Noname manuscript No.
(will be inserted by the editor)

Constraint-aware Interior Layout Exploration for Precast Concrete-based Buildings

Han Liu · Yong-Liang Yang · Sawsan AlHalawani · Niloy J. Mitra

Received: date / Accepted: date

Abstract Creating desirable layouts of building interiors is a complex task as designers have to manually adhere to various local and global considerations arising from competing practical and design considerations. In this work, we present an interactive design tool to create desirable floorplans by computationally conforming to such design constraints. Specifically, we support three types of constraints: (i) *functional constraints* such as number of rooms, connectivity among the rooms, target room areas, etc.; (ii) *design considerations* such as user modifications and preferences; and (iii) *fabrication constraints* such as cost and convenience of manufacturing. Based on user specifications, our system automatically generates multiple floor layouts with associated 3D geometry that all satisfy the design specifications and constraints, thus exposing only the desirable family of interior layouts to the user. In this work, we focus on precast concrete-based constructions, which lead to interesting discrete and continuous optimization possibilities. We test our framework on a range of complex real-world specifications and demonstrate the control and expressiveness of the exposed design space relieving the users of the task of manually adhering to non-local functional and fabrication constraints.

Keywords interior layout · constraint-aware form exploration · precast concrete · functional layouts · computational design

H. Liu · Y-L.Yang · S.AlHalawani
King Abdullah University of Science and Technology

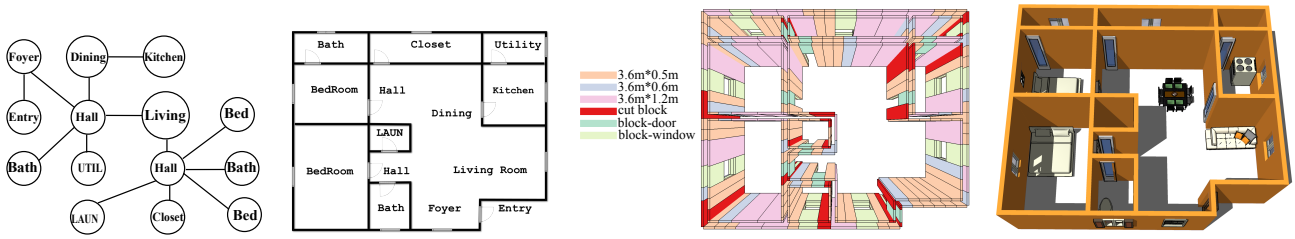
N. J. Mitra
University College London
E-mail: n.mitra@cs.ucl.ac.uk

1 Introduction

Good room layouts in building designs have to consider functional and personal specifications coming from the user, while also taking into account the final manufacturing cost. For most designers, manually accounting for such heterogeneous requirements remains a challenge, and encourages conventional layouts that often ignore manufacturing implications (i.e., cost).

Interior layout design amounts to arranging input rooms in a given *envelope* or approximate space area under scale and connectivity constraints (see Figure 1). Additional design objectives are geometric and topological, e.g., denoting room sizes and relations, respectively [1]. There can also be accessibility considerations, for example, a bathroom is preferably attached to the main bedroom; the dining room should be next to the kitchen; etc. Further, as a sanity measure, all the rooms should be accessible by connected paths from the main door. In the case of multi-storied buildings, there are additional constraints, e.g., the position and the dimensions of stairs have to match across the different floors. Interestingly, a large number of such constraints are applicable to both virtual scenes and real physical setups.

Automatic generation of interior layouts has been studied recently based on user specified requirements such as expected room area and connectivity among the rooms (c.f., [1,13]). In our setup, the user first indicates such global functional specifications by prescribing approximate area of each room and a connectivity graph indicating the desired inter-room connections. Subsequently, the system returns realizations conforming to the input constraints. Even when such a layout satisfies all the predefined conditions, users may still want to fine tune the design, such as change room areas or swap pairs of rooms. More importantly, being oblivious



(a) design graph (b) optimized layout (c) casting concretes (d) 3D model

Fig. 1 Interactive floor plan generation by our system based on constrained optimization and precast concrete based construction cost minimization. Concrete slabs that were cut are highlighted in red. (Please see supplementary video and demo.)

to the final construction, the user designed floor plan may be ill-suited to the actual manufacturing process, and thus be expensive. Hence, besides ensuring functional floor plan when the user manipulates any suggested design, we specifically optimize the geometric realization to produce economic constructions. Thus, both functional and fabrication constraints are directly accounted for in our proposed interactive design setup.

In this paper, we focus on a particular construction methodology, namely on precast concrete based buildings. The standard practice is to pre-fabricate a (discrete) number of concrete slabs of certain dimensions (i.e., length, width, and thickness) using reusable molds, transport the slabs to the construction site, and directly assemble the slabs to form the buildings. Further, to encourage mold reuse, only a small number of slab sizes are supported. Hence, to build a wall of arbitrary dimension (e.g., length), it is often necessary to custom cut bigger pieces of slabs, a process that can be both expensive and time consuming. Figure 2 shows some real precast concrete based designs. Note that some of the slabs are sliced to fit the wall length, or have been drilled to make space for widows and doors.

Thus, while it is possible to create custom-made concrete slabs, they are preferably avoided due to cost considerations. Hence, our goal is to reduce costs in precast buildings by decreasing the number of cuts required. Figure 1 shows an instance of a layout with double-walled precast concrete slabs using our approach, wherein the gaps in between the concrete slabs are filled with insulation material (or left void).

After generating an initial layout, all the wall segments become candidate elements to support user interactions. In order to support interactivity, the system approximates rooms as rectangular *proxies* to optimize and later regenerate a layout following a set of predefined rules. The regeneration is challenging since the users cannot easily predict possible consequences after arbitrary interactions such as irregular rooms leading to undesired corners, or scattered short wall segments that result in manufacturing difficulties. In such cases, the system proposes local changes reflecting the user intents as well as global optimization targeting functional and fabrication efficacy.

