Int J Artif Intell Educ (2013) 23:71–93
DOI 10.1007/s40593-013-0005-5

RESEARCH ARTICLE

Problem Order Implications for Learning

Nan Li · William W. Cohen · Kenneth R. Koedinger

Published online: 12 October 2013

Abstract The order of problems presented to students is an important variable that affects learning effectiveness. Previous studies have shown that solving problems in a blocked order, in which all problems of one type are completed before the student is switched to the next problem type, results in less effective performance than does solving the problems in an interleaved order. However, we have no precise understanding of the reason for this effect. In addition to existing theoretical results, we use a machine-learning agent that learns cognitive skills from examples and problem solving experience, SimStudent, to provide a computational model of the problem order question. We conduct a controlled simulation study in three different math and science domains (i.e., fraction addition, equation solving and stoichiometry), where SimStudent is tutored by automatic tutors given problems that have been used to teach human students. We compare two problem orders: the blocked problem order, and the interleaved problem order. The results show that the interleaved problem order yields as effective or more effective learning in all three domains, because the interleaved problem order provides more or better opportunities for error detection and correction to the learning agent. Examination of the agent's performance shows

N. Li (✉)
Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213 USA
e-mail: nli1@cs.cmu.edu

W. W. Cohen
Machine Learning Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213 USA
e-mail: wcohen@cs.cmu.edu

K. R. Koedinger
Human Computer Interaction Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213 USA
e-mail: koedinger@cs.cmu.edu

that learning *when* to apply a skill benefits more from interleaved problem orders, and suggests that learning *how* to apply a skill benefits more from blocked problem orders.

Keywords Learning transfer · Learner modeling · Interleaved problem order · Blocked problem order

Introduction

One of the most important variables that affects learning effectiveness is the order of problems presented to students. While most existing textbooks organize problems in a blocked order, in which all problems of one type (e.g., learning to solve equations of the form $S_1/V = S_2$) are completed before the student is switched to the next problem type, it is surprising that problems in an interleaved order often yield more effective learning.

A considerable amount of research has demonstrated the effectiveness of interleaved problem orders. Shea and Morgan (1979) were the first that showed problems of a random order yields better performance in retention and transfer tests than students trained on problems of a blocked order, and named this effect as the *contextual interference (CI) effect*. The CI effect compares random problem orders and blocked problem orders, not interleaved problem orders and blocked orders, but the results should be similar since the main point is whether consecutive problems should be of the same or different types. That is, random problem orders have lots of interleaving. After that, a growing number of studies (e.g., Gabriele et al. 1987; Carnahan et al. 1990; Lee and Magill 1983; Young et al. 1993; Del Rey 1982; Sekiya et al. 1996; Jelsma and Pieters 1989) have repeatedly observed the CI effect in different tasks. Other studies on relatively complex tasks (e.g., Tsutsui et al. 1998) or novices (e.g., French et al. 1990) have yielded mixed results. To explain the CI phenomenon, researchers have proposed several hypotheses including the elaboration hypothesis (Shea and Morgan 1979), the forgetting or reconstruction hypothesis (Lee and Magill 1983), etc. More details on these hypotheses are available in Wulf and Shea (2002). Research on task switching (Monsell 2003) shows that subjects' responses are substantially slower and more error-prone immediately after a task switch, but not within the context of learning tasks. More generally, all of the above hypotheses are described in fairly ambiguous language and none have the precision of a computational theory. A computational model that demonstrates such behavior would be a great help in better understanding this widely-observed phenomena, and might reveal insights that can improve current education technologies.

In this paper, to better explain the theoretical results gathered from previous studies, we conducted a controlled-simulation study using a machine-learning agent, SimStudent, as a computational model. It provides a precise, unambiguous implementation of how and why interleaving may be effective. SimStudent was tutored by automatic tutors that simulate the ones used by human students in in classroom

studies, and was trained on real-student problems that were of blocked orders or interleaved orders. We then tested whether the advantages of interleaved problem orders over blocked problem orders are exhibited in all three domains. After that, we carefully inspected what causes such effect by inspecting SimStudent's learning processes and learning outcomes, which are not easily obtainable from human subjects. Other research on creating simulated students (VanLehn 1990; Chan and Chou 1997; Pentti Hietala 1998) and simulating expert memory (Richman et al. 1995) share some resemblance to our work. VanLehn (1990) created a learning system and evaluated whether it was able to learn procedural "bugs" like real students. To the best of our knowledge, none of the above approaches made use of the models to simulate the advantage of interleaved or random problem orders over blocked problem orders.

Additionally, there are several fruitful future steps. As we will discuss in later sections, despite the fact that SimStudent has been shown to find high-quality models of student learning across domains (Li et al. 2011), there are limitations of SimStudent as a model of student learning. In future studies, we would like to carry out more extensive experiments in response to these limitations. First, SimStudent does not assume a memory limitation, which differs from human students. It would be interesting to carry out more studies in which SimStudent has limited memory, and validate which type of learning (e.g., "how" learning or "when" learning) gains more from blocked problem orders in this case. SimStudent's learning mechanism also presents a bias towards more generalized skills over specific ones without considering computational cost. A controlled study that uses a more advanced skill selection mechanism would help us in getting a better understanding how this bias affects the learning effectiveness achieved through different problem orders. Besides, there are various aspects in human learning such as motivation that are not modeled in SimStudent. To validate our hypothesis, future studies are needed on human subjects. For example, future research could design a controlled study on human students that focuses on "when" or "how" learning, and evaluate how different problem orders affect the different learning aspects. Last, the current study shows results in well-defined domains. The content of the problems is explicitly represented in forms that do not need further interpretation. A worthy future step is to incorporate a natural language processing component into SimStudent so that SimStudent also supports problems stated in natural language. In addition, in current domains, there is a known set of rules to follow in solving the problems. We would like to extend SimStudent to less well defined domains in the future, where there is no clear definition of what is the correct solution, and test whether the same results still hold.

A Brief Review of SimStudent

SimStudent is a machine-learning agent that inductively learns skills to solve problems from demonstrated solutions and from problem solving experience. It is an extension of programming by demonstration (Lau and Weld 1998) using a variation of the version space algorithm (Mitchell 1982), inductive logic programming (Muggleton and de Raedt 1994), and iterative-deepening depth-first search

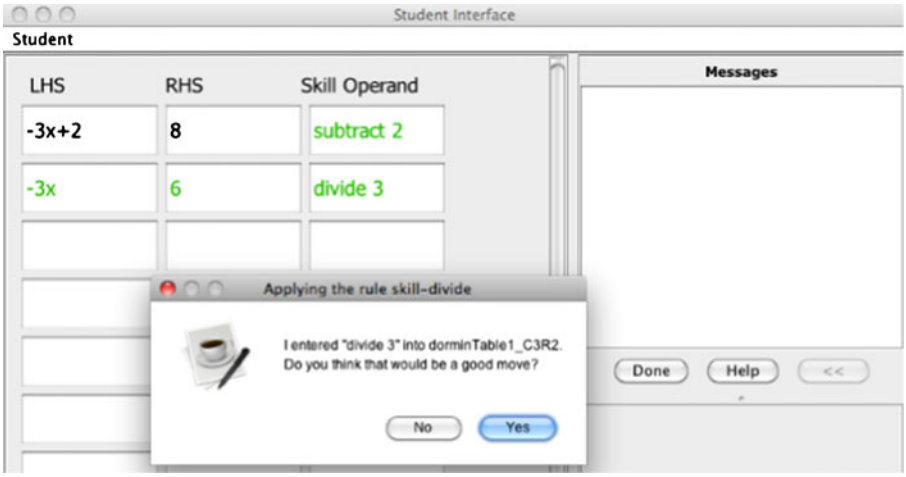

Fig. 1 The interface where SimStudent is being tutored in an equation solving domain. The given problem was $-3x + 2 = 8$, and SimStudent has been tutored to subtract both sides by 2. After entering *divide 3*, SimStudent asks the author user/tutor whether this step is correct. If SimStudent has not learned an applicable production rule, it asks the author to demonstrate a good next step and then learns a production rule to reproduce steps like this

as underlying learning techniques. Figure 1 shows a screenshot of the interface used to tutor SimStudent to solve algebra equations. Figure 2 presents how SimStudent traces each demonstrated step to learn skill knowledge. We have recently integrated representation learning (Li et al. 2010) to SimStudent's skill learning mechanisms, and have shown that the extended SimStudent learns skills that are comparable to or better than the skills acquired by the original SimStudent, with minimal manual construction of prior knowledge (Li et al. 2012a). Moreover, the extended SimStudent can be used to discover student models that predict human student behavior better than human-generated student models (Li et al. 2011). This suggests that SimStudent is a good model of human learning, since it is able to pick up important instructional details that may be overlooked by domain experts.

