Available online at www.sciencedirect.com

SciVerse ScienceDirect

Int. J. Human-Computer Studies 70 (2012) 866–887

International Journal of Human-Computer Studies

www.elsevier.com/locate/ijhcs

A paradigm for handwriting-based intelligent tutors

Lisa Anthony*, Jie Yang, Kenneth R. Koedinger

Human-Computer Interaction Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Received 13 September 2011; received in revised form 4 April 2012; accepted 10 April 2012
Available online 30 April 2012

Abstract

This paper presents the interaction design of, and demonstration of technical feasibility for, intelligent tutoring systems that can accept handwriting input from students. Handwriting and pen input offer several affordances for students that traditional typing-based interactions do not. To illustrate these affordances, we present evidence, from tutoring mathematics, that the ability to enter problem solutions via pen input enables students to record algebraic equations more quickly, more smoothly (fewer errors), and with increased transfer to non-computer-based tasks. Furthermore our evidence shows that students tend to like pen input for these types of problems more than typing. However, a clear downside to introducing handwriting input into intelligent tutors is that the recognition of such input is not reliable. In our work, we have found that handwriting input is more likely to be useful and reliable when *context* is considered, for example, the context of the problem being solved. We present an intelligent tutoring system for algebra equation solving via pen-based input that is able to use context to decrease recognition errors by 18% and to reduce recognition error recovery interactions to occur on one out of every four problems. We applied user-centered design principles to reduce the negative impact of recognition errors in the following ways: (1) though students handwrite their problem-solving process, they type their final answer to reduce ambiguity for tutoring purposes, and (2) in the small number of cases in which the system must involve the student in *recognition* error recovery, the interaction focuses on identifying the student's *problem-solving* error to keep the emphasis on tutoring. Many potential recognition errors can thus be ignored and distracting interactions are avoided. This work can inform the design of future systems for students using pen and sketch input for math or other topics by motivating the use of context and pragmatics to decrease the impact of recognition errors and put user focus on the task at hand.

Keywords: Intelligent tutoring systems; Pen input; Handwriting recognition; Mathematics; Cognitive tutors; Interaction design; Human-computer interaction; Educational technology

1. Introduction

Pen-based input is one of the more *transparent* interface modalities, increasing the naturalness of an interaction by removing the "physical interface as a barrier between the user and the work [he or] she wishes to accomplish" (Abowd, 1999). Rather than focusing on translating one's intent into special-purpose input for a specific system, a user can use pen input to quickly sketch or write the intent in his or her normal mode of expressing such concepts. For example, sketching user interface diagrams (Landay and Myers, 1995; Lin et al., 2000), drawing and animating physics simulations (Cheema and LaViola, 2010; LaViola and Zeleznik, 2004), or entering mathematical formulae (Anthony et al., 2005, 2007a) are all domains in which pen input improves on the transparency of traditional interaction modalities. In our work, we have explored the affordances of handwriting and pen-based input in the domain of intelligent tutoring systems for mathematics, specifically algebra equation solving. Through a series of laboratory and classroom studies, we found evidence that using pen input instead of typing enables students to record algebraic equations more quickly, more smoothly (e.g., with fewer user errors), and with increased transfer to non-computer-based learning tasks such as tests and other assessments. We also found that both students and adults tend to

*Corresponding author. Present address: Information Systems Department, UMBC, 1000 Hilltop Circle, Baltimore, MD 21250, USA. Tel.: +1 410 755 6395; fax: +1 410 455 1073.
E-mail addresses: lanthony@cs.cmu.edu (L. Anthony), jie.yang@cs.cmu.edu (J. Yang), koedinger@cs.cmu.edu (K.R. Koedinger).

1071-5819/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.ijhcs.2012.04.003

prefer pen input for doing math tasks over other modalities such as typing or speaking.

A challenge of introducing handwriting input into intelligent tutors is that the recognition of pen input is not reliable. In no domains has it achieved 100% recognition accuracy (cf. Liu et al., 2010; Märgner and Abed, 2010). Depending on the domain, recognition errors may be more or less impactful; in tutoring systems, recognition errors might cause faulty tutoring to occur, confusing the student and harming learning. To mitigate the negative impact such recognition errors would have on the interaction, and on the student's learning, we introduced the use of *context* to improve the recognition accuracy and increase the utility and reliability of handwriting input. The tutoring system we developed considers the context of the problem being solved (e.g., what is the correct next step) to refine the handwriting recognizer's confidence about the hypothesized student input. The system design also takes advantage of what we call *task pragmatics*, that is, the needs of the task at hand (tutoring, in our case), to limit distracting interactions about potential recognition errors.

