Front.Comput.Sci.
DOI

RESEARCH ARTICLE

Rich-text Document Styling Restoration via Reinforcement Learning

Hongwei LI [1,2], Yingpeng HU [1,2], Yixuan CAO [1,2], Ganbin ZHOU [3], Ping LUO (✉)[1,2]

1 Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China
2 University of Chinese Academy of Sciences, Beijing 100049, China
3 Search Product Center, WeChat Search Application Department, Tencent, Beijing 100080, China

Abstract Richly formatted documents, such as financial disclosures, scientific articles, government regulations, widely exist on Web. However, since most of these documents are only for public reading, the styling information inside them is usually missing, making them improper or even burdensome to be displayed and edited in different formats and platforms. In this study we formulate the task of document styling restoration as an optimization problem, which aims to identify the styling settings on the document elements, e.g. lines, table cells, text, so that rendering with the output styling settings results in a document, where each element inside it holds the (closely) exact position with the one in the original document. Considering that each styling setting is a decision, this problem can be transformed as a multi-step decision-making task over all the document elements, and then be solved by reinforcement learning. Specifically, Monte-Carlo Tree Search (MCTS) is leveraged to explore the different styling settings, and the policy function is learnt under the supervision of the delayed rewards. As a case study, we restore the styling information inside tables, where structural and functional data in the documents are usually presented. Experiment shows that, our best reinforcement method successfully restores the stylings in 87.65% of the tables, with 25.75% absolute improvement over the greedy method. We also discuss the tradeoff between the inference time and restoration success rate, and argue that although the reinforcement methods cannot be used in real-time scenarios, it is suitable for the offline tasks with high-quality requirement. Finally, this model has been applied in a PDF parser to support cross-format display.

Keywords styling restoration, monte-carlo tree search, reinforcement learning, richly formatted documents, tables

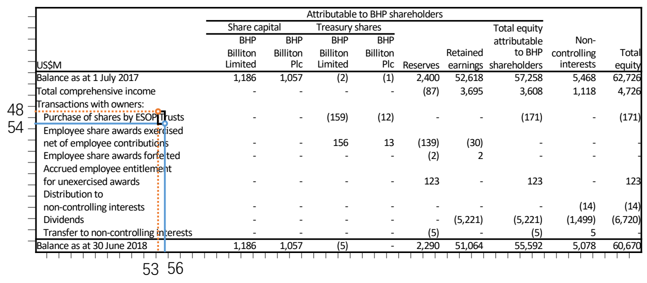
Received month dd, yyyy; accepted month dd, yyyy

E-mail: *luop@ict.ac.cn*

1 Introduction

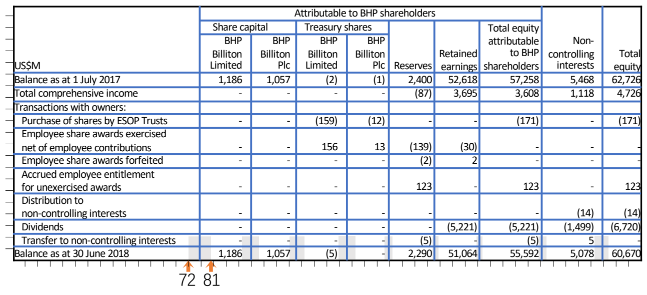
People around us, academicians, financial practitioners, lawyers, government officers, and ones from other industries, always complain to us about the documents every day they need to deal with, most of which are Richly Formatted Documents, like PDF files. These documents publicly disclose the critical information (the factual data, activities, events, regularities and so on) from academia, finance, law, government, and other verticals, and thus play a cornerstone role to reveal and guide the operation of socio-economics. However, since most of these documents are only for public reading, the styling information inside them is usually missing, making them improper or even burdensome to be displayed and edited in different formats and platforms.

Before detailing how these documents trouble us, let us see how they are represented first. Since these documents are mostly for reading but not for editing, they are designed to only store the visual and textual information. For the example in Fig. 1(a), a screenshot of a table in a typical



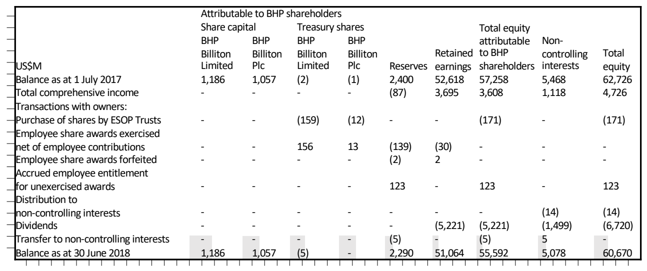
US$M	Attributable to BHP shareholders								
	Share capital		Treasury shares				Total equity attributable to BHP shareholders	Non-controlling interests	Total equity
	BHP Billiton Limited	BHP Billiton Plc	BHP Billiton Limited	BHP Billiton Plc	Reserves	Retained earnings			
Balance as at 1 July 2017	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Total comprehensive income	-	-	-	-	(87)	3,695	3,608	1,118	4,726
Transactions with owners:									
Purchase of shares by ESOP Trusts	-	-	(159)	(12)	-	-	(171)	-	(171)
Employee share awards exercised net of employee contributions	-	-	156	13	(139)	(30)	-	-	-
Employee share awards forfeited	-	-	-	-	(2)	2	-	-	-
Accrued employee entitlement for unexercised awards	-	-	-	-	123	-	123	-	123
Distribution to non-controlling interests	-	-	-	-	-	-	-	(14)	(14)
Dividends	-	-	-	-	-	(5,221)	(5,221)	(1,499)	(6,720)
Transfer to non-controlling interests	-	-	-	-	(5)	-	(5)	5	-
Balance as at 30 June 2018	1,186	1,057	(5)	-	2,290	51,064	55,592	5,078	60,670

(a) A screenshot of a table in a typical financial document. The black box gives the position of the character “T" on the page. The coordinates of its upper left and lower right corners are (48, 53), (54,56).



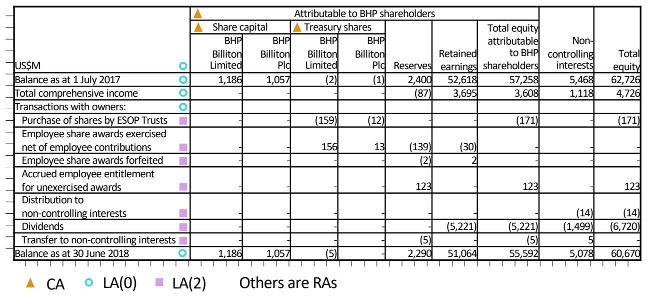
US$M	Attributable to BHP shareholders								
	Share capital		Treasury shares				Total equity attributable to BHP shareholders	Non-controlling interests	Total equity
	BHP Billiton Limited	BHP Billiton Plc	BHP Billiton Limited	BHP Billiton Plc	Reserves	Retained earnings			
Balance as at 1 July 2017	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Total comprehensive income	-	-	-	-	(87)	3,695	3,608	1,118	4,726
Transactions with owners:									
Purchase of shares by ESOP Trusts	-	-	(159)	(12)	-	-	(171)	-	(171)
Employee share awards exercised net of employee contributions	-	-	156	13	(139)	(30)	-	-	-
Employee share awards forfeited	-	-	-	-	(2)	2	-	-	-
Accrued employee entitlement for unexercised awards	-	-	-	-	123	-	123	-	123
Distribution to non-controlling interests	-	-	-	-	-	-	-	(14)	(14)
Dividends	-	-	-	-	-	(5,221)	(5,221)	(1,499)	(6,720)
Transfer to non-controlling interests	-	-	-	-	(5)	-	(5)	5	-
Balance as at 30 June 2018	1,186	1,057	(5)	-	2,290	51,064	55,592	5,078	60,670

(b) We assume that the internal table lines in blue are recognized correctly in advance. However, this predecessor step can only output its position range, shown as the grey shadow for each vertical line. For example, the position range of the leftmost vertical line is $[72, 81]$. This range is decided so that any vertical line in it does not intersect with the text on its left and right sides. Here, the vertical lines are put at the middle of their position ranges.



