Computers and Electrical Engineering 93 (2021) 107269

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A study on the automatic generation of banner layouts

Hao Hu [a,1], Chao Zhang [b,1], Yanxue Liang [b,*]

[a] Westlake University, 18 Shilongshan Road, Cloud Town, Xihu District, 310024, Hangzhou, PR China
[b] Intelligent Industry Research Institute of Westlake University, 181 Liansheng Road, Yuhang District, 310023, Hangzhou, PR China

ARTICLE INFO

Keywords:
banner design
layout generation
traditional method
deep reinforcement learning

ABSTRACT

This paper addresses the automatic banner design problem as a layout generation task, where a banner is generated by filling design elements to a given layout. We initially introduce the traditional method of generating design layouts used by designers. After that, we introduce a deep reinforcement learning (DRL) based method to learn the policy of generating design layouts. Evaluation metrics are introduced to assess the quality of design layouts generated by the two proposed methods. The experiment shows that, both methods can generate a layout for a banner of given size, and the DRL based method outperforms the traditional method by generating layouts of better quality. However, results based on the DRL method are affected by reward definitions and a long training process is needed to achieve a better performance.

1. Introduction

In modern life, banners appear in various medias such as advertisements, websites, and mobile phones since they play a key role in visual communication of various messages. Creating an original design is more artistic and some design principles should be complied to achieve pleasing visual appearance. Designing a beautiful and good quality banner is a difficult and time-consuming task, to refine it, since such tasks require professional skills to convey information clearly while also satisfying aesthetic goals. Moreover, many banners are created by non-experts, which consumes a lot of time and the quality is unsatisfying. As the need of creating new banners keeps growing rapidly, automatic tools for creating, adapting, and improving banners design could not only release the effort of designers, but also encourage new ideas and educate novice users.

A banner is often composed of an underlying theme, a background image, the brand logo in focus and affiliated text phrases [1]. These elements are usually taken from a library, placed according to a design layout and applied few image transformations to create a banner (Fig. 1). For designers, a layout is a blueprint gathering design elements to guide them to create design works. It determines the spatial arrangement of various design elements like text, logo, and others. It can be used to describe a group of banners, and to generate new banners with the same blueprint. Following design principles, it is composed of a set of rectangles, which determines the position and size of a design element. A high-quality layout can generate aesthetic pleasing banners and thus enhance information presentation, guide reader attention [2]. In recent years, design layout generation has received a growing interest in computer graphic community. Prior works focusing on layout generation are involved with design style, aesthetic measures, and perception [3,4].

This paper is for special section VSI-hci. Reviews processed and recommended for publication by Guest Editor Dr. Shengzong Zhou.This paper was recommended for publication by Associate Editor Dr. M. Malek.

* Corresponding author.
E-mail address: liangyanxue@westlake-iiri.com (Y. Liang).

[1] Equal contribution

https://doi.org/10.1016/j.compeleceng.2021.107269
Received 24 July 2020; Received in revised form 2 January 2021; Accepted 14 June 2021
Available online 9 July 2021

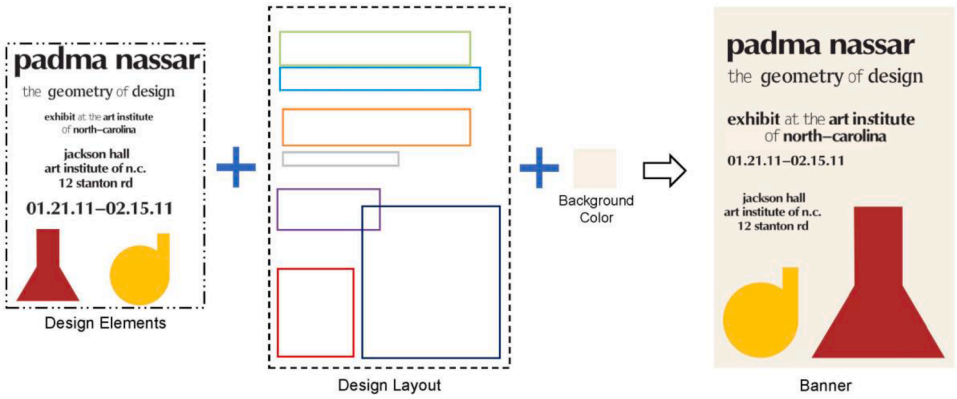

Fig. 1. Banner generation with design layout. Original image is from the work of [5].

Banner design is a complex problem involving layout, colors, fonts, design intent, and requirements gathering. In this work, however, we specifically focus on the layout generation. The layout generation is defined as specifying the locations and sizes of all elements in a banner. In this work, we assume that a set of texts and graphic elements are provided as inputs along with associated meta-data and their corresponding roles. The goal is to generate a visually satisfying banner by arranging these elements properly according to design principles. Although banner design involves a set of choices including colors, fonts and lines breaks, we leave selecting these variables for future work.

We present an automatic tool capable of creating and improving design layouts for design elements of arbitrary sizes. Due to the complexity of banner design, our approach has two stages. Given a set of design elements and their corresponding roles, we first initialize design layouts according to elements' sizes. After that, we use energy terms to evaluate the quality of initial layouts and improve those with low quality. Since we do not consider elements' style, color etc. in a design process, our quality evaluation metrics is different from those described in [6], which takes the darkness, brightness, HSV distribution of design elements into account. Finally, we introduce two methods for layout improvement. One is based on designers' experience of adjusting bad designs. The other one is based on reinforcement learning, where agent learns the policy of layout optimization through interacting with environment. We believe this tool enables to assist designers to create new ideas and process large amount of designs in their daily work.

Our key technical contributions include:

- We propose a traditional method that is used by designers to adjust design layouts according to design principles. Actions for adjusting positions of design elements are transformed into algorithms separately.
- A deep reinforcement learning based method with two different reward functions for layout generation is introduced. Under our evaluation metrics, experiments are conducted to test the performance of different methods.

We test the proposed methods on real banner designs, producing competitive results with those created by professional designers. In the rest of this paper, we will first investigate prior work in Section 2 and clarify layout generation and evaluation in Section 3, then describe how to optimize design layouts in Section 4. Finally, we show an experimental study of proposed methods in Section 5 and conclude this work in Section 6.

2. Related works

In this section, approaches of automatic layout generation are discussed. They are classified into learning-based methods, constraint-based methods, and probabilistic graphical model-based methods.

A state-of-the-art learning approaches have been proposed to generate multi-size banners automatically by utilizing a triangle interpolation with learned style parameters and a multi-size style transfer technique [7]. LayoutGAN is used to synthesize layout by modeling geometric relations of different types of elements [8]. However, the generated layouts are not convincing due to the uncertain performance of GAN. In the facility layout design [9], container marshalling at container yard terminals [10], Q-learning has been implemented in optimizing elements layout. But in graphical-textual layout and optimization, to the best of our knowledge, reinforcement learning has not been applied yet, which makes our method is one of the very few instances.

