Learning to Perceive Two-Dimensional Displays Using Probabilistic Grammars

Nan Li, William W. Cohen, and Kenneth R. Koedinger

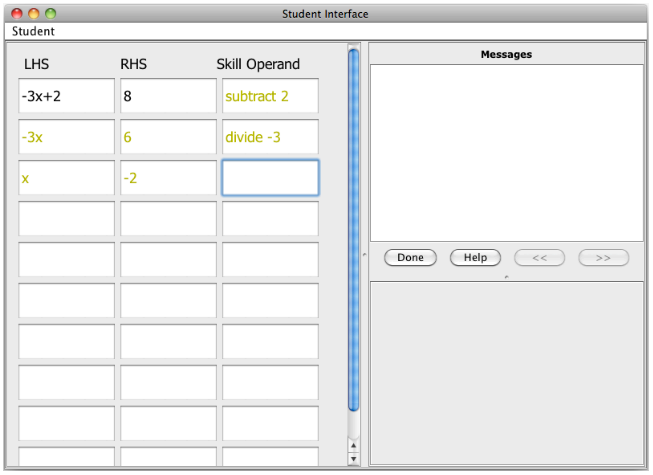
Carnegie Mellon University, Pittsburgh PA 15232, USA,
nli1@cs.cmu.edu, wcohen@cs.cmu.edu, koedinger@cs.cmu.edu

Abstract. People learn to read and understand various displays (e.g., tables on webpages and software user interfaces) every day. How do humans learn to process such displays? Can computers be efficiently taught to understand and use such displays? In this paper, we use statistical learning to model how humans learn to perceive visual displays. We extend an existing probabilistic context-free grammar learner to support learning within a two-dimensional space by incorporating spatial and temporal information. Experimental results in both synthetic domains and real world domains show that the proposed learning algorithm is effective in acquiring user interface layout. Furthermore, we evaluate the effectiveness of the proposed algorithm within an intelligent tutoring agent, *SimStudent*, by integrating the learned display representation into the agent. Experimental results in learning complex problem solving skills in three domains show that the learned display representation is as good as one created by a human expert, in that skill learning using the learned representation is as effective as using a manually created representation.

Keywords: two-dimensional grammar learning, learning to perceive displays, intelligent agent, cognitive modeling

1 Introduction

Every day, people view and understand many novel two-dimensional (2-D) displays such as tables on webpages and software user interfaces. How do humans learn to process such displays? As an example, Figure 1 shows a screenshot of one interface to an intelligent tutoring system that is used to teach students how to solve algebraic equations. The interface should be viewed as a table of three columns, where the first two columns of each row contain the left-hand side and right-hand side of the equation, and the third column names the skill applied. In tutoring, students enter data row by row, a strategy which requires a correct intuitive understanding of how the interface is organized. SimStudent [1] is a system that uses programming-by-demonstration [2] to develop a rule-based tutor on an arbitrary interface, and to learn effectively, it needs a similar understanding of the way the interface is organized. Incorrect representation of the interface may lead to inappropriate generalization of the acquired skill knowledge, such



LHS	RHS	Skill Operand
-3x+2	8	subtract 2
-3x	6	divide -3
x	-2	

Fig. 1. The interface where SimStudent is being tutored in an equation solving domain.

as generalizing the skill for adding two numerators to adding two denominators in fraction addition. Past instances of SimStudent have used a hand-coded hierarchical representation of the interface, which is both time-consuming, and less psychologically plausible. Here we consider replacing that hand-coded element with a learned representation.

More generally, we consider using a two-dimensional variant of a probabilistic context-free grammar (pCFG) to model how a user perceives the structure of a user interface, and propose a novel 2-D pCFG learning algorithm to model acquisition of this representation. Our learning method exploits both the spatial layout of the interface, and temporal information about when users interact with the interface. The alphabet of the grammar is a vocabulary of symbols representing primitive interface-element types. For example, in Figure 1, the type of the cells in the first two columns is *Expression*, and the type of the last cell in the each column is *Skill*. (In SimStudent, these primitive types can be learned from prior experience.) We extend an ordinary one-dimensional (1-D) pCFG learner [3] to acquire two-dimensional grammar rules, using a two-dimensional probabilistic version of the Viterbi training algorithm to learn parameter weights and a structure hypothesizer that uses spatial and temporal information to propose grammar rules.

We then integrate this two-dimensional representation learner into SimStudent. SimStudent is used to model the learning of human students in tutoring domains such as algebra. Many students learn quickly, from few examples; however, some learn more slowly. Previous work in cognitive science [4] showed that one of the key factors that differentiates experts and novices in a field is their different prior knowledge of world state representation. Previously, we had to manually encode such representation, which is both time consuming and error prone. We now extend SimStudent by replacing the hand-coded display representation with the statistically learned display representation. We demonstrate the proposed algorithm in tutoring systems, and for simplicity will refer to terminal symbols in the grammar as interface element, but we emphasize that the proposed algorithm should work for two-dimensional displays of other types as well. We evaluate the proposed algorithms in both synthetic domains and real world domains, with and without integration into SimStudent. Experimental results show that the proposed learning algorithm is effective in acquiring user interface layouts. The SimStudent with the proposed representation learner acquired domain knowledge at similar rates to a system with hand-coded knowledge. The main contribution of this paper is to use probabilistic grammar induction to model learning to perceive two-dimensional visual displays.