Our algorithm focuses on two key considerations. First, we generate an initial design based on user inputs, and then iteratively regenerate the layout based on user refinements. Such layouts satisfy predefined specifications and reflect user prescribed changes. Second, we closely follow real world manufacturing implications to improve the designed floor plan. Figure 1 shows a typical work flow and a final layout produced in a design session. We tested our system on a range of different examples and demonstrated the strengths and limitations of such a system for computational design (see also supplementary video and demo).

2 Related Work

Architectural floor plan design has been widely studied in the context of automatic space planning. Layout design was initially treated as packing problem and cate-

Fig. 2 Concrete casting in real-world building manufacture.

gorized as slicing and non-slicing structures [2,18]. Typical approaches include using tree and graph structures to search over possible configurations of slicing floor plans [11,12,16,22]; or employing branch-and-bound to place a number of blocks inside a given layout envelope [18]. In other contexts, non-slicing floor plans are evolved from circuit layout problem to minimize chip areas [6,8,17]. More recently, layout design has been extended to simplify indoor furniture placement [14].

Most approaches search over possible placements in a design space. Galle et al. [4] implemented an exhaustive algorithm to select the rectangular arrangements satisfying constraints among all the possible generations. However, due to computational restrictions, the method could only handle layouts with up to ten rooms. Wong et al. [21] searched feasible solutions by simulated annealing. Alternately, physically based modeling [1,7] has been introduced to simulate rooms as mechanical elements whose motions are manipulated by forces defined between inter-room spacing. Arvin et al. [1] summarize geometric and topological objectives to search space planning solutions, e.g., alignment, area, and proportion objectives.

Evolutionary algorithms have been applied to search layout possibilities by treating design variants via crossovers and mutations operators [3,15,19]. In these approaches, evolutionary strategy is used to fit rooms into target envelopes, while improving appropriately designed fitness functions. Further, mutations are allowed during such evolution, for example to switch rooms (crossover) in order to optimize connectivity [9]. Recently, an automated layout generation has been proposed to sample and efficiently explore the layout search space [13]. The method, however, is not designed to support interactive design refinements.

In a related attempt, Harada et al. [7] design a system that allows users to interactively drag rooms, but the possible layouts are pre-defined. Specifically, the algorithm searches for a matched state that best reflects user intents from a set of constructed transformations for mapping states. Further, only limited sets of constraints are considered, and the method does not generalize to handle manufacturing constraints. More recently, physical and manufacturing considerations have also been explored in the context of geometric form finding [20]. Similarly, in this work, we focus on pre-cast concrete based constructions and consider its implications in design and layout problems.

We integrate the users in the design process by supporting direct manipulation of the floor plan, followed by a constrained optimization under room layout and construction guidelines. The optimization, based on abstracted proxy geometry, proceeds by updating the layouts responding to user interactions including pulling walls forward or backward, and swapping rooms. More importantly, the layouts are adjusted to make them more amenable to subsequent construction considerations. Specifically, we introduce new constraints due to construction cost, e.g., casting walls with certain types of precast concretes to minimize cutting of prefabricated concrete slabs.