US$M	Attributable to BHP shareholders								
	Share capital		Treasury shares				Total equity attributable to BHP shareholders	Non-controlling interests	Total equity
	BHP Billiton Limited	BHP Billiton Plc	BHP Billiton Limited	BHP Billiton Plc	Reserves	Retained earnings			
Balance as at 1 July 2017	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Total comprehensive income	-	-	-	-	(87)	3,695	3,608	1,118	4,726
Transactions with owners:									
Purchase of shares by ESOP Trusts	-	-	(159)	(12)	-	-	(171)	-	(171)
Employee share awards exercised net of employee contributions	-	-	156	13	(139)	(30)	-	-	-
Employee share awards forfeited	-	-	-	-	(2)	2	-	-	-
Accrued employee entitlement for unexercised awards	-	-	-	-	123	-	123	-	123
Distribution to non-controlling interests	-	-	-	-	-	-	-	(14)	(14)
Dividends	-	-	-	-	-	(5,221)	(5,221)	(1,499)	(6,720)
Transfer to non-controlling interests	-	-	-	-	(5)	-	(5)	5	-
Balance as at 30 June 2018	1,186	1,057	(5)	-	2,290	51,064	55,592	5,078	60,670

(c) This table is rendered with a default styling settings, where each vertical line is put at the middle of its position range and each cell is set to left-alignment.


▲ CA
○ LA(0)
■ LA(2)
Others are RAs

US$M ○	▲ Attributable to BHP shareholders								
	▲ Share capital		▲ Treasury shares				Total equity attributable to BHP shareholders	Non-controlling interests	Total equity
	BHP Billiton Limited	BHP Billiton Plc	BHP Billiton Limited	BHP Billiton Plc	Reserves	Retained earnings			
Balance as at 1 July 2017 ○	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Total comprehensive income ○	-	-	-	-	(87)	3,695	3,608	1,118	4,726
Transactions with owners: ○									
Purchase of shares by ESOP Trusts ■	-	-	(159)	(12)	-	-	(171)	-	(171)
Employee share awards exercised net of employee contributions ■	-	-	156	13	(139)	(30)	-	-	-
Employee share awards forfeited ■	-	-	-	-	(2)	2	-	-	-
Accrued employee entitlement for unexercised awards ■	-	-	-	-	123	-	123	-	123
Distribution to non-controlling interests ■	-	-	-	-	-	-	-	(14)	(14)
Dividends ■	-	-	-	-	-	(5,221)	(5,221)	(1,499)	(6,720)
Transfer to non-controlling interests ■	-	-	-	-	(5)	-	(5)	5	-
Balance as at 30 June 2018 ○	1,186	1,057	(5)	-	2,290	51,064	55,592	5,078	60,670

(d) After styling restoration, each vertical line has its exact position and each table cell has its right setting of text alignment (shown by the icon). The cells without any icon are all with the Right Alignment setting. The abbreviations of text alignment settings are defined in Table 1.

Fig. 1 A demonstration of table styling restoration problem by using a table in a typical financial document.

financial document[1]. This PDF file only records the absolute position of each character inside it. Specifically, each character in the file corresponds to a bounding box, which gives its exact position on the page. For instance, the black box in Fig. 1(a) with the coordinates of its upper left and lower right corners gives the position of the character “T" on the page. With all the visual and textual information, the pages can be shown properly so that people can cognitively understand the semantics of table cells even though all the internal table lines (the internal table structure) are missing.

However, in many business use cases people always have huge demands in reusing, displaying, editing these tables in different formats and platforms. For example, we need to copy the table in Fig. 1(a) into HTML, Word, or LaTex files, and even edit the text inside the table cells. However, since the table styling settings are missing in the original file, people constantly have the experiences that the copied result in the destination file, rendered with the default styling settings on cell alignments and line positions, is far from the original table. As shown in Fig. 1(c), with all *left-alignment* on each table cell the positions of cell texts change dramatically, and the table semantics become confused so that human might fail to recognize the hierarchical relationships among the horizontal and vertical headers in the table. Hence, it still takes lots of manual work to modify the styling settings so that the target table can be displayed elegantly in the new file type.

People also try many ways to cope with this issue, like saving the table as a bitmap with high resolution to support cross-platform and cross-format display. However, the bitmap cannot support the interactive editing (e.g. removing a column, revising the text in a cell) in the target table. Hence, only with the detailed styling settings, the table can be rendered properly in the target platform to support interactive editing and displaying.

As a case study towards the general purpose of rich-text document styling restoration, we focus on restoring the styling information inside tables, where structural and functional data in the documents are usually presented. For this task we assume that the table visual information and internal table structure are given. For instance, the vertical and horizontal table lines in blue in Fig. 1(b) are recognized in a previous step. However, this predecessor step can only output the position range of each vertical line, shown in the grey shadow in Fig. 1(b) (detailed in Section 3). The text

[1] BHP Annual Report 2018. Retrieved from BHP website.

alignment settings on table cells are all unknown, either. Thus, for a given table this task is to restore its styling information, including the exact position of each vertical line, the text alignment on each cell, and so on[2]. Fig. 1(d) shows the output of this problem. Only with these styling settings, the table can be displayed and edited in different formats and platforms.

Although recent years have witnessed an increasing interests in layout analysis and information extraction in these documents [1], including analyzing the document structure [2], extracting paragraphs, tables and figures [3–7], and the their reading order [8, 9], to our best knowledge there is no previous studies on restoring styling information of richly formatted documents. As a pioneer work towards this end, we focus on restoring the styling settings for tables as tables always contain important structural and functional data in the documents. We also argue that the proposed algorithm can be easily extended to restore the styling information for all the content inside richly formatted documents.

Since there are no styling information but only ground truth positions of cells, we propose an improved reinforcement learning algorithm to the problem of *table styling restoration*. Specifically, we define a state as the current styling settings on a table, and leverage Monte-Carlo Tree Search (MCTS) to explore the state space. The policy function of state transition is learnt so that rendering with the output styling settings results in a table, where the text in each cell holds the (closely) exact position with the one in the original table. In other words, the reward in reinforcement learning is developed as the similarity between rendered and original positions of all the elements inside the table.

Based on a data set of 20,000 tables from the financial market, we empirically show that the proposed algorithm can solve the problem of table styling restoration well. In two conditions (namely one strict and one loose condition) in measuring the success for table styling restoration, our method achieves the success rate of 69.65% and 87.65% on the test tables, respectively. Comparing with the greedy method, our algorithm has 35.9% and 25.75% absolute improvement. Additionally, we study the different success rate for the tables with different number of table cells. We find that the improvement mostly comes from the tables with more cells, indicating more complicated table structures. We also discuss the tradeoff between the inference time and restoration success rate, and argue that although the reinforcement methods cannot be used in real-time scenarios, it is suitable for the offline tasks with high-quality requirement. As some case studies, we also make the videos to show how the reinforcement method and greedy method adjust the styling settings step by step for table styling restoration. We strongly recommend readers to watch these videos[3].

With the support of this study, we have published a tool of PDF parser[4] for open access (detailed in Section 5.4), where users can copy a table from PDF files into other file types, or even paste the LaTex code of this table into the editor. The styling settings obtained in this study are embedded in this tool to ensure that the editable target table appears (almost) the same with the original one.

The contributions of our work are summarized as follows:

- To the best of our knowledge, this is the first study that formulates the problem of document styling restoration as an optimization problem, where we aim to identify the styling settings on the table elements, which minimizes the distance between their rendered positions and the ground truth positions (detailed in Section 3).
- Considering that the styling setting on each document element is a decision, this optimization problem can be transformed into a multi-step decision-making problem, and then solved by reinforcement learning. We also develop some heuristics to guide the search in the learning (detailed in Section 4).
- We empirically show the effectiveness of the proposed method, and release a tool of PDF parser with styling restoration (detailed in Section 5).

2 Related Work

Table-related tasks. Most studies in this area focus on table understanding from richly formatted documents. Since the important structural and functional data in the documents are usually stored in tables, table understanding is the key step to automatic building of knowledge graphs. Table understanding can be further decomposed into three sub-tasks, including table region detection, table structure recognition, and table semantic interpretation. Table detection is to detect the location of tables in a page [4–7]. Table structure recognition is to recognize the internal table

[2] We do not consider table horizontal lines in this study since they are usually put at the middle of their position range.

[3] See StyRes repository on Github website.

[4] A PDF parser for open access, see PDFlux website.

structure [10–13], namely the the grid structure and the spanning cells. With the table structure, table semantic interpretation is to interpret the two-dimensional table into relational database [14]. In this study we recognize the table region and the table structure for each page in a predecessor step. However, this preprocess cannot tell the exact positions of table lines, nor the text alignment on table cells. Thus, we aim to restore these styling settings for cross-platform display and editing.