Except for these learning based methods, several works have also focused on constraint-based methods. Donovan et al. introduced an energy-based model for evaluating layouts based on graphic design principles and stylistic goals [5,11]. They initially analyzed hidden variables of perceptual properties. Those variables were then used as part of the model to evaluate a banner design. Finally, they used optimization methods to synthesize a new layout. Moreover, their method has been extended to a user-interactive mode in the work of [12]. Yang et al. formulated the typography as an energy optimization problem by minimizing the cost of perceptual properties constrained by automatically selected template and further preserving color harmonization [4]. Harrington et al. proposed an energy function to measure the aesthetics of a layout that includes terms such as alignment, regularity, and balance [13].

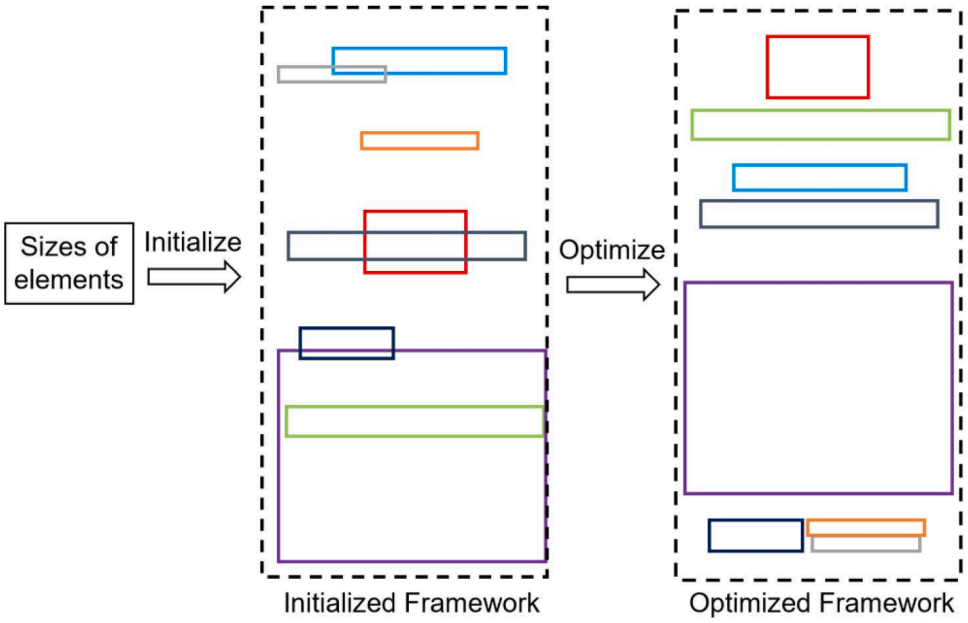

Fig. 2. Layout Initialization and Optimization.

Some researchers focused on automating layout by using probabilistic graphical models. Venkata et al. presented a probabilistic document model (PDM) of adaptive document layout that supports automatic generation of paginated documents for variable content [3]. Cao et al. proposed a two-stage approach to generate a manga page [14]. They firstly used a generative probabilistic framework to create an initial layout that best fits the layout structure model. Then, they used optimization methods to refine the initial layout. Qiang et al. introduced a data-driven method for automatic scientific poster generation [15]. A probabilistic graphical model and a tree structure were proposed to infer panel attributes and to represent panel layout in a poster, respectively. These works, however, do not consider layout elements with scale changes and style transfer, which is one of our focuses in this work.

3. Layout generation

This section describes our approach of generating layouts for a given set of design elements. The main idea is to initialize the position of bounding boxes of design elements of a layout and then optimize the layout with our approaches.

3.1. Layout generation process

An initial layout is generated by randomly initializing positions of bounding boxes (rectangles with different colors in the Fig. 2). We assume that sizes of bounding boxes are fixed once the design elements are given. As it is shown in the Fig. 2, it is possible that bounding boxes may overlap, be misaligned with each other. Layout in such case should be optimized before filling elements in it. That is, moving the positions of bounding boxes to proper positions to achieve aesthetics pleasing layouts. Aesthetic measures of design layouts are introduced in the next section.

3.2. Aesthetics measures

Based on the work in [5], we define text alignment, overlap, visual balance as evaluation terms to measure the impact of elements' positions on the layout aesthetics of a layout. A sigmoid function is used to reshape an evaluation term $e(X)$ as: $E(e(X)) = arctan(e(X)\alpha)/artan(\alpha)$, where larger value of α makes $E(e(X))$ more sensitive to small changes.

Alignment: We consider only the alignment between texts elements. The alignment types are center-X, center-Y, left, right, top, and bottom alignment. The value of the text alignment is defined as:

$$e^{a}_{align} = \frac{\sum_{i\in(all)} \sum_{j\in(all)} I^{a}_{ij}}{n^2} \tag{1}$$

$$E_{alignment} = max\left(E\left(e^{a}_{align}\right)\right) \tag{2}$$

where I^{a}_{ij} is the fraction of element i and element j in a-aligned manner.

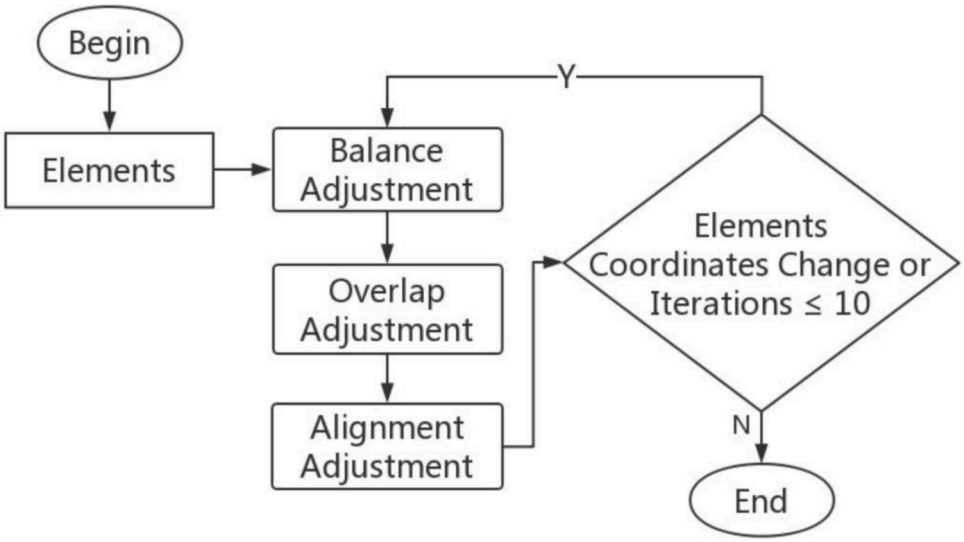

Fig. 3. Framework of the traditional method.

