. We apply a hierarchical framework for large-scale layout generation. Fig. 8 shows an example of a shopping mall that is generated with four levels. In practice, the large-scale layouts can be decomposed into regions and rooms in each region. Our idea is to generate the layout for regions first, then generate the layout inside each region. Given the layout domain and constraints for regions, we apply the hierarchical framework to generate the layout for regions. The polygonal regions are generated in a coarse-to-fine manner by the aforementioned framework. Then, aisles are generated by expanding the gap between two regions with a fixed width if needed. Next, we set each region as a sub-domain and assign constraints for each sub-domain. The number of rooms in each sub-domain and high-level constraints for them are specified by the user. Then, we formulate the objective function for each sub-domain and generate the corresponding layouts. We repeat this procedure to get the final layout. Fig. 1(b) shows another example of a layout of an office building.

5.4. Implementation Details

We implement our algorithm in C++ using Gurobi [GO16] to solve the MIQP problem. As it is difficult to find a globally optimal solution, we also accept sub-optimal solutions that fulfill all constraints computed within reasonable time limits.

Users can set some of the high-level constraints on a subset of rooms or on all of them. Since modeling realistic constraints is a challenging task, we also provide an option for automatic constraint modeling for residential buildings. We collect 200 floorplans of residential buildings and learn the size constraints and adjacency constraints. We first train polynomial regression models for width and depth of each kind of room with respect to the square root of the domain area. Given a new domain, the range of the width and depth of each room are set as the range of the 90% confidence band of the corresponding model. We also collect connectivity information from the data to randomly sample a useful set of adjacency constraints.

The performance of synthesizing a good layout is related to the size constraints of rooms. In practice, we apply several optimization strategies for acceleration. If target sizes of rooms are given, we apply a heuristic to set up the minimum and maximum sizes for specified rooms to help Gurobi quickly find the feasible solution. We first set 90% and 110% of the target value as the minimum and maximum size, respectively. Then, we gradually relax the size of the constraints by 10% until Gurobi finds a feasible solution within reasonable time limits.

6. Results and Applications

All of our experiments are performed on a laptop with dual 2.9 GHz Intel Core i5 processors and 8 GB main memory.

Interior layout for residential buildings. First, we apply our method to an apartment layout design. The layout domain, the number of rooms and the functionality of each room are given. In Fig. 9, we show two interior layouts designed by our algorithm with the same layout domain and constraints. Boundary constraints are
Figure 8: An interior layout of a shopping mall is generated by our hierarchical framework. (a) Starting from an empty layout domain with obstacles. (b) First, we generate the layouts for five regions shown in different colors. (c) Then aisles are generated by expanding the gap between two regions with a fixed width. (d) Next, we generate the layouts inside each region. (e-h) We repeat this procedure to get (i) the final layout. The number of objects (regions and rooms) and constraints on each level are specified by the user.

Figure 9: Two apartment layouts are generated by our algorithm with the same input. Boundary constraints are added for bedrooms to ensure that all bedrooms receive sufficient sunshine from the south east direction. The north direction is shown.

Figure 10: An interior layout of a bungalow-style house generated by our algorithm. The floorplan is shown in the middle. Two sides of the 3D rendering are shown.

Figure 11: An interior layout of a two-storey house. The stairs are considered as special rooms that should be consistent between two floors. The floorplan of each floor is shown on the left, 3D renderings are shown on the right.

Figure 12: Regeneration of a layout according to the constraints extracted from a given layout. Left: the ground-truth layout. Right: an interior layout regenerated from the size and adjacency constraints extracted from the left layout.

Evaluation of our method. First, we evaluate our approach by reproducing interior layouts of buildings. A good layout algorithm should reproduce reasonable results from constraints extracted from real designs. We extract only size and room adjacency constraints from the given design shown in Fig. 12 left. A regenerated layout is shown in Fig. 12 right. We find that all connectivities are maintained and that the size error of each room is small. Second, we perform a comparison between layouts generated by the basic method and by our hierarchical approach with the same layout domain and high-level constraints. Since the basic approach only tiles rectangles on one level, there are some small regions remaining (see Fig. 13 left). In contrast, our hierarchical approach can generate a layout that completely covers the whole domain. Third, we
evaluate the effect of each constraint by performing leave-one-out tests. Fig. 14 shows the effect of each constraint. We can see that the kitchen size is unreasonable if the size error term is excluded. The foyer is not connected to the main door without the position constraint. When the boundary constraint for the kitchen is excluded, there is no pathway to other rooms except the study. The performance of our algorithm is shown in Table 1. In our experiments, our algorithm can generate a mid-size layout in 15 seconds on average.

We also evaluate our method on large-scale examples, e.g., Fig. 14 shows the effect of each constraint. We can see that the kitchen size is unreasonable if the size error term is excluded. The foyer is not connected to the main door without the position constraint. When the boundary constraint for the kitchen is excluded, there is no pathway to other rooms except the study. The performance of our algorithm is shown in Table 1. In our experiments, our algorithm can generate a mid-size layout in 15 seconds on average.

We also evaluate our method on large-scale examples, e.g.,}(369,728,515,740)
The effect of each constraint. (a) We extract the attributes from a real layout design. (b-f) Ablation of individual constraints and the effect on the results. The unreasonable part of each layout is highlighted.

Figure 14: The effect of each constraints. (a) We extract the attributes from a real layout design. (b-f) Ablation of individual constraints and the effect on the results. The unreasonable part of each layout is highlighted.

Figure 15: Interior layouts of office buildings generated by our method. Polygons in the same color are from the same parent rectangle in a higher level.

Figure 16: Interior layouts of supermarkets generated by our method. Shelves for products in the same category are in the same color.

Figure 17: Given the same input (i.e., the layout domain and constraints), layouts generated by our method and the stochastic method are shown. The stochastic method took 10 times longer than our method. The empty region is highlighted in red, and rooms that are too small for furniture are highlighted in yellow.

Figure 18: Performance comparison between the stochastic method and our algorithm. For the stochastic method, the running time increases significantly when the complexity (i.e., the sum of the number of rooms and the number of constraints) of the problem increases, while the running time of our approach increases slowly. The running time of the stochastic method for the last experiment is longer than 2 hours. We terminated it before obtaining a result.

the Visual Computing Center at KAUST. Ligang Liu is supported by the National Natural Science Foundation of China (61672482, 61672481, 11626253) and the One Hundred Talent Project of the Chinese Academy of Sciences. We would like to thank Virginia Unkefer for proofreading the paper.

References


[CC06] CHEN T.-C., CHANG Y.-W.: Modern floorplanning based on b*-tree and fast simulated annealing. IEEE Transactions on Computer-