As representation learning is not the key to this paper, in the rest of this section, we will focus on reviewing the skill learning mechanisms of the extended SimStudent. For full details on both the representation learning and skill learning mechanisms, please refer to Li et al. (2012c).

Production Rules

SimStudent learns production rules as skills to solve problems. During the learning process, given the current state of the problem (e.g., $-3x = 6$), SimStudent first tries to find an appropriate production rule that proposes a plan that generates an action for the next step (e.g., *(coefficient* $-3x$ *?coef) (divide ?coef)*). If it finds a plan and receives positive feedback, it continues to the next step. If the proposed next step is

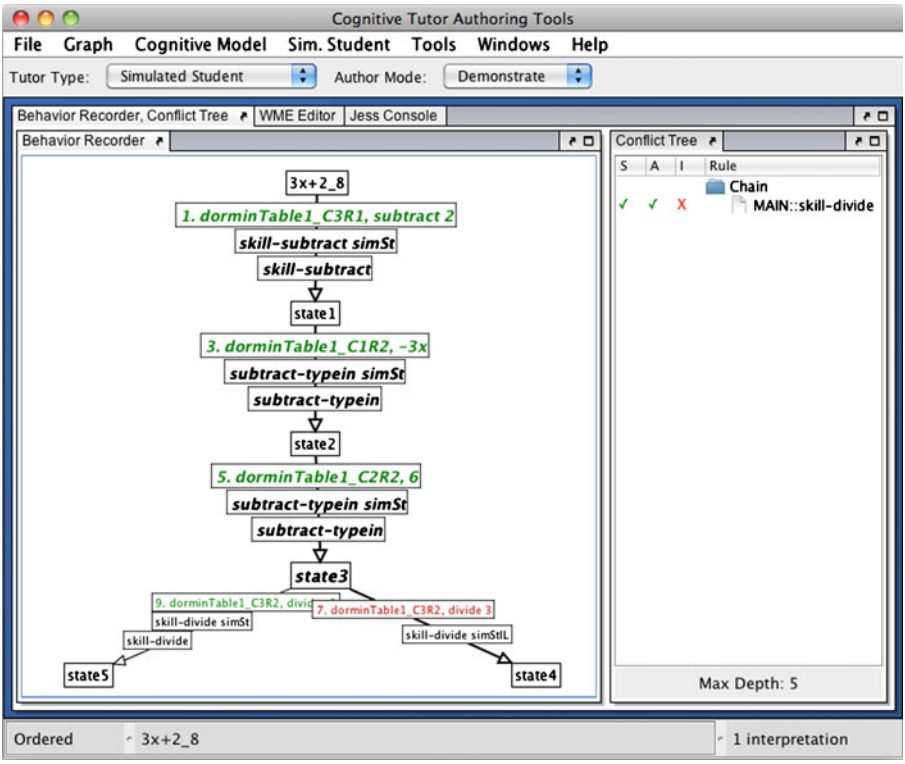

Fig. 2 CTAT's (Aleven et al. 2009) behavior recorder is used to show how SimStudent traces each demonstrated step. Each entry in a GUI element is traced. In this example, the state changes (steps) are the four entries in the table cells shown in Fig. 1. These are either traced by an existing production rule or used to learn a new production rule. The conflict tree panel in the figure shows that SimStudent applied the skill "divide" incorrectly (i.e., dividing both sides by 3 instead of -3)

incorrect, negative feedback and a correct next step demonstration are provided to SimStudent. The learning agent will attempt to learn or modify its production rules accordingly. If it has not learned enough skill knowledge and fails to find a plan, a correct next step is directly demonstrated to SimStudent for later learning. While other feedback mechanisms such as a human tutor are possible, in this paper, we use automated tutors to provide positive and negative feedback, and demonstrate the correct next step to SimStudent.

Figure 3 shows an example of a production rule learned by SimStudent in LISP format. A production rule indicates "where" to look for information in the interface, "how" to change the problem state, and "when" to apply a rule. For example, the perceptual information part of the production shown in Fig. 3 shows three paths that point to three cells in the interface, the two input cells (i.e., the left-hand side and the right-hand side), and the output cell. The rule to "divide both sides of $-3x = 6$ by -3" shown in Fig. 3 would be read as "given a left-hand side ($-3x$) and a right-hand side (6) of the equation, when the left-hand side does not have a constant term,

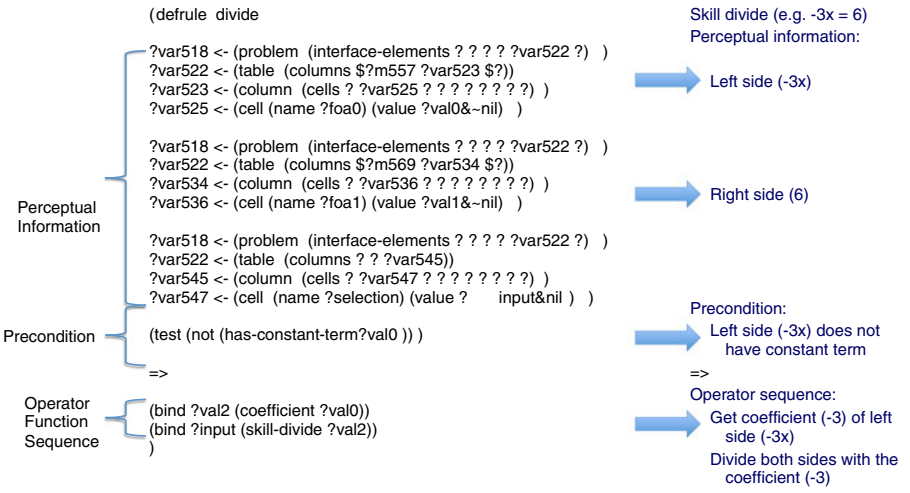

Fig. 3 A production rule for the skill divide in JESS (Forgy 1982), a LISP-like production rule representation implemented in Java, on the *left* and in English on the *right*. Notice the three parts of the production rule, the perceptual information (where), the precondition (when), and the operator function sequence (how), which correspond with different learning mechanisms

then get the coefficient of the term on the left-hand side and divide both sides by the coefficient."

Note that besides the common if-part and then-part in traditional production rules (i.e, the when-part and how-part), there is a separate where-part in the problem-solving skill. This is one of the distinctive elements of SimStudent, which is the emphasis on perceptual information retrieval. This emphasis on perceptual learning has been shown to be one essential component in human knowledge acquisition (Chase and Simon 1973; Koedinger and Anderson 1990). It is different from other cognitive architectures as most of the existing cognitive architectures (e.g., Laird et al. 1987; Anderson 1993) do not distinguish perceptual knowledge from conceptual knowledge. One exception comes from ICARUS (Langley and Choi 2006), which does have a perceptual memory and a conceptual knowledge base, but focuses more on concept knowledge learning (Li et al. 2012b), and does not model perceptual learning.