Overlap: We consider the overlap between design elements whose value can be calculated as:

$$e_{overlap} = 1 - \frac{\sum_{i=1}^{n} A_{overlap\_i}}{wh} \tag{3}$$

$$E_{overlap} = E\left(e_{overlap}\right) \tag{4}$$

where $A_{overlap\_i}$ is the overlap area between two elements. w and h are the width and height of a layout.

Visual balance: We define two types of balance: left-right $(l-r)$ balance and top-bottom $(t-b)$ balance. The values of them can be calculated as:

$$e^{l-r}_{balance} = \frac{min(A_l, A_r)}{max(A_l, A_r)}, e^{t-b}_{balance} = \frac{min(A_t, A_b)}{max(A_t, A_b)} \tag{5}$$

$$E^{l-r}_{balance} = E\left(e^{l-r}_{balance}\right), E^{t-b}_{balance} = E\left(e^{t-b}_{balance}\right) \tag{6}$$

where A_l is the overlap area between design elements and the left 1/3 part of a layout. Similarly, A_r, A_b, A_b are the overlap areas between design elements and the right, top, bottom 1/3 part of a layout, respectively. A layout is divided equally into 3 parts along horizontal and vertical direction when computing these terms.

Finally, we define

$$\begin{aligned} E = {} & \alpha_1 * E_{alignment} + \alpha_2 * E_{overlap} + \\ & \alpha_3 * E^{l-r}_{balance} + \alpha_4 * E^{t-b}_{balance} \end{aligned} \tag{7}$$

as the evaluation score to measure the quality of design layout. $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are the weights of each term, which are constant value

4. Layout optimization

In this section, we propose two methods to optimize the initial layouts based on evaluation metrics described above.

The first one is based on designer's experience of moving elements where algorithms are proposed for moving actions. The other one is based on reinforcement learning where DDPG [16] with different reward functions is introduced.

4.1. Traditional method

Fig. 3 shows the main framework of the traditional method, where the input is the coordinates of each element in a layout. The whole adjustment process is carried out by adjusting the balance, overlap and alignment iteratively. Iteration terminates when certain conditions are satisfied. In the following, we will introduce the content of each adjustment with pseudocode.

4.1.1. Balance adjustment

In balance adjustment, we define the physical center and the visual center as the central points of a layout and the minimum outer

border of all elements, respectively. We adjust the visual balance by moving all the elements to certain positions where the visual center coincides with the physical center.

We introduce how to adjust visual balance in accordance with the Algorithm 1. The parameters for this algorithm is the information of a layout, including coordinates of all elements, the width and height as shown in line 1 and 2. The minimum and maximum coordinates of the 1st to the *n*th elements from on *X* and *Y* axes are assigned by F.coordinate as shown in line 3. Next, the algorithm calculates the physical center of a layout by line 4. Line 5 obtains the coordinates of all elements, and the visual center in line 6 is calculated similarly but slightly moving it upward since experts believe the visual center is a little higher from human's perception of beauty. Line 7 calculates the distances in width and height between the physical and visual centers. In the next step, the algorithm updates the new coordinates of all elements by approaching the physical center to the visual center in line 10. Finally, the algorithm returns an adjusted layout in line 11.

4.1.2. Overlap adjustment

For overlap adjustment, with all the pairs of overlapped elements, we take the smaller one as the moving object, and then moving the elements to a certain direction according to their align styles.

We introduce how to adjust visual balance in accordance with the Algorithm 2. Similar with the Algorithm 1, Algorithm 2 initialize the layout parameters and coordinates of elements in line 2 and 3. Line 4 and 5 define two align styles. In each iteration of determining overlap relations of two elements (line 6 and 7), two counters of *K* and *S* are defined to record the adjustment times as shown in line 8 and 9. Line 10 stores the width and height of the overlapped area of elements e_i and e_k, and line 11 stores the width and height of one of the element whose area is smaller. Line 12 to 32 show the procedure of how to decide the moving direction of elements due to different conditions if two elements overlap each other and iteration is within 1000 times. Firstly, line 13 to 16 show the assignment of the flag which indicates the moving direction. After that, from line 22 to 32 the algorithm separates the two elements along Y axis if the assignment of flag is 1, otherwise it separates the two elements along X axis; and the coordinates of elements and counters are updated. Note that, it takes iterative steps to separate two elements in line 26 and 31, and if the separation is unsatisfied in one direction in an axis, it will take the opposite direction in the same axes. Moreover, if it may cause new overlaps in line 23 and 28, the procedure jumps to the other flag trying to separate the elements from other directions in line 24 to 25 and line 29 to 30. Finally, the algorithm returns an adjusted layout in line 33.

4.1.3. Alignment adjustment

In terms of alignment adjustment, we adjust the alignment of two adjacent elements by repositioning their positions if they are not aligned with each other.

We introduce how to align elements in accordance with the Algorithm 3. Similar with the Algorithm 1 and 2, Algorithm 3 initialize the layout parameters and coordinates of elements from line 2 to 4, what different is that coordinates of elements are kept in an order from Y_{min} to Y_{max} of a layout. For two elements that are not aligned, the algorithm calculates the minimum differences of their four coordinates and X, Y centers in line 7. An align style is assigned to the two elements according to the minimum difference d in line 8. Line 9 and 10 get the initial positions of the two adjacent elements. If the difference d satisfies the align condition in line 11, the algorithm move the lower element toward to the upper one to meet their align type, and update the position of the lower element in line 12 and 13. However, this adjustment may cause overlaps, lines from 14 to 21 show that the algorithm simply move the lower element back to its initial position, and move the upper element to meet their align type; the upper element will also be moved back to its initial position if its movement causes overlaps. If the difference d does not satisfy the align condition in line 11, it means the two adjacent elements do not have any align relations, hence, they will not be moved. Finally, the algorithm returns an adjusted layout in line 24.

4.2. Method based on deep reinforcement learning

In this section, details of our implementation of deep reinforcement learning for layout optimization are discussed.

4.2.1. Training environment

The training environment interacts with the agent during the training process by returning numerical rewards where the value is dependent on the action the agent takes on the environment. An episode is defined as an instance of dynamic propagation from initial condition to termination, resulting in a set of state transitions: $(S_0...S_N,a_0...a_N,r_0...r_N,S_1...S_{N+1})$.

Environment: Corresponding to a layout to be optimized.

State: $s^t \in S$, Corresponding to sizes and positions of all rectangles (elements) in a layout.

Action: $a^t \in A$, Corresponding to scale ratios and moving directions (up↑, down↓, left ← and right →) of a rectangle.

State Transition: Corresponding to a layout generation model which returns the transition probability for s^{t+1} given previous state s^t.

Reward function:

Reward is one of the key points in reinforcement learning, which takes a great part in determining the performance of algorithm. In this work, it is intuitive to make use of layout evaluation metrics. At each step, the policy evaluates the value of state of a layout and chooses an action of moving elements. The movement task is defined simply by providing a reward to the learner during training: a successful movement results in a reward of *R*, and a failed movement result in a reward of 0. A movement is considered successful if the evaluation score of a layout getting higher. We propose two definitions of *R*. One is defined as:

$$R_1 = 1, E(s^t) + \alpha < E\left(s^{t+1}\right) \tag{8}$$

where α is the parameter. The other one is defined as:

$$\text{Given} E(s^*) + \alpha < E\left(s^{t+1}\right),$$

$$R_2 = \begin{cases} k_1 * \left(E\left(s^{t+1}\right) - E(s^*)\right), & s_1 < E\left(s^{t+1}\right) \le 1; \\ k_2 * \left(E\left(s^{t+1}\right) - E(s^*)\right), & s_2 < E\left(s^{t+1}\right) \le s_1; \\ k_3 * \left(E\left(s^{t+1}\right) - E(s^*)\right), & s_3 < E\left(s^{t+1}\right) \le s_2 \end{cases} \tag{9}$$

where $E(s^{t+1}) \in [0, 1]$, $s_3 \ge 0$, $k_1 \ge k_2 \ge k_3 \ge 0$. s^* is the state when the action of an agent gets the highest reward in one episode. It is updated as s^{t+1} when $E(s^*) + \alpha < E(s^{t+1})$.

4.2.2. Deep deterministic policy gradients

In this section, we describe the reinforcement learning based approach for layout optimization. We use the Deep Deterministic Policy Gradients (DDPG) model, which output deterministic actions instead of stochastic ones [18].

a) **Reinforcement learning with DDPG**

We consider a classic RL setting that can be represented as a Markov Decision Process [17] defined as a 5-tuple (*S, A, P, r*, γ), where *S* is a set of full states of the environment, *A* is the set of actions, *r:* $S \times A \leftrightarrow \Re$ is the reward function, γ is a discount factor and *P:* $S \times A \times S \leftrightarrow \Re$ is a state transition probability function. Q-learning is a model free algorithm to solve the MDP problem, which means decision making process is independent from the system dynamics. The decision process is partially observable, and the agent receives observations *o* from the set of observations *O*. The reward function in our work is dense (section 4.2.1), so the agent receives the reward every time on accomplishing the full task. Hence, retraining the agent for a new task only requires defining the conditions of success and possibly a new hyper-parameter search, which can be parallelized and automated.

The goal of the agent is to learn a deterministic policy $\pi: O \rightarrow A$ such that taking action $a_t = \pi(o_t)$ maximizes the return from the state s_t (sum of discounted future rewards), $R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$. After taking an action *a*, the environment transitions from state s_t to state s_{t+1} sampled from probability distribution $P(s_t, a_t, .)$. The quality of taking an action a_t in state s_t can be measured by a *Q* function $Q(s_t, a_t) = \mathbb{E}[R_t + \gamma Q(s_{t+1}, a_{t+1})]$.

- DDPG is an off-policy algorithm that builds upon the Deterministic Policy Gradient (DPG) algorithm [19]. DDPG uses an actor-critic architecture with neural networks as function approximators. The actor network is used to make the action decisions while the critic network is used for estimating the Q function. Here, we give a brief definition of key concepts related to DDPG as following:
- **Deterministic action policy *μ*:** It is used to determine which action a_t to take at time step *t* through $a_t = \mu(s_t)$.
- **Policy network:** It is used to approximate μ with neural networks, which is parameterized by θ^{μ}. It corresponds to the actor network in the actor-critic architecture.
- **Behavior policy *β*:** It is used to help agent to explore more action policies during training. It is realized by adding exploration noise to our action policy according to:

$$\beta(s_t) = \mu\left(s_t \middle| \theta_t^{\mu}\right) + \mathcal{N} \tag{10}$$

where $\mathcal{N}$ refers to an Orstein-Uhlenbeck process [20].

- **Behavior policy distribution ρ^{β}:** Given behavior policy β, a status collection generated by an agent attempts to satisfy distribution ρ^{β}.
- **Value function Q:** The quality of acting a_t at state s_t under deterministic action policy μ is measured by Bellman function:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))] \tag{11}$$

- **Value network:** It is used to approximate *Q* with neural networks, which is parameterized by θ^{Q}. It corresponds to the critic network in the actor-critic architecture.

The objective function is defined as:

$$J_{\beta}(\mu) = \mathbb{E}_{\mu}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots + \gamma^n r_n\right] \tag{12}$$

where $\mathbb{E}$ is a function expectation, γ is a discount factor. The optimal deterministic action policy μ^* can be found by maximizing the policy μ in the objective function:

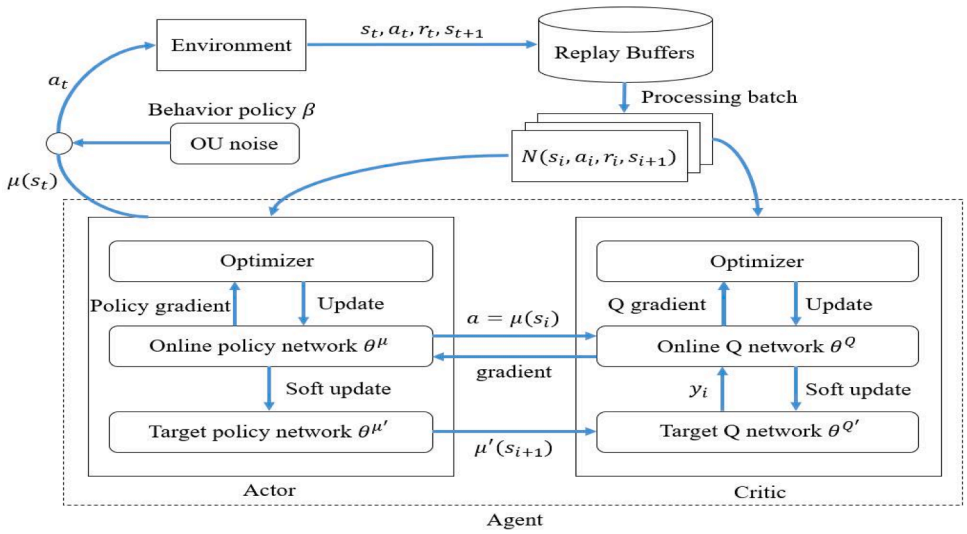

Fig. 4. Flow framework of DDPG.

$$\mu^* = \underset{\mu}{\operatorname{argmax}} J(\mu) \tag{13}$$

In the work of [19], Silver et al have demonstrated that gradient of $J_\beta(\mu)$ with respect to θ^μ is equivalent to the expectation of the gradient of $Q(s, a; \theta^Q)$ with respect to θ^μ. Following the chain rule, parameters of the policy network is updated by:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q_\mu(s_t, \mu(s_t))\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a; \theta^Q)|_{s=s_t, a=\mu(s_t;\theta^\mu)}\right] \end{aligned} \tag{14}$$

where $Q_\mu(s; \mu(s))$ is the action state value Q when an action is selected according to deterministic policy μ at state s.