In this paper, we present an intelligent tutoring system for algebra equation solving via pen-based input that is able to reduce recognition errors by 18% and reduce unnecessary interruption of the student's learning process to one out of every four problems. The tutoring system is based on the Cognitive Tutor family of tutoring systems (Koedinger and Corbett, 2006), and uses the character recognizer and spatial math parser of the Freehand Formula Entry System (FFES) (Smithies et al., 2001) for recognition of handwritten input. We applied user-centered design principles, focusing the design of the system on the student's learning needs rather than requiring the student to change his or her behavior (e.g., to correct recognition errors) to successfully solve math problems. This approach enabled us to reduce the negative impact of the remaining recognition errors in the following ways: (1) though students handwrite their problem-solving process, they type their final answer to reduce ambiguity for tutoring purposes, and (2) in the small number of cases in which the system must involve the student in *recognition* error recovery, the interaction focuses on identifying the student's first *problem-solving* error to keep the emphasis on tutoring. Thus, recognition errors that might occur in full processing of correct solutions or in steps after the first error can be ignored. We describe in detail the evolution of the interaction paradigm, as a model for other technologies, especially educational technology, that may be improved by accepting handwriting or pen input.

In today and tomorrow's classrooms, touchscreens and tablet computers are becoming increasingly affordable and commonplace (Roschelle et al., 2007). Pen input is becoming a standard input modality whose availability allows all students to take advantage of its benefits during computer-based learning. The work presented in this paper, including the design recommendations and interaction paradigm, can inform the design of future systems for students using pen and sketch input for math or other topics. The use of a variety of types of context, such as the problem-solving context, the user's handwriting or drawing style, knowledge about the student's mastery of the material, and needs of the task (pragmatics), can be used to decrease the impact of recognition errors on the student, allowing him or her to focus on, and achieve higher marks in, the learning at hand.

1.1. Affordances of handwriting input

The use of handwriting interfaces has particular pedagogical advantages in the domain of learning environments, especially for the mathematics domain. Studies conducted as part of our motivating work found that handwriting input for math is faster than in typing interfaces (Anthony et al., 2005, 2007a). The efficiency of a handwriting interface for a math tutor allows students to complete more problems in the same amount of time (cf. Glaser, 1976). Second, the use of handwriting rather than a menu-based typing interface may result in a reduction of extraneous cognitive load on students during learning. Extraneous cognitive load (cf. Sweller, 1988), in this context, is a measure of how much mental overhead is experienced as a result of interface-related tasks while one is also trying to learn a mathematical concept. Additionally, students may prefer handwriting, especially if it makes the problem-solving process easier or more natural for them, and therefore leads to increased engagement during tutoring (cf. Elliott and Dweck, 1988).

Furthermore, in mathematics, the spatial relationships among symbols have inherent meaning, even more so than in other forms of writing. For example, the spatial placement of the x in the following expressions significantly changes the meaning: $2x$ vs. 2^x. Handwriting is a much more flexible and robust modality for representing and manipulating such spatial relationships, which become more prevalent as students advance in math training. Finally, students still learn to write before they can type, lending a higher degree of *fluency* to their written work that takes longer to develop in typing (cf. Read et al., 2001a). In our work, we have found support for this fluency factor in that students experience greater degrees of transfer to non-computer-based tasks when they enter their problem-solving solutions via handwriting than via typing or menus (Anthony et al., 2007a). These affordances encourage the adoption of handwriting and pen input into learning environments, as long as the technology is available to support it.

1.2. Motivation and approach

The technology to support handwriting input has not always been successful, affordable or prevalent enough to use in the classroom. However, in recent years, the cost of tablets and digital stylus devices has become reasonable for widespread use, reflected in their increased appearance and adoption in mainstream markets. Still, recognition and understanding of users' handwritten input is not a solved problem. Accuracy rates vary greatly depending on the task and evaluation dataset(s). For example, in the ICFHR 2010 Arabic handwriting recognition competition, recognition accuracies from six teams on seven different test

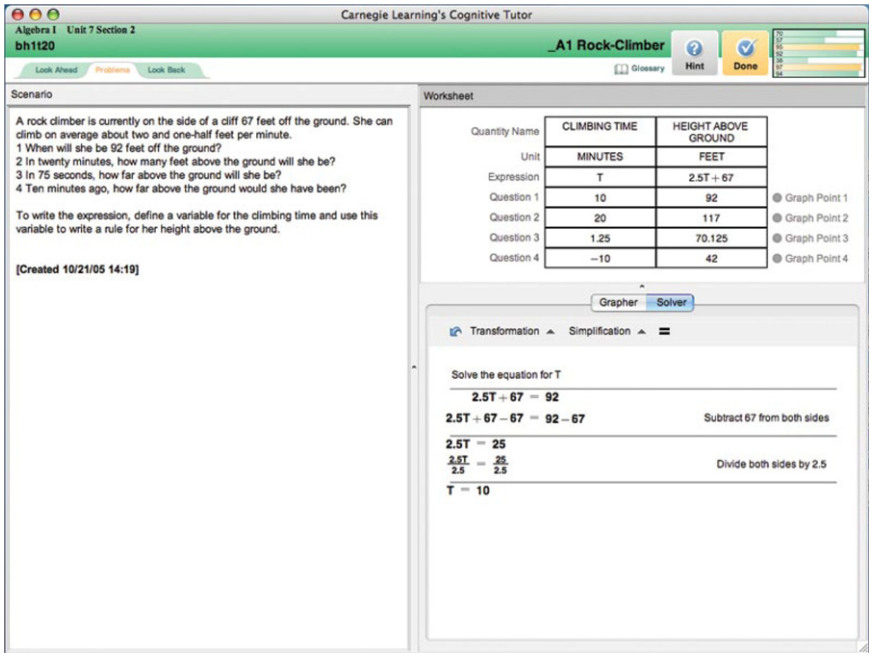


Fig. 1. A screenshot of the Cognitive Tutor interface for an algebra unit involving formulating the relationship between two variables.

datasets ranged from 67.9% to 99.7% (Märgner and Abed, 2010). Recognition rates with children on various tasks can be as low as 49.6% to 72.2% (Read, 2007). In our work, we determined that, in order to enable students to reap the benefits of using handwriting-based interaction with intelligent tutors, the interaction would have to be designed with the limitations of the recognition technology in mind. We took several steps to accomplish this: (a) we trained the recognition engine using students' writing to improve *a priori* accuracy on input from the target population in the target domain; (b) we used domain-specific context to improve accuracy even further; (c) we altered the default pedagogical intervention to avoid use of step-targeted feedback, which could be error-prone in handwriting; and (d) we designed an interaction paradigm to minimize the impact on the student of recognition errors by avoiding directly requesting the student to correct the system's recognition hypotheses. Steps (a) and (b) we regard as taking advantage of domain-specific context, whereas steps (c) and (d) we regard as capitalizing on the pragmatic needs of the learning task. In the end, a realistic user interaction paradigm was achieved, in spite of modest baseline recognition accuracy. In the next few sections, we provide background for our approach, including the intelligent tutoring system we adapted and related work on handwriting input in general, for math and for children.

1.2.1. Intelligent tutoring systems and cognitive tutors

An intelligent tutoring system (ITS) is educational software containing an artificial intelligence component (Corbett et al., 1997). Many ITSs present complex, multi-step problems and provide the individualized support that students need to complete them. The software monitors the student as he or she works at his or her own pace. By collecting information on a particular student's performance, the software can make inferences about her strengths and weaknesses, and can tailor the curriculum to address her needs.

Cognitive Tutors comprise a specific class of ITSs that are designed based on cognitive psychology theory and methods; they pose authentic problems to students and emphasize learning-by-doing (Koedinger and Corbett, 2006). Each Cognitive Tutor is constructed around a *cognitive model* of the knowledge students are acquiring, and can provide step-by-step feedback and help as students work. These tutors have been created for a variety of learning domains, including algebra, geometry, foreign languages, chemistry, computer programming and more. Cognitive Tutors for mathematics are in use in over 2600 schools in the United States. A screenshot of a typical Cognitive Tutor interface for an algebra unit is shown in Fig. 1, showing important interface components such as the worksheet and equation solver tool. In this solver, students type equations or steps of the problem and must use the Transformation and Simplification menus to perform manipulations on the equation to solve it.

1.2.2. Handwriting input for math tutors and learning

One area in which tutoring systems may be improved is with respect to the interface they provide to students for problem solving. Most ITSs use keyboard- and mouse-based windows-icons-menus-pointing (WIMP) interfaces. Such interfaces may not be ideally suited for math tutoring systems. These interfaces impose cognitive load (Sweller, 1988) on the student, *extraneous* to learning because using and learning the interface is (and should be) separable from the math concepts being learned. A more natural

interface that can directly support the standard notations for the mathematics that the student is learning could reduce extraneous cognitive load and therefore yield increased learning (cf. Sweller, 1988).

Furthermore, young children may be a particularly good audience for handwriting-based interfaces, even without considering learning. Studies have shown that children experience difficulties with the standard QWERTY keyboard, making text entry laborious and causing them to lose their train of thought – a sign of high cognitive load – even given the rise in computer use by children (Read et al., 2000). There is also some evidence that children may write more fluently when using a handwriting-based interface than a standard keyboard-and-mouse interface when entering unconstrained text (Read et al., 2001a).

Anecdotally, teachers say that students have difficulty moving from the computer tutor to working on paper. Teachers report seeing students having trouble solving problems on paper that are just like problems they recently solved on the computer with no trouble. The WIMP interface may act as a crutch: the knowledge students acquire may become most strongly activated by (or linked to) the visual cues of the interface, making it difficult for students to access their conceptual knowledge without those cues.

1.2.3. Pen input and handwriting recognition

Handwriting recognition has been an active area of research since the late 1950s (Brown, 1964; Dimond, 1957), even for mathematics (Anderson, 1968). Techniques for the recognition of handwritten mathematics range from the recognition of a page of notes after it has already been written (offline) (Fateman et al., 1996; Miller and Viola, 1998), to the recognition of a user's handwriting even while he or she is in the process of writing (online) (Belaid and Haton, 1984; Dimitriadis and Coronado, 1995). For a survey of the techniques used in handwriting recognition systems, see (Chan and Yeung, 2000). Each of the many techniques presents different speed, accuracy, and memory tradeoffs, but none of them significantly outperforms all others in every respect (Guyon and Warwick, 1998). It is difficult to quote a state-of-the-art handwriting recognition accuracy rate because recognition can be highly dependent on the task and individual writer (cf. Read, 2007; Märgner and Abed, 2010). Furthermore, citing and comparing performance evaluations can be difficult because authors do not always report all the pertinent details of the evaluation needed for interpretation and reproducibility (Lapointe and Blostein, 2009), and because public benchmark datasets are not widely available (Awal et al., 2010). New standardized metrics for evaluation of pen input performance are still being proposed and validated (Blostein et al., 2002; Zanibbi et al., 2011). A second problem is that few rigorous evaluations have been done from a usability perspective on handwriting recognizers for any domain, a weakness identified in the literature (Goldberg and Goodisman, 1991) but not strongly pursued. Many of the evaluations that do exist are now out-dated (MacKenzie and Chang, 1999; Santos et al., 1992) as recognition technology has continued to advance over the past decades.

Handwriting recognition systems for math are especially lacking in formal evaluations and are rarely evaluated for usability or other human factors. MathPad^2 is one of the few recent systems to perform a complete user study designed to gauge factors such as user performance, satisfaction, ease-of-use, and learnability along with recognition engine performance (LaViola, 2006). In that study, seven adult participants performed a variety of math-related tasks in MathPad^2, such as writing and evaluating equations and making mathematics and physics sketches. The handwriting recognizer was writer-dependent and yielded accuracy rates of 95.1%. Participants generally noted that MathPad^2 was easy to use during the study and useful for accomplishing math tasks. PenProof (Jiang et al., 2010), a system for sketching and writing geometric proofs, reported results of a simple user study with 12 students that showed positive user feedback and yielded writer-independent recognition rates of 92.1% (symbols) to 87.3% (full proofs). AlgoSketch (Li et al., 2008), an interaction layer on top of MathPaper (Zeleznik et al., 2008), is another system that has reported results of a (positive) usability evaluation (O'Connell et al., 2009), but did not also record recognition accuracy during that study. More studies are needed at the intersection of pen input recognition and human-computer interaction.

Early work in computer-aided instruction (CAI) explored the interplay between tutoring system context and recognition of handwritten math (Purcell, 1977). Our work builds on this early pioneering effort: (a) moving from the first generation of educational software into the modern generation of intelligent tutoring systems running on desktop and tablet computers rather than mainframes, and (b) moving from the early approaches to character recognition which required user training and neat printing to writer-independent recognition of potentially messy student writing.

1.2.4. Usability and user acceptance of recognition errors

In terms of handwriting recognition performance levels that are acceptable to users, LaLomia (1994) provided evidence that adults will tolerate accuracy rates in handwriting recognition for a variety of tasks (not including math) only as low as 97%. Note that human recognition rates of handwriting are around this level (Santos et al., 1992). In contrast to the adult figures, Read et al. (2003b) found that children are more tolerant of recognition errors, finding acceptance among children for accuracy rates of only 91%. Reasons for this difference in acceptability of errors include the fact that children find handwriting input to be very appealing and engaging, thus increasing their overall tolerance for the system making errors (Read, 2002). Frankish et al. (1995) explored the relationship between recognition accuracy and user satisfaction and

found that it was highly task-dependent: some tasks (such as a form-filling task) were rated as very suitable for pen-based input no matter what the recognition accuracy level was, whereas others (such as a diary task) were only rated highly when accuracy was also high. Therefore, recognition error acceptance is domain- and population-dependent. Based on the range of values reviewed here, we use the 91–97% range as a goal for an ITS that accepts handwriting input. Our approach also uses pragmatics of the task to mitigate the need for such high rates.

1.3. Evidence from foundational studies

In our work we have conducted three foundational studies that build on the prior work described in the previous sections and that provide concrete evidence for the theoretical affordances of handwriting input for learning math. In this section, we summarize the main details and results of those studies.

1.3.1. Usability of different modalities for math input

The Math Input Study (Anthony et al., 2005) focused on the following research questions: (a) Which of the common desktop input modalities is the fastest or least error-prone when entering mathematics on the computer? (b) Do these effects change significantly as the mathematics being entered increase in complexity? and (c) Which modality do users rate most highly as being natural for entering mathematics on the computer? In this within-subjects study, 48 paid participants were asked to enter mathematical equations of varying complexity using four different modalities: (1) traditional keyboard-and-mouse (typing) using the common Microsoft Equation Editor (MSEE), (2) pen-based handwriting entry (handwriting), (3) speech entry (speaking), and (4) handwriting-plus-speech (multimodal). There was no automatic handwriting or speech recognition in this study; users simply input the equations and did not get feedback about computer recognition of their input. The equations the participants entered were varied in terms of the equation length (e.g., number of symbols) and the equation complexity (e.g., number of non-keyboard symbols such as exponents).

The results indicated that handwriting was *three times faster* for entering calculus-level equations on the computer than typing using a template-based editor, and this speed impact increased as equations got more complex (namely, as characters not on the keyboard were included). In addition, user errors were *three times higher* when typing than when writing for entering math on the computer. Finally, users rated the handwriting modality as the most natural, suitable modality for entering math on the computer out of the ones they used during this study (on a 5-point Likert scale). The increased efficiency of a handwriting interface for a math tutor would allow students to accomplish more problems in the same amount of time, and the fact that students prefer handwriting might lead to increased engagement during tutoring (cf. Elliott and Dweck, 1988). As a result, the Math Input Study established that, even when not considering learning as a task, math input via handwriting is generally more usable than typical typing interfaces for math.

1.3.2. Learning of math with different input modalities

The Lab Learning Study (Anthony et al., 2007a) focused on the following research questions: (a) Do students experience differences in learning due to the modality in which they generate their answers? and (b) Do the results reported in the Math Input Study generalize to a younger population and simpler equations that can be typed easily? The study was a laboratory experiment in which 48 middle and high school students were paid to participate. Two of the modalities used in this study were: (1) typing, in which students typed out the solution in a blank text box (not MSEE); (2) handwriting, in which students wrote the solution using a stylus in a blank space on the screen. Students first copied equations in all modalities and then solved nine equations in one of the modalities. When solving problems, students saw a worked example (Pashler et al., 2007) of the same type of problem they were about to solve before the introduction of each new problem type to act as an instructional intervention. No automatic recognition of student solutions was done in this study. Feedback on their solutions was provided via a Wizard-of-Oz format: an experimenter received a screenshot of the student's solution when the student clicked a "Check my Answer!" button; the Wizard was only able to respond "Yes" or "No" and the student had to try again if he got the problem wrong. To prevent students from becoming stuck on a problem, after the third incorrect try to solve a problem, the problem turned into an example and students were required to copy the solution.

Findings from the Lab Learning Study did in fact show that the usability advantages found in the Math Input Study generalized to students performing a learning task with simpler equations. Students completed the tutoring lesson in handwriting in half the time than in typing, but experienced no significant difference in learning—the extra time students spent in the typing condition did not help their learning. In addition, students overwhelmingly chose handwriting as their favorite of the modalities that they tried during this study. In the Math Input Study, handwriting was three times as fast; here in the Lab Learning Study, handwriting was twice as fast. This difference is likely due to the simpler nature of the equations given in this study (fewer advanced characters used), the simpler nature of the typing interface used (to be more like the handwriting interface), and the addition of the learning task. Students experienced sizeable learning gains from pre-test to post-test in spite of being given only answer-level feedback, in the context of worked-examples-based instruction. This finding supports the hypