Automating Layout Synthesis with Constructive Preference Elicitation

Luca Erculiani[1], Paolo Dragone[13,⋆], Stefano Teso[2] ✉, and Andrea Passerini[1]

[1] University of Trento, Italy
{luca.erculiani,paolo.dragone,andrea.passerini}@unitn.it
[2] KU Leuven
stefano.teso@cs.kuleuven.be
[3] TIM-SKIL, Trento, Italy

Abstract. Layout synthesis refers to the problem of arranging objects subject to design preferences and structural constraints. Applications include furniture arrangement, space partitioning (e.g. subdividing a house into rooms), urban planning, and other design tasks. Computer-aided support systems are essential tools for architects and designers to produce custom, functional layouts. Existing systems, however, do not learn the designer's preferences, and therefore fail to generalize across sessions or instances. We propose addressing layout synthesis by casting it as a *constructive preference elicitation* task. Our solution employs a coactive interaction protocol, whereby the system and the designer interact by mutually improving each other's proposals. The system iteratively recommends layouts to the user, and learns the user's preferences by observing her improvements to the recommendations. We apply our system to two design tasks, furniture arrangement and space partitioning, and report promising quantitative and qualitative results on both.

Keywords: Constructive Learning, Preference Elicitation, Layout Synthesis, Furniture Arrangement, Space Partitioning.

1 Introduction

Layout synthesis refers to the problem of arranging objects in accordance to design preferences and hard constraints. It encompasses tasks like arranging furniture within a room [32, 8, 31], planning entire floors [16], and designing block- or city-sized urban spaces [19]. The constraints are meant to encode functional and structural requirements, as well as any applicable human design guideline [18, 3] (visibility, accessibility, *etc.*). In this paper we focus on the challenging problem of synthesizing layouts customized for a particular user.

Striking the right balance between personal taste and hard requirements is notoriously difficult. For instance, an apartment may be furnished in a minimalist or industrial style depending on the owner's preferences. But when furnishing it, the tenants, who often have no knowledge of interior design principles, proceed

⋆ PD is a fellow of TIM-SKIL Trento and is supported by a TIM scholarship.

intuitively, producing layouts that are not entirely functional or do not "look or feel right" [32]. Even experienced designers often rely on trial-and-error [32, 14]. Understendably, the more complex the constraints (regulatory and design guidelines, engineering requirements), the harder it is for the user to produce a functional design that she actually enjoys. Layout synthesis tools assist both expert and amateur designers in this endeavor, enhancing productivity and outcome quality.

Now, consider a customer wishing to buy an apartment from a set of alternatives. To evaluate a candidate, she could use a layout synthesis tool to "fill in" the missing furniture according to her taste. Existing tools assist the user in this task [14, 13]. However, they do not explicitly learn the user's preferences, and thus can not generalize across synthesis instances or design sessions. This implies that, in order to compare the alternative flats, the customer would have to furnish every one of them individually, a tedious and time consuming activity. In stark constrast, a tool capable of interactively learning her preferences could automatically furnish any flat based on the preferences estimated on the previous ones. To the best of our knowledge, this challenging learning problem has not been considered before.

Building on recent work on constructive learning [27, 26], we propose a *constructive preference elicitation* approach to synthesizing custom layouts. Our method leverages Coactive Learning [24], a framework for interactive preference learning from manipulative interaction intended specifically for structured domains. As in previous work [14, 13], our interaction protocol involves iteratively presenting high-scoring recommendations to the user, and then asking the latter to improve them. The user is free to perform small, large, or even multiple adjustments at each round. The algorithm automatically acquires a preference estimate from the observed improvements.

In our approach, layouts are represented as ensembles of components whose properties (position, size, *etc.*) determine the feasible space of candidates. The salient properties of the layouts are captured by features of its components (rooms, furniture) and their arrangement (e.g. the maximal distance between the bedrooms and the bathroom), whose weight is learned during the interaction process. In contrast to standard preference elicitation [20], where the set of alternatives can be easily searched over (e.g. looking up a movie in a catalogue), in layout synthesis the number of possible arrangements is large or infinite. For this reason, synthesizing a layout to be recommended to the user requires full-fledged mathematical optimization [16]. Borrowing from previous work [16, 26], we cast this synthesis step as a constrained combinatorial optimization problem, which can be readily handled by off-the-shelf solvers.

We apply the proposed method to two tasks: arranging furniture in a room and planning the layout of an apartment. In the first case, the user can interactively adjust the values of features of interest (e.g. the average distance between tables in a cafè), while in the second the user modifies the layout directly by, for instance, moving or removing walls. We empirically evaluate our algorithm on instances of increasing complexity in terms of recommendation quality and

computational cost, showing that it can effectively learn the user's preferences and recommend better layouts over time. We also apply it to larger synthesis instances and show how to tame the runtime of the synthesis step by employing approximation techniques. Our results show that, even in this case, the algorithm can reliably learn the user's preferences with only a minor loss in recommendation quality. Finally, we also positively evaluate the ability of the algorithm to deal with users with very different preferences and learn how to generate layouts that suit their taste.

2 Related work

Synthesizing a custom layout requires to solve two problems: generating a layout consistent with the known requirements and preferences (*synthesis*), and biasing the synthesis process toward layouts preferred by the user (*customization*).

Broadly speaking, synthesis can be solved in two ways. One is to design a parameterized distribution over layouts (e.g. a probabilistic graphical model [31] or a probabilistic grammar [11]), whose structure encodes the set of validity constraints on objects and arrangements. Synthesis equates to sampling from the distribution via MCMC. A major downside of probabilistic approaches is that enforcing hard constraints (other than those implied by the structure of the distribution) may severely degrade the performance of the sampler, potentially compromising convergence, as discussed for instance in [27].