Given $a = \mu(s; \theta^\mu)$, equation 14 can be rewritten as:

$$\nabla_{\theta^\mu} J = \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a; \theta^Q)|_{s=s_t, a=\mu(s_t)} = \nabla_{\theta^\mu} \mu(s_t; \theta^\mu)|_{s=s_t}\right] \tag{15}$$

By applying gradient ascent algorithm to equation 15, the value of $J_\beta(\mu)$ increases until μ^* is found. Meanwhile, θ^μ updates along with the increasing value of $Q(s, a; \theta^Q)$. Parameters of the value network is updated by minimizing:

$$L(\theta^Q) = \mathbb{E}_{s,a,r,s' \sim R}\left[\left(Target\ Q - Q(s, a; \theta^Q)\right)\right] \tag{16}$$

where

$$Target\ Q = r + \gamma max Q'(s', \mu(s'; \theta^{\mu'})) \tag{17}$$

is the temporal difference with n-step bootstrapping of $Q(s, a; \theta^Q)$, with $\theta^{\mu'}$ and $\theta^{Q'}$ are parameters of Target policy network and Target value network respectively.

Gradient of the value network is defined as:

$$\nabla_{\theta^Q} = \mathbb{E}_{s,a,r,s' \sim R}\left[\left(Target\ Q - Q(s, a; \theta^Q)\right) \nabla_{\theta^Q} Q(s, a; \theta^Q)\right] \tag{18}$$

They are updated by applying gradient descent algorithm to equation 18 during the process of training the value network. In general, the goal of DDPG algorithm is to maximize the objective function $J_\beta(\mu)$ of policy network while minimizing the loss function of the value network.

Using only a single neural network may lead to the learning process of action value (Q value) unstable, since the value network parameters are updated frequently and they are used for calculating the gradient in the policy network at the same time. To address the above issue, the DDPG algorithm creates two neural networks for the policy network and the value network, one is the Online network and the other one is the Target network, which are shown in the following:

$$policy\ network \begin{cases} online: \mu(s; \theta^\mu), update\ \theta^\mu \\ target: \mu'(s; \theta^{\mu'}), update\ \theta^{\mu'} \end{cases} \tag{19}$$

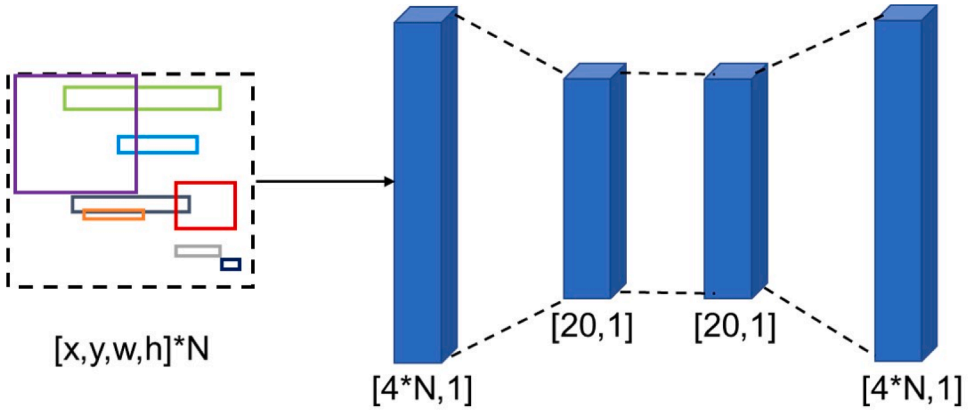

Fig. 5. The architecture of policy network.

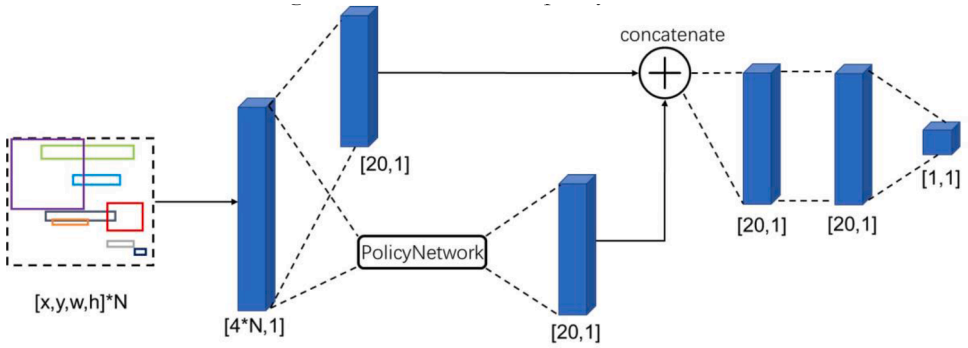

Fig. 6. The architecture of value network.

$$value\ network\begin{cases} online: Q(s;\theta^{Q}), update\ \theta^{Q} \\ target: Q^{'}(s;\theta^{Q^{'}}), update\ \theta^{Q^{'}} \end{cases} \tag{20}$$

The DDPG algorithm adopts four network models. After training of a batch of data, parameters of the online network are updated first by the gradient ascent or descent algorithm. Parameters of the target network are updated afterwards by the Soft update algorithm [21].

From the Fig. 4 we can see that the DDPG model is a MDP process. Both the Actor and Critic contain online and target network models, where the Actor is responsible for the policy network and the Critic is responsible for the value network. Through the interaction process between the Actor and the environment, the episodes generated by the process are stored in the Replay Buffers (experience memory). At the next time step, the Replay Buffers transmit the batch of episodes to the Actor and Critic for further calculation.

a) **Network architecture**

The architecture of policy network is shown in the Fig. 5. Position (*x, y*) and size (*w, h*) of rectangles (*x, y, w, h*)**N* of a layout are input into a fully connected network, where *N* is the number of rectangles. Length of the first layer is 4**N* where 4 is the dimension of a rectangle (*x, y, w, h*). Following the first layer is the two middle layers with 20 neurons. The last layer output the probability of each action of rectangles. Dimension of the output layer is 4**N*, since each of the *N* rectangles has 4 moving directions (up, down, left, right). We use the same architecture for both the Online and Target policy network.

The architecture of the value network is shown in the Fig. 6. The input to the network is the same as those of the policy network. Following the input layer is two branches of networks: one is a fully connected layer with 20 neurons, the other is the policy network. Here, the policy network computes only the possibility of moving actions of input rectangles and does not update parameters through back-propagation. Its output is further connected by fully connected layer with 20 neurons. The two branches of layers with 20 neurons are then concatenated together forming a layer with 40 neurons. Following two fully connected layers of 20 neurons, the final layer output the Q values of all input rectangles. We use the same architecture for both the Online and Target value network.