There are also some studies for the applications over the extracted tables, including the semantic or ad hoc search over tables [15,16], table classification and clustering [17], on-the-fly table generation for a given query [18], and populating rows and columns for entity-focused tables [19]. It is worth mentioning that all these table-related applications need the table styling information if we need to display and edit them on different platforms.

Reinforcement learning. Because of the huge state spaces in our problem (detailed in Section 3), it is impossible to use *tabular solution* methods of reinforcement learning [20] to find the exact optimal solution. In recent years, there are some studies that combined tabular solution methods and deep neural network to approximate value function and policy, such as DQN [21–23]. In 2006, Coulom et al. [24] first proposed Monte-Carlo Tree Search (MCTS) to improve a Go-playing program, which provides the flexible control and efficient selectivity in simulation. Since 2016, Silver et al. [25, 26] combined MCTS with reinforcement learning to train a better Go-playing policy than humans. MCTS is considered a powerful method to improve policy [20]. These methods have been proven to perform well in large state spaces, shown in the great success in AlphaGo. Hence, to find optimal solution of table styling restoration problem, we propose a policy iteration method that employs a deep neural network to evaluate policy and leverages MCTS to improve policy.

3 Problem Formulation with Notions and Denotations

First, we define the notion and denotation used in this study. Here, each table E consists of a set of table elements, namely $\boldsymbol{e} = (e_1, \cdots, e_i, \cdots, e_k)$. In this study we consider two types of table elements, namely table cells and vertical lines. Thus, e_i $(i = 1, \cdots, k)$ refers to a cell or a vertical line in the table. Then, s_i refers to the styling setting on the element e_i, and $\boldsymbol{s} = (s_1, \cdots, s_i, \cdots, s_k)$ refers to all the settings on these elements. Different table elements have different styling settings, summarized in Table 1. For a table cell, its styling can be set to `CA` (center alignment), `RA` (right alignment), or `LA(Δ)` (left alignment with Δ space indentation). For a vertical line, its styling setting refers to its exact horizontal position within a range.

Table 1 The stylings of the cell and line.

Element	Styling
Cell	`CA`: Center Alignment `RA`: Right Alignment `LA(Δ)`: Left Alignment with Δ space indentation ($\Delta = 0, 1, 2, 3, 4$)
Line	x: Horizontal position

With the styling settings $\boldsymbol{s}$, a rendering function G outputs the positions of all these objects, namely

$$G(\boldsymbol{s}) = (x_1, \cdots, x_i, \cdots, x_k)$$

where x_i is the horizontal position of element e_i. Specifically, when e_i is a vertical line, x_i can be directly obtained from s_i. When e_i is a cell, the horizontal position of the text within this cell can be computed as

$$x_i = \begin{cases} (x_l + x_r - length)/2, & s_i \text{ is } \texttt{CA} \\ x_r - length - 0.5\bar{w}, & s_i \text{ is } \texttt{RA} \\ x_l + (0.5 + \Delta)\bar{w}, & s_i \text{ is } \texttt{LA}(\Delta) \end{cases} \tag{1}$$

where x_l and x_r are the horizontal position of the vertical lines on its left and right side, *length* is the length of the text within the cell, $\bar{w}$ is the average length of the characters in the table.

Given the ground truth positions of elements, namely $\boldsymbol{x}^* = (x_1^*, \cdots, x_i^*, \cdots, x_k^*)$ where x_i^* is the ground truth position for e_i[5)], we calculate the distance between $\boldsymbol{x}^*$ and $G(\boldsymbol{s})$ as

$$distance(G(\boldsymbol{s}), \boldsymbol{x}^*) = \max_{e_i \text{ is a cell}} |x_i - x_i^*| \tag{2}$$

In other words, for the text in each cell we calculate the distance between its ground truth position and rendered position, and the distance between $\boldsymbol{x}^*$ and $G(\boldsymbol{s})$ is the maximal distance among them.

Problem Formulation. Given all the table elements $\boldsymbol{e}$, the table structure, and the ground truth positions $\boldsymbol{x}^*$ we aim to find the styling setting $\boldsymbol{s}^*$ which satisfies

$$\boldsymbol{s}^* = \arg\min_{\boldsymbol{s}} distance(G(\boldsymbol{s}), \boldsymbol{x}^*) \tag{3}$$

In other words, we aim to identify the styling settings on the table elements, which minimizes the distance between their

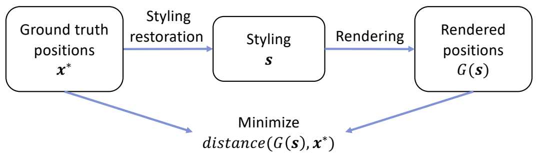

Ground truth positions x^*
Styling restoration
Styling s
Rendering
Rendered positions $G(s)$
Minimize $distance(G(s), x^*)$

Fig. 2 The problem of table styling restoration.

rendered positions and the ground truth positions. This problem formulation is shown in Fig. 2.

Since there may not exist an optimal styling setting $\boldsymbol{s}$ such that $distance(G(\boldsymbol{s}), \boldsymbol{x}^*) = 0$, we introduce the fault-tolerant hyper-parameter α into this problem, such that only if

$$distance(G(\boldsymbol{s}), \boldsymbol{x}^*) \leqslant \alpha * \bar{w} \tag{4}$$

we call this "styling restoration" is successful. It is clear that smaller α makes this problem more difficult to success.

Problem Analysis. The challenge to this problem is that the position of an element depends not only on the styling setting of itself, but also on the settings of the elements around it. As you can see in Eq. (1), the position of a table vertical line directly affects the positions of the text inside the cells on its two sides. Additionally, this intertwined relationship can propagate throughout the neighboring table cells and lines. Thus, the greedy method may not work well on this problem, which will be validated in the experiment section. Meanwhile, if each element has l possible settings and there are k elements in total, the state space of styling settings is l^k. When k is up to the order of 100 (usual case for financial tables), it is impossible to enumerate all the possible styling choices to find the optimal one. Hence, we leverage reinforcement learning to search this optimal solution.

We also summarize the differences between our problem and other general reinforcement learning tasks as follows. First, to avoid redundant actions and even action deadlock in the output action sequence, we need to consider all the predecessor actions which have been taken before the current state. We expect to learn a "concise" policy from the initial state to the target state without redundant actions. Second, for each state we have its distance to the ground truth position. Although this distance cannot be used as reward for reinforcement learning, we expect to check how this distance can be used as prior to guide the search in MCTS for more efficiency. Third, different from the games with fixed-size boards (e.g., Chess, Go, etc.), the tables for styling restoration may have different numbers of rows and columns, and the internal table structure in terms of merging cells may also be totally different. We expect to see whether the reinforcement method still works in this problem scenario.

[5] Note that there are no ground truth positions of vertical lines, they are set to none in this study.

4 Solutions

Considering that each styling setting is a decision, this problem can be transformed into a multi-step decision-making task over all the document elements. Thus, we use a Reinforcement Learning (RL) algorithm to solve the problem. Specifically, in a table, we consider the styling setting $\boldsymbol{s} \in \mathcal{S}$ as the state, where $\mathcal{S}$ is the *state space* that contains all styling settings of this table. Then, $\boldsymbol{s}_t = (s_{t1}, \cdots, s_{ti}, \cdots, s_{tk})$ refers to the state at time t. We take an action $a \in \mathcal{A}$ that transforms the state from $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$. The *action space* $\mathcal{A}$ is

$$\mathcal{A} = \{a_e | a_e \in \text{Action}(e), e \in \boldsymbol{e}\}, \tag{5}$$

where Action(e) indicates the actions of element e that is defined in Table 2 by the element type. In this study, each action a changes the styling setting of only one element in the table. For example, it changes the styling of a cell to `CA`, or moves a line to left by 1 pixel. We define *success value* of a state as 1 if the styling restoration of the table is successful from the state, otherwise -1. We use success value as the *reward* of a state. Note that the reward for this reinforcement learning is delayed. In other words, a state could get a reward only if it meets the termination conditions, namely the styling restoration is successful or the action space of the current state is empty. Then, the reward of the final state can be regarded as the rewards of all the predecessor states in the search sequence.

Table 2 The actions of the cell and line.