An alternative strategy, adopted by our method, is to define a scoring function that ranks candidate layouts based on the arrangement of their constituents. In this case, synthesis amounts to finding a high-scoring layout subject to design and feasibility constraints. This optimization problem may be solved using stochastic local search [32, 1], mathematical optimization [14], or constraint programming [16]. We opted for the latter: constraint programming [22] allows to easily encode expressive local and global constraints, and is supported by many efficient off-the-shelf solvers. Further, in many cases it is easy to instruct the solver to look for (reasonably) sub-optimal solutions, allowing to trade-off solution quality for runtime, for enhanced scalability. This synergizes with our learning method, which is robust against approximations, both theoretically [23] and experimentally (see Section 4).

Many of the existing tools are concerned with synthesis only, and do not include a customization step: their main goal is to automate procedural generation of realistic-looking scenes or to produce concrete examples for simplifying requirement acquisition from customers [30]. Other approaches bias the underlying model (distribution or scoring function) toward "good" layouts by fitting it on sensibly furnished examples [31, 13, 32]. However, the generated configurations are not customized for the target user[1]. More generally, offline model

[1] While the examples may be provided by the end user, it is unreasonable to expect the latter to manually select the large number of examples required for fine-grained model estimation. Through interaction, our system allows a more direct control over the end result.

estimation may be used in conjunction with our method to accelerate layout fine-tuning for the end user. We will explore this possibility in a future work.

Akase and colleagues proposed two interactive methods based on iterative evolutionary optimization [1, 2]. Upon seeing a candidate furniture arrangement, the user can tweak the fitness function either directly using sliders [1] or indirectly via conjoint analysis [2]. In both works the number of customizable parameters is small and not directly related to the scene composition (e.g. illumination, furniture crowdedness). Contrary to these methods, we enable the user to graphically or physically manipulate a proposed layout to produce an improved one. This kind of interaction was successfully employed by a number of systems [28, 14, 13] using ideas from direct manipulation interfaces [25, 10]. We stress that our method works even if user improvements are small, as shown by our empirical tests. The major difference to the work of Akase *et al.*, however, is that we leverage constraint programming rather than generic evolutionary optimization algorithms. This enables our method naturally handle arbitrary feasibility constraints on the synthesized layouts, extending its applicability to a variety of layout synthesis settings.

Among interactive methods, the one of Merrell *et al.* [13] is the closest to ours. Both methods rely on a scoring function, and both require the user and system to interact by suggesting modifications to each other. In contrast to our approach, the method of Merrel *et al.* does not learn the scoring function in response to the user suggestions, i.e., it will always suggest configurations in line with fixed design guidelines. Since no user model is learned, this method does not allow to transfer information across distinct design session.

3 Coactive Learning for Automated Layout Synthesis

In this section we frame custom layout synthesis as a *constructive preference elicitation* task. In constructive preference elicitation [7], the candidate objects are complex configurations composed of multiple components and subject to feasibility constraints. Choosing a recommendation amounts to synthesizing a novel configuration that suits the user's preferences and satisfies all the feasibility constraints. In this setting, learning can be cast as an interactive *structured-output prediction* problem [4]. The goal of the system is to learn a *utility* function u over objects y that mimics the user's preferences. The higher the utility, the better the object. In layout synthesis, the objects y may represent, for instance, the positions of all furniture pieces in a given room or the positions and shapes of all rooms on a given floor. The utility can optionally depend on extra contextual information x, e.g., the size and shape of the target apartment. More formally, the utility is a function $u : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of all the contexts and objects respectively. Typically, the utility is assumed to be a linear model of the type: $u(x, y) = \langle \boldsymbol{w}, \boldsymbol{\phi}(x, y) \rangle$. Here the *feature function* $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$ maps an object to a vector of d features summarizing its high-level properties. For instance, in furniture arrangement the features might capture the maximum distance of the furniture pieces from the walls or the minimum distance between

1 **Procedure** `PreferencePerceptron`(T)
2 Initialize $\boldsymbol{w}_1$
3 **for** $t = 1, \ldots, T$ **do**
4 Receive user context x_t
5 $y_t \leftarrow \text{argmax}_{y \in \mathcal{Y}} \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y) \rangle$
6 Receive improvement $\bar{y}_t$ from the user
7 $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \boldsymbol{\phi}(x_t, \bar{y}_t) - \boldsymbol{\phi}(x_t, y_t)$
8 **end**
9 **end**

Algorithm 1: The Preference Perceptron from Coactive Learning [24].

each other. The weights $\boldsymbol{w} \in \mathbb{R}^d$ associated to the features represent the user preferences and are estimated by interacting with the user.

In our approach, the learning step is based on Coactive Learning [24], whereby the system iteratively proposes recommendations y to the user and the user returns an improvement $\bar{y}$, i.e. a new object with a (even slightly) higher utility than y. Using this information, the algorithm updates the current estimate of the parameters $\boldsymbol{w}$ in order to make better recommendations in the future.

Perhaps the simplest, yet effective, Coactive Learning algorithm is the Preference Perceptron [24], listed in Algorithm 1. The algorithm iteratively elicits the user preferences by making recommendations and getting user feedback in response. Across the iterations $t \in [1, T]$, the algorithm keeps an estimate of the parameters $\boldsymbol{w}_t$ which are continuously updated as new feedback is observed. At each iteration t, the algorithm first receives the context x_t and then produces a new object y_t by maximizing the current utility estimate $y_t = \text{argmax}_{y \in \mathcal{Y}} u_t(x_t, y) = \langle \boldsymbol{w}_t, \boldsymbol{\phi}(x_t, y) \rangle$ (line 5). The object y_t is recommended to the user, who then provides an improvement $\bar{y}_t$. Lastly, the algorithm obtains a new parameter vector $\boldsymbol{w}_{t+1}$ using a simple perceptron update (line 7).