Table 1
Workflow of the DDPG algorithm.

Framework	Network	Network model	Input
Actor	Policy network	Online u	Current state
		Target μ'	Next state
Critic	Value network	Online Q	Current state and action
		Target Q'	Action of target network

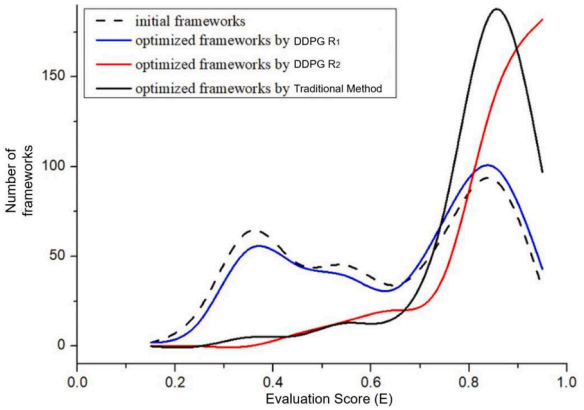

Fig. 7. *E* curve for generated layouts on testing dataset.

Table 2
Number of layouts distributed at different level of evaluation score.

Level	bad	acceptable	good	excellent
Evaluation Score	0.1~0.5	0.5~0.7	0.7~0.9	0.9~1
Initial Layouts	133	79	156	32
DDPG R1†	115	71	171	43
Traditional Method	11	29	263	97
DDPG R2†	7	34	177	182

†: training time is 11h on 6 cores of Intel i7-8700 CPU.

Algorithm 1
Visual Balance Adjustment

1: **Function** *Adjust_VisualBalanceLayout*
2: [F.coordinate,F.width,F.height] ← GetParameters(Layout)
3: [(X1*min*, Y1*min*, X1*max*, Y1*max*), ..., (X*nmin*,Y*nmin*,X*nmax*,Y*nmax*)]← F.coordinate
4: physicalcenter ← (F.width/2,F.height/2)
5: (X_{min},Y_{min},X_{max},Y_{max}) ← outer border coordinates of all elements
6: visualcenter ← ((X_{min}+X_{max})/2, (Y_{min}+Y_{max})/2-F.height/20)
7: (w,h) ← (F.width/2,F.height/2)-visual center
8: **For** i ← 1 to n
9: F.coordinate ← (X*imin*+w,Y*imin*+h,X*imax*+w,Y*imax*+h)
10: **end for**
11: **return** Layout
12: **end function**

5. Experimental study

5.1. Training

We train the model on a dataset with 1800 layouts. Positions of bounding boxes within a layout are initialized randomly. The implementation details are as following. We simulate 500000 episodes. Each episode is limited to 500ms, which means there are at most 200 steps in a single episode. Initial state for each episode is sampled randomly in the state space. The learning rate is set to 0.00025 and the discounted factor is set to 0.99. The batch size is set to 128. The capacity of the transition memory buffer is set to 1000000. We choose Adam as the optimizer. Momentum and gamma of the optimizer are set to 0.9 and 0.99, respectively. We implement our DDPG based on the public deep learning platform TensorFlow. The computing server is the Intel I7 8700 @ 3.20GHz

Algorithm 2
Overlap Adjustment.

```
1: Function Adjust_OverLapLayout
2: [F.coordinate,F.width,F.height] ← GetParameters(Layout)
3: [(X1min,Y1min,X1max,Y1max),...,(Xnmin,Ynmin,Xnmax,Y^n_max)] ← F.coordinate
4: Align style_1 ← (left,right,X_center)
5: Align style_2 ← (top,bottom,Y_center)
6: for i ← 1 to n-1
7: for k ← i+1 to n
8: K ← 0
9: S ← 0
10: (W_overlap,H_overlap) ← the width and height of the overlap area of e_i and e_k % e_n is the nth element
11: (W_min,H_min) ← the width and height of e_i or e_k, whose area is minimum
12: while (e_i,e_k) overlaps and S<1001 and K<1001
13: if (e_i,e_k) Aligns then
14: if (e_i,e_k) is Align style_1 or S==1000 flag ← 1
15: elif (e_i,e_k) is Align style_2 or K==1000
16: flag ← 2
17: else
18: if Woverlap/Wmin ≥ Hoverlap/Hmin
19: flag ← 1
20: else
21: flag ← 2
22: if flag==1 then
23: if Dis(e_i,e_m) < threshold or Dis(e_k,e_m) < threshold
24: goto line 30
25: F.coordinate ← separate(e_i,e_k) along Y direction
26: K ← K+1
27: elif flag==2 then
28: if Dis(e_i,e_m) < threshold or Dis(e_k,e_m) < threshold
29: goto line 25
30: F.coordinate ← separate(e_i,e_k) along X direction
31: S ← S+1
32: return Layout
33: end function
```

Algorithm 3
Alignment Adjustment.

```
2: [F.coordinate,F.width,F.height] ← GetParameters(Layout)
3: keep orders of elements from F.Ymin to F.Ymax
4: [(X1min,Y1min,X1max,Y1max),...,(Xnmin,Ynmin,Xnmax,
Ynmax)] ← elements coordinates
5: for i ← 1,n
6: if (ei,ei+1) do not align then
7: d ← min(|Yimin-Yimin+1 |, |Yimax-Yimax+1 |,|Ximin-Xmini+1 |,
|Ximax-Ximax+1 |,|Xicenter-Xcenteri+1 |, |Yicenter-Yicenter+1 |)
8: Align style(d) ←{top,bottom,left,right,Xcenter,Ycenter} acoording to d's source of coordinates
9: ei_initial ← ei.position()
10: ei+1_initial ← ei+1.position()
11: if 0<d<max(min(F.width,F.height)/α,β). % α and β tuning parameters
12: move ei+1 forward to ei satisfying Align style(d)
13: update(ei+1.position())
14: for e in [e1,...,ei,ei+2,...,en]
15: if (e,ei+1) overlaps then
16: ei.position() ← ei_initial
17: move ei forward to ei+1 satisfying Align_style(d)
18: update(ei.position())
19: for e in [e1,...,ei−1,ei+1,...,en]
20: if (e,ei) overlaps
21: ei.position() ← ei_initial
22: else
23: continue
24: return Layout
25: end function
```