Element	Action
Cell	Change styling to s (s = `CA, RA, LA(Δ)`)
Line	Move to left by Λ pixel (Λ = 1, 2, 4, 8)
	Move to right by Λ pixel (Λ = 1, 2, 4, 8)

To avoid redundant actions and even action deadlock in both training and inference, we require that only one action can be performed onto each cell; each line can be moved only towards one direction. It indicates that the action space for the current state might depends on all the predecessor actions. Thus, we need to consider all the predecessor actions to the current state in developing the *value network* (detailed in Section 4.2). Through reinforcement learning,

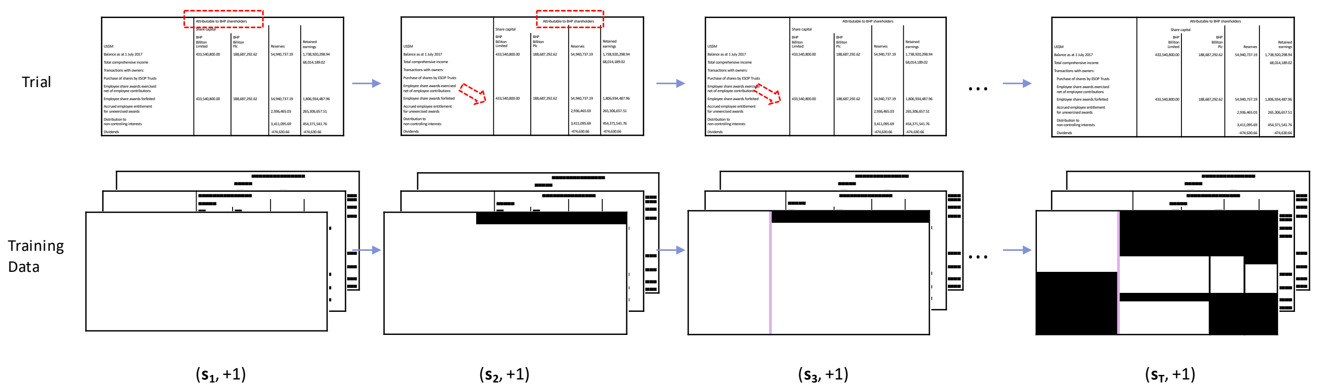


Fig. 3 A successful trial. As shown by the two red rectangles, the first action changes the alignment setting of the corresponding cell from `LA(0)` to `CA`. As shown by the two red arrows, the second action moves the corresponding vertical line to left.

we hope to learn a "concise" policy from the initial state to the target state without redundant actions.

Next, we first introduce the framework of RL and propose the used deep learning model. Then, we incorporate a heuristic feature into RL to further improve the model performance.

4.1 The Framework of RL

The main idea of RL is to use the MCTS-based policy to select an action at each state. At a state, an MCTS guided by a value network (detailed in Section 4.2) is executed to calculate the probability distribution π to select an action. For training the policy, we conduct several trials that produce the training data. In return, we update the policy based on the training data. We consider the process of generating data and updating policy a round and loop the process until convergence. After we training the policy, we use it to restore the styling of a table. This inference step is very similar to the trial step, except that each action is taken not by sampling from the probability distribution π, but directly using the mode of π. We elaborate on RL from three aspects: generating data, updating policy, and MCTS.

Generating training data and model update. We conduct trials to generate training data. In a trial, we use MCTS at each state to generate an action probability distribution π, which indicates the probability of taking an action at this state. Namely, $\pi(a)$ is the probability of taking the action a at this state. Then, we sample an action a by π, which changes the current state to the next state. We perform this trial until meeting the *end conditions* that is reaching a state $\boldsymbol{s}$ that satisfies Eq. (4), or exceeding a specified number of states, or no next state to go. Finally, we get the final styling setting $\boldsymbol{s}$. If $\boldsymbol{s}$ satisfies Eq. (4), we set the success value of all the states in this trial to 1. Otherwise, if the end is caused by exceeding a specified number of states or no next state to go, we set the success value of all these states to -1. Thus, we get one training sample per state in a trial. The success value of the state is used as the supervision signal to train the value network. Fig. 3 shows the training data we get from a successful trial.

At the beginning of the training, we randomly sample a table from the table training set uniformly. Then, we run the trial on this table and put all the generated training samples into the buffer. If the buffer is full, we replace the earliest training samples. We sample training data uniformly from the buffer as a batch for stochastic gradient descent. The loss function of the model is as follows:

$$loss = (z(\boldsymbol{s}) - V_\theta(\boldsymbol{s}))^2 \qquad (6)$$

where $\boldsymbol{s}$ is the state, z is the success value of the state, V_θ is the value network with parameters θ.

Next, we will detail the process of MCTS for action generation.

MCTS. We adopt the MCTS that is similar to [26]. The process of generating the action probability distribution π is shown in Fig. 4. We first create a MC tree, whose root node is the current state. Then, set Q, U and N of the root node to be 0. Here, $Q(\boldsymbol{s}, a)$ is the action value of taking action a at state $\boldsymbol{s}$; $U(\boldsymbol{s}, a)$ is the visit-count-adjusted score; $N(\boldsymbol{s}, a)$ is the number of visits. We plot these values on the resultant node of the action. These three values will be updated in the continuous simulation. The process of generating π will run simulation K times at most. Each simulation consists of the following three steps:

1. Selection. We first select a leaf node of MC tree for following expansion. The selection starts from the root node and chooses the child node that has the largest $Q +$

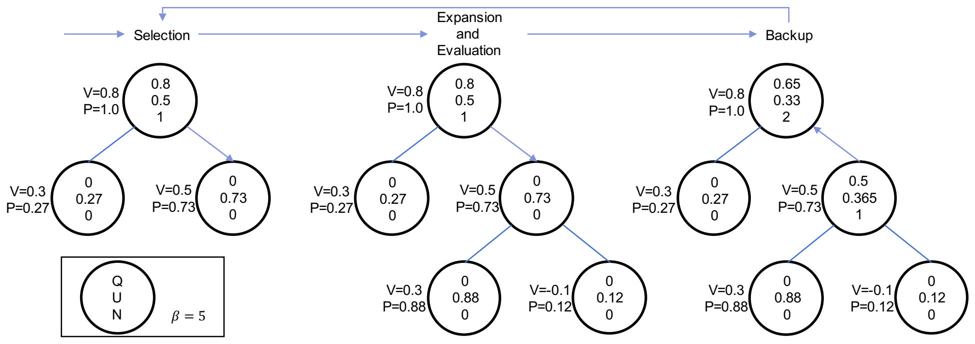


Fig. 4 The process of generating the action probability distribution π. There shows one simulation: 1) the root node chooses its right child node as the leaf node; 2) expand the leaf node and evaluate its child nodes; 3) update the leaf node and its ancestors.

U value as the next node. Iterate the process until it reaches the leaf node.

2. Expansion and Evaluation. If the leaf node is a termination node that meets the end conditions, we set the success value $V(\boldsymbol{s}_l)$ of the leaf node as 1 if it is a success state, or -1 if it is not. Otherwise, we expand all child nodes for the selected leaf node and set Q, U and N of each of its child nodes to 0. This process is called expansion. Next, we use the value network to evaluate the success value $V(\boldsymbol{s}_l)$ of the leaf node and the success value of each of its child nodes. Meanwhile, we calculate the prior probability $P(\boldsymbol{s}, a)$ of each of its child nodes (detailed in Section 4.2 and 4.3), and update U of each of its child nodes according to Eq. (8).
3. Backup. We update Q, U and N of the leaf node and all its ancestors according to Eqs (7-9), where $\boldsymbol{s}$ is the state of a node.

$$N(\boldsymbol{s}, a) \leftarrow N(\boldsymbol{s}, a) + 1 \tag{7}$$

$$U(\boldsymbol{s}, a) \leftarrow \frac{P(\boldsymbol{s}, a)}{1 + N(\boldsymbol{s}, a)} \tag{8}$$

$$Q(\boldsymbol{s}, a) \leftarrow Q(\boldsymbol{s}, a) + \frac{V(\boldsymbol{s}_l) - Q(\boldsymbol{s}, a)}{N(\boldsymbol{s}, a)} \tag{9}$$

It can be seen from Eqs (7-9) that Q is affected by the output of the value network, and U is determined by the prior probability P and the number of visits N. The selection step at each simulation is determined by Q and U. This ensures that the nodes with the higher Q are preferentially visited while the number of visits is the same. When Q is not much different, the nodes with fewer visits have an opportunity to be selected, which makes the simulations more diverse.