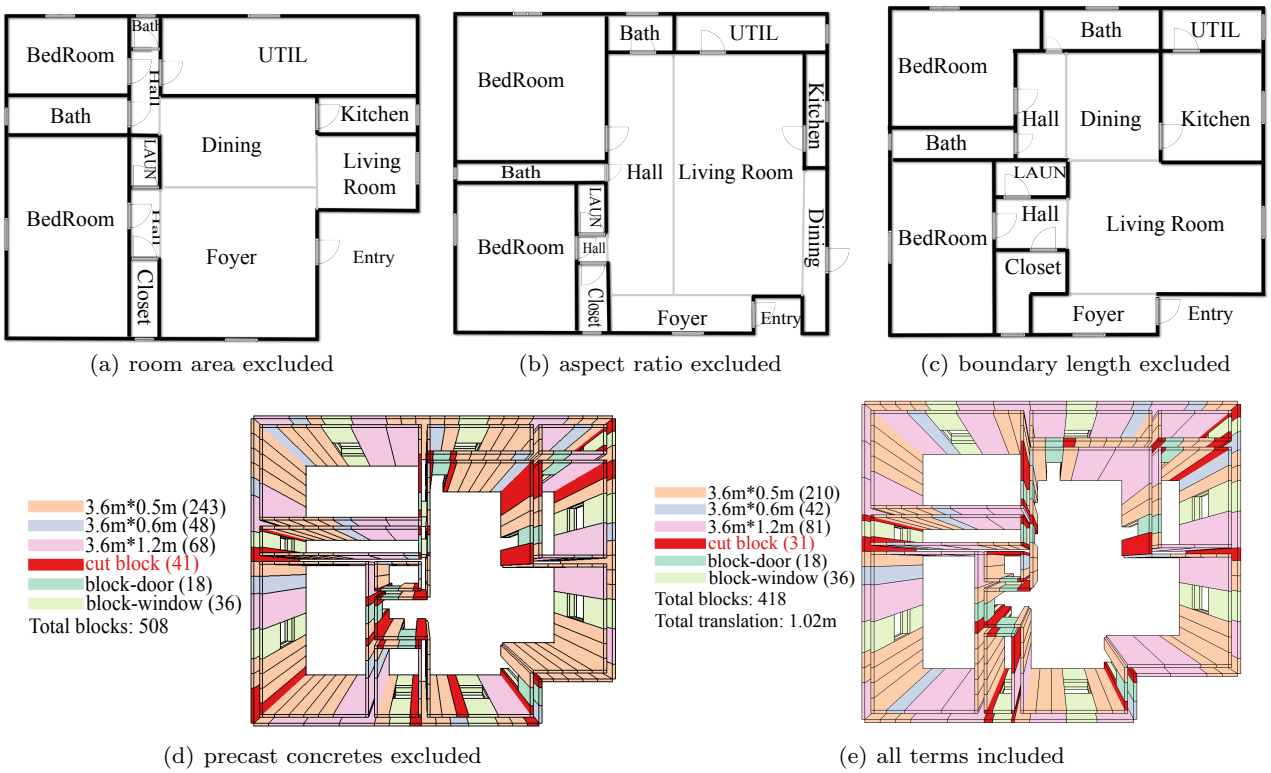
3 Overview

We introduce an interactive system to create precast concrete based building layouts satisfying design constraints. The algorithm runs in three phases: (i) A layout is initially formed by space division complying with user specifications. (ii) Users can then interactively manipulate the suggested layouts by sliding an arbitrary wall segment forward or backward as well as swapping rooms. Such interactions might cause topological changes and produce unreasonable layouts, e.g., unbalanced room size, irregular room shapes, and many corners in the building etc. We use a proxy-based optimization, followed by layout regeneration, to explore the possible layouts and generate the one that best conforms to the user requirements and design considerations. (iii) We build a model to support pre-cast concrete based construction with a given library of concrete slabs by automatically adjusting wall positions. Specifically, after casting refinement, the final layout still closely approximates the original layout produced by user interactions and optimization, and can directly be used for pre-cast concrete constructions involving minimal cutting. Figure 1 shows an example.

4 Layout Initialization

In this section, we describe the approach used to initialize the floor plan according to the design requirements. Given the dimensions of the rectangular outline region R, the user first specifies the number of rooms, approximate room areas, and the connectivity between the rooms. We encode the information as a connection graph (see Figure 1(a)), where each room is represented according to its relative room area, while the graph edges specify the room connectivity.

Based on the design requirements encoded in the graph, we generate an initial floor plan layout f_0, which decomposes the outline region into n rectangular rooms. Each room is represented by its center point $\mathbf{c}_i = (x_i, y_i)$, room length u_i, and width v_i. Note that room height is assumed to be given and fixed. Each wall segment is defined by connecting neighboring room corners. The



(a) room area excluded (b) aspect ratio excluded (c) boundary length excluded

(d) precast concretes excluded (e) all terms included

Fig. 3 The effect of individual terms in the cost function for layout optimization. To highlight the precast concrete term, we also show the layout with different types of concretes.

initial layout f_0 serves as the input for the subsequent optimization and exploration.

We use a binary partitioning strategy to iteratively divide the outline region and generate the initial floor plan. We observe that room layouts are heavily influenced by certain dominant rooms (e.g., living room) that are well connected with the other parts of an apartment. Hence, as a first step, we sort all the rooms according to their importance, which is determined by the number of their adjacent rooms (i.e., their valence in the connection graph). In case of tie, we use a predefined ordering, e.g., living room$\succ$dining$\succ$bedroom, etc.

Two key rooms $\bar{r}_1$ and $\bar{r}_2$ with maximum number of connections are selected. Then, we create two groups of rooms based on their connectivity to $\bar{r}_1$ and $\bar{r}_2$. If there exists a room r_i that is connected to both $\bar{r}_1$ and $\bar{r}_2$, we check the number of joint connected rooms between r_i against $\bar{r}_1$ and $\bar{r}_2$; and assign r_i to the group with the larger number of overlapping connections. Else if r_i is not connected to either $\bar{r}_1$ or $\bar{r}_2$, we assign it to the one with less importance. After we have two groups of rooms, we split the given region into two subregions with the area of each subregion proportional to the sum of room areas within the respective group. We introduce a horizontal split, if the length of the region is larger than its width; otherwise we introduce a vertical split. We recurse this process for each subregion until each group consists of a single room. Finally, in order to eliminate wrong connections that may have been introduced during this division process, we swap rooms with comparable areas if the operation decreases the number of missing connections.

Further, for buildings with multiple floors, we ensure the same horizontal location for the staircase. Therefore, we perform space partition on all the floors without staircases except the first floor, then slice the space for the staircases on every other floor according to the assigned location of the staircase on the first floor. Thus, at the end of this stage, we have a set of sliced rectangles abstracting an initial layout.

5 Layout Optimization

In this section, we describe the system behavior as the user interactively manipulates the layout. Starting from an initial floor plan that complies with the design specifications, the user can further interact with the suggested layout by manipulating rooms (see Figure 4(a) and supplementary video) accompanied by an optimization. Since the dimension and position of an individual room are organized as variables to be optimized, the resultant layout might have overlapping rooms or gaps

among rooms. Therefore, we employ a topology validation after layout optimization to regenerate a valid layout.

5.1 User interactions

Unlike most existing approaches that generate a fixed layout, we allow the user to evaluate and refine the suggested floor plan, while the system suggests changes to conform to the predefined constraints. We support two types of interactions to change room shapes and locations.

(i) Moving wall segments. The basic elements of a layout are its room shapes and locations. To assist the users in the layout exploration within the outline region R, we present the layout as a collection of wall segments that split the region into different rooms. The user can drag wall segments along their normals to update the area of incident rooms (see the highlighted wall in Figure 4(a)). More specifically, suppose an inner wall segment ω_i is shared by two rooms r_1 and r_2 with respective areas a_1 and a_2, the trajectory of ω_i is a rectangular area measured by α_i, then the new room areas will be $a_1' \leftarrow a_1 \pm \alpha_i$ and $a_2' \leftarrow a_2 \mp \alpha_i$. This can result in a unbalanced layout if the walls are moved arbitrarily. Therefore, in a subsequent optimization, we adjust the room locations and areas according to several quality measurements and validate the topology of the whole layout.