Skill Learning Mechanisms

Ohlsson's (2008) argues that there should be multiple learning models employed during different learning phases in intelligent systems. We follow this line of research, and model how one learning mechanism is able to aid other learning processes in an intelligent system. As there are three main parts in a production rule, SimStudent's learning mechanism also consists of three parts: a "where" learner, a "when" learner, and a "how" learner.

Before learning, SimStudent is given a perceptual hierarchy, a set of (ideally simple) *feature predicates* and a set of (ideally simple) *operator functions* as prior

knowledge. A perceptual hierarchy specifies the layout of the elements in the graphical user interface (GUI) that SimStudent is interacting with. The elements in the interface are typically organized in a tree structure. For example, in the equation solving domain, interface elements can be of type table, column and cell. In Fig. 1, the interface contains one table node as the root of the tree. This table node links to three column nodes. For each column node, it has multiple cells as its children, which are also leaves of the tree.

Each feature predicate is a boolean function that describes relations among objects in the domain. For example, *(has-coefficient* $-3x$*)* means $-3x$ has a coefficient. SimStudent uses these feature predicates to understand the state of the given problems.

Operator functions specify basic functions (e.g., add two numbers, get the coefficient) that SimStudent can apply to aspects of the problem representation. Operator functions are divided into two groups, domain-independent operator functions and domain-specific operator functions. Domain-independent operator functions can be used across multiple domains, and tend to be simpler (like standard operations on a programming language). Examples of such operator functions include adding two numbers, *(add 1 2)* or copying a string, *(copy* $-3x$*)*. These operator functions are not only useful in solving equations, but can also be used in other domains such as multi-column addition and fraction addition. Because these domain-general functions are involved in domains that are acquired before algebra, we can assume that real students know them prior to algebra instruction. Because these domain-general functions can be used in multiple domains, there is a potential engineering benefit in reducing or eliminating a need to write new operator functions when applying SimStudent to a new domain.

Domain-specific operator functions, on the other hand, are more complicated functions, such as getting the coefficient of a term, *(coefficient* $-3x$*)*, or adding two terms. Performing such operator functions implies some domain expertise that real students are less likely to have. Domain-specific operator functions tend to require more knowledge engineering or programming effort than domain-independent operator functions. For example, compare the "add" domain-independent operator function with the "add-term" domain-specific operator function. Adding two numbers is one step among the many steps in adding two terms together (i.e., parsing the input terms into sub-terms, applying an addition strategy for each term format, and concatenating all of the sub-terms together).

Note that operator functions are different from *operators* in traditional planning systems, operator functions have no explicit encoding of preconditions and may not produce correct results when applied in context. Thus, SimStudent is different from traditional planning algorithms, which can engage on speed-up learning. SimStudent engages in knowledge-level learning (Dietterich 1986), and inductively acquires complex reasoning rules represented as production rules.

The "where" learner acquires knowledge about where to find useful information in the GUI. For example, for the step *divide* -3, $-3x$ and 6 are the useful information, the GUI elements associated with them are *Cell21* and *Cell22*. The learning task is to find paths that identify such elements. Recall that all of the elements in the interface are organized in a tree structure. For each cell, SimStudent uses a *deep*

feature representation learning mechanism that acquires knowledge on how to further parse the content in each cell into a cell parse tree Li et al. (2010). The representation learner extends an existing grammar induction algorithm to support feature learning and transfer learning from unlabeled or lightly annotated input (e.g., $-3x = 6$).[1]

When given a set of positive examples (i.e., GUI elements associated with useful information in the steps), the learner carries out a specific-to-general learning process (e.g., from *Cell21* to *Cell?1* to *Cell??*). It finds the most specific paths that cover all of the positive examples. There are two ways to reach a percept node in the interface: 1) by the exact path to its exact position in the tree, or 2) by a generalized path to a set of GUI elements that may have a specific relationship with the GUI element where the next step is entered (e.g., cells above next step). A generalized path has one or more levels in the tree that are bound to more than one node. For example, a cell in the second column and the third row, *Cell23*, can be generalized to any cell in the second column, *Cell2?*, or any cell in the table, *Cell??*. In the example shown in Fig. 3, the where-part of the production rule, i.e., *?var525* and *?var536*, specifies two cells in row two, and is thus overly-specific. The production produces a next step only when the left-hand side and right-hand side of the current step are in row two. The learner searches for the least general path in the version space formed by the set of paths to training examples (Mitchell 1982). This process is done by a brute-force depth-first search. For example, if only given the example $-3x = 6$ in row two, the production rule learned as shown in Fig. 3 has an over-specific where-part. If given more examples in other rows (e.g., $4x = 12$ in row three), the where-part will be generalized to any row in the table.

The "when" learner acquires the precondition of the production rule that describes the desired situation to apply the rule (e.g., *(not (has-constant ?var0))*) given a set of *feature predicates*. Each predicate is a boolean function of the arguments that describes relations among objects in the domain. For example, *(has-coefficient* $-3x$*)* means $-3x$ has a coefficient. The "when" learner utilizes FOIL (Quinlan 1990) to acquire the precondition as a set of feature tests. FOIL is an inductive logic programming system that learns Horn clauses from both positive and negative examples expressed as relations. For each rule, the feature test learner creates a new predicate that corresponds to the precondition of the rule, and sets it as the target relation for FOIL to learn. The arguments of the new predicate are associated with the percepts. If a step is either demonstrated to SimStudent or receives positive feedback, that step is a positive example for FOIL; otherwise, a negative example. FOIL carries out an iterative learning process, where it acquires clauses that separate positive examples from negative examples. For example, *(precondition-divide* $?percept_1$ $?percept_2$*)* is the precondition predicate associated with the production rule named "divide". *(precondition-divide* $-3x$ *6)* is a positive example for it. The feature test learner computes the truthfulness of all predicates bound with all possible permutations of percept values, and sends it as input to FOIL. Given these inputs, FOIL will acquire a set of clauses formed by feature predicates describing the precondition predicate.

[1] An example of lightly annotated input would be the expression "$3x = -6$" with the substring "3" labeled as salient.

The learning process is from general to specific. When FOIL only has positive examples, it starts with the most general condition, in which any situation is considered to be desirable. Later, it gradually narrows its condition based on negative examples. Hence, negative examples are important in avoiding to learn overly general skills.

The last component is the "how" learner which acquires knowledge about how to change the problem state. Given all of the positive examples and a set of *basic operator functions* (e.g., *(divide ?var)*), the "how" learner attempts to find a shortest operator function sequence that explains all of the training examples using iterative-deepening depth-first search. For each positive example action record, R_i, the learner takes the percepts, $R_i.percepts$, as the initial state, and sets the step, $R_i.step$, as the goal state. We say an operator function sequence *explains* a percepts-step pair, $\langle R_i.percepts,\ R_i.step\rangle$, if the system takes $R_i.percepts$ as an initial state and yields $step_i$ after applying the operator functions. For example, if SimStudent first receives a percepts-step pair, $\langle (2x,\ 2),\ (divide\ 2)\rangle$, both the operator function sequence that directly divides both sides with the right-hand side (i.e., *(bind ?output (divide 2))*), and the sequence that first gets the coefficient, and then divides both sides with the coefficient (i.e., *(bind ?coef (coefficient 2x ?coef)) (bind ?output (divide ?coef))*) are possible explanations for the given pair. Since we have multiple example action records for each skill, it is not sufficient to find one operator function sequence for each example action record. Instead, the learner attempts to find a shortest operator function sequence that explains all of the $\langle percepts,\ step\rangle$ pairs using iterative-deepening depth-first search within some depth-limit. As in the above example, since *(bind ?output (divide 2))* is shorter than (i.e., *(bind ?coef (coefficient 2x ?coef)) (bind ?output (divide ?coef))*), SimStudent will learn this operator function sequence as the how-part. Later, it receives another example, $-3x = 6$, and another percepts-step pair, $\langle (-3x,\ 6),\ (divide\ -3)\rangle$. The operator function sequence that divides both sides with the right-hand side is not a possible explanation any more. Hence, SimStudent modifies the how-part to be the longer operator function sequence *(bind ?coef (coefficient ?rhs)) (bind ?output (divide ?coef))*.

Problem Order Study

To get a better understanding of how and why problem orders affect learning efficiency, we carried out a controlled simulation study on SimStudent given different problem orders.

Methods

To ensure the generality of the results, we selected three different math and science domains: fraction addition, equation solving, and stoichiometry, all of which have associated cognitive tutors and student data in Datashop (Koedinger et al. 2010). Both the training and testing problems were selected from problems solved by human students in prior classroom studies. In these prior studies, human students were taught by automated tutors in the three domains. In order to get a more precise computational

model, in this study, we replaced human students with a learning agent, SimStudent, and tutored SimStudent with automatic tutors that simulate the automatic tutors used by human students. The set of feature predicates and the set of operator functions provided to SimStudent as prior knowledge were sufficient to solve the problems.

Fraction Addition

In the fraction addition domain, SimStudent was given a series of fraction addition problems of the form

$$\frac{numerator_1}{denominator_1} + \frac{numerator_2}{denominator_2}$$

All numerators and denominators are positive integers. The problems are of three types in the order of increasing difficulty:

1. *Easy problems*, where the two addends share the same denominators (i.e., $denominator_1 = denominator_2$, e.g., $1/4 + 3/4$).
2. *Medium problems*, where one denominator is a multiple of the other denominator (i.e., $GCD(denominator_1, denominator_2) = denominator_1$ *or* $denominator_2$, e.g., $1/2 + 3/4$).
3. *Hard problems*, where no denominator is a multiple of the other denominator (e.g., $1/3 + 3/4$). In this case, students need to find the common denominator (e.g., 12 for $1/3 + 3/4$) by themselves.

Both the training and testing problems were selected from a classroom study of 80 human students using an automatic fraction addition tutor (Stampfer et al. 2011). The number of training problems is 20, and the number of testing problems is 6. Among the 20 training problems, there were 5 problems of type 1, 5 problems of type 2, and 10 problems of type 3. Among the 6 testing problems, there were 3 problems of type 1, 1 problems of type 2, and 2 problems of type 3, as shown in Table 1.

Equation Solving

The second domain in which we tested SimStudent is equation solving. Equation solving is a more challenging domain since it requires more complicated prior knowledge to solve the problem. For example, it is hard for human students to learn what

Table 1 Number of problems used for each domain

Domain	# of problems in training phase			# of problems in testing phase		
	Type 1	Type 2	Type 3	Type 1	Type 2	Type 3
Fraction Addition	5	5	10	3	1	3
Equation Solving	6	4	2	9	1	1
Stoichiometry	4	2	2	1	1	1

is a coefficient, and what is a constant. Also, adding two terms together is more complicated than adding two numbers.

In this experiment, we evaluated SimStudent based on a dataset of 71 human students in a classroom study using an automatic tutor, CTAT (Aleven et al. 2009). The problems are also in three types:

1. Problems of the form $S_1 + S_2V = S_3$,
2. Problems of the form $V/S_1 = S_2$,
3. Problems of the form $S_1/V = S_2$,

where S_1 and S_2 are signed numbers, and V is a variable. Note that the terms in the above problem forms can appear in any order, and may be surrounded with parenthesis. As summarized in Table 1, there were 12 training problems, and 11 testing problems in the experiment. Among the 12 training problems, there were 6 problems of type 1, 4 problems of type 2, and 2 problems of type 3. Among the 11 testing problems, there were 9 problems of type 1, 1 problems of type 2, and 1 problems of type 3.

Stoichiometry

Lastly, we evaluated SimStudent in a chemistry domain, stoichiometry. Stoichiometry is a branch of chemistry that deals with the relative quantities of reactants and products in chemical reactions. We selected stoichiometry because it is different in nature from equation solving and fraction addition. In the stoichiometry domain, SimStudent was asked to solve problems such as "How many moles of atomic oxygen (O) are in 250 grams of P_4O_{10}? (Hint: the molecular weight of P_4O_{10} is 283.88 g P_4O_{10} / mol P_4O_{10}.)". Eight training problems and three testing problems were selected from a classroom study of 81 human students using an automatic stoichiometry tutor (McLaren et al. 2008).

To solve the problems, SimStudent needs to acquire three types of skills:

1. Unit conversion (e.g., 0.6 kg H_2O = 600 g H_2O). An example of a type 1 problem is "How many grams (g) are in 10.6 milligrams (mg) of wood alcohol (COH4)?".
2. Molecular weight (e.g., There are 2 moles of P_4O_{10} in 283.88 × 2 g P_4O_{10}). A type 2 problem is "What is the number of moles of alcohol / kg of H2O in a solution of 6.00 g COH4 in 100.0 g of H2O? (Hint: the molecular weight of COH4 is 32.04 g COH4 / mol of COH4)".
3. Composition stoichiometry (e.g., There are 10 moles of O in each mole of P_4O_{10}). A type 3 problem is similar to "How many grams of Ba are in exactly 3.00 moles of YBa2Cu3O7 (a superconductor)? (Hint: the molecular weight of Ba is 137.33 g Ba / mol Ba.)".

The problems are of three types ordered in increasing difficulty, where each later type adds one more skill comparing with its former type.[2] As presented in Table 1,

[2] Although the problems are written in natural language here, when SimStudent being tutored, the problems are transformed into machine readable format.

there were 8 training problems, and 3 testing problems in the experiment. Among the 8 training problems, there were 4 problems of type 1, 2 problems of type 2, and 2 problems of type 3. Among the 3 testing problems, there were 1 problems of type 1, 1 problems of type 2, and 1 problems of type 3.

To test the generality of our computational model, the three selected domains represent skill knowledge of different types. In the fraction addition domain, the production rules of higher order are more general and can replace the production rules of lower order (i.e., the production rules acquired from problems of type 3 are enough to solve the problem in every case). In the equation solving domain, some production rules acquired from one type of problems are separate from the other production rules and can only be applied to this specific type of problems. In the stoichiometry domain, production rules learned from problems of lower order can be used to partially solve problems of higher order, but new production rules need to be acquired to solve problems of higher order. The different nature of the three domains presents different challenges to the when-part and how-part learning. As we will see in later parts of this paper, this difference causes distinctive behaviors of SimStudent in the learning procedure. More specifically, in fraction addition, the key to success of learning is the how-part learning. On the contrary, in the other two domains, when-part learning is more essential in the learning procedure than the how-part learning. The question we ask in this experiment is that despite the differences among the domains, whether interleaved-order curricula yield more effective learning than blocked-order curricula across these three domains.

Blocked vs. Interleaved Problem Orders

To manipulate the order of problems given to SimStudent, for each domain, we first grouped the problems of the same type together. Since there were three types of problems, we had three groups in each domain: *group1*, *group2*, and *group3*. Although textbooks often start with easier problems followed by hard problems, to carry out a more extensive study, we also included curricula that start with harder problems. There were six different orders of these three groups. For each order (e.g., *[group1, group2, group3]*), we generated one blocked-ordering curriculum by repeating the same problems in each group right after that group's training was done (e.g., *[group1, group1′, group2, group2′, group3, group3′]*). To generate the interleaved-ordering curriculum, the same problems will be repeated once the whole set of problems were done (e.g, *[group1, group2, group3, group1′, group2′, group3′]*). For example, as shown in the first row of Fig. 4, in the fraction addition domain, the blocked order curriculum would be of the form, [1/4 + 3/4, 1/4 + 3/4, 1/2 + 3/4, 1/2 + 3/4, 1/3 + 3/4, 1/3 + 3/4], but with more problems. For the interleaved order curriculum, the problems would be shown in the order presented in the second row of Fig. 4, [1/4 + 3/4, 1/2 + 3/4, 1/3 + 3/4, 1/4 + 3/4, 1/2 + 3/4, 1/3 + 3/4]. Since we repeated the problems in different orders, the total number of training problems shown to SimStudent is double of the number of the original training problems given to human students.