2 Related Work

In previous work, we have developed a one-dimensional (1-D) pCFG learner to acquire representations of 1-D strings (e.g., the parse structure of *-3x*), and showed that the acquired representations yield effective learning, while reducing the amount of knowledge engineering required in building an intelligent agent [5]. Moreover, it has been shown that with this extension, the intelligent agent becomes a better model of human students [6], and can be used to better understand human student learning behavior [7]. In this work, we further extend the representation learner to acquire representations in a 2-D space using a two-dimensional variant of pCFG.

One closely related research area that also uses two-dimensional pCFGs is learning to recognize equations (e.g., [8, 9]). Algorithms in this direction often assume the structure of the grammar is given, and use a two-dimensional parsing algorithm to find the most likely parse of the observed image. Our system differs from their approaches in that we model the acquisition of the grammar structure, and apply the technique to another domain, learning to perceive user interface.

Research on extracting structured data on the web (e.g., [10–12]) shares a clear resemblance with our work, as it also concerns on understanding structures embedded in a two-dimensional space. It differs from our work in that webpages have an observable hierarchical structure in the form of their HTML parse trees, whereas we only observe the 2-D visual displays, which have no such structural information.

3 Problem Definition

To learn the representation of a 2-D display, we first need to formally define the input and output of the problem.

3.1 Input

The input to the algorithm is a set of records, $\mathcal{R} = \{R_1, R_2, ..., R_n\}$, associated with examples shown on the display observed by people. Figure 1 shows one problem example in this algebra tutor interface. Each record, R_i $(i = 1, 2, ...n)$, records how and when the elements in the display are filled out by users. Thus, R_i is a sequence of tuples, $\langle T_{i1}, T_{i2}, ..., T_{im} \rangle$, where each tuple, T_{ik} $(k = 1, 2, ..., m)$, is associated with one display element that is used in solving the problem. The tuples in a record are ordered by time. For example, to solve the problem, *-3x+2 = 8*, shown in Figure 1, the cells in the first three rows (except for the last cell of the third row) are used. We do not assume that meta-elements such as columns and rows are given, but we will assume that each display element occupies a rectangular region, and that we can detect when regions are adjacent. In this case, R_i will contain 12 tuples, $\langle T_{i1}, T_{i2}, ..., T_{i12} \rangle$, that correspond to the eight cells, *Cell 11*, *Cell 12*, *Cell 13*, *Cell 21*, *Cell 22*, *Cell 23*, *Cell 31*, and *Cell 32*, and the four buttons, *done*, *help*, <<, and >>.

Each tuple consists of seven items,

$$T_{ik} = \langle type, x_{left}, x_{right}, y_{up}, y_{bottom}, timestamp_{start}, timestamp_{end} \rangle$$

where *type* is the type of the input to the display element, x_{left}, x_{right}, y_{up}, and y_{bottom} define the x and y coordinates of the space the element ranges over, and $timestamp_{start}$ and $timestamp_{end}$ are the start and ending time when the display element is filled out by the user. For example, given the problem *-3x+2 = 8*, the tuple associated with *Cell 11* is $T_{i1} = \langle Expression, 0, 1, 0, 1, 0, 0 \rangle$. The timestamp of *Cell 11* is *0*, since both *Cell 11* and *Cell 21* were entered first by the tutor as the given problem. As mentioned above, we have developed a 1-D pCFG learner that acquires parse structures of 1-D strings. The type of the input is the non-terminal symbol associated with the parse tree of the content. Hence, the type of *-3x+2* is *Expression.*

3.2 Output

Given the input, the objective of the grammar learner is to acquire a 2-D pCFG, $\mathcal{G}$, that best captures the structural layout given the training records, that is,

$$\arg\max_{\mathcal{G}} \; p\left(\mathcal{R} \mid \mathcal{G}\right)$$

under the constraint that all records share the same parse structure (i.e., layout). We will explain this in more detail in the algorithm description section.

The output of the layout learner is a two-dimensional variant of pCFG [8], which we define below. When used to parse a display, this grammar will generate a tree-like hierarchical grouping of the display elements.

Two-Dimensional pCFG 2-D pCFG is an extended version of 1-D pCFG. Each 2-D pCFG, $\mathcal{G}$, is defined by a four-tuple, $\langle \mathcal{V}, \mathcal{E}, \mathcal{R}ules, S \rangle$. $\mathcal{V}$ is a finite set of non-terminal symbols that can be further decomposed to other non-terminal or terminal symbols. $\mathcal{E}$ is a finite set of terminal symbols, that makes up the actual content of the “2-D sentence”. In our algebra example, the terminal symbols of the visual display are the input types associated with the display elements

Table 1. Part of the two-dimensional probabilistic context free grammar for the equation solving interface

Terminal symbols: *Expression*, *Skill*;
Non-terminal symbols: **Table**, **Row**, **Equation**, **Exp**, **Ski**
Table → 0.7, [v] **Table Row**
Table → 0.3, [d] **Row**
Row → 1.0, [h] **Equation Ski**
Equation → 1.0, [h] **Exp Exp**
Exp → 1.0, [d] *Expression*
Ski → 0.5, [d] *Skill*

(e.g., *Expression*, *Skill*). $\mathcal{R}ules$ is a finite set of 2-D grammar rules. S is the start symbol.