(ii) Swapping rooms. We also provide the user with additional design freedom to swap two rooms. Note that such a manipulation (i.e., swapping of room ids) only leads to position exchange, while the expected room areas remain unchanged. Hence, subsequently local adjustments are made via optimization to get the rooms closer to the predefined sizes. To overcome the connectivity violation due to room swaps, we restore connections using doors or open walls by searching neighboring room spaces during a post processing step.

5.2 Optimization

Next, the system optimizes the current layout to assist the users to correct implausible deviations from predefined requirements on floor plan quality and affordance considerations, as well as to facilitate real fabrications. The deviations include: (a) unusual room sizes, such as relatively small living room or a large laundry; (b) unbalanced dimensions resulting in unusually narrow room spaces; (c) irregular room outlines with too many corners; (d) inconsistent staircases locations across multiple stories of a building; and (e) lack of manufacturing considerations, e.g., requiring unnecessarily large number of cutting of pre-cast concrete slabs.

In the optimization, each room r_i is viewed as a rectangle that is represented by four variables (x_i, y_i, u_i, v_i), where x_i and y_i denote the center coordinates, u_i and v_i denote the width and length (i.e., the x-range and y-range, respectively). Currently, we only handle axis-aligned rooms in our implementation. We formulate the optimization problem by minimizing a weighted cost function (over the free variables parameterizing the current layout f):

$$C(f) := \lambda_a C_a(f) + \lambda_r C_r(f) + \lambda_b C_b(f) + \lambda_c C_c(f). \quad (1)$$

The individual terms are defined as follows while their relative weights are specified by the user. The effect of each term is illustrated in Figure 3. In our experiments, as a default all the weights $(\lambda_a, \lambda_r, \lambda_b, \lambda_c)$ are set to 1.

Room area. As the user specified the expected room areas, we try to make the room area meet the initial area requirements after the user interaction. The area cost is defined as:

$$C_a(f) = \sum_{i=1}^{n} (A_i - u_i \times v_i)^2 \quad (2)$$

where, A_i is the user specified area of room r_i.

Aspect ratio. In order to avoid skewed room shapes for bedroom, living room, etc., we optimize the aspect ratio of each room by making room width and length similar, i.e.,

$$C_r(f) = \sum_{i=1}^{n} \left(\frac{u_i - v_i}{\max(u_i, v_i)} \right)^2. \quad (3)$$

Note that the aspect ratio is not applied on the rooms mainly for spatial connectivity, such as hall, entry, porch and stair.

Boundary length. We also aim to ensure that the optimized layout makes the most use of the area bounded by the given rectangular outline. Therefore, we compare the total length of the wall segments lying on a boundary line against the length of this boundary. It is measured by:

$$C_b(f) = \sum_{i=1}^{n_b} (l_{b[i]} - \sum_{j=1}^{n_{b[i]}} l_j^i)^2 \quad (4)$$

where, n_b is the number of line segments of the outline ($n_b = 4$ in our implementation), $l_{b[i]}$ is the length of the i-th boundary line, $n_{b[i]}$ and l_j^i are the number and lengths of the wall segments associated with the i-th boundary line. Note that l_j^i can be the length or width of a certain room based on the current layout.

Precast concretes. Since only concrete slabs with restricted dimensions are preferred, our goal is to find out the optimal combination of concretes for casting each wall segment. Namely, the number of required cuts needs to be minimized to save cost in the real construction. The height of a concrete usually equals to the height of the wall of a single floor, thus we only consider the length of each wall segment. Suppose $w_1, w_2, \dots w_m$ are the widths of m concrete types and the optimal number of precast concretes for a wall segment ω_i are $n_1^i, n_2^i, \dots n_m^i$ (in Section 6, we explain how to estimate these numbers). We optimize the length of each wall segment so that it is close to the optimal length (with no cuts required)

$$C_c(f) = \sum_{i=1}^{n_\omega} (l_i - \sum_{j=1}^{m} w_j \times n_j^i)^2, \tag{5}$$

where, n_ω is the number of wall segments, l_i is the length of wall segment ω_i, which can either be the length or width of a certain room. Note that the numbers of the required concretes in the optimization are only approximately estimated and the minimization for casting concrete will be discussed in the next section.

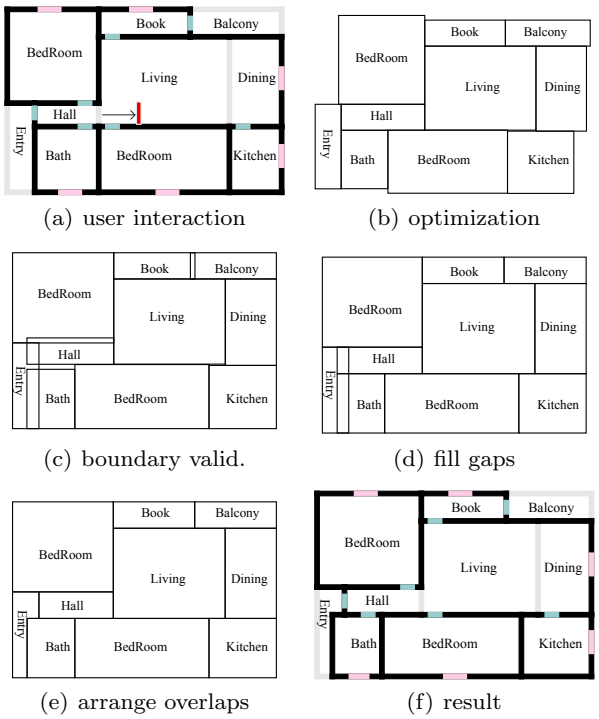
Furthermore, the combined energy function $C(f)$ is optimized subject to a set of constraints defined below.

Neighborhood consistency. For each room r_i, we want to preserve its connectivity with neighboring rooms in the layout. We identify the neighboring rooms along four canonical directions (left, right, above, below) and preserve the distance between r_i and the neighboring room. For example, if r_j is the left neighbor, the constraint is, $x_i - x_j = u_i/2 + u_j/2$, etc.