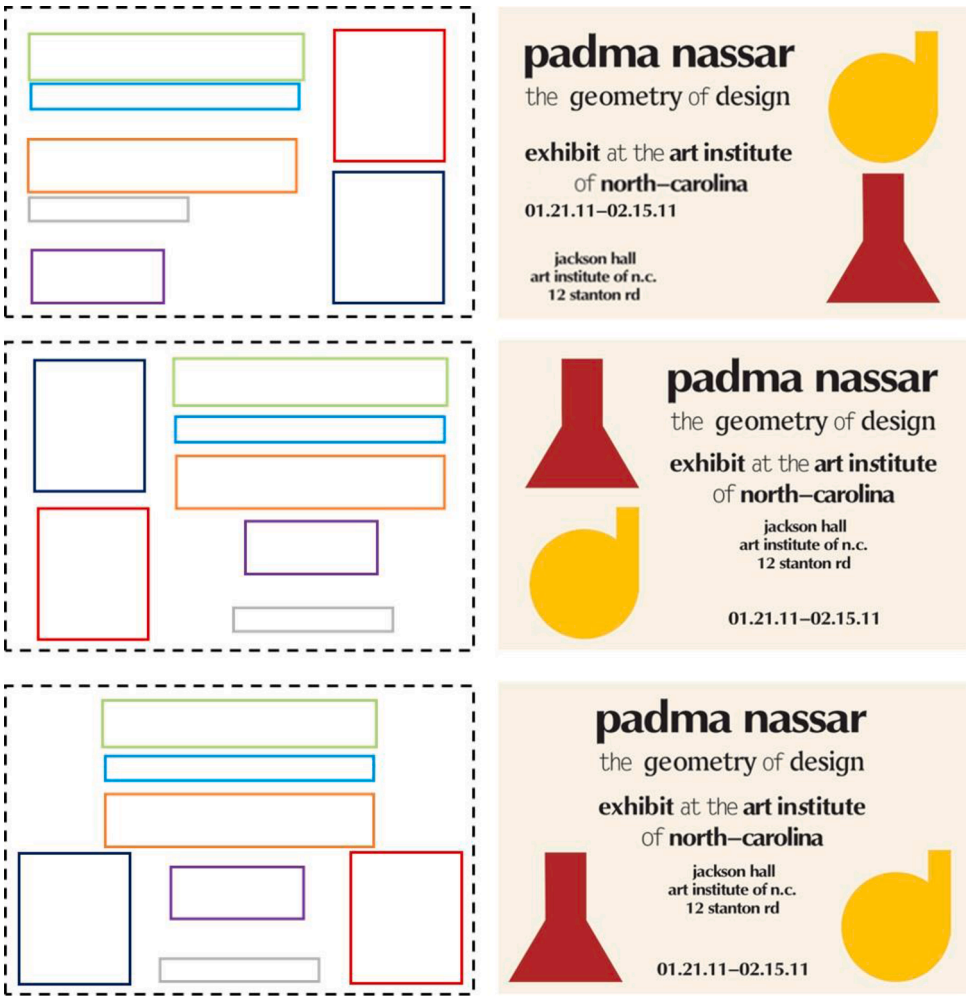

Fig. 8. Sample result1. Up: banner in the work of [5]; Middle: optimized layout using traditional method; Down: optimized layout with DDPG R_2.

with a NVIDIA GTX 1080 GPU.

5.2. *Testing*

We test different models on a dataset with 400 layouts. They are initialized in the same way as those of the training set. Results are summarized as follows.

Table 1

One can first notice that all our proposed methods (traditional and reinforcement learning based) can improve layouts (Fig. 7). Initial layouts (black dashed curve) are improved slightly by DDPG with R_1 reward (blue curve), where the number of layouts of bad or acceptable level is decreased by 12.3% (Table 2).

However, traditional method shows a significant improvement (black curve): number of layouts of bad or acceptable level is decreased by 81.1%; those of good or excellent level are increased by 68.6% and 203% respectively. Among improved layouts, 62.2% of them are increased to good level while 37.8% of them reach the excellent level. Designer's experience can optimize most layouts to have a good design layout but lacks the ability of being excellent.

Finally, for the DDPG with R_2 reward (red curve), one can notice that it has almost the same performance with the traditional method but a little improvement on good and excellent levels. That is, 12.3% of them is increased to good level while 87.7% of them reaches the excellent level, which is almost 5 times more than those of initial layouts. Compared to designer's experience, this model is more capable of optimizing layouts to achieve the layout of better quality. This is because the DDPG model is trained with less prior knowledge and explores more state spaces through interacting with the environment. However, there is no free lunch. Training DDPG models with two different reward functions takes 11 hours to obtain satisfying results while the traditional method does not need the training process. The success of reinforcement learning is based on a lot of simulations for exploration, features selection, and reward design. Good features help the agent fitting scores quickly, while carefully designed reward guides the exploration in stalemates.

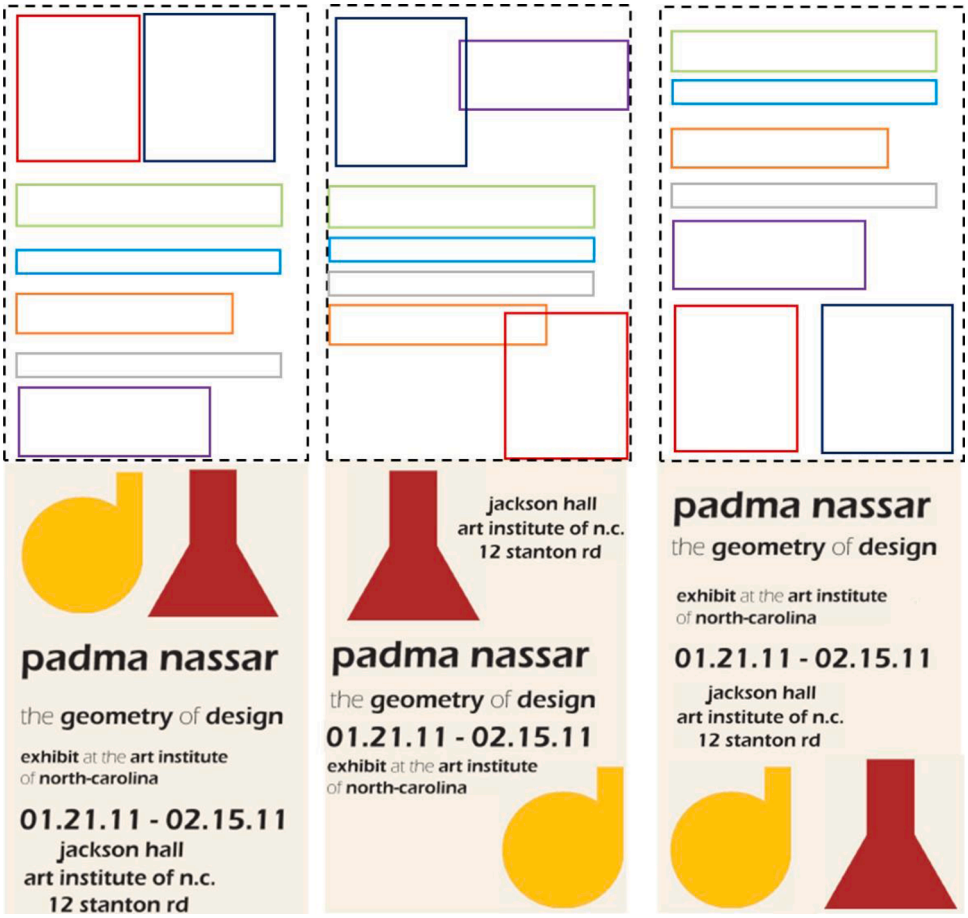

Fig. 9. Sample result2. Left: banner in the work of [5]; Middle: optimized layout using traditional method; Right: optimized layout with DDPG R_2.