Depending on the type of objects $y \in \mathcal{Y}$, inference (line 5) may be solved in different ways. In layout synthesis, the objects $y \in \mathcal{Y}$ are sets of variable assignments representing the components of the layouts. For instance, in furniture arrangement y may contain the position of each piece of furniture, its size, type, color, and so on. The space of potential configurations (i.e. possible assignments of attributes) is combinatorial or even infinite (for continuous variables) in the number of attributes. The space $\mathcal{Y}$ is also typically restricted by feasibility constraints, enforcing functional and structural requirements, e.g. non-overlap between furniture pieces. In this setting, the inference problem can be formalized in the general case as a mixed integer program (MIP). In practice, to make inference computationally feasible, we often restrict it to the mixed integer *linear* program (MILP) case by imposing that constraints and features be linear in y. Exploring in which cases more general constrained optimization problems (e.g. mixed integer quadratic programs) are computationally feasible is an interesting future direction (as outlined in the Conclusion). While being NP-hard in the general case, MILP problems with hundreds of variables can be quickly solved to optimality by existing off-the-shelf solvers. For problems that

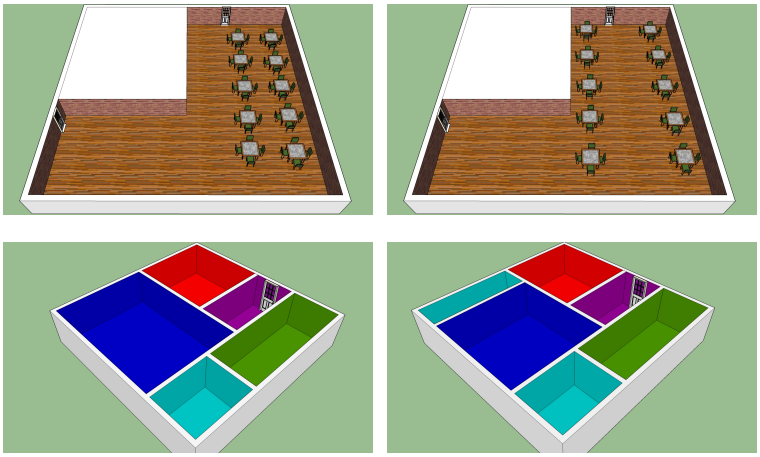

Fig. 1. Illustration of feature- and object-level improvements. Left: recommended configuration. Right: corresponding user-provided improvement. The two top images illustrate a *feature-level* improvement where the user increased the minimum distance between tables (a feature) directly, affecting all tables. The bottom images show an *object-level* improvement where the user modified the object itself by adding a new wall. Best viewed in color.

are too complex for exact solution strategies, approximation techniques can be used to speed-up the inference process. Reasonably sub-optimal synthesized layouts do not significantly alter the performance of Coactive Learning, as proven theoretically in [21] and shown empirically by our experiments.

The Coactive Learning paradigm can be seen as a cooperation between the learning system and the user to mutually improve each other's design proposals. This is particularly appealing for devising a layout synthesis system, since coactive feedback may be acquired through visual or physical manipulation of the recommended objects (see e.g. [28, 14, 13]). Such a system could be integrated into graphical design applications and used by architects and designers to automatically improve their products. Depending on the type and complexity of the design task, a visual layout recommendation system may also be used by an amateur designer.

4 Experiments

In this section we evaluate the proposed system on two layout synthesis applications. The first is a furniture arrangement problem in which the system recommends arrangements of tables in a room, for furnishing, e.g., bars versus offices. The second is a space partitioning problem in which the system suggests how to partition the area of an apartment into rooms. In our empirical analysis, we [i] perform a quantitative evaluation of the system's ability to learn the user

preferences, with an emphasis on the trade-off between exact and approximate synthesis; and [ii] perform a qualitative evaluation by illustrating the layouts recommended at different stages of the learning process.

In our experiments we simulate a user's behavior according to standard feedback models [24]. Namely, we assume that the user judges objects according to her own true utility[2] $u^*(x, y) = \langle \boldsymbol{w}^*, \boldsymbol{\phi}(x, y) \rangle$, and measure the quality of recommendations by their *regret*. The regret at iteration t is simply the difference between the true utility of the best possible object and that of the actual recommendation y_t, that is, $\mathrm{REG}(x_t, y_t) = (\max_{y \in \mathcal{Y}} u^*(x_t, y)) - u^*(x_t, y_t)$. Since varying contexts makes comparing regrets at different iterations impossible [24], in the experiments we measure the *average regret* $\frac{1}{T} \sum_{t=1}^{T} \mathrm{REG}(x_t, y_t)$.

We observe that, very intuitively, good recommendations can't be improved by much, while bad recommendations offer ample room for improvement. For this reason, we assume the user's feedback to follow the α-informative feedback model [24], which states that the true utility of $\bar{y}_t$ is larger than the true utility of y_t by some fraction $\alpha \in (0, 1]$ of the regret $\mathrm{REG}(x_t, y_t)$, that is $u^*(x_t, \bar{y}_t) - u^*(x_t, y_t) \geq \alpha \mathrm{REG}(x_t, y_t)$. Here α represents the user "expertise". Higher values of α imply better improvements and fewer iterations for learning to suggest good (or optimal) recommendations. Indeed, if α-informative holds the average regret of the Preference Perceptron decreases at a rate of $\mathcal{O}(1/(\alpha\sqrt{T}))$ (see [24]).