After this manipulation, we ended up having 12 curricula of different orders for each domain as shown in Table 2. Six of them were blocked-ordering curricula,

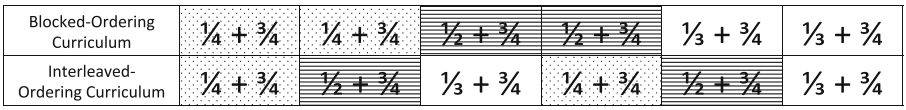

Fig. 4 Example blocked-ordering curriculum and interleaved-ordering curriculum in fraction addition. Problems in cells of the same pattern are of the same type

whereas the other six were interleaved-ordering curricula. SimStudent was trained on all these curricula, and tested by the set of testing problems. In the training phase, we record the current set of production rules SimStudent acquired every time SimStudent finishes a new training problem. Then, in the testing phase, we test the sequence of production rule sets by all of the testing problems, and report the progression of the step scores by number of training examples. Note that during the testing phase, no new production rules are acquired, and thus the order of the testing problems presented to SimStudent does not affect the step score. The results are the step scores averaged over the 6 curricula of the same type (blocked or interleaved).

Measurement

To measure learning gain, the production rules learned by SimStudent were evaluated on the set of testing problems. More specifically, during the training phase, SimStudent recorded the production rules it learned. Then, SimStudent was asked to solve problems in the test phase without resorting to any external help. In math and science problems, there is often more than one way to solve one problem. Hence, at each step, there is usually more than one production rule that is applicable. Using the knowledge it acquired in the training phase, SimStudent proposed all possible next steps in solving the problem. For each step in the testing problems, we measure a *step score* for all of the next steps proposed by SimStudent. Even if SimStudent cannot solve the whole problem, partial credit is given to the steps where SimStudent knows how to proceed. Among all possible correct next steps, we count the number of correct steps that are actually proposed by some applicable production rule, and report the step score as the number of correct next steps covered by learned rules divided by the

Table 2 12 Curricula of different orders for each domain

Blocked-ordering curricula	Interleaved-ordering curricula
1, 1′, 2, 2′, 3, 3′	1, 2, 3, 1′, 2′, 3′
1, 1′, 3, 3′, 2, 2′	1, 3, 2, 1′, 3, 2′
2, 2′, 1, 1′, 3, 3′	2, 1, 3, 2′, 1′, 3′
2, 2′, 3, 3′, 1, 1′	2, 3, 1, 2′, 3′, 1′
3, 3′, 1, 1′, 2, 2′	3, 1, 2, 3′, 1′, 2′
3, 3′, 2, 2′, 1, 1′	3, 2, 1, 3′, 2′, 1′

total number of correct next steps plus the number of incorrect next steps proposed by SimStudent, i.e.,

$$\frac{\#\text{ of Correct Next Steps Proposed}}{\text{Total \# of Correct Next Steps} + \#\text{ of Incorrect Next Steps Proposed}}$$

For example, if there are four possible correct next steps, and SimStudent proposes three, of which two are correct, and one is incorrect, then only two correct next steps are covered, and thus the step score is $2/(4+1) = 0.4$. In other words, SimStudent gets a step score 1 only if it knows all correct ways of solving the problem, and does not propose any incorrect next step. Since all possible next steps proposed by SimStudent are considered, this measurement is not sensitive to the order that production rules are applied. We report the step score averaged over all testing problem steps for each curriculum.

In addition, to better understand the cause of the difference between two problem orders, we further measured the amount of negative feedback received by SimStudent. In a previous study (Matsuda et al. 2008), it has been shown that more negative feedback often leads to more effective learning. The amount of negative feedback is evaluated by the average number of times SimStudent received negative feedback for each skill.

Results

Figures 5, 6, and 7 show the learning curves of SimStudent trained on blocked-ordering or interleaved-ordering curricula. As we can see in the graph, in all three domains, the interleaved-ordering curricula yielded as effective or more effective learning than the blocked-ordering curricula.

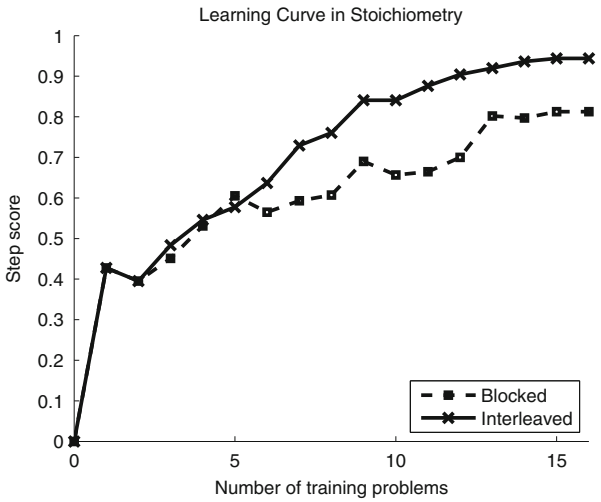

Fig. 5 Learning curves of blocked-ordering curricula vs. interleaved-ordering curricula in stoichiometry

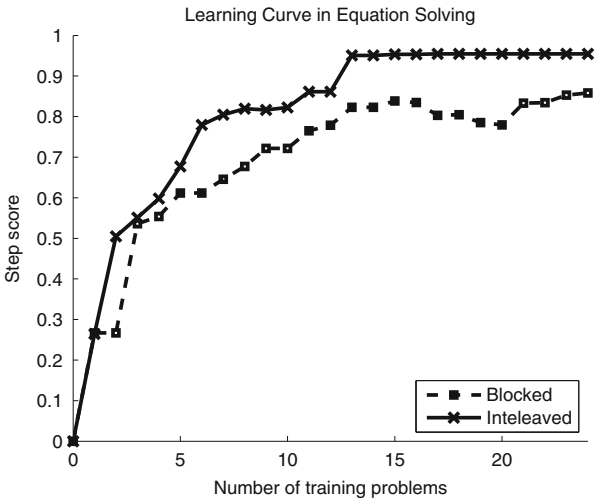

Fig. 6 Learning curves of blocked-ordering curricula vs. interleaved-ordering curricula in equation solving

In all three domains, the SimStudent trained on interleaved-ordering curricula got faster learning curves than the SimStudents given blocked-ordering curricula. This is not surprising, since the SimStudent trained on interleaved-ordering curricula got to see problems of different types sooner than the SimStudent given blocked-ordering curricula. In addition, as we will explain later, SimStudent given

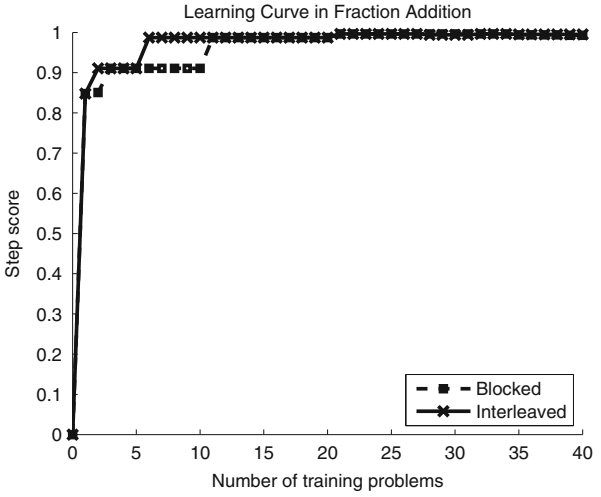

Fig. 7 Learning curves of blocked-ordering curricula vs. interleaved-ordering curricula in fraction addition