Each 2-D grammar rule is of the form

$$V \rightarrow p, [direction]\ \gamma_1\ \gamma_2\ ...\gamma_n$$

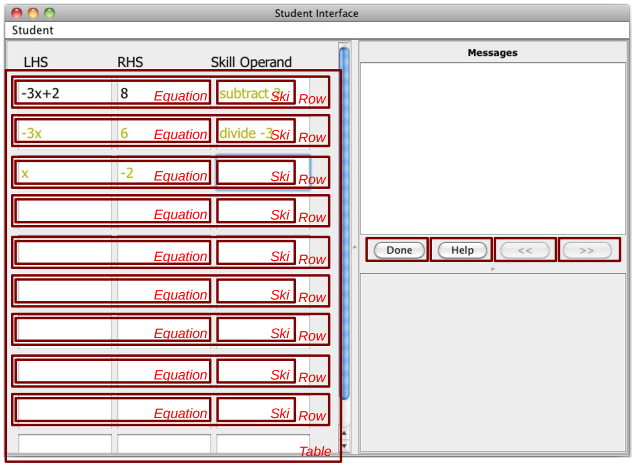
where $V \in \mathcal{V}$, p is the probability of the grammar rule used in derivations[1], and $\gamma_1, \gamma_2, ...\gamma_n$ is either a sequence of terminal symbols or a sequence of non-terminal symbols. Without loss of generality, in this case, we only consider grammar rules that have one or two symbols at the right side of the arrow.

direction is a new field added for the 2-D grammar. It specifies the spatial relation among its children. The value of the direction field can be d, h, or v. d is the default value set for grammar rules that have only one child, in which case there is no direction among the children. h (v) means the children generated by the grammar rule should be placed horizontally (vertically) with respect to each other. An example of a two-dimensional pCFG of the equation solving interface is shown in Table 1[2]. The corresponding layout is presented in Figure 2. The rows in the table are placed vertically with respect to other rows. Thus, the direction field in the grammar rule "**Table** → 0.7, [v] **Table Row**" is set to be v. On the other hand, the equation should be placed horizontally with the skill cell in the third column, so the direction field of "**Row** → 1.0, [h] **Equation Ski**" is h. These three direction values form the original direction value set.

Since the interface elements may not form a rectangle sometimes (e.g., the table and the buttons in the equation solving interface), we further extend the direction field to have two additional values *pv* and "ph". *pv* (*ph*) means that the children of the grammar rule should be placed vertically (horizontally) with respect to each other, but the parts in the interface associated with these children do not have to form a rectangle. As shown in Figure 2, the table in the left side and the buttons in the right side can be placed horizontally, but do not form a rectangle. In this case, the grammar rule should use *ph* instead of h as the directional field value. These direction values are less-preferred than the original

[1] The sum of the probabilities associated with rules that share the same head, V, equals to 1.

[2] The non-terminal symbols are replaced with meaningful names here. The symbols in the learned grammars are synthetic-generated symbols.



LHS	RHS	Skill Operand
-3x+2	8	subtract 2
-3x	6	divide -3
x	-2	

Fig. 2. An example layout of the interface where SimStudent is being tutored in an equation solving domain.

values. Grammar rules that have such direction values will only be added if no more rules with directions d, h, or v can be found.

Layout Given the 2-D pCFG, the final output of the display representation is a hierarchical grouping of the display elements, which we will call a layout, L. Figure 2 shows an example layout of the equation solving interface. The left side of the interface contains a row-ordered table, where each row is further divided into an equation and a skill. The right side of the interface contains a list of buttons that can be pressed by students to ask for help or to indicate when he/she considers the problem is solved.

4 Learning Two-Dimensional Display Layout Using Probabilistic Grammars

Now that we have formally defined the learning task, we are ready to describe the 2-D display layout learner. Recently, we have proposed a 1-D grammar learner [3], and have shown that the 1-D grammar learner acquires knowledge more effectively and runs faster than the inside-outside algorithm [13][3]. Hence, we further extend the one-dimensional grammar learner to acquire a 2-D pCFG from two-dimensional training records.

Algorithm 1 shows the pseudo code of the 2-D display layout learner. The learning algorithm iterates between a greedy structure hypothesizer (GSH) and a Viterbi training phase. The GSH tries to construct non-terminal symbols as well as grammar rules that could parse all input records, $\mathcal{R}$. The set of constructed

[3] rakaposhi.eas.asu.edu/nan-tist.pdf.

Algorithm 1: *2D-Layout-Learner* constructs a set of grammar rules, $\mathcal{R}ules$, from the training records, $\mathcal{R}$, and a set of terminal symbols $\mathcal{E}$.

```
Input: Record Set $\mathcal{R}$, Terminal Symbol Set $\mathcal{E}$
$\mathcal{R}ules := \phi$;
while not-all-records-have-one-layout($\mathcal{R}$, $\mathcal{R}ules$) do
    $\mathcal{R}ules :=$ GSH($\mathcal{R}$, $\mathcal{E}$, $\mathcal{R}ules$);
    $\mathcal{R}ules :=$ Viterbi-training($\mathcal{R}$, $\mathcal{R}ules$);
end
return $\mathcal{R}ules$
```

rules are then set as the start point for the Viterbi training algorithm. Next, the Viterbi training algorithm iteratively re-estimates the probabilities associated with all grammar rules until convergence. If the grammar rules are not sufficient in generating a layout in the Viterbi training algorithm, GSH is called again to add more grammar rules. This process continues until at least one layout can be found.