Wall length. In order to generate a valid layout within the envelope, the width/length of each room should be bounded by the horizontal/vertical boundary line respectively. In addition, to avoid that a wall segment is too short to place a door or a window, we place a lower bound on room width/length as $\max(l_b)/10$, where l_b denotes the length of a boundary line.

Staircase in multi-story buildings. For buildings with multiple floors, the staircase of each floor has to be located at the same position with similar dimensions. Therefore, after we optimize the active floor f_a edited by the user, we use its staircase as a reference to align the staircases of the other floors and adjust the rooms accordingly.

The above optimization is a non-linear least squares problem subject to linear equality and inequality constraints. We use the active set algorithm implemented by the `minbleic` package in `ALGLIB` library to solve for the optimum configuration.



(a) user interaction (b) optimization

(c) boundary valid. (d) fill gaps

(e) arrange overlaps (f) result

Fig. 4 Layout optimization via abstracted rectangular proxies, followed by layout regeneration and validation.

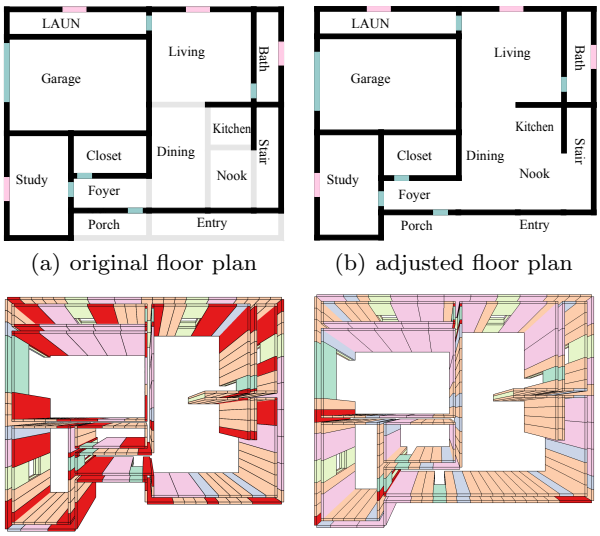
5.3 Layout regeneration

After the optimization, some rooms (abstracted by rectangular proxies) can overlap, have gaps in between, or cause irregularity at the boundary (see Figure 4(b)). Therefore, we next regenerate a valid layout by resolving the topology violations.

Irregularity at the boundary. We define a room r_i as boundary room if one of its wall segments ω_j lies on a given boundary line b_k. After the optimization, we translate r_i if necessary to make sure ω_j is still aligned with the same boundary line b_k (see Figure 4(c)).

Gaps and overlaps. To detect gaps and overlaps among the rooms, we generate a grid structure of the given outline region based on the room vertices. Each grid cell is a rectangle, denoted by g_j. Thus, there exists a gap if the center of g_j is not covered by any room. If a cell is covered by more than one room, we mark it as an overlap.

A gap is merged by snapping to the closest neighboring room. We define closeness as follows: if a room r_i and a gap g_j share a horizontal wall segment, the closeness is the difference along y-axis between their two centers. Similarly, if they share a vertical wall segment, the closeness is measured by their difference along the x-axis. To avoid unnecessarily irregular room shapes,



(a) original floor plan (b) adjusted floor plan

(c) direct casting (49 cuts) (d) optimized result (9 cuts)

Fig. 5 Our system can significantly reduce the number of cuts for precast concretes by making small layout adjustment on the wall segments as shown in bottom-right versus the unoptimized solution shown on bottom-left.

we merge parts by translating the shared wall segment of r_i so that r_i covers g_j (see Figure 4(d)). Note that the gap merging process may generate new overlaps.

An overlapping region is merged to one of the incident rooms that retains regularity as much as possible. Suppose room r_i is one of the rooms that covers g_j, if merging r_i and g_j will not increase the number of wall segments in r_i and the resultant area of r_i is closer to its expected area compared to the merging effect on the other incident rooms, then r_i will be selected to merge g_j (see Figure 4(e)). After validation, we update the room parameters according to the resultant layout (see Figure 4(f)).

6 Fabrication-aware Reshaping

Precast concretes are widely favored in building constructions due to two key advantages: (i) concrete casting happens in controlled factory environments; and (ii) the pre-cast concrete blocks are simply transported and assembled at the construction site, thus vastly simplifying casting logistics and scaffolding requirements. Also, on-site construction proceeds faster as one does not have to wait for concrete casts to solidify. The convenience, however, comes at the cost of having to tailor pre-cast concrete blocks to fit building dimensions. Specifically, if the available concrete blocks cannot fit the wall segments well, the concrete blocks have to be either redesigned or sliced leading to high manufacturing cost. We observe that slight wall resizing of the layout without sacrificing layout desirability can avoid

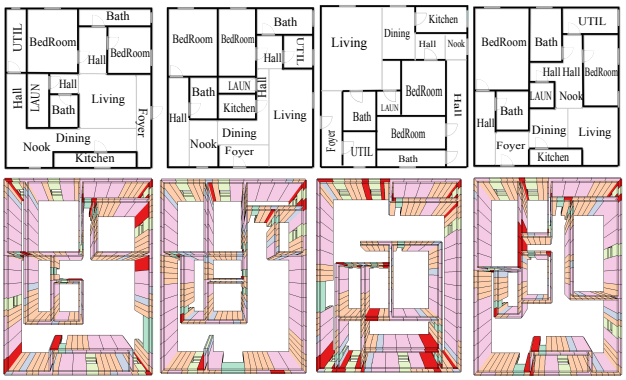


Fig. 6 Starting from a single layout (left), the user can create multiple design variations (middle-left, middle-right, right), while the system ensures that the variations respect the prescribed constraints.

such extra cost (and inconvenience). Hence, we focus on adjustments that do not cause obvious size variations in the entire design, or result in topological changes.