The root node of the MC tree corresponds to a state $\boldsymbol{s}_r$. After simulations, the probability of selecting an action a at the root node is

$$\pi(a) = \frac{N(\boldsymbol{s}_r, a)}{\sum_{b \in \mathcal{A}} N(\boldsymbol{s}_r, b)} \tag{10}$$

where π is the action probability distribution of the state $\boldsymbol{s}_r$.

4.2 Value Network

For the studied problem we develop the following value network, shown in Fig. 5. We find that the *target position*, *current state* and *historical actions* are all relevant to the current state value. Specifically, the target position contains the ground truth position of all elements. The current state is the current styling setting $\boldsymbol{s}$. The historical actions record all actions that has been taken from the initial state to the current state. We put these three types of information into target image, current image and historical image. Thus, the input of value network consists of these three images. The process of generating these images is as follows:

- Target image: draw an image based on the ground truth position of each element. Each pixel in the image has a value. Specifically, the value is set to 1 when there is a character or line at this pixel; otherwise, set it to 0. This image is called target image.
- Current image: after rendering with the current styling setting $\boldsymbol{s}$, we draw an image for the current state. The value at each pixel is set in a similar way to that in the target image.
- Historical image: draw an image similar to the current image. However, we set the value at each pixel in a different way. We set it to 1 (or -1) when this pixel is in a cell or on a line onto which an action has taken towards the right (or left) direction. Otherwise, we set it to 0.

Eventually, we concatenate these three images into a three-dimensional tensor as input to a convolutional neural

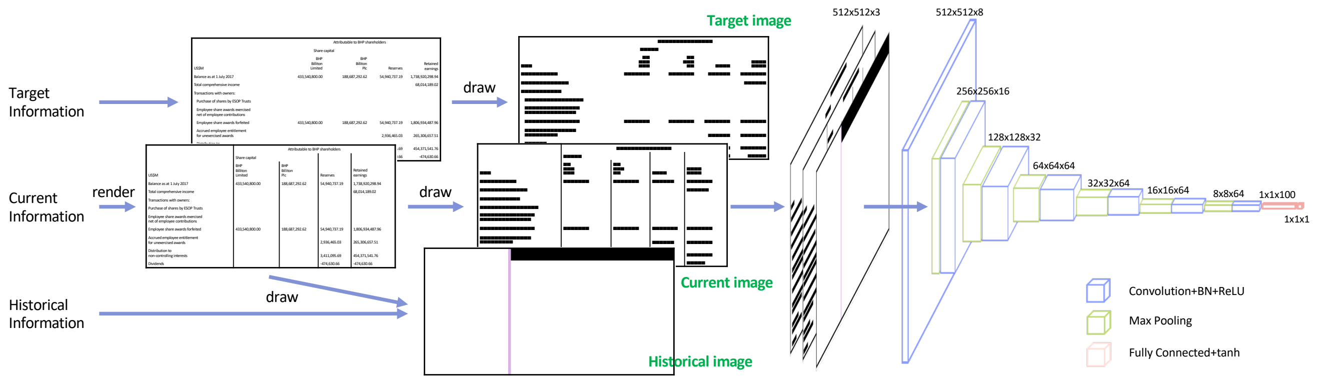


Fig. 5 The value network of RL. In these images, a black pixel indicates 1; a white pixel indicates 0; a purple pixel indicates -1. These images are concatenated as input to a convolutional neural network.

network (CNN). For higher speed, we use a simple CNN structure, as shown in Fig. 5. The CNN consists of 6 convolutional layers, 6 batch normalization layers, 6 max pooling layers and 2 fully connected layers. Each kernel of the convolution layers is 3×3. Each kernel of the max pooling layers is 2×2. The entire network only has a total of 454,308 parameters. The final output of CNN is $V_\theta(\boldsymbol{s}) \in [-1, 1]$. We define the prior probability $P(\boldsymbol{s}, a)$ of taking action a at state $\boldsymbol{s}$ as:

$$P(\boldsymbol{s}, a) = \frac{e^{\beta V_\theta(\boldsymbol{s}_a)}}{\sum_{b \in \mathcal{A}} e^{\beta V_\theta(\boldsymbol{s}_b)}} \in [0, 1] \tag{11}$$

where $\boldsymbol{s}_a$ is the resultant state of taking action a at state $\boldsymbol{s}$; β is a hyper-parameter; $V_\theta(\boldsymbol{s}_a)$ is transformed from $[-1, 1]$ to $[e^{-\beta}, e^{\beta}]$.

4.3 Greedy Method and RL with greedy heuristics

We also develop a Greedy Algorithm (GA) for this problem. Namely, at each state, we select an action that minimizes the distance between $G(\boldsymbol{s}_{t+1})$ and $\boldsymbol{x}^*$ at state $\boldsymbol{s}_t$. The process ends until Eq. (4) is satisfied or the distance does not decline any more. Clearly, GA has strong prior knowledge. It explicitly uses the difference between ground truth positions and rendered positions of the states. This prior knowledge may have the problem of local optimum.

To introduce this prior knowledge into reinforcement learning, we develop the following Reinforcement Learning algorithm with the Greedy heuristics (RLG). Specifically, RLG is the same as RL except that the prior probability in Eq. (11) is replaced with:

$$P(\boldsymbol{s}, a) = \frac{e^{H(\boldsymbol{s}_a)V_\theta(\boldsymbol{s}_a)}}{\sum_{b \in \mathcal{A}} e^{H(\boldsymbol{s}_b)V_\theta(\boldsymbol{s}_b)}}, \tag{12}$$

where

$$H(\boldsymbol{s}_a) = \frac{1}{distance(G(\boldsymbol{s}_a), \boldsymbol{x}^*) + \gamma} \tag{13}$$

Here, γ is smoothing coefficient, H is a function that transforms distance from $[0, +\infty)$ to $(0, \frac{1}{\gamma}]$.

It is worth mentioning that this distance between a state and the ground truth position cannot be uses as *true* reward for reinforcement learning. That is, a state with a small distance might not indicate that it is a good intermediate state towards the final success. Thus, this distance can only be used as the heuristic to guide the search in MCTS.

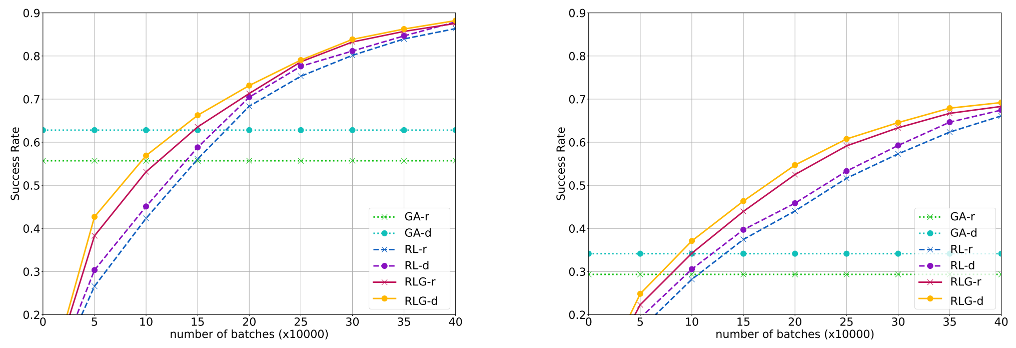
5 Experiment

5.1 Data and Settings

We downloaded a total of 1,819 public PDF documents from CNINFO[6]. Most of these documents are the annual reports of the listed companies. Based on a table recognition tool, we get 20,000 tables with their internal structures. However, the styling settings of these tables are all missing. We split these tables into a training set with 16,000 tables, a validation set with 2,000 tables and a test set with 2,000 tables. The detailed information of these tables in terms of the number of cells, rows, and columns are shown in Table 3.

In each experiment, we set two values of hyper-parameter α, 0.3 and 0.1. Six methods are tested: GA-r, GA-d, RL-r, RL-d, RLG-r and RLG-d. The "-r" means the random initialization that sets the initial styling of cell alignment randomly; the "-d" means the default initialization that sets the initial styling for each cell by selecting the closest

[6] An information disclosure website by the China Securities Regulatory Commission.



(a) $\alpha = 0.3$ (b) $\alpha = 0.1$

Fig. 6 The success rate of each model on the validation set.

Table 3 The detailed dataset statistics.

Documents	1,819
Tables	20,000
Training tables	16,000
Validation tables	2,000
test tables	2,000
Average number of cells	51.49
Average number of rows	10.27
Average number of columns	5.33
Maximum number of cells	630
Maximum number of rows	68
Maximum number of columns	33

alignment to the target. The initial position range of each vertical line is obtained from the previous table recognition step.