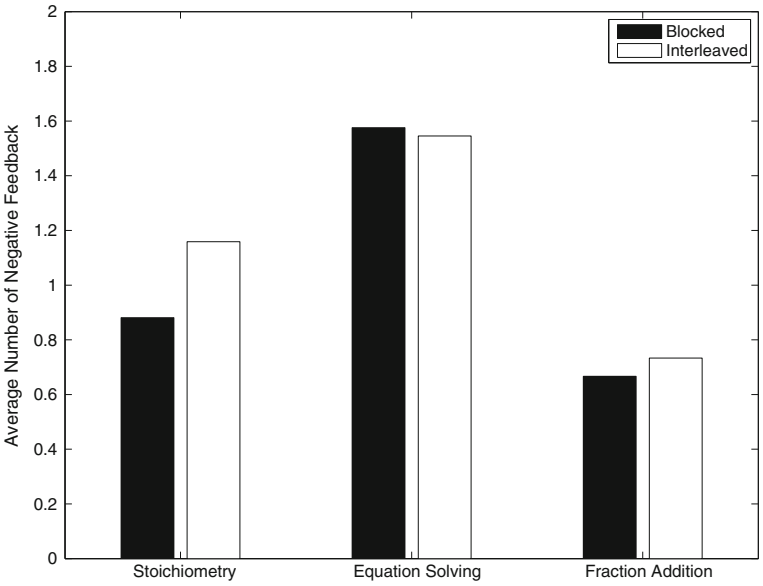

Fig. 8 The average number of times SimStudent receives negative feedback for each skill across three domains

interleaved-ordering curricula also received more negative feedback than SimStudent given blocked-ordering curricula, which assisted it in achieving more efficient learning.

More interestingly, after both SimStudents have been trained on the same set of problems, SimStudent given the interleaved-ordering curricula achieved higher or equal step scores than SimStudent given blocked-ordering curricula. In the domain of stoichiometry, the step score of the interleaved-ordering curricula was 0.944 ($SD = 0.045$), whereas the step score of the blocked-ordering curricula was 0.813 ($SD = 0.057$). A sign test between pairs of step scores achieved by the associated interleaved-ordering and blocked-ordering curricula (e.g., *[group1, group2, group3, group1′, group2′, group3′]* vs. *[group1, group1′, group2, group2′, group3, group3′]*) showed that, after trained on 40 problems, the interleaved-ordering curricula is significantly ($p < 0.05$) more effective than the blocked-ordering curricula.

Similar results were also observed in the equation solving domain. The interleaved-ordering curricula again showed a benefit ($mean = 0.955,\ SD < 0.001$ vs. $mean = 0.858,\ SD = 0.056$) over blocked-ordering curricula. The sign test also demonstrated significant ($p < 0.05$) advantages of interleaved-ordering curricula over the blocked-ordering curricula.

In fraction addition, SimStudent got a step score of 0.995, $SD = 0.008$ when trained with interleaved-ordering curricula, which is slightly higher than the step score SimStudent received (0.993, $SD = 0.007$) when trained with blocked-ordering curricula. There was no significant difference ($p = 0.50$) between the two conditions.

Implications for Instructional Design

We can inspect the data more closely to get a better qualitative understanding of why the interleaved ordering seems better and what implications there might be for improved instruction. In two of three of our domains, interleaved-ordering curricula are more advantageous than blocked-ordering curricula. Since previous studies have suggested that SimStudent is a good model of student learning (Li et al. 2011), these results can potentially provide theoretical support for the hypothesis that when teaching human students in math and science domains, an interleaved problem order yields better learning than a blocked problem order.

Impact of Negative Feedback

To better understand the cause of the advantages of interleaved ordering, we further measured the amount of negative feedback received by SimStudent, as it is one of the important factors in achieving effective learning (Matsuda et al. 2008). The amount of negative feedback is assessed by the average number of times SimStudent received negative feedback for each skill. As presented in Fig. 8, the SimStudent given interleaved-ordering problems receives significantly ($p < 0.05$, 31.5 %) more negative feedback than the SimStudent trained on blocked-ordering problems in stoichiometry, and 10.0 % more negative feedback in fraction addition.

One possible explanation for this result is when problems are of an interleaved order, SimStudent may incorrectly apply the production rules learned from previous problem types to the current problem, even if the current problem is of another type. In this case, SimStudent receives explicit negative feedback from the tutor. In contrast, when trained on blocked-ordering curricula, SimStudent has fewer opportunities for incorrect rule applications, and thus receives less negative feedback. Since the negative feedback serves as negative training examples of the "when" learning, more negative feedback in the interleaved problem order case enables SimStudent to yield more effective "when" learning compared to blocked problem orders.

For example, in stoichiometry, since the skill composition stoichiometry was only taught in problems of type three, if SimStudent was given the blocked-ordering curriculum *[group1, group1′, group2, group2′, group3, group3′]*, all of the negative examples explained for composition stoichiometry production rules were from problems of type three. In fact, one of the skills, which decides O is a substance in P_4O_{10} and outputs 1 mol P_4O_{10}, does not receive any negative feedback, since it works as originally acquired throughout *group3* and *group3′*. In this case, the when-part of the acquired skill is simply empty, which considers all situations applicable to the skill, and thus the skill was overly general. When given the interleaved-ordering curriculum *[group1, group2, group3, group1′, group2′, group3′]*, SimStudent incorrectly applied this composition stoichiometry skill to problems that need unit conversion (in *group1′*). Given the problem of how many grams (g) of COH_4 are in 10.6 mg COH_4, SimStudent returned 1 mg COH_4, which was incorrect. Given this negative feedback, SimStudent updated its overly-general production rule, and learned that to

apply this composition stoichiometry rule, the unit of the given value (e.g., mg) and the targeted unit (e.g., g) should not be convertible.

In equation solving, SimStudent received approximately the same amount of negative feedback ($p = 1, -1.9\ \%$) in the blocked problem order case and interleaved problem order case. However, a careful inspection shows that negative examples from other problem types, which one experienced more often in interleaved ordering, are sometimes more informative than those from the same problem type. Consider the blocked-ordering curriculum *[group1, group1′, group3, group3′, group2, group2′]* and its associated interleaved-ordering curriculum *[group1, group3, group2, group1′, group3′, group2′]*. During the acquisition of the skill "subtract", the SimStudent given blocked-ordering problems was first trained in *group1* to solve problems of the form $S_1V + S_2 = S_3$. It learned that when there is a constant term in the left-hand side of the equation (e.g., term S_2 is a number in $S_1V + S_2 = S_3$), subtract both sides with that number (e.g., $(subtract\ S_2)$). But it failed to learn that there must be more than one term in the left-hand side connected by a plus sign (e.g., $S_1V + S_2$). In the interleaved condition, SimStudent received negative feedback from problems of *group3* (i.e., problems of the form $S_1 = S_2/V$). It tried to subtract both sides with S_1 when given problems of type $S_1 = S_2/V$. SimStudent given interleaved-ordering problems modified its when-part when given negative feedback on such problems. The updated production rule became, "when there is a constant term that follows a plus sign in the left-hand side of the equation, subtract both sides with that number." Since this negative feedback was given to SimStudent earlier in the training process, SimStudent acquired the skill knowledge faster than the one given the blocked-ordering curriculum. Thus, in the following problems, the SimStudent given the interleaved-ordering curriculum received less negative feedback than the SimStudent given the blocked-ordering curriculum, and had a faster learning curve.