Since an appropriate way of transferring previously acquired knowledge to later learning process could potentially improve the learning speed, we further designed a learning mechanism that transfers the acquired grammar with the application frequency of each rule from previous tasks to future tasks. Due to the limited space, we will not present the detail of this extension in this paper.

4.1 Viterbi Training

Given a set of grammar rules from the GSH step, the Viterbi training algorithm tunes the probabilities on the grammar set, and removes unused rules.[4] We consider an iterative process. Each iteration involves two steps.

One key difference between learning the parse trees of 1-D strings and learning the GUI element layout is that the parse trees for different input contents are different (e.g., *-3x* vs. *5x+6*), whereas the GUI elements should always be organized in the same way even if the input contents in the GUI elements have changed from problem to problem. For instance, students will always perceive the equation solving interface as multiple rows, where each row consists of an equation along with a skill operation, no matter which problem they are given. Therefore, instead of finding a grammar that parses the interface given specific input, the learning algorithm should acquire one layout for the interface across different problems. This effectively adds a constraint on the learning algorithm.

In the first step, the algorithm computes the most probable parse trees, $\mathcal{T}$, for all training records using the current rules, under the constraint that the

[4] More detailed discussion on why a Viterbi training algorithm instead of the standard CKY is used can be found in [14], which is mainly because of overfitting.

parse structure among these trees should be the same, that is,

$$
\begin{aligned}
\mathcal{T} &= \arg\max_{\mathcal{T}}\ p\left(\mathcal{T} \mid \mathcal{R}, \mathcal{G}, S\right) \\
&= \bigcup_{i=1,2,...n} \arg\max_{T_i}\ p\left(T_i \mid R_i, \mathcal{G}, S\right) \\
& s.t.\ \ parse(T_1) = parse(T_2) = ... = parse(T_n)\ \forall\ T_i \in \mathcal{T}
\end{aligned}
$$

where T_i is the parse tree with root S for record R_i given the current grammar $\mathcal{G}$, and $parse(T_i)$ denotes the parse structure of T_i ignoring the symbols associated with the parse nodes[5]

Since any subtree of a most probable parse tree is also a most probable parse subtree, we have

$$
\begin{aligned}
& p\left(T_i \mid R_i, \mathcal{G}, S_i\right) \\
&= \max_{rule, idx} \begin{cases}
p\left(rule \mid \mathcal{G}\right) \times p\left(T_{i,1} \mid R_{i,1}, \mathcal{G}, S_{i,1}\right) \times p\left(T_{i,2} \mid R_{i,2}, \mathcal{G}, S_{i,2}\right) \\
\qquad\qquad \text{if } rule \text{ is } S_i \rightarrow p(rule|\mathcal{G}), [direction]\ S_{i,1}\ S_{i,2}, \\
p\left(rule \mid \mathcal{G}\right) \times p\left(T_{i,1} \mid R_i, \mathcal{G}, S_{i,1}\right) \\
\qquad\qquad \text{if } rule \text{ is } S_i \rightarrow p(rule|\mathcal{G}), [direction]\ S_{i,1}, \\
p\left(rule \mid \mathcal{G}\right) \\
\qquad\qquad \text{if } rule \text{ is } S_i \rightarrow p(rule|\mathcal{G}), [direction]\ E_{i,1}, \text{ and } E_{i,1} \in \mathcal{E}.
\end{cases}
\end{aligned}
$$

where $rule$ is the rule that is used to parse the current record R_i, $p\left(rule \mid \mathcal{G}\right)$ is the probability of $rule$ used among all grammar rules (in all directions) that have head S_i, $R_{i,1}$ and $R_{i,2}$ are the split traces based on the direction of the rule, $direction$, and the place of the split, idx, and T_i, $T_{i,1}$ and $T_{i,2}$ are the most probable parse trees for R_i, $R_{i,1}$ and $R_{i,2}$ respectively. Using this recursive equation, the algorithm builds the most probable parse trees in a bottom-up fashion.

After getting the parse trees for all records, the algorithm moves on to the second step. In this step, the algorithm updates the selection probabilities associated with the rules. For a rule with head V, the new probability of getting chosen is simply the total number of times that rule appearing in the Viterbi parse trees divided by the total number of times that V appears in the parse trees, that is,

$$
p(rule_i|\mathcal{G}) = \frac{|rule_i \text{ appearing in parse trees}|}{|V_i \text{ appearing in parse trees}|}
$$

where $rule_i$ is of the form $V_i \rightarrow p, [direction], \gamma_1, \gamma_2, ...\gamma_n$, $n = 1\ or\ 2$.

After finishing the second step, the algorithm starts a new iteration until convergence. This learning procedure is a fast approximation of expectation-maximization, which approximates the posterior distribution of trees given parameters by the single MAP hypothesis. The output of the algorithm is an updated 2-D pCFG, $\mathcal{G}$, and the most probable layout of the interface. For elements

[5] In the case that some record uses less elements than the other records (e.g., simpler problems that require less steps), $parse(T_i)$ is considered equal to $parse(T_j)$ as long as the parse structures of the shared elements are the same.