There are some other hyper-parameters. The height and width of the rendered image are both 256; the buffer size is 4096; the batch size is 16; β in Eq. (11) is set to 5; the smoothing coefficient γ is 0.05. the number of simulations K is 50. We use the Adam optimization method and the learning rate is 0.01.

We used 4 GPUs (GeForce GTX 1080 Ti) to train reinforcement learning models: 1 GPU for updating model, 3 GPUs for generating data. We have a total of 48 processes for generating data, 16 processes per GPU. And we use 64 processes for evaluation.

5.2 Evaluation Measure

Given a dataset $D = (\boldsymbol{e}^{(1)}, ..., \boldsymbol{e}^{(n)})$, we use the following two measures to evaluate the effectiveness and efficiency of the proposed methods. To measure the effectiveness, we define the first measure, *success rate*, as follow:

$$\text{Success Rate} = \frac{|\{\bar{\boldsymbol{s}}^{(i)} \mid \bar{\boldsymbol{s}}^{(i)} \text{ satisfies Eq. (4)}\}|}{n} \tag{14}$$

where $\boldsymbol{e}^{(i)}$ is the i-th table in the dataset D, $\bar{\boldsymbol{s}}^{(i)}$ is the final styling setting of $\boldsymbol{e}^{(i)}$ after the inference.

To measure the efficiency for inference, we define the second measure, *inference time*, as follow:

$$\text{Inference Time} = \frac{\sum_{i=1}^{n} \bar{t}^{(i)}}{n} \tag{15}$$

where $\bar{t}^{(i)}$ is the running time of the inference of the i-th table.

5.3 Experimental Results

We conduct the experiments to answer the following questions in two aspects:

1) In terms of success rate, is the reinforcement method better than the greedy one? Does introducing the greedy heuristics into RL further improve the performance? How does the success rate is affected by the initialization of the table styling settings and the table complexity in terms of numbers of rows, columns, and cells?

2) In terms of inference time, how much time difference does there exist between the reinforcement method and the greedy one? What is the time percentage of each module in the inference process of the reinforcement method?

To this end, we train the value network on the training set. Along the training process, we evaluate the success rate on the validation set every 50,000 batches of training. Curves of success rate are shown in Fig. 6. After 400,000 batches of training (almost converged), we get the final model for each method and then use it for the evaluation on the test set.

5.3.1 Evaluation on Success Rate

Table 4 shows the success rate of all the methods on the validation and test sets. Clearly, for both $\alpha = 0.1$ and

Table 4 Success rates of models on validation and test set.

Method	Validation set/%		Test set/%	
	$\alpha = 0.3$	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.1$
GA-r	55.70	29.35	56.15	30.65
GA-d	62.80	34.15	61.90	33.75
RL-r	86.35	66.10	85.90	67.15
RL-d	87.90	67.45	87.40	68.60
RLG-r	87.55	68.30	86.95	67.90
RLG-d	**88.20**	**69.20**	**87.65**	**69.65**

$\alpha = 0.3$, the success rate of RL and RLG are much greater than GA. RLG achieves the best success rate of 69.65% and 87.65% on the test set when $\alpha = 0.1$ and $\alpha = 0.3$ respectively. Comparing with GA, it has 35.9% and 25.75% improvement. Note that performance on the validation and test sets are quite similar.

Table 4 also shows that the success rate of RLG is greater than RL and the absolute improvement is between 0.5% and 2.00%. Note that this evaluation is based on the final model after training. Fig. 6 also shows the model performances along the training process. As we can see, at the 50,000-th batch, the gap between the success rate of RLG-d and RL-d is 12.35%. However, at the 400,000-th batch, the gap becomes 0.65%. Thus, we argue that RLG works better in the early stage of training and can converge faster.

Table 4 also shows that a model with default initialization for styling settings has higher success rate than with random initialization. The reason might be that a table with random initialization is more complex for styling restoration and prone to make algorithms get stuck in local optimum. Also, the improvement from GA-r to GA-d is greater than from RL-r to RL-d and from RLG-r to RLG-d. It indicates that RL and RLG are more tolerant to the initialization conditions.

Additionally, we divide the tables in the test set into groups according to the number of table cells (namely, [1, 30), [30, 60), ..., [180, 210)), and calculate the success rate of each model on each group. As shown in Fig. 7, GA, RL, and RLG have the similar performance when the cell number is in [1, 30). As the cell number increases from 30 to 120, RL and RLG performs much better than GA. The performance improvement mainly comes from the tables in this range. Lastly, as the cell number is bigger than 120, the gap between RL and GA becomes smaller. This is mainly because the number of training data in this range is small.

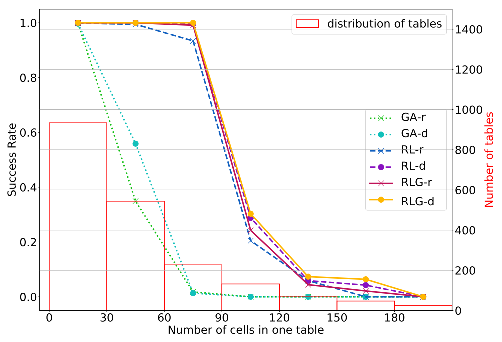


Fig. 7 The success rate of each model on each table group (with $\alpha = 0.3$).

5.3.2 Evaluation on Inference Time

Given a table, the reinforcement method uses MCTS for styling restoration. For each action, it needs K simulations. And for each simulation, it needs to render the resultant states into images as the input of the value network. Thus, it is time-consuming. Here, we detail the techniques used to reduce the inference time, and then discuss the tradeoff between inference time and success rate.

The following techniques are proposed to reduce the inference time.

Local rendering. Since each action only changes the state of one table element, only the positions of some local elements will be changed accordingly. Thus, for a new state $\boldsymbol{s}$ we adopt local rendering to get its image and calculate $distance(G(\boldsymbol{s}), \boldsymbol{x}^*)$ more efficiently.

Grouping of table elements. It is more likely that some adjacent table cells have the same styling settings. For example, if the left position of the cells in a column are all the same, we can put them into one group and this group of cells share the same styling setting. Since the number of groups is much smaller than elements, this rule dramatically reduces the search space.

Avoiding redundant actions and action deadlock. We also use some rules to avoid the taken actions form a deadlock. Specifically, we can take only one action onto a table cell; for a vertical line we can only move it towards one direction. Without these rules, it is possible to have redundant actions on an element, and even form a deadlock. Through reinforcement learning, we hope to learn a "concise" policy from the initial state to the target state without redundant actions.

After the optimizations above, the detailed statistics about time on test set are shown in Table 5. The inference time

Table 5 Inference time of each models on test set.

α	Method	#Actions[a]	#States[b]	@Rendering[c]		@Image[d]		@Network[e]		Inference time/s
				Time/s	%	Time/s	%	Time/s	%	
0.3	GA-r	9.39	397.61	0.26	86.67	-	-	-	-	0.30
	GA-d	6.07	218.10	0.15	71.43	-	-	-	-	0.21
	RL-r	25.89	4283.31	2.77	5.80	16.04	33.57	28.35	59.33	47.78
	RL-d	17.79	3239.52	2.14	6.06	11.07	31.34	21.28	60.25	35.32
	RLG-r	21.31	3820.13	2.49	5.80	14.55	33.89	25.31	58.96	42.93
	RLG-d	16.29	2810.12	1.81	5.68	10.09	31.66	19.13	60.03	**31.87**
0.1	GA-r	12.89	536.24	0.36	90.00	-	-	-	-	0.40
	GA-d	10.45	437.29	0.29	82.86	-	-	-	-	0.35
	RL-r	33.81	5591.76	3.67	6.18	19.08	32.11	36.31	61.11	59.42
	RL-d	22.35	3716.23	2.43	6.01	12.79	31.61	24.83	61.37	40.46
	RLG-r	29.21	4857.68	3.20	6.06	16.93	32.06	32.24	61.05	52.81
	RLG-d	19.89	3316.51	2.17	5.91	11.38	31.00	22.56	61.45	**36.71**

[a] #Actions is the number of actions taken from the initial state to the final state.
[b] #States is the number of all the visited states during inference.
[c] @Rendering is the time consumed to execute the rendering function G.
[d] @Image is the time consumed to draw images for value network.
[e] @Network is the time consumed to calculate in the value network.

mainly is comprised of 3 parts: @Rendering, @Image, @Network, standing for the time consumed to execute the rendering function G, draw images for value network, and calculate in the value network. We also record the number of actions taken from the initial state to the final state, and the number of all the visited states during inference.

Since there is no @Image and @Network, the cost of executing GA depends mostly on @Rendering. Besides, comparing with RL and RLG, GA take less actions which means it visit less states and execute less rendering function G. Therefore, GA costs much less inference time than RL and RLG on average, which reveals that RL and RLG can only handle offline tasks and cannot response in real time.

Table 5 shows that @Rendering of RL and RLG is only about 6% of their inference time. Because each state needs to evaluate the success value, RL and RLG spend a lot of time on drawing the input images and calculating in the value network. They approximately take up 32% and 60% of the inference time, respectively. With the same initialization, RLG takes fewer action steps than RL and their times of each step are almost the same. Therefore, with the same initialization, RLG spends less time than RL on restoring the styling of a table. It is interesting to see that RLG is more efficient than RL while RLG achieves better success rate.

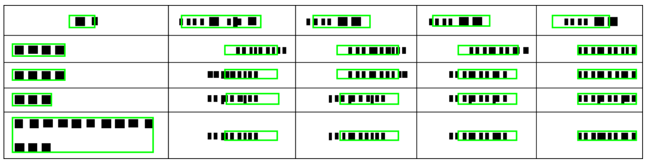
Tradeoff between the success rate and inference time. We evaluate the test set by using RLG-d with $\alpha = 0.3$ in different number of simulations K for MCTS. Table 6 shows the result. As K increases, both success rate and inference time increase. Specifically, as K increases, the number of visited states increases; the inference time is approximately linear to the number of visited states; the success rate is approximately sub-linear to the number of visited states. It is also interesting to see that more simulations tend to restore the styling with fewer number of taken actions.

Table 6 The result of RLG with $\alpha = 0.3$ in different K on test set.

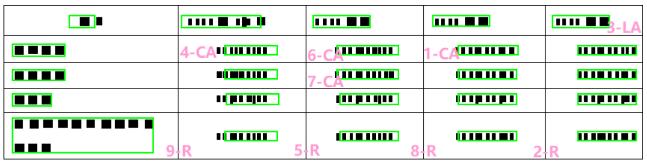
K	#Actions	#States	Inference time/s	Success rate/%
10	21.31	733.18	**8.79**	72.55
20	19.41	1349.27	15.89	77.95
30	17.80	1842.76	21.44	81.30
40	16.65	2317.75	26.69	84.80
50	16.29	2810.12	31.87	87.65
60	16.04	3320.29	36.98	88.15
70	15.81	3822.43	42.85	88.50
80	15.57	4303.17	48.41	88.95
90	15.34	4766.23	53.48	**89.10**

5.4 Case Studies and Real-world Deployment

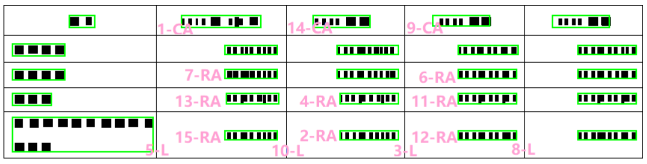
Fig. 8 shows the results of each method on an example table. All the characters are converted to black blocks for easy reading, and each green box shows the ground truth position of the text inside each cell. Fig. 8(a) is the table with the randomly initialized styling information. And the following three figures show the tables with the restored styling settings from the methods of GA, RL, and RLG. Additionally, the red text in a cell or beside a vertical line records the action applied onto it and the action sequence. For example, “1-CA” in Fig. 8(b) means that it is the first step in this adjustment process and it sets the text alignment of this cell as CA, while “2-R" means that in this second move it moves the line to the right. As we can see in Fig. 8(b), GA moves the second right-most vertical line to the right, which causes that all the other lines move to the right. Finally, the texts in second left-most column fail to meet the ground truth positions, and the restoration fails. By contrast, RL and RLG take more actions, but they bypass the local optimum and restore the styling successfully.



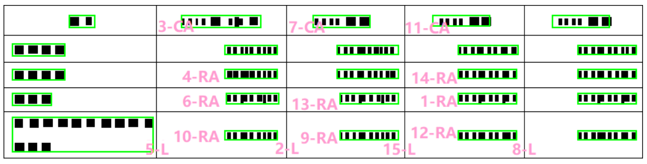
(a) The table with randomly initialized styling.



(b) The table with the restored styling by GA.

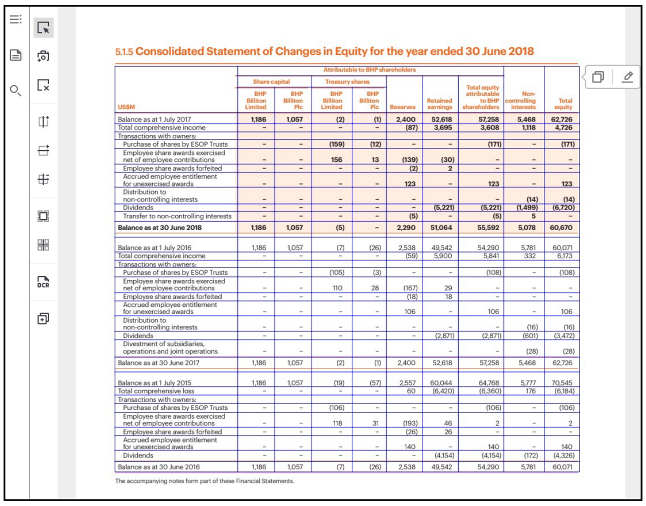


(c) The table with the restored styling by RL.



(d) The table with the restored styling by RLG.

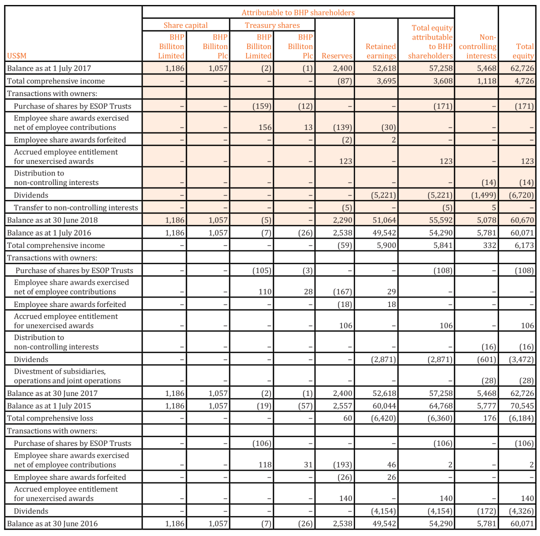
Fig. 8 The tables with the styling settings from different methods.



5.1.5 Consolidated Statement of Changes in Equity for the year ended 30 June 2018

The accompanying notes form part of these Financial Statements.

(a) The snapshot of the PDF parser.