In fraction addition, as we have seen above, although SimStudent given interleaved-ordering curricula learned faster than SimStudent trained on blocked-ordering curricula, there was no significant difference between the step scores of the production rules acquired based on all the training problems. Unlike the two other domains, SimStudent in fraction addition does not have to learn different sets of skills to solve problems of different types. Instead, SimStudent learns one set of rules that handles fraction addition problems of all types. Thus, "when" learning is less essential in achieving effective learning in this domain. Suppose SimStudent was trained on the blocked-ordering curriculum *[group3, group3′, group2, group2′, group1, group1′]*. Having trained on problems of type 3 (e.g., $2/5+1/3$), SimStudent learned that it should first calculate the least common multiple of the two denominators of the addends, and then covert the fractions to get the answer. This set of skills also applies to problems of type 1 (e.g., $1/2 + 1/2$) and 2 (e.g., $1/2 + 3/4$). Therefore, no negative feedback is needed. The interleaved-ordering curriculum is no more beneficial than the blocked-ordering curriculum. This bias towards more generalized skills over specific ones without considering computational cost appears to be a limitation of SimStudent as a model of student learning. In cases where a more general strategy invokes a more complicated procedure (like calculating the common denominators), human students may prefer to use a less general but simple strategy (such as directly copying the addend's denominator). We have recently developed a

conflict resolution strategy which could be used to prefer skills of smaller computational cost (Li et al. 2013). This extension potentially addresses this limitation of SimStudent as a student model.

Impact of Memory Limitations

We conjecture that the frequent use of blocked examples in textbooks might relate to perceived memory limitations of students. The version of SimStudent used in these experiments, currently does not have any severe memory (or retrieval) limitations (e.g., it remembers all past examples no matter how long ago). SimStudent would need to have some memory limitations if it were to have a bigger knowledge base or to better model humans. If it did, the benefits for blocking may go up, and in particular for "how" learning. There are different models of memory limitations. To see how memory limitation changes the behavior of SimStudent, consider a fixed memory size for SimStudent, which means SimStudent is only able to remember a fixed number of most recent training examples. SimStudent receives positive training examples of "how" learning only when the current step is demonstrated or SimStudent applies a production rule correctly. Hence, in the blocked problem order case, SimStudent maintains all the training examples of the current problem type unless the number of training examples exceeds the memory limit. In contrast, when trained on interleaved-ordering curricula, SimStudent needs to remember training examples for multiple problem types. For any specific production rule, the number of stored training examples within the threshold will be smaller than that given a blocked-ordering curricula, which could result in less effective learning than the blocked-ordering case. This suggests that theoretical results may change when memory limitations are modeled.

"When" learning, on the other hand, is not affected as much by memory limitations because of a different inductive bias. "When" learning starts with the most general condition and makes the condition more specific when negative examples are received. In contrast, function operator sequence (how) learning is driven by positive examples and will create more complex sequences only when multiple positive examples are received.

This also relates to VanLehn's work on "learning one subprocedure per lesson" (VanLehn 1987). If a subprocedure is achieved in the same way, that is, with the same how-part in the production rule, then as Vanlehn suggested, problems of blocked orders are more beneficial. However, for production rules/procedures to differentiate across subgoals, the when-part needs to be acquired and in that case, interleaving problems of different types is important.

Impact on Error Patterns

Additionally, we have carried out a study of how different problem orders affect the types of errors SimStudent makes. We analyzed the dataset of the in-classroom study, in which human-students were taught to add fractions using an automated tutor. There are 3 most common types of errors: directly copying the denominator of the addends as the common denominator, directly copying the numerator of addends

without conversion, and getting the smaller denominator of the addends as the common denominator. The list of feature predicates and the list of operator functions provided to SimStudent are shown in Table 3. The feature predicates and operator functions that are used in the final production rules are in bold. As we can see, the provided operator functions are basic skills that are often used in maths domains.

During training, both the SimStudent trained on interleaved-ordering curricula and the SimStudent given blocked-ordering curricula produced human-like errors. In fact, all of the errors made by SimStudent were among these three types of errors. The SimStudent trained on interleaved-ordering curricula made mistakes earlier during training than the SimStudent given blocked-ordering curricula, since SimStudent got more opportunities to apply skills to problems of incorrect types in interleaved-ordering curricula. The blocked order problems yield more errors than the interleaved order problems. Among the curricula, the ones that start with simpler problems (e.g., $1/4 + 3/4$) yielded more errors than the other curricula, since the production rules acquired from these problems are overly specific for harder problems such as $1/3 + 1/4$. On the other hand, for curricula that start with the hardest problems (i.e., problems of type 3), SimStudent directly learned the most general production rules that apply to the simpler cases as well. In this case, fewer errors were generated in later segments of the curricula. Interestingly, none of the provided feature predicates was used in the acquired production rules. We took a closer look at the data and found out that since the most general set of production rules applies to all fraction addition problems, the information retrieval paths were sufficient in differentiating which production rule to apply. Thus, there was no need in acquiring additional feature tests. The errors produced were mostly caused by the incorrectly acquired how-part in production rules. Since all of the errors made were human-like errors, there was no difference between the types of errors made by SimStudent given problems in different orders.

Summary

In summary, the study provides theoretical support for the claim that learning when to apply a skill benefits more from interleaved problem orders. We also argued that

Table 3 Prior knowledge provided to SimStudent in fraction addition

Feature predicates	Operator functions
is-greater-number(?val0, ?val1)	**copy(?val0)**
is-coprimed(?val0, ?val1)	greater-number(?val0, ?val1)
is-multiple-of(?val0, ?val1)	**add(?val0, ?val1)**
	subtract(?val0, ?val1)
	multiply(?val0, ?val1)
	divide(?val0, ?val1)
	least-common-multiple(?va0, ?val1)

learning how to apply a skill benefits more from blocked problem orders. This specialization of optimal instruction to when vs. how learning may explain why the differences on step scores of SimStudent between interleaved-ordering curricula and blocked-ordering curricula in equation solving and stoichiometry were larger than the difference in fraction addition. "When" learning is more challenging in the equation solving and stoichiometry domains, while "how" learning is more essential in the fraction addition domain. Therefore, when tutoring students in domains that are more challenging in "how" learning, we suggest that the problems presented to students should start with more blocked orders. If the learning task requires more rigorous "when" learning, interleaved-ordering problems should be preferred.

Related Work

The main objective of this work is to better understand how and why problem orders affect learning outcome. A considerable amount of research has demonstrated the effectiveness of interleaved problem orders. Shea and Morgan (1979) were the first that showed problems of a random order yields better performance in retention and transfer tests than students trained on problems of a blocked order, and named this effect as the *contextual interference (CI) effect*. The CI effect compares random problem orders and blocked problem orders, not interleaved problem orders and blocked orders, but the results should be similar since the main point is whether consecutive problems should be of the same or different types. That is, random problem orders have lots of interleaving. After that, a growing number of studies (e.g., Gabriele et al. 1987; Carnahan et al. 1990; Lee and Magill 1983; Young et al. 1993; Del Rey 1982; Sekiya et al. 1996; Jelsma and Pieters 1989) have repeatedly observed the CI effect in different tasks. Other studies on relatively complex tasks (e.g., Tsutsui et al. 1998) or novices (e.g., French et al. 1990) have yielded mixed results. To explain the CI phenomenon, researchers have proposed several hypotheses including the elaboration hypothesis (Shea and Morgan 1979), the forgetting or reconstruction hypothesis (Lee and Magill 1983), etc. More details on these hypotheses are available in Wulf and Shea (2002), however, all are described in fairly ambiguous language and none have the precision of a computational theory. In contrast, SimStudent provides a precise, unambiguous implementation of how and why interleaving may be effective.

Research on task switching (Monsell 2003) shares a resemblance with our work. It shows that subjects' responses are substantially slower and more error-prone immediately after a task switch. Our work differs from this research in that we focus on learning tasks. During the learning process, switching among problems of different types also increases the cognitive load, but causes more effective learning.