The concrete block can be parameterized as $h \times w \times t$, where h, w, t are the height, width, and thickness, respectively. Then, realizing a layout with pre-cast slabs is similar to solving the packing problem [10] with an additional requirement that each wall segment should be fully packed. Since the provided concretes have the same thickness in practice and the concrete height is usually determined by the wall height of a single floor, the problem amounts to solving a 1D problem and can be formulated as follows: Given a set of possible concrete slabs with different widths, how to assemble all the wall segments with minimal number of cuts by adjusting wall positions and minimizing the resultant adjustments?

We ignore the thickness of the wall segments in the layout optimization. In real construction with double wall, each wall segment is enclosed by two layers of precast concretes with the intermediate space used for insulation material. We first update the layout so that each wall segment is of a certain thickness (see Figure 7). Thus, each wall segment structure is represented by a polygon (the height is irrelevant here). The two longest parallel edges are the two layers which will be formed by precast concretes. Other edges are cross sections for conjoining neighboring walls. Let s_1 and s_2 denote the sizes of the two layers corresponding to a wall segment ω and h is the uniform height (same as the height of precast concretes). Our algorithm considers three scenarios: a single wall, a wall with doors or windows, and wall traverse. Without loss of generality, assume we have three types of concretes with the same height and thickness but different widths, denoted by w_1, w_2, and w_3.

(i) Single wall. Take the layer with size s_1 as an example, the goal is to fill it with given smaller concrete slabs of widths w_1, w_2, and w_3. In order to calculate the optimal combination of concretes leading to the minimal change requirement of the layer, we use dynamic programming to solve a classical Knapsack problem [5]. Having obtained the optimal numbers of precast concretes with different types, we compute the error between the sum of their widths s_w and s_1 and minimize it. Here $s_w = n_1 \times w_1 + n_2 \times w_2 + n_3 \times w_3$ $(n_1, n_2, n_3 \geq 0)$. After that, s_1 is updated to s_w and s_2 is similarly updated.

Although it seems the two layers of a wall segment can be cast with the same solution, the two layer sizes are actually different. A wall segment can have one, two, or three neighboring walls, which can be observed from the updated layout in Figure 7. We model four types of wall segments to facilitate concrete casting for the corresponding layers.

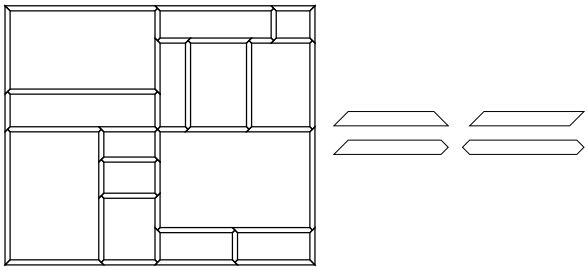


Fig. 7 The floor plan layout is updated to have certain thickness of each wall segment for casting concretes (left). Four types of wall segments with different layer configurations (right).

(ii) Wall with doors or windows. The wall segments with doors or windows need special treatment. For a given layer, we first cast the area above/below a door or a window using specialized concretes. Here, we assume all the doors have the same size, so as the windows. We solve the rest of the parts separately as presented before. Note although a global optimization can alternately be used, we preferred this greedy approach for interactive speed.

(iii) Wall traverse. After the two layers of a single wall are updated, we propagate the change to the layers of neighboring wall segments so that the orthogonality and collinearity between wall segments are preserved. We also have a flag for each wall segment so that the size of its layer is only updated once to avoid conflicts.

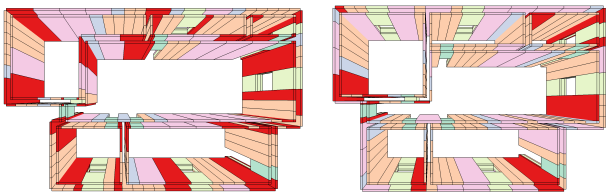
Finally, during the concrete casting, two neighboring collinear layers are preferably combined into a longer one, if applicable, since it helps to save unnecessary cuts during casting optimization. Figure 5 shows a concrete casting result for a floor plan layout of a single story. Note that although the layout does not change significantly, the number of cuts is significantly reduced.

For multi-storied buildings, we perform the above concrete casting algorithm floor by floor since the room layouts can be different for each floor, and thus can be solved independently (with only stairway consistency).

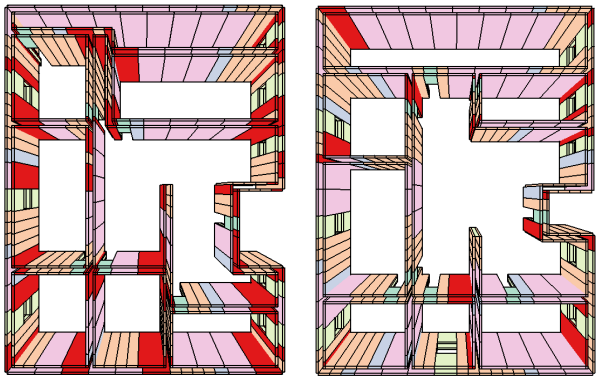
7 Results

We evaluated our interactive floor plan designer on a variety of examples with different design specifications (varying room numbers, stories, etc.). The users succeeded in interactively creating floor plan layouts under different types of constraints (see also supplementary material). These constraints ensure that the resultant layouts are both functionally plausible and optimized according to fabrication considerations.