Algorithm 2: *GSH* constructs a set of grammar rules, $\mathcal{R}ules$, a set of terminal symbols $\mathcal{E}$, and from the training records, $\mathcal{R}$.

```
Input: Record Set $\mathcal{R}$, Terminal Symbol Set $\mathcal{E}$, Grammar Rule Set $\mathcal{R}ules$
if is-empty-set($\mathcal{R}ules$) then
    $\mathcal{R}ules$ := generate-terminal-grammar-rules($\mathcal{E}$);
end
while not-all-records-are-parsable($\mathcal{R}$, $\mathcal{R}ules$) do
    if has-recursive-structure($\mathcal{R}$) then
        $rule$ := generate-recursive-rule($\mathcal{R}$);
    else
        $rule$ := generate-most-frequent-non-added-rule($\mathcal{R}$);
    end
    $\mathcal{R}ules$ := $\mathcal{R}ules$ + $rule$;
    $\mathcal{R}$ := update-record-set-with-rule($\mathcal{R}$, $rule$, $\mathcal{R}ules$);  // First, update the
    record set using $rule$; second, update the record set using all
    acquired $\mathcal{R}ules$
end
$\mathcal{R}ules$ = initialize-probabilities($\mathcal{R}ules$);
return $\mathcal{R}ules$
```

that have never been used in the training examples, the acquired layout will not include them in it as there is no information for them in the record. But the acquired grammar may be able to generalize to those elements. For example, if the acquired grammar learns a recursive rule across rows, it will be able to generalize to more rows than the training records have reached.

The complexity of the Viterbi training phase is $\mathcal{O}(|iter| \times |\mathcal{R}| \times |\mathcal{R}ules_{nt}| \times |\max R_i.length|!)$, where $|iter|$ is the number of iterations, $|\mathcal{R}|$ is the number of records, $|\mathcal{R}ules_{nt}|$ is the number of rules that reduce to non-terminal symbols, $|\max R_i.length|$ is the length of the longest record. In practice, since the number of rules generated by GSH is small, and we cache previously calculated parse trees in memory, as we will see in the experiment section, all learning tasks are completed within a reasonable amount of time.

4.2 Greedy Structure Hypothesizer (GSH)

As with the standard Viterbi training algorithm, the output of the algorithm converges toward only a local optimum. It often requires more iterations to converge if the starting point is not good. Moreover, since the complexity of the Viterbi training phase increases as the number of grammar rules increases, we designed a greedy structure hypothesizer (GSH) that greedily adds grammar rules for frequently observed "adjacent" symbol pairs. Note that instead of building a structure learner from scratch, we extend an existing one [3] to accommodate the 2-D space. Extending other learning mechanisms is also possible. To formally define adjacency, let's first define two terms, *temporally adjacent*, and *horizontally (vertically) adjacent.*

Definition 1 *Two tuples, T_{i1} and T_{i2}, are temporally adjacent, iff the two tuples' time intervals overlap, i.e.*

$$[T_{i1}.timestamp_{start}, T_{i1}.timestamp_{start}) \cap$$
$$[T_{i2}.timestamp_{start}, T_{i2}.timestamp_{start}) \neq \emptyset$$

Definition 2 *Two tuples, T_{i1} and T_{i2}, are horizontally adjacent, iff the spaces taken up by the two tuples are horizontally next to each other, and form a rectangle, i.e.*

$$T_{i1}.x_{right} = T_{i2}.x_{left} \text{ or } T_{i2}.x_{right} = T_{i1}.x_{left}$$
$$T_{i1}.y_{up} = T_{i2}.y_{up}$$
$$T_{i1}.y_{bottom} = T_{i2}.y_{bottom}$$

Definition 3 *Two tuples, T_{i1} and T_{i2}, are vertically adjacent, iff the spaces took up by the two tuples are vertically next to each other, and form a rectangle, i.e.*

$$T_{i1}.y_{bottom} = T_{i2}.y_{up} \text{ or } T_{i2}.y_{bottom} = T_{i1}.y_{up}$$
$$T_{i1}.x_{left} = T_{i2}.x_{right}$$
$$T_{i1}.x_{right} = T_{i2}.x_{left}$$

Now, we can define what is a *2D-mergeable pair.*

Definition 4 *Two tuples, T_{i1} and T_{i2}, are 2D-mergeable, iff the two tuples are both temporally adjacent and horizontally (vertically) adjacent.*

The structure hypothesizer learns grammar rules in a bottom-up fashion. The pseudo code of the structure hypothesizer is shown in Algorithm 2. The grammar rule set, $\mathcal{R}ules$, is initialized to contain rules associated with terminal symbols, when GSH is called for the first time. Then the algorithm detects whether there are recursive structures embedded in the records (e.g., $\mathbf{Row}, \mathbf{Row}, ...\mathbf{Row}$) , and learns a recursive rule for it if finds one (e.g., $\mathbf{Table} \rightarrow 0.7,\ [v]\ \ \mathbf{Table}\ \ \mathbf{Row}$). If the algorithm fails to find recursive structures, it starts to search for the 2D-mergeable pair (e.g., $\langle\mathbf{Equation}, \mathbf{Ski}\rangle$) that appears in the record set most frequently, and constructs a grammar rule (e.g., $\mathbf{Row} \rightarrow 1.0,\ [h]\ \mathbf{Equation\ Ski}$) for that 2D-mergeable pair. The direction field value is set based on whether the 2D-mergeable pairs are horizontally or vertically adjacent. If the Viterbi training phase cannot find a layout based on these rules, less frequent pairs are added later. When there is no more pair that is 2D-mergeable, it is possible that some training record has not been fully parsed, since some symbol pairs that are horizontally (vertically) ordered may not form rectangles. The grammar rules constructed for these symbol pairs in this case will use the extended direction values (e.g., ph, pv). After getting the new rule, the system updates the current record set with this rule by replacing the pairs in the records with the head of the rule.