Other research on creating simulated students (VanLehn 1990; Chan and Chou 1997; Pentti Hietala 1998) and simulating expert memory (Richman et al. 1995) also share some resemblance to our work. VanLehn (1990) created a learning system and evaluated whether it was able to learn procedural "bugs" like real students. To the best of our knowledge, none of the above approaches made use of the models to simulate the advantage of interleaved or random problem orders over blocked problem orders.

Concluding Remarks

In this paper, we carried out a controlled simulation study to gain a better understanding of why interleaved problem orders generate more effective learning than blocked problem orders. We used a machine-learning agent, SimStudent, which has demonstrated previous success as a theory of student learning (Li et al. 2011), in the study. We measured the learning effectiveness of SimStudent in three domains given different problem orders. The results show that given the current implementation of the intelligent agent, since the interleaved problem order yields more opportunities for error detection and correction, the SimStudent trained by interleaved-ordering curricula achieved better performance than the SimStudent trained by blocked-ordering curricula. We also discussed limitations of SimStudent (e.g., unlimited memory, bias towards more generalized skills) as a model of student learning, their potential impacts on learning effectiveness with different problem orders, and how future extensions can be made in addressing these limitations.

Acknowledgments Our thanks to the National Science Foundation (#SBE-0354420) for funding of the Pittsburgh Science of Learning Center.

References

Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R. (2009). A new paradigm for intelligent tutoring systems: example-tracing tutors. *International Journal of Artificial Intelligence in Education*, *19*, 105–154.

Anderson, J.R. (1993). *Rules of the mind*. Hillsdale: Lawrence Erlbaum Associates.

Carnahan, H., Van Eerd, D.L., Allard, F. (1990). A note on the relationship between task requirements and the contextual interference effect. *Journal of Motor Behavior*, *22*(1), 159–169.

Chan, T.-W., & Chou, C.-Y. (1997). Exploring the design of computer supports for reciprocal tutoring. *International Journal of Artificial Intelligence in Education*, *8*, 1–29.

Chase, W.G., & Simon, H.A. (1973). Perception in chess. *Cognitive Psychology*, *4*(1), 55–81.

Del Rey, P. (1982). Effects of contextual interference on the memory of older females differing in levels of physical activity. *Perceptual and Motor Skills*, *55*(1), 171–180.

Dietterich, T.G. (1986). Learning at the knowledge level. *Machine Learning*, *1*(3), 287–315.

Forgy, C. (1982). Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, *19*(1), 17–37.

French, K.E., Rink, J.E., Werner, P.F. (1990). Effects of contextual interference on retention of three volleyball skills. *Peceptual and Motor Skills*, *71*, 179–186.

Gabriele, T.E., Hall, C.R., Buckolz, E.E. (1987). Practice schedule effects on the acquisition and retention of a motor skill. *Human Movement Science*, *6*, 1–16.

Jelsma, O., & Pieters, J.M. (1989). Practice schedule and cognitive style interaction in learning a maze task. *Applied Cognitive Psychology*, *3*(1), 73–83.

Koedinger, K.R., & Anderson, J.R. (1990). Abstract planning and perceptual chunks: elements of expertise in geometry. *Cognitive Science*, *14*, 511–550.

Koedinger, K.R., Baker, R.S.J., Cunningham, K., Skogsholm, A., Leber, B., Stamper, J. (2010). A data repository for the EDM community: the PSLC DataShop. *Handbook of educational data mining* (pp. 43–55).

Laird, J.E., Newell, A., Rosenbloom, P.S. (1987). Soar: an architecture for general intelligence. *Artificial Intelligence*, *33*(1), 1–64.

Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. In *Proceedings of the twenty-first national conference on artificial intelligence* (pp. 1469–1474).

Lau, T., & Weld, D.S. (1998). Programming by demonstration: An inductive learning formulation. In *Proceedings of the 1999 international conference on intelligence user interfaces* (pp. 145–152).

Lee, T.D., & Magill, R.A. (1983). The locus of contextual interference in motor-skill acquisition. *Journal Of Experimental Psychology, Learning Memory And Cognition*, *9*(4), 730–746.

Li, N., Cohen, W.W., Koedinger, K.R. (2010). A computational model of accelerated future learning through feature recognition. In *Proceedings of 10th international conference on intelligent tutoring systems* (pp. 368–370).

Li, N., Cohen, W.W., Matsuda, N., Koedinger, K.R. (2011). A machine learning approach for automatic student model discovery. In *Proceedings of the 4th international conference on educational data mining* (pp. 31–40).

Li, N., Cohen, W.W., Koedinger, K.R. (2012a). Efficient cross-domain learning of complex skills. In *Proceedings of the 11th international conference on intelligent tutoring systems* (pp. 493–498).

Li, N., Stracuzzi, D., Langley, P. (2012b). Improving acquisition of teleoreactive logic programs through representation change. *Advances in Cognitive Systems*, *1*, 109–126.

Li, N., Cohen, W.W., Koedinger, K.R. (2012c). *Integrating representation learning and skill learning in a human-like intelligent agent*. Technical Report CMU-MLD-12-1001, Carnegie Mellon University.

Li, N., Tian, Y., Cohen, W.W., Koedinger, K.R. (2013). Integrating perceptual learning with external world knowledge in a simulated student. In *16th international conference on artificial intelligence in education* (pp. 400–410).

Matsuda, N., Cohen, W.W., Sewall, J., Lacerda, G., Koedinger, K.R. (2008). Why tutored problem solving may be better than example study: Theoretical implications from a simulated-student study. In *Proceedings of the 9th international conference on intelligent tutoring system* (pp. 111–121).

McLaren, B.M., Lim, S.-j., Koedinger, K.R. (2008). When and how often should worked examples be given to students? New results and a summary of the current state of research why isn't the science done? *Cognitive Science*, 2176–2181.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, *18*(2), 203–226.

Monsell, S. (2003). Task switching. *Trends in Cognitive Sciences*, *7*(3), 134–140.

Muggleton, S., & de Raedt, L. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming*, *19*, 629–679.

Ohlsson, S. (2008). *Computational models of skill acquisition, chapter 13* (pp. 359–395). Cambridge University Press.

Pentti Hietala, T.N. (1998). The competence of learning companion agents. *International Journal of Artificial Intelligence in Education*, *9*, 178–192.

Quinlan, J.R. (1990). Learning logical definitions from relations. *Machine Learning*, *5*(3), 239–266.

Richman, H.B., Staszewski, J.J., Simon, H.A. (1995). Simulation of expert memory using EPAM IV. *Psychological Review*, *102*(2), 305–330.

Sekiya, H., Magill, R.A., Anderson, D.I. (1996). The contextual interference effect in parameter modifications of the same generalized motor program. *Research Quarterly for Exercise and Sport*, *67*(1), 59–68.

Shea, J.B., & Morgan, R.L. (1979). Contextual interference effects on the acquisition, retention, and transfer of a motor skill. *Journal of Experimental Psychology Human Learning Memory*, *5*(2), 179–187.

Stampfer, E., Long, Y., Aleven, V., Koedinger, K.R. (2011). Eliciting intelligent novice behaviors with grounded feedback in a fraction addition tutor. In *Proceedings of the 15th international conference on artificial intelligence in education*, AIED'11 (pp. 560–562). Springer-Verlag.

Tsutsui, S., Lee, T.D., Hodges, N.J. (1998). Contextual interference in learning new patterns of bimanual coordination. *Journal of Motor Behavior*, *30*(2), 151–157.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, *31*, 1–40.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions*. Cambridge: MIT Press.

Wulf, G., & Shea, C.H. (2002). Principles derived from the study of simple skills do not generalize to complex skill learning. *Psychonomic Bulletin Review*, *9*(2), 185–211.

Young, D.E., Cohen, M.J., Husak, W.S. (1993). Contextual interference and motor skill acquisition: on the processes that influence retention. *Human Movement Science*, *12*(5), 577–600.