We first evaluate single-story floor plans. Starting from the layout specifications encoded in a connected graph, the user interacts with the initial layout while the optimization automatically involves design and fabrication considerations (see Figure 1). Further, our system supports easy creation of multiple design suggestions respecting prescribed constraints. Figure 6 illus-

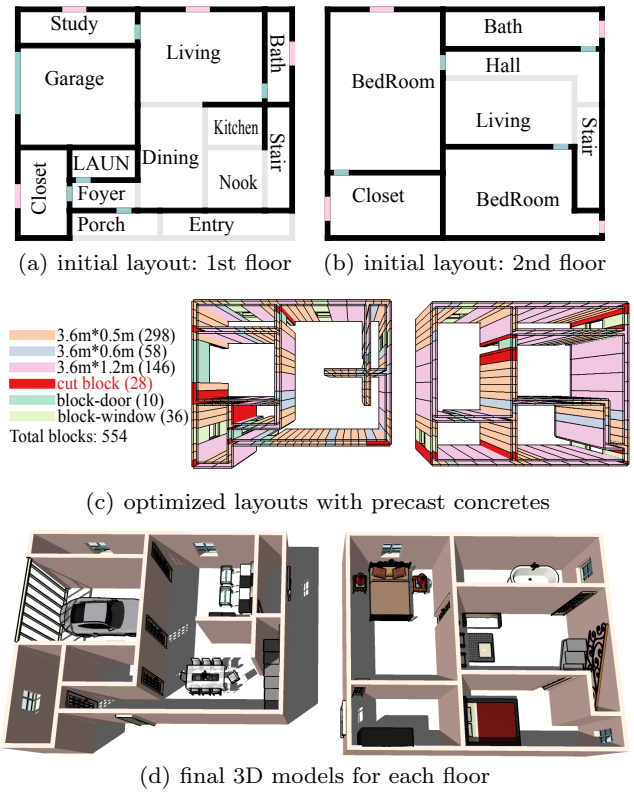


(a) floor plan #1 - without optimization (56 cuts) (b) floor plan #1 - after optimization (20 cuts)



(c) floor plan #2 - without optimization (72 cuts) (d) floor plan #2 - after optimization (26 cuts)

Fig. 8 Optimization results in cost savings due to large reductions in the required cuttings of precast slabs.

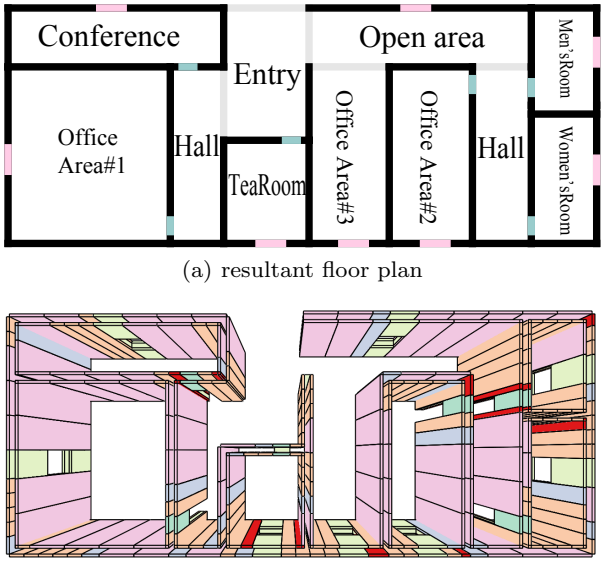


(a) initial layout: 1st floor (b) initial layout: 2nd floor

(c) optimized layouts with precast concretes

(d) final 3D models for each floor

Fig. 9 Our system also supports floor plan layout generation for multi-storied buildings.



(a) resultant floor plan

(b) precast concretes

Fig. 10 Office building layout created by our system.

trates three interactively generated layouts originated from the same initial layout.

Figure 8 compares two layouts with and without optimization and precast concrete adjustment. Note that the optimized layouts have more regular room shapes, while the number of required concrete cuts has been significantly reduced. Floor plans of multi-story buildings can also be created by our system. Figure 9 shows an example of a two-story building.

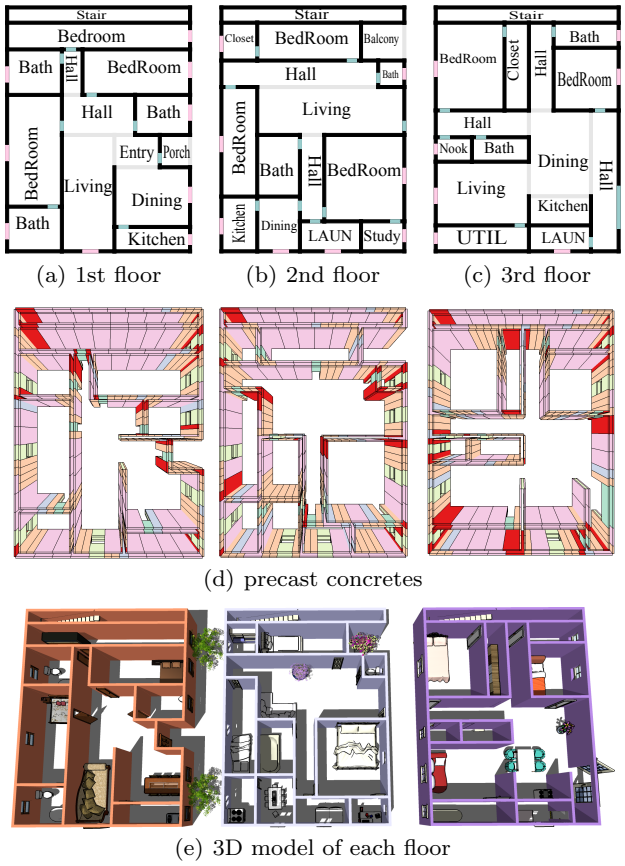
Our work can be applied to different kinds of interior layout, such as office design. Figure 10 shows an office layout generated using our interactive system, as well as the corresponding concrete casting layouts. Traditional apartment buildings have same layouts for residents on different floors. However, based on our approach, each floor can be designed in a user-specified style without additional construction cost in terms of precast concretes (note that the support structure for the floors are not considered). Figure 11 shows an example of a three-story building with different interior layouts on each floor according to different design specifications.