After learning the grammar rules, the GSH assigns probabilities associated with these grammar rules. For each rule with head V, p is assigned to 1 divided by the number of rule that have V as the head. In order to break the symmetry among all rules, the algorithm adds a small random number to each probability and normalizes the values again. This structure learning algorithm provides a redundant set of grammar rules to the Viterbi algorithm.

5 Experimental Results of the Two-Dimensional Display Learner

In order to evaluate whether the proposed layout learner is able to acquire the correct layout, we carried out three experiments in progressively more realistic settings. All experiments were performed on a machine with a 3.06 GHz CPU and 4 GB Memory. The time the layout learner takes to learn ranges from less than 1 millisecond to 442 milliseconds per training record.

5.1 Experiment Design

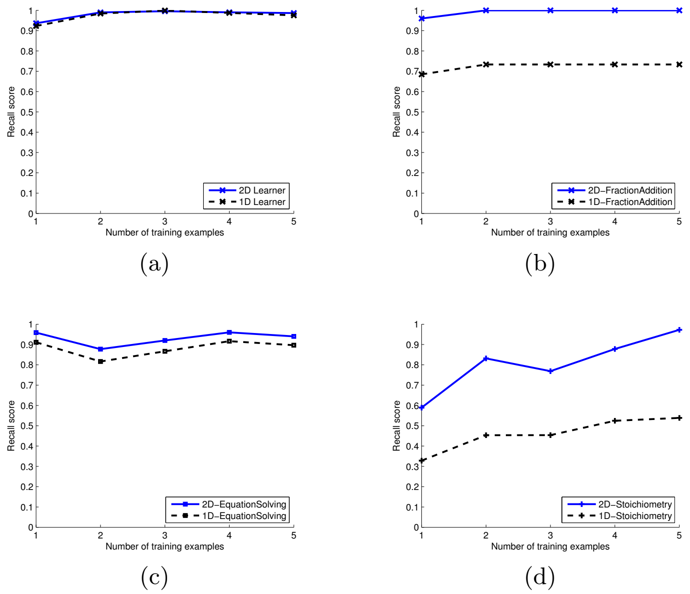
In this section, we use the 1-D layout learner (i.e., 1-D pCFG learner) as a baseline, and compare it with the proposed 2-D layout learner. In order to make the training records learnable by the 1-D layout learner, we first transform each training record into a row-ordered 1-D record, and then call the 1-D layout learner on the transformed records.

We evaluate the quality of the learned parses with the most widely-used evaluation measurements [15]: (1) the *Crossing Parentheses* score, which is the number of times that the learned parse has a structure such as ((A B) C) and the oracle parse has one or more structures such as (A (B C)) which "cross" with the learned parse structure; (2) the *Recall* score, which is the number of parenthesis pairs in the intersection of the learned and oracle parses (L intersection O) divided by the number of parenthesis pairs in the oracle parse O, i.e., (L intersection O) / O. To better understand the crossing parentheses score, we further normalize it so that it ranges from zero to one.

5.2 Experiments in Randomly Generated Synthetic Domains

In the first experiment, we randomly generate 50 oracle two-dimensional grammars. For each oracle grammar, we randomly generate a sequence of 15 training layouts[6] based on the oracle grammar. Each randomly-generated oracle grammar forms an and-or tree, where each non-terminal symbol can be decomposed by either a non-recursive or a recursive rule. Each grammar has 50 non-terminal symbols in it. For each layout, we give the layout learners a fixed number of training records. The two layout learners (i.e., the 1-D layout learner with row-based transformation and the 2-D layout learner) are trained on the 15 layouts sequentially using a transfer learning mechanism developed for the layout learner. The transfer learning mechanism is not described here due to the limited space. Then,

[6] Some layouts may be the same.



(a)

(b)

(c)

(d)

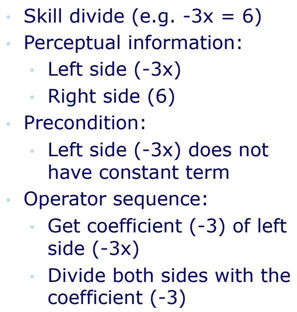
Fig. 3. Recall scores in a) randomly-generated domains, and three synthetic domains, b) fraction addition, c) equation solving, d) stoichiometry.

we generate another layout with a fixed number of testing records by the oracle grammar, and test whether the grammars acquired by the two layout learners are able to correctly parse the testing records.