	Attributable to BHP shareholders								
	Share capital		Treasury shares						
US$M	BHP Billiton Limited	BHP Billiton Plc	BHP Billiton Limited	BHP Billiton Plc	Reserves	Retained earnings	Total equity attributable to BHP shareholders	Non-controlling interests	Total equity
Balance as at 1 July 2017	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Total comprehensive income	–	–	–	–	(87)	3,695	3,608	1,118	4,726
Transactions with owners:									
Purchase of shares by ESOP Trusts	–	–	(159)	(12)	–	–	(171)	–	(171)
Employee share awards exercised net of employee contributions	–	–	156	13	(139)	(30)	–	–	–
Employee share awards forfeited	–	–	–	–	(2)	2	–	–	–
Accrued employee entitlement for unexercised awards	–	–	–	–	123	–	123	–	123
Distribution to non-controlling interests	–	–	–	–	–	–	–	(14)	(14)
Dividends	–	–	–	–	–	(5,221)	(5,221)	(1,499)	(6,720)
Transfer to non-controlling interests	–	–	–	–	(5)	–	(5)	5	–
Balance as at 30 June 2018	1,186	1,057	(5)	–	2,290	51,064	55,592	5,078	60,670
Balance as at 1 July 2016	1,186	1,057	(7)	(26)	2,538	49,542	54,290	5,781	60,071
Total comprehensive income	–	–	–	–	(59)	5,900	5,841	332	6,173
Transactions with owners:									
Purchase of shares by ESOP Trusts	–	–	(105)	(3)	–	–	(108)	–	(108)
Employee share awards exercised net of employee contributions	–	–	110	28	(167)	29	–	–	–
Employee share awards forfeited	–	–	–	–	(18)	18	–	–	–
Accrued employee entitlement for unexercised awards	–	–	–	–	106	–	106	–	106
Distribution to non-controlling interests	–	–	–	–	–	–	–	(16)	(16)
Dividends	–	–	–	–	–	(2,871)	(2,871)	(601)	(3,472)
Divestment of subsidiaries, operations and joint operations	–	–	–	–	–	–	–	(28)	(28)
Balance as at 30 June 2017	1,186	1,057	(2)	(1)	2,400	52,618	57,258	5,468	62,726
Balance as at 1 July 2015	1,186	1,057	(19)	(57)	2,557	60,044	64,768	5,777	70,545
Total comprehensive loss	–	–	–	–	60	(6,420)	(6,360)	176	(6,184)
Transactions with owners:									
Purchase of shares by ESOP Trusts	–	–	(106)	–	–	–	(106)	–	(106)
Employee share awards exercised net of employee contributions	–	–	118	31	(193)	46	2	–	2
Employee share awards forfeited	–	–	–	–	(26)	26	–	–	–
Accrued employee entitlement for unexercised awards	–	–	–	–	140	–	140	–	140
Dividends	–	–	–	–	–	(4,154)	(4,154)	(172)	(4,326)
Balance as at 30 June 2016	1,186	1,057	(7)	(26)	2,538	49,542	54,290	5,781	60,071

(b) The table rendered with the restored styling by the PDF parser.

Fig. 9 The PDF parser for open access.

For displaying more complex cases, we also make the videos[3)] to show how the reinforcement and greedy methods adjust the styling settings step by step. They reveal that the greedy method often reach the local optimum while the reinforcement method learns the ability to make the restoration successful.

It is worth mentioning that the proposed model has been deployed into a PDF parser to support cross-format table display. This tool can extract the tables from PDF files. Fig. 9(a) shows the snapshot of a PDF page, where the table region and table structure are recognized in a predecessor step, while Fig. 9(b) shows the table rendered with the restored styling. In this tool, users can also copy the LaTex code of the extracted table. Note that the results from the reinforcement models are shown only for public disclosure documents after we process them in advance.

6 Conclusion

In this paper, we formulate the problem of table styling restoration, which is useful in the scenarios that we need to interactively edit the table content in different platforms and formats. Since there are no styling information but only ground truth positions of cells, we propose an improved reinforcement learning algorithm. We show that our best reinforcement method successfully restores the stylings in 87.65% of the tables, with 25.75% absolute improvement over the greedy method. We also discuss the tradeoff between the inference time and restoration success rate, and argue that although the reinforcement methods cannot be used in real-time scenarios, it is suitable for the offline tasks with high-quality requirement. In the future, we will extend our solution to restore the styling information of the whole page area, not just in the table area. And we will consider more types of the styling settings.

7 Acknowledgements

This work was supported by the National Key Research and Development Program of China under Grant No. 2017YFB1002104, the National Natural Science Foundation of China under Grant No. U1811461, and the Innovation Program of Institute of Computing Technology, CAS.

References

1. Wu S, Hsiao L, Cheng X, Hancock B, Rekatsinas T, Levis P, Ré C. Fonduer: Knowledge base construction from richly formatted data. In: Proceedings of the 2018 International Conference on Management of Data. 2018, 1301–1316
2. Chao H, Fan J. Layout and content extraction for pdf documents. In: Proceedings of 6th International Workshop on Document Analysis Systems. 2004, 213–224
3. Oro E, Ruffolo M. Pdf-trex: An approach for recognizing and extracting tables from pdf documents. In: Proceedings of the 10th International Conference on Document Analysis and Recognition. 2009, 906–910
4. Wang Y, Hu J. A machine learning based approach for table detection on the web. In: Proceedings of the 11th international conference on World Wide Web. 2002, 242–250
5. Gilani A, Qasim S R, Malik I, Shafait F. Table detection using deep learning. In: Proceedings of 2017 14th IAPR International Conference on Document Analysis and Recognition. 2017, 771–776
6. He D, Cohen S, Price B, Kifer D, Giles C L. Multi-scale multi-task fcn for semantic page segmentation and table detection. In: Proceedings of 2017 14th IAPR International Conference on Document Analysis and Recognition. 2017, 254–261
7. Rashid S F, Akmal A, Adnan M, Aslam A A, Dengel A. Table recognition in heterogeneous documents using machine learning. In: Proceedings of 2017 14th IAPR International Conference on Document Analysis and Recognition. 2017, 777–782
8. Meunier J L. Optimized xy-cut for determining a page reading order. In: Proceedings of Eighth International Conference on Document Analysis and Recognition. 2005, 347–351
9. Malerba D, Ceci M, Berardi M. Machine learning for reading order detection in document image understanding. In: Machine Learning in Document Analysis and Recognition, volume 90, 45–69. Springer, 2008
10. Fang J, Mitra P, Tang Z, Giles C L. Table header detection and classification. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. 2012
11. Schreiber S, Agne S, Wolf I, Dengel A, Ahmed S. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In: Proceedings of 2017 14th IAPR International Conference on Document Analysis and Recognition. 2017, 1162–1167
12. Pinto D, McCallum A, Wei X, Croft W B. Table extraction using conditional random fields. In: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. 2003, 235–242
13. Nagy G, Seth S C, Jin D, Embley D W, Machado S, Krishnamoorthy M. Data extraction from web tables: The devil is in the details. In: Proceedings of 11th International Conference on Document Analysis and Recognition. 2011, 242–246
14. Chen X, Chiticariu L, Danilevsky M, Evfimievski A, Sen P. A rectangle mining method for understanding the semantics of financial tables. In: Proceedings of 2017 14th IAPR International Conference on Document Analysis and Recognition. 2017, 268–273
15. Wang H L, Wu S H, Wang I, Sung C L, Hsu W L, Shih W K. Semantic search on internet tabular information extraction for answering queries. In: Proceedings of the ninth international conference on Information and knowledge management. 2000, 243–249
16. Zhang S, Balog K. Ad hoc table retrieval using semantic similarity. In: Proceedings of the 2018 World Wide Web Conference. 2018, 1553–1562
17. Ghasemi-Gol M, Szekely P A. Tabvec: Table vectors for classification of web tables. Computing Research Repository, 2018, abs/1802.06290
18. Zhang S, Balog K. On-the-fly table generation. In: Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. 2018, 595–604
19. Zhang S, Balog K. Entitables: Smart assistance for entity-focused tables. In: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2017, 255–264
20. Sutton R S, Barto A G. Reinforcement learning: An introduction. MIT press, 2018
21. Mnih V, Kavukcuoglu K, Silver D, Rusu A A, Veness J, Bellemare M G, Graves A, Riedmiller M, Fidjeland A K, Ostrovski G, others . Human-level control through deep reinforcement learning. Nature, 2015, 518: 529–533
22. Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. 2016, 2094–2100
23. Anschel O, Baram N, Shimkin N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning. 2017, 176–185
24. Coulom R. Efficient selectivity and backup operators in monte-carlo tree search. In: Proceedings of the 5th International Conference on Computer and Games. 2006, 72–83
25. Silver D, Huang A, Maddison C J, Guez A, Sifre L, Van Den Driessche

G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, others . Mastering the game of go with deep neural networks and tree search. Nature, 2016, 529: 484–489

26. Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, others . Mastering the game of go without human knowledge. Nature, 2017, 550: 354–359

Hongwei Li received the BE degree in software engineering from Fuzhou University, China in 2015 and now is a PhD student at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests focus on machine learning, natural language processing and information extraction.

Yingpeng Hu received the BE degree in computer science and technology from University of Chinese Academy of Sciences at 2018 and now is a MS student at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests focus on machine learning, natural language processing.

Yixuan Cao received the BE degree in transportation engineering from Tongji University in 2015 and now is a PhD student at the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include natural language processing and information extraction.

Ganbin Zhou received the BE degree in software engineering from Dongbei University, China in 2013. He received his a Ph.D. degree at the Institute of Computing Technology, Chinese Academe of Sciences at 2018. His research interests mainly focus on dialog systems, machine learning and data mining.

Ping Luo received the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences. He is an associate professor in the Institute of Computing Technology, Chinese Academy of Science (CAS). His general area of research is knowledge discovery and machine learning.