Performance. Our system is implemented in C# on a Windows 7 machine with Intel X5550 CPU (2.66GHz) and 4GB memory. The experimental layouts have 10-15 rooms on a single floor. The constrained optimization for a 14-room layout (Figure 3) takes around 2 seconds, and 3.5 seconds for the two-story layout in Figure 9. The layout validation and fabrication-aware reshaping are in real time (see also video). Note that in the video, to better visualize the layout validation, we inserted small time intervals (0.8 second) between individual steps.

User study. We conducted a pilot user study to evaluate our system. The participants are five students from CS department. All of them have experiences on software UI design but no experience on layout design. We provided them a single floor layout example with two bedrooms and asked them to design similar layouts. All the users could successfully create new layouts by starting from the connectivity graph representing the example layout. They all felt that the system is easy to use and appreciated the various layout design possibilities. In the user study, we also tried to disable the underlying optimization and fabrication cost minimization. The resultant design became undesirable due to irregular rooms shapes.

8 Conclusion

We presented an interactive system to create interior room layouts subject to design constraints, user preferences, and manufacturing considerations. Specifically, we focused on constructions with precast concrete slabs with the goal to minimize the number of necessary cutting of the concrete slabs. We demonstrated our system



(a) 1st floor (b) 2nd floor (c) 3rd floor

(d) precast concretes

(e) 3D model of each floor

Fig. 11 A three-storied apartment building layout with consistent stairway placement as generated using our system.

on a range of different examples and evaluated the effectiveness of the system via a pilot user study.

Our system can be extended in multiple ways. For example, a data-driven approach similar as in [13] would be very helpful to generate initial layouts for interactive exploration. Other layout design constraints can be added such as favorable position and orientation for certain rooms. Also, more expressive room proxies other than rectangles would be interesting to consider. Finally, the current framework cannot reliably handle large changes prescribed by the users, or when the constrained optimization fails to find a valid solution. In the future, we would like to explore a multi-resolution approach to address this limitation.

Acknowledgements We thank Raed Al Rabiah from Rabiah&Zamil Concrete Industries for professional suggestions on precast concrete building construction, Mohamed Shalaby for discussions, Dong-Ming Yan for discussions on gap detection and Youyi Zheng for preparing the demo program.

References

1. Arvin, S., House, D.: Modeling architectural design objectives in physically based space planning. Automation in Construction **11**(2), 213–225 (2002)
2. Dasgupta, P., Sur-Kolay, S.: Slicible rectangular graphs and their optimal floorplans. ACM Trans. on Design Automation of Electronic Systems **6**(4), 447–470 (2001)
3. Elezkurtaj, T., Franck, G.: Algorithmic support of creative architectural design. Umbau **2**, 16 (2002)
4. Galle, P.: An algorithm for exhaustive generation of building floor plans. Communications of the ACM **24**(12), 813–825 (1981)
5. Gary, M., Johnson, D.: Computers and intractability: A guide to the theory of np-completeness (1979)
6. Guo, P., Cheng, C., Yoshimura, T.: An o-tree representation of non-slicing floorplan and its applications. In: Proc. Design Automation Conf., pp. 268–273. IEEE (1999)
7. Harada, M., Witkin, A., Baraff, D.: Interactive physically-based manipulation of discrete/continuous models. In: Proc. CGIT, pp. 199–208 (1995)
8. Jiang, L., Xu, Q., Chakrabarty, K., Mak, T.: Layout-driven test-architecture design and optimization for 3d socs under pre-bond test-pin-count constraint. In: IEEE/ACM CADD, pp. 191–196 (2009)
9. Knecht, K., Koenig, R.: Generating floor plan layouts with k-d trees and evolutionary algorithms. In: Generative Art Conf., pp. 238–253 (2010)
10. Korf, R.: Optimal rectangle packing: Initial results. In: ICAPS, pp. 287–295 (2003)
11. Marson, F., Musse, S.: Automatic real-time generation of floor plans based on squarified treemaps algorithm. Int. Jnl. of Computer Games Technology **2010**, 7 (2010)
12. Medjdoub, B., Yannou, B.: Dynamic space ordering at a topological level in space planning. Artificial Intelligence in Engg. **15**(1), 47–60 (2001)
13. Merrell, P., Schkufza, E., Koltun, V.: Computer-generated residential building layouts. p. 181. ACM (2010)
14. Merrell, P., Schkufza, E., Li, Z., Agrawala, M., Koltun, V.: Interactive furniture layout using interior design guidelines. p. 87. ACM (2011)
15. Michalek, J., Choudhary, R., Papalambros, P.: Architectural layout design optimization. Engineering optimization **34**(5), 461–484 (2002)
16. Otten, R.: Automatic floorplan design. In: Proc. Design Automation Conf., pp. 261–267 (1982)
17. Pan, P., Liu, C.: Area minimization for floorplans. IEEE Computer-Aided Design of Integrated Circuits and Systems **14**(1), 123–132 (1995)
18. Preas, B., VanCleemput, W.: Placement algorithms for arbitrarily shaped blocks. In: Proc. Design Automation Conf., pp. 474–480 (1979)
19. Schneider, S., Fischer, J., König, R.: Rethinking automated layout design: Developing a creative evolutionary design method for the layout problems in architecture and urban design. DCC pp. 367–386 (2011)
20. Umetani, N., Igarashi, T., Mitra, N.J.: Guided exploration of physically valid shapes for furniture design. ACM Trans. on Graphics **31**(4), 86:1–86:11 (2012)
21. Wong, D., Liu, C.: A new algorithm for floorplan design. In: Proc. Design Automation Conf., pp. 101–107 (1986)
22. Zhou, H., Wang, J.: Acg-adjacent constraint graph for general floorplans. In: IEEE Computer Design: VLSI in Computers and Processors, pp. 572–575. IEEE (2004)