Figure 3(a) presents the recall scores of the layout learners averaged over 50 grammars. Both learners perform surprisingly well. They are able to achieve close to one recall scores, and close to zero crossing parentheses scores with only five training examples per layout. To better understand the result, we take a close look at the data. Since the oracle grammar is randomly generated, the probability of getting a hard-to-learn grammar is very low. In fact, many of the training records are traces of single rows or columns, which makes learning easy. Hence, to challenge the layout learner more, we carried out a second experiment.

5.3 Experiments in Three Synthetic Domains

We examine three tutoring systems used by human students: fraction addition, equation solving, and stoichiometry, and manually construct an oracle grammar that is able to parse these three domains. Moreover, the oracle grammar can further generate variants of the existing user interfaces. For example, instead of adding two fractions together, the oracle grammar can generate interfaces that can be used to add three factions. We carry out the same training process based on this manually-constructed oracle grammar, and test the quality of the acquired grammar in three domain variants.



- Skill divide (e.g. -3x = 6)
- Perceptual information:
  - Left side (-3x)
  - Right side (6)
- Precondition:
  - Left side (-3x) does not have constant term
- Operator sequence:
  - Get coefficient (-3) of left side (-3x)
  - Divide both sides with the coefficient (-3)

Fig. 4. Original and extended production rules for divide in a readable format.

The interface of the fraction addition tutor has four rows, where the upper two rows are filled with the problem (e.g., $\frac{3}{5} + \frac{2}{3}$), and the lower two rows are empty cells for the human students to fill in. The equation solving tutor's interface is shown in Figure 1. The interface of the stoichiometry domain contains four tables of different sizes. The four tables are used to provide given values, to perform conversion, to self-explain for the current step, and to compute intermediate results. All tables are of column-based orders.

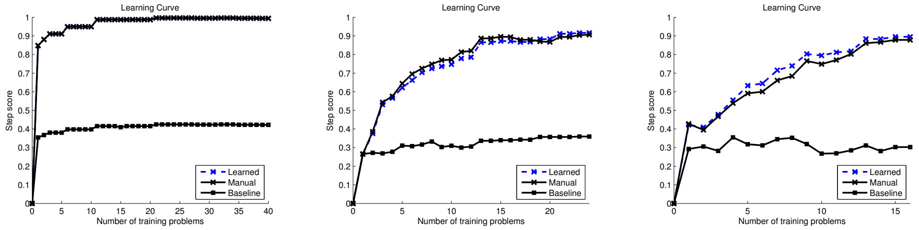
Figure 3(b), 3(c), 3(d) show the recall scores of the three domains averaged over 50 runs. Both learners achieve better performance with more training examples. We also see that the 2-D layout learner has significantly ($p < 0.0001$) higher recall scores than the 1-D layout learner in all three domains. Both fraction addition and stoichiometry contain tables/subtables of column-based orders. The row-based transformation of the 1-D layout learner removes the column information, and thus hurts the learning performance. The crossing parentheses scores for both learners are always close to zero across three domains, which indicates the acquired grammar does not generate bad "crosses" often.

6 Experimental Results within an Intelligent Agent

In order to understand how display representation learning affects agent learning effectiveness, the last experiment that we carry out is within an intelligent agent, *SimStudent*. SimStudent is an intelligent agent that inductively learns skills to solve problems from demonstrated solutions and from problem solving experience. It is an extension of programming by demonstration [2] using inductive logic programming [16] as an underlying learning technique.

Given a sequence of problem examples, the knowledge acquired by SimStudent defines "where" to look for useful information in the GUI, and "when" the useful information satisfies certain conditions, "how" to proceed. This skill knowledge is represented as production rules. Figure 4 shows an example of a production rule learned by SimStudent in its readable format[7]. The perceptual information part is acquired by the "where" learner. The precondition part is

[7] Actual production rules follows the LISP format.



(a) (b) (c)

Fig. 5. Learning curves of three SimStudents in three domains, a) fraction addition, b) equation solving, c) stoichiometry.

learned by the "when" learner. The operator function sequence part is created by the "how" learner. The rule to "divide both sides of $-3x = 6$ by -3" shown in Figure 4 would be read as "given a left-hand side (i.e., $-3x$) and a right-hand side (6) of the equation, when the left-hand side does not have a constant term, then get the coefficient of the term of the left-hand side and divide both sides by the coefficient." The "where" learner requires the layout of the interface to be given as input, which is essential for constraining the search space of the other two learning components. Previously, the agent developers need to manually encode such layout as prior knowledge, which hurts the usability of SimStudent as an authoring tool for building cognitive tutors, and fails to model display representation learning. With the layout learner, we are now able to acquire the layout based on the training problems SimStudent observes.

6.1 Experiment Design

We use the actual tutor interfaces in three tutoring domains. The 2-D layout learner is first trained on no more than five problems used to tutor human students, and sends its output to SimStudent. An automatic tutor (also used by human students) then teaches the SimStudent with the constructed/acquired layouts with one set of problems, and tests SimStudents' performance on another set of problems. Both the training and testing problems are problems used by human students. In each domain, SimStudent is trained on 12 problem sequences. Three SimStudents are compared in the experiment. One SimStudent (*manual*) is given the manually-constructed layout, one SimStudent (*learned*) is given the acquired layout, and one SimStudent (*baseline*) is given a row-based layout[8].