Crucially, the α-informative model is very general, since it is always possible to find an α, no matter how small, that satisfies the feedback model[3]. In our experiments we assume that no user expertise is required to interact usefully with our system, and thus set $\alpha = 0.1$ to simulate a non-expert user. Furthermore, we assume that changes made by the user are small, in order to keep her effort to a minimum. We thus take a conservative approach and simulate the user behavior by selecting a "minimal" α-informative improvement (more details below).

For both settings, in the quantitative experiment we compare the average regret of the system on instances of increasing complexity. As the complexity increases, approximate solutions become necessary for guaranteeing real-time interaction. We perform approximate synthesis by setting a time cut-off on the solver and choosing the best solution found in that time. Regret bounds similar to the aforementioned one can be found also in approximate settings [21]. We evaluate empirically the effect of using approximate inference on the quality of the recommendations in both settings. For the quantitative experiments we simulated 20 users with randomly generated true parameter vectors $\boldsymbol{w}^*$ and plotted the median performance.

We consider two types of user improvements, as exemplified in Figure 1. In the first experiment we use a feature-based improvement, in which a user may variate the value of a feature (e.g. with a simple set of UI controllers) to generate a better configuration. In the top example of Figure 1, the user sets

[2] Note that the learner can observe the user's feedback, but has no access to the true utility itself.

[3] Assuming the user makes no mistakes, i.e., $u^*(x_t, \bar{y}_t) > u^*(x_t, y_t) \forall t \in [T]$. This model can be easily extended to allow user mistakes (see [24]).

the minimum distance between the tables to a higher value. The second type of improvement considered is an object-based improvement, in which the user directly shapes the configuration by adding, moving or removing parts. This is the case of the bottom example of Figure 1, in which the user adds a wall to create a new room. The details of the user feedback simulation models are reported in the corresponding subsections. In both experimental settings, we also report a qualitative evaluation showcasing the behavior of the system in interacting with some "prototypical" type of user (e.g. a cafè owner arranging the tables in her place). We show that the system achieves the goal of finding good configurations matching the user taste.

The system is implemented in Python[4] and uses MiniZinc to model the constrained optimization problems [17], and an external MILP solver[5] for the inference and the improvement problems. All the experiments were run on a 2.8 GHz Intel Xeon CPU with 8 cores and 32 GiB of RAM.

4.1 Furniture arrangement

In the first experimental setting, the goal of the system is to learn to arrange tables in a room according to the user preferences. Rooms are 2D spaces of different shapes. We model the rooms as squared bounding boxes, plus several inaccessible areas making up internal walls. The size of the bounding box and the inaccessible areas are given in the context x, together with the number of tables to place. The available space is discretized into unit squares of fixed size. Tables are rectangles of different shape occupying one or more unit squares. The objects y consist in the table arrangements in the given room. More precisely, tables are represented by their bottom-left coordinates (h, v) in the bounding box and their sizes (dh, dv) in horizontal and vertical directions. The object y contains the coordinates (h_t, v_t) and the sizes (dh_t, dv_t) of each table t. Several constraints are imposed to define the feasible configurations. Tables are constrained to fit all in the bounding box, to not overlap, and to not be positioned in unfeasible areas. Tables must keep a minimum "walking" distance between each other. Doors are also placed on the room walls (in the context) and tables are required to keep a minimum distance from the doors.

In our experiment the total size of the bounding box is 12×12. Tables are either 1×1 squares (occupying one unit square) or 1×2 rectangles (occupying two unit squares). Room shapes were selected randomly at each iteration from a pool of five candidates.

The feature vector $\boldsymbol{\phi}(x, y)$ is composed of several numeric properties of the configuration, such as the maximum and minimum distance between tables, the maximum and minimum distance between tables and walls, and the number of tables per type (1×1 and 1×2). The upper part of Table 1 contains a detailed summary of the structure of x, y and $\boldsymbol{\phi}(x, y)$ in this setting.

[4] See `https://github.com/unitn-sml/constructive-layout-synthesis` for the complete implementation.

[5] Opturion CPX: `http://opturion.com`

Furniture arrangement	
Context x	- Size of bounding box - Position of doors - Inaccessible areas - Number of tables
Object y	- Position (h, v) of all tables - Sizes (dh, dv) of all tables
Features $\boldsymbol{\phi}(x, y)$	- Max and min distance of tables from bounding box: $\max_{t \in Tables} bbdist(t)$; $\min_{t \in Tables} bbdist(t)$ - Max and min distance of tables from inaccessible areas: $\max_{t \in Tables} wdist(t)$; $\min_{t \in Tables} wdist(t)$ - Max and min distance between tables: $\max_{t_1, t_2 \in Tables} dist(t_1, t_2)$; $\min_{t_1, t_2 \in Tables} dist(t_1, t_2)$ - Number of tables per type (1×1 and 1×2): $\lvert\{t \in Tables \mid dh_t + dv_t \leq 2\}\rvert$; $\lvert\{t \in Tables \mid dh_t + dv_t \geq 3\}\rvert$

Floor planning	
Context x	- Size of bounding box - Position of entrance door - Inaccessible areas - Max and min rooms per type
Object y	- Position (h, v) of all rooms - Type t_r of each room r - Sizes (dh, dv) of all rooms
Features $\boldsymbol{\phi}(x, y)$	- Ranges of occupied space (percent) per room type: $\forall t \in Types \;\; \sum_{r \in R_t} A_r \leq 15\%$ $\forall t \in Types \;\; 15\% < \sum_{r \in R_t} A_r \leq 30\%$ $\forall t \in Types \;\; \sum_{r \in R_t} A_r > 30\%$ - Upper bound D_r of difference of sides for each room r: $\forall r \in Rooms \;\; D_r$ s.t. $\lvert dh_r - dv_r \rvert \leq D_r$ - Number of rooms per type: $\forall t \in Types \;\; \lvert R_t \rvert$ - Room with entrance door r_{door} is of type t: $\forall t \in Types \;\; t = type(r_{\text{door}})$ - Sum of pairwise difference of room areas per type: $\forall t \in Types \;\; \sum_{i,j \in R_t} \lvert A_i - A_j \rvert$ - Number of rooms adjacent to corridors: $\lvert\{r \in Rooms \mid \exists s \in R_{\text{corridor}} \; adj(r, s)\}\rvert$ - Distance of each room from South (bottom edge): $\forall r \in Rooms \;\; sdist(r)$

Table 1. Summary of the structure of the objects in the two experimental settings. In the furniture arrangement task, *Tables* is the set of tables, $bbdist(t)$ is the distance of table t from the bounding box, $wdist(t)$ is the distance of table t from the inaccessible areas (walls), $dist(t_1, t_2)$ is distance between tables t_1 and t_2. In the floor planning setting, instead, *Types* is the set of room types, *Rooms* is the set of rooms, R_t the set of rooms of type t, A_r the area of room r (number of unit squares), dh_r and dv_r the horizontal and vertical size of room r, $type(r)$ the type of room r, $adj(r, s)$ is a boolean function denoting the adjacency between rooms r and s, $sdist(r)$ is the distance between r and the south edge of the bounding box. All distances considered here are Manhattan distances.

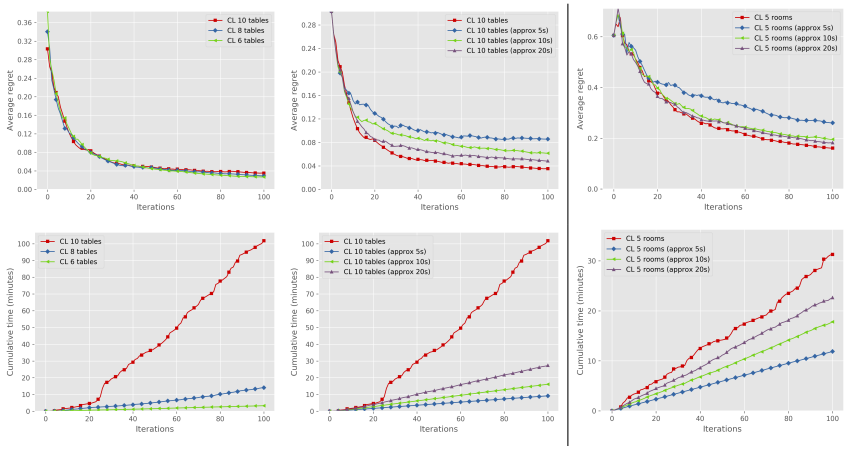

Fig. 2. Median average regret (top) and median cumulative time (bottom) in three settings. Left to right: furniture arrangement with exact inference on 6, 8 and 10 tables; comparison between exact and approximate inference in furniture arrangement with 10 tables; exact versus approximate inference on floor planning. Best viewed in color.

As mentioned, in this setting we employ a feature-based improvement scheme to simulate the user behavior. In particular, the following constrained problem is solved to generate an α-informative improvement $\bar{y}_t$:

$$\begin{aligned} \bar{y}_t = &\operatorname*{argmin}_{y \in \mathcal{Y}} \|\boldsymbol{\phi}(x_t, y) - \boldsymbol{\phi}(x_t, y_t)\|_0 \\ \text{s.t.} \quad & u^*(x_t, \bar{y}_t) - u^*(x_t, y_t) \geq \alpha(u^*(x_t, y_t^*) - u^*(x_t, y_t)) \end{aligned}$$

where $\|\boldsymbol{\phi}(x_t, y) - \boldsymbol{\phi}(x_t, y_t)\|_0$ is the ℓ_0 norm of the difference between the feature vectors, i.e. the number of different features between $\boldsymbol{\phi}(x_t, y)$ and $\boldsymbol{\phi}(x_t, y_t)$. This is in line with the assumption made on the minimal user effort.

In the quantitative evaluation we run the recommendation algorithm for an increasing number of tables to be placed. A high number of tables makes the inference problem more complex, as it involves more optimization variables and constraints. We test the algorithm on problems with 6, 8 and 10 tables. We compare the average regret and the running time of the system in each of these scenarios. Figure 2 shows the median results (over all users) on settings with different number of tables. The plots show the median average regret (top) and the median cumulative inference time (bottom). The first column of Figure 2 shows the results for the table arrangement task with exact inference on problems with different numbers of tables. Using exact inference, the difference in regret decay between different levels of complexity is minimal. This means that when the system is able to solve the inference problem to optimality, the complexity of the problem does not affect much the performance of the system. Inference time, however, increases drastically with the increasing complexity. Exact inference in the 10 tables setting is already largely impractical for an interactive system.

The second column of Figure 2 shows a comparison of the results of exact and approximate inference on the furniture arrangement setting with 10 tables, for time cut-offs at 5, 10 and 20 seconds[6]. When using approximate inference, the running times drop to a much lower rate, while the regret suffers a slight increase but keeps decreasing at a similar pace as the exact variant. We can see that the time cut-off can be modulated to achieve the desired balance between recommendation quality and inference time. This is a promising behaviour suggesting that the method can scale with the problem size with predictable running time without compromising performance.

In order to get a visual grasp of the quality of the recommended solutions, we also evaluated our system on two propotypical arrangement problems, namely a user interested in furnishing a café and one willlling to furnish an office. Cafés are usually furnished with small tables (1×1), positioned along the walls in a regular fashion. Offices, instead, contain mostly desks (1×2) positioned along the walls or in rows/columns across the room. We sampled two users according to the above criteria. Figure 3 showcases the recommendations of the system at different stages of the learning procedure. Initially, tables are randomly spread across the room. Gradually, the system learns to position tables in a more meaningful way. In the case of the café, the intermediate image shows that the algorithm has learned that a café should have mostly 1×1 tables and that they should be placed along the walls. For the office, the intermediate figure shows that the algorithm has roughly figured out the position of tables, but not their correct type. At the end of the elicitation, the final configurations match the user desiderata.

4.2 Floor planning

Our second experimental setting is on floor planning, that is recommending partitionings of apartments into separate rooms. The outer shape of the apartment is provided by the context, while the user and the system cooperate on the placement of the inner walls defining the room boundaries. As in the previous setting, the space is discretized into unit squares. Each room is a rectangle described by four variables: (h, v) indicate its position, (dh, dv) its size. Coordinates and sizes are measured in unit squares. Rooms must fit in the apartment and must not overlap. Rooms can be of one among five types, namely kitchen, living room, bedroom, bathroom and corridor. In the context, the user can also specify an upper and lower bound on the number of rooms of each type. For instance, a user may look for an apartment with exactly one kitchen, one living room, and between one and two bathrooms and bedrooms. After placing all the rooms, the spaces left in the apartment are considered corridors. The context also specifies the position of the entrance door to the apartment.

In this experiment we consider a 10×10 bounding box. We define five different apartment shapes and generate random contexts with any combination of room types, summing to a maximum of five rooms, with random lower bounds.

[6] The time cut-off is on the solver time, but the actual inference time has some more computational overhead, taking on average 2.21 seconds more.

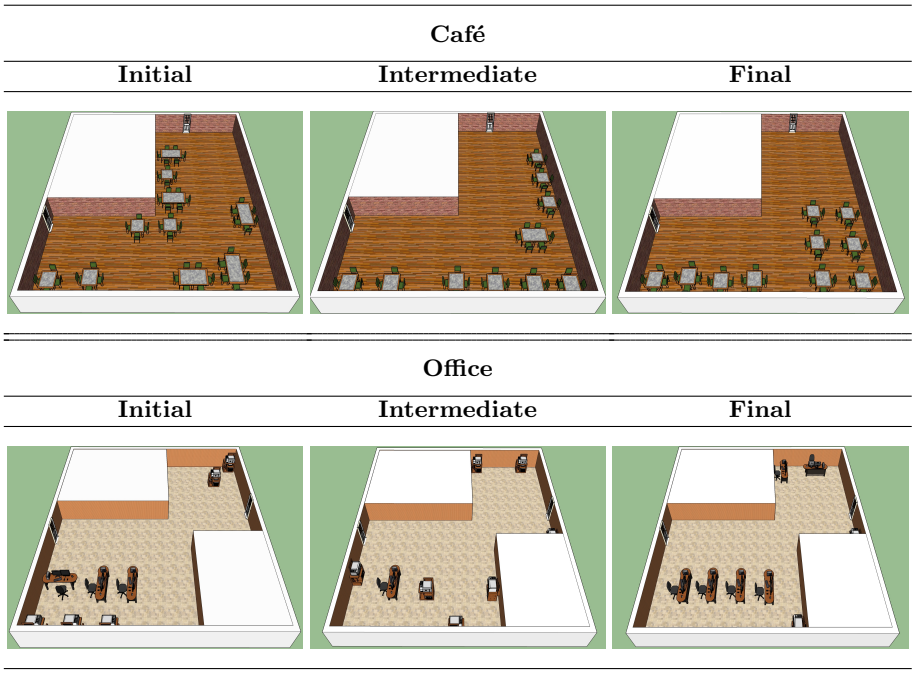

Fig. 3. Two use cases of our system. The images are 3D renderings of configurations recommended by our system when interacting with users whose goal is to furnish a café (top) and an office (bottom). Horizontally, the figures show different stages of the elicitation process. In the café, 1×1 and 1×2 tables are seen as dining tables of different sizes, whereas in the office 1×2 tables represent desks while 1×1 tables contain utilities such as printers. Best viewed in colors.

While the number of generated rooms is variable, we impose a maximum. The dimension of feature vector $\boldsymbol{\phi}(x, y)$ depends on the maximum number of rooms, hence it must be fixed in advance and cannot change throughout the iterations. Features are normalized in order to generalize different contexts and different numbers of rooms. The features include: [i] the percentage of space occupied by the rooms of each type, discretized in several ranges of values, each denoting a certain target size for each room type; [ii] an upper-bound on the difference between the sides dh_r and dv_r of each room r, which is used to modulate how "squared" the room should be; [iii] the actual number of rooms per type; [iv] a boolean value for each room type indicating whether the entrance door is in a room of that type; [v] the sum of the pairwise difference between the areas of rooms of the same type, to modulate how similar in size rooms of a certain type should be; [vi] the number of rooms that are adjacent to corridors; [vii] the distance of each room from the south border of the apartment, as living rooms are usually made to look south and bedrooms look north for lighting purposes. A summary of all the features of this setting is listed in the lower part of Table 1.

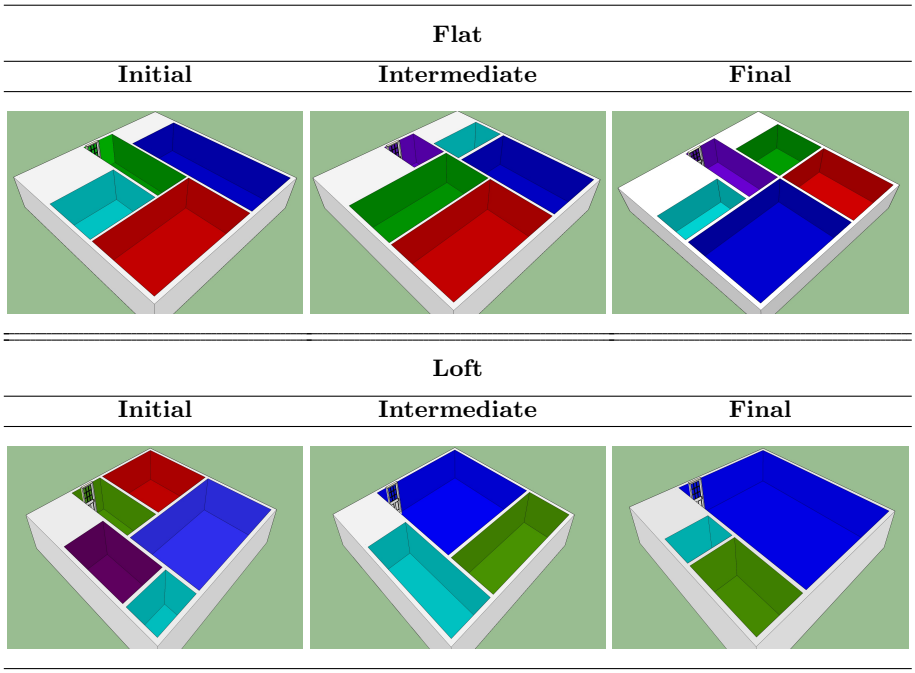

Fig. 4. Two use cases of our system for the task of floor planning. The images are 3D renderings of configurations recommended by our system when interacting with users whose goal is to build a flat (top) and an loft (bottom). Horizontally, the figures show different stages of the elicitation process. Room colors are associated to room types: the kitchen is in red, the living room is in blue, the bathroom is in turquoise, the bedroom in green, the corridor is in violet. Best viewed in colors.

Differently from the previous setting, here we employ an object-based improvement schema. We simulate the user feedback by solving the following optimization problem:

$$\bar{y}_t = \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \ \|U_y - U_{y_t}\|_0$$
$$\text{s.t.} \quad u^*(x_t, \bar{y}_t) \geq u^*(x_t, y_t) + \alpha(u^*(x_t, y_t^*) - u^*(x_t, y_t))$$

where U_y is the matrix 10×10 containing the room types per unit square. We assume a user to perform a change that involves as least rooms as possible and we simulate this behavior by minimizing the number of unit squares affected by the improvement. This is done to simulate a minimal effort for the user.

In this case, the problem complexity is mainly given by the maximum number of rooms to be placed in the apartment. Notice that this problem is more difficult than the previous one, as it has more optimization variables, more features and it has to learn from a more diverse set of possible contexts. We evaluate this setting only on a scenario with a maximum of five rooms. As in the pre-

vious experiment, we report a comparison of the results of exact inference and approximate inference. We again run approximate inference with time cut-offs at 5, 10, and 20 seconds. The last column of Figure 2 shows the median average regret and the median cumulative inference time in this setting. Both regret and times follow the same trend as the ones in the previous experiment. Approximate inference allows for substantial computational savings[7] at the cost of a small reduction in recommendation quality.

In the qualitative experiment we compare two users who are interested in different kinds of apartments. In the first case, the user is interested in a "traditional" apartment (here dubbed "flat" to avoid ambiguities), which contains a corridor from the entrance door to the rooms, two separate rooms for the kitchen and living room, with the former slightly smaller that the latter, a bedroom and a bathroom. The second user is interested in a loft, which is composed by fewer rooms, usually a big living room with a kitchenette, a bedroom and a bathroom. In Figure 4 we can see different stages of the learning process for both users. At the beginning the recommended configurations are random. The system then is able to learn that a flat should have a corridor as entrance and a smaller bathroom, and that a loft should have only a moderately large living room plus a bedroom and a bathroom of approximately equal size. Finally the system reaches good recommendations that meet the preferences of the users: an apartment with a a kitchen smaller than the living room and a corridor connecting rooms, and a loft with a big living room, a bedroom and a small bathroom.

5 Conclusion

We presented an approach to layout synthesis suitable for suggesting layouts customized for a particular user. We cast layout synthesis as a constructive preference elicitation problem, where the set of potential arrangements is determined by hard constraints encoding the functional and structural requirements. Contrary to previous solutions, our approach learns the user's preferences and thus generalizes across synthesis instances and design sessions. Our interactive learning strategy pairs a very natural interactive protocol based on direct manipulation with a simple but principled learning framework for manipulative feedback [24]. We applied our system to two design tasks, namely furniture arrangement and floor planning, and evaluated it on instances of increasing complexity. The results show that our approach can reliably learn the user's preferences even when synthesis is (moderately) sub-optimal, for improved scalability, at the cost of a minor degradation of recommendation quality. We showcased the flexibilty of our system by learning from users with radically different preferences, e.g., users that prefer lofts to highly partitioned apartments and vice-versa.

This work can be extended in several directions. First, it makes sense to bias the learner toward "known working" layouts, as done in [31, 13, 32], to accelerate convergence towards promising candidates and reduce the amount of user

[7] Exact inference becomes impractical for more that five rooms.

intervention. Second, although our presentation focuses on linear constraints and features (which yield a MILP synthesis problem) our learning procedure is not restricted to this setup. As solver technology for mixed integer quadratic problems matures[8], it becomes progressively more viable to employ non-linear terms to model even richer layout properties, such as surface areas, Euclidean distances, and variances. The increased computational requirements could be handled with appropriate quality-runtime trade-offs, as done in our experiments. Third, decomposition strategies like those presented in [6] offer a promising direction for aggressively reducing the cognitive and computational costs of layout synthesis. This is especially fitting for inherently modular layouts such as buildings, which can be decomposed into progressively simpler parts (floors, rooms, *etc.*). This would also facilitate the introduction of more computationally demanding features, as hinted to above, by restricting inference to portions of layouts. We are actively investigating these research directions. Finally, the proposed method can in principle be employed to automate tasks other than layout synthesis, like environmental [12] and chemical engineering [29] and synthetic biology [9].

Acknowledgments This work has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. [694980] SYNTH: Synthesising Inductive Data Models).

References

1. Akase, R., Okada, Y.: Automatic 3d furniture layout based on interactive evolutionary computation. In: CISIS. pp. 726–731. IEEE (2013)
2. Akase, R., Okada, Y.: Web-based multiuser 3d room layout system using interactive evolutionary computation with conjoint analysis. In: VINCI. p. 178. ACM (2014)
3. Alexander, C.: A pattern language: towns, buildings, construction (1977)
4. Bakir, G.H., Hofmann, T., Schölkopf, B., Smola, A.J., Taskar, B., Vishwanathan, S.V.N.: Predicting Structured Data (2007)
5. Boyd, S., Kim, S.J., Vandenberghe, L., Hassibi, A.: A tutorial on geometric programming. Optimization and engineering 8(1), 67 (2007)
6. Dragone, P., Teso, S., Kumar, M., Passerini, A.: Decomposition strategies for constructive preference elicitation. In: AAAI (2018)
7. Dragone, P., Teso, S., Passerini, A.: Constructive preference elicitation. Frontiers in Robotics and AI 4, 71 (2017)
8. Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., Hanrahan, P.: Example-based synthesis of 3d object arrangements. ACM TOG 31(6), 135 (2012)
9. Galdzicki, M., et al.: The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. Nature biotechnology 32(6), 545 (2014)
10. Harada, M., Witkin, A., Baraff, D.: Interactive physically-based manipulation of discrete/continuous models. In: SIGGRAPH. pp. 199–208. ACM (1995)

[8] Especially for expressive but restricted fragments such as mixed integer second order cone programming [33, 15] and mixed integer geometric programming [5].

11. Liu, T., Chaudhuri, S., Kim, V.G., Huang, Q., Mitra, N.J., Funkhouser, T.: Creating consistent scene graphs using a probabilistic grammar. ACM Transactions on Graphics (TOG) 33(6), 211 (2014)
12. Masters, G.M., Ela, W.P.: Introduction to environmental engineering and science, vol. 3 (1991)
13. Merrell, P., Schkufza, E., Li, Z., Agrawala, M., Koltun, V.: Interactive furniture layout using interior design guidelines. In: ACM Transactions on Graphics (TOG). vol. 30, p. 87. ACM (2011)
14. Michalek, J., Papalambros, P.: Interactive design optimization of architectural layouts. Engineering Optimization 34(5), 485–501 (2002)
15. Misener, R., Smadbeck, J.B., Floudas, C.A.: Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into glomiqo 2. Optimization Methods and Software 30(1), 215–249 (2015)
16. Mitchell, W.J., Steadman, J.P., Liggett, R.S.: Synthesis and optimization of small rectangular floor plans. Environment and Planning B: Planning and Design 3(1), 37–70 (1976)
17. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: Minizinc: Towards a standard cp modelling language. In: CP, pp. 529–543 (2007)
18. Panero, J., Zelnik, M.: Human dimension and interior space: a source book of design reference standards (1979)
19. Parish, Y.I., Müller, P.: Procedural modeling of cities. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 301–308. ACM (2001)
20. Pigozzi, G., Tsoukiàs, A., Viappiani, P.: Preferences in artificial intelligence. Ann. Math. Artif. Intell. 77(3-4), 361–401 (2016)
21. Raman, K., Shivaswamy, P., Joachims, T.: Online learning to diversify from implicit feedback. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 705–713. ACM (2012)
22. Rossi, F., Van Beek, P., Walsh, T.: Handbook of constraint programming (2006)
23. Shivaswamy, P., Joachims, T.: Online structured prediction via coactive learning. In: ICML. pp. 1431–1438 (2012)
24. Shivaswamy, P., Joachims, T.: Coactive Learning. JAIR 53, 1–40 (2015)
25. Sutherland, I.E.: Sketchpad a man-machine graphical communication system. Transactions of the Society for Computer Simulation 2(5), R–3 (1964)
26. Teso, S., Passerini, A., Viappiani, P.: Constructive preference elicitation by setwise max-margin learning. In: IJCAI. pp. 2067–2073 (2016)
27. Teso, S., Sebastiani, R., Passerini, A.: Structured learning modulo theories. Artificial Intelligence (2015)
28. Tidd, W.F., Rinderle, J.R., Witkin, A.: Design refinement via interactive manipulation of design parameters and behaviors. In: ASME DTM (1992)
29. Turton, R., Bailie, R.C., Whiting, W.B., Shaeiwitz, J.A.: Analysis, synthesis and design of chemical processes (2008)
30. Xu, W., Wang, B., Yan, D.M.: Wall grid structure for interior scene synthesis. Computers & Graphics 46, 231–243 (2015)
31. Yeh, Y.T., Yang, L., Watson, M., Goodman, N.D., Hanrahan, P.: Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. ACM TOG 31(4), 56 (2012)
32. Yu, L.F., et al.: Make it home: automatic optimization of furniture arrangement. SIGGRAPH 30(4) (2011)
33. Zhao, Y., Liu, S.: Global optimization algorithm for mixed integer quadratically constrained quadratic program. J. Comput. Appl. Math. 319, 159–169 (2017)