Figs. 8 and 9 show sample results of generated layouts. Banners are generated by filling design elements to the generated layouts. Although the quality of evaluating design layout should not only consider the general aesthetic measures but also design intent [22], requirements of designers [23], design constraints [24], we leave these variables as multi-level evaluation for future work.

6. Conclusion

In this paper, we propose methods to automatically generate layouts of design banners. With the purpose of generating a design layout satisfying aesthetic measures, a layout is initialized by randomly positioning design elements, and then reorganize them with proposed methods. Proposed methods are tested and compared on a given banner dataset. Results of experiments show that the deep reinforcement learning based method outperforms the one of traditional method with respect to aesthetic measures. However, performance of the reinforcement learning based method is affected by reward definitions. Also, it requires a long training process to obtain satisfiable results.

The proposed DRL based tool is readily usable for automatic production and it would be interesting to explore a collaborative scenario where both human and machine creatives can be joint together in the design process. Designers may use our tools to generate results as a starting point and refine them into more personalized layouts. We believe that such human-machine collaborative can possibly inspire new approaches and expand our creativity for banner designs.

It is also planned in the future that our work will be extended to banner retargeting to specific size conditioned on the original layout of a given banner. To this end, a hierarchical reinforcement learning (HRL) model might be useful to decompose the task into resize and reposition design elements sub-tasks. Moreover, it would be interesting to incorporate long short-term memory (LSTM) to input the arbitrary number of design elements, since the number of input neurons of the current network is fixed.

Author statement

Hao Hu: Conceptualization, Methodology, Software, Writing- Original draft preparation
Chao Zhang: Data curation, Validation, Visualization, Investigation.

Yanxue Liang: Writing- Reviewing and Editing, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Vempati, Sreekanth, and Korah T. Malayil. "Enabling hyper-personalization: Automated ad creative generation and ranking for fashion e-commerce," arXiv preprint arXiv:1908.10139, 2019.
[2] Cao Y, Lau RWH, Chan AB. Look over here: Attention-directing composition of manga elements. ACM Trans Graph July 2014;33. 94:1–94:11.
[3] Damera-Venkata N, Bento J, O'Brien-Strain E. Probabilistic document model for automated document composition. In: Proceedings of the 11th ACM symposium on Document engineering. ACM; 2011. p. 3–12.
[4] Yang X, Mei T, Xu Y-Q, Rui Y, Li S. Automatic generation of visual-textual presentation layout. ACM Trans Multimedia Comput, Commun Appl (TOMM) 2016;12 (2):33.
[5] O'Donovan P. Learning Design: Aesthetic Models for Color, Layout, and Typography. University of Toronto (Canada; 2015. PhD thesis.
[6] Wu Y, Shen X, Mei T, Tian X, Yu N, Rui Y. Monet: A system for reliving your memories by theme-based photo storytelling. IEEE Trans Multimedia 2016;18(11): 2206–16.
[7] Zhang Y, Hu K, Ren P, Yang C, Xu W, Hua X-S. Layout style modeling for automating banner design. In: Proceedings of the on Thematic Workshops of ACM Multimedia 2017. ACM; 2017. p. 451–9.
[8] J. Li, J. Yang, A. Hertzmann, J. Zhang, and T. Xu, "Layoutgan: Generating graphic layouts with wireframe discriminators," arXiv preprint arXiv:1901.06767, 2019.
[9] Tarkesh H, Atighehchian A, Nookabadi AS. Facility layout design using virtual multi-agent system. J Intell Manuf 2009;20(4):347.
[10] Hirashima Y. A q-learning system for container marshalling with group-based learning model at container yard terminals. In: Proceedings of the International Multiconference of Engineers and Computer Scientists 2009. 1. IMECS; 2009. p. 2009.
[11] ODonovan P, Agarwala A, Hertzmann A. Learning layouts for single-page graphic designs. IEEE Trans Vis Comput Graph 2014;20(8):1200–13.
[12] O'Donovan P, Agarwala A, Hertzmann A. Designs cape: Design with interactive layout suggestions. In: Proceedings of the 33rd annual ACM conference on human factors in computing systems. ACM; 2015. p. 1221–4.
[13] Harrington SJ, Naveda JF, Jones RP, Roetling P, Thakkar N. Aesthetic measures for automated document layout. In: Proceedings of the 2004 ACM symposium on Document engineering. Citeseer; 2004. p. 109–11.
[14] Cao Y, Chan AB, Lau RW. Automatic stylistic manga layout. ACM Trans Graph (TOG) 2012;31(6):141.
[15] Qiang Y-T, Fu Y-W, Yu X, Guo Y-W, Zhou Z-H, Sigal L. Learning to generate posters of scientific papers by probabilistic graphical models. J Comput Sci Tech 2019;34(1):155–69.
[16] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," arXiv preprint arXiv: 1803.00933, 2018.
[17] White C. Markov decision processes. Springer; 2001.
[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.
[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.
[20] Uhlenbeck GE, Ornstein LS. On the theory of the Brownian motion. Phys Rev 1930;36(5):823.
[21] R. Fox, A. Pakman, and N. Tishby, "Taming the noise in reinforcement learning via soft updates," arXiv preprint arXiv:1512.08562, 2015.
[22] Hu Hao, Kleiner Mathias, Pernot Jean-Philippe. Over-constraints detection and resolution in geometric equation systems. Comput-Aided Des 2017;90:84–94.
[23] Olszewska J. D7-R4: Software Development Lifecycle for Intelligent Vision Systems. In: Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume. 2. KEOD; 2019. p. 435–41. ISBN 978-989-758-382-7, pages.
[24] Hu Hao, Chao Zhang, Yanjia Huang, Qian Zhao, Sunny Yeung. Geometric Over-Constraints Detection: A Survey. Archives of Computational Methods in Engineering 2021. https://doi.org/10.1007/s11831-020-09509-y. In press.

Hao Hu received the PhD degree from Arts et Metiers ParisTech, France, in 2018. He is a senior researcher in Westlake University. His research interests include computer vision, perception, computer aided design and manufacturing.

Chao Zhang received his master's degree from Chang'An University in 2019. He is a computer vision engineer in Intelligent Industry Research Institute of Westlake University. His research interests include Deep Learning and Reinforcement Learning applications.

Yanxue Liang Received his master's degree from Shanghai JiaoTong University in 2003 and PhD degree from Tokyo Institute of Technology in 2008. He is now a professor in Westlake University. His current research interests include perception of robotics, CAD, and CAM.