To measure learning gain, we calculated a *step score* for each step in the testing problem. Among all possible correct next steps, we counted the number of correct steps that were actually proposed by some applicable production rule, and reported the step score as the number of the correct next steps proposed by learned rules divided by the total number of correct next steps plus the number of incorrect next steps proposed. For example, if there were four possible correct next steps, and SimStudent proposed three, of which two were correct, and

[8] A fully flat layout performs so badly that SimStudent cannot finish learning.

one was incorrect, then only two correct next steps were covered, and thus the step score is *2/(4+1)=0.4*. Step score measures both recall and precision of the proposed next steps. We report the average step score over all testing problem steps for each curriculum.

6.2 Results

Figure 5 shows the learning curves of the three SimStudents across three domains. In all three cases, the SimStudent with a row-based layout (*baseline*) performs significantly ($p < 0.0001$) worse than the other two SimStudents. This shows the importance of the layout in achieving effective learning. Both the SimStudent with the manually-constructed layout (*manual*) and the SimStudent with the learned layout (*learn*) perform well across three domains. There is no significant difference between the two SimStudents, which suggests that the acquired layouts are as good as the manually constructed layouts.

7 Future Work

Although in this paper, we mainly focus on using the two-dimensional grammar learner to model interface layouts, the algorithm is not limited to this specific task. We would like to explore the generality of the proposed approach in other tasks. Reading tables on webpages or notes on paper are potentially interesting tasks. Sometimes, notes on a paper may not be well-aligned. In this case, the layout learning algorithm will need to be able to align these contents.

Moreover, we would like to test whether the layout learner can be used to recognize two-dimensional complex math equations. Correct 2-D layouts of tables are also important in completing calculation tasks in Excel. We would like to see whether the 2-D grammar learner can be used to help learning to perform tasks in Excel.

Finally, the complexity of the current Viterbi training algorithm increases rapidly with the lengths of the training records. Although the GSH and the caching mechanism speed up the learning process a lot, we would like to further optimize the Viterbi training phase to ensure scalability of the learning algorithm.

8 Concluding Remarks

In summary, we proposed a novel approach that models learning to perceive visual displays by grammar induction. More specifically, we extend an existing one-dimensional pCFG learning algorithm to support acquisition of a two-dimensional variant of pCFG by incorporating spatial and temporal information. We showed that the two-dimensional layout learner is more effective than the one-dimensional layout learner in general. When integrated into an intelligent agent, the SimStudent using the acquired layouts performs equally well comparing with the SimStudent given manually constructed layouts.

References

1. Li, N., Matsuda, N., Cohen, W.W., Koedinger, K.R.: Integrating representation learning and skill learning in a human-like intelligent agent. Technical Report CMU-MLD-12-1001, Carnegie Mellon University (January 2012)
2. Lau, T., Weld, D.S.: Programming by demonstration: An inductive learning formulation. In: Proceedings of the 1999 International Conference on Intelligence User Interfaces. (1998) 145–152
3. Li, N., Cohen, W.W., Koedinger, K.R.: A computational model of accelerated future learning through feature recognition. In: Proceedings of 10th International Conference on Intelligent Tutoring Systems. (2010) 368–370
4. Chi, M.T.H., Feltovich, P.J., Glaser, R.: Categorization and representation of physics problems by experts and novices. Cognitive Science **5**(2) (June 1981) 121–152
5. Li, N., Cohen, W.W., Koedinger, K.R.: Efficient cross-domain learning of complex skills. In: Proceedings of the 11th International Conference on Intelligent Tutoring Systems. (2012)
6. Li, N., Matsuda, N., Cohen, W.W., Koedinger, K.R.: A machine learning approach for automatic student model discovery. In: Proceedings of the 4th International Conference on Educational Data Minin. (2011) 31–40
7. Li, N., Cohen, W.W., Koedinger, K.R.: Problem order implications for learning transfer. In: Proceedings of the 11th International Conference on Intelligent Tutoring Systems. (2012)
8. Chou, P.A.: Recognition of Equations Using a Two-Dimensional Stochastic Context-Free Grammar. In: Proceedings of Visual Communications and Image Processing. Volume 1199. (November 1989) 852–863
9. Vanlehn, K.: Learning one subprocedure per lesson. Artificial Intelligence **31** (January 1987) 1–40
10. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM (2003) 337–348
11. Cafarella, M.J., Halevy, A.Y., Wang, D.Z., 0002, E.W., Zhang, Y.: Webtables: exploring the power of tables on the web. Proceedings of the VLDB Endowment **1**(1) (2008) 538–549
12. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 109–118
13. Lari, K., Young, S.J.: The estimation of stochastic context-free grammars using the inside-outside algorithm. Computer Speech and Language **4** (1990) 35–56
14. Li, N., Cushing, W., Kambhampati, S., Yoon, S.: Learning probabilistic hierarchical task networks as probabilistic context-free grammars to capture user preferences. Technical Report arxiv:1006.0274 (Revised), Arizona State University (2011)
15. Harrison, P., Abney, S., Black, E., Gdaniec, C., Grishman, R., Hindle, D., Ingria, R., Marcus, M.P., Santorini, B., Strzalkowski, T.: Evaluating syntax performance of parser/grammars of English. In: Natural Language Processing Systems Evaluation Workshop. Technical Report, Griffis Air Force Base, NY (1991) 71–78
16. Muggleton, S., de Raedt, L.: Inductive logic programming: Theory and methods. Journal of Logic Programming **19** (1994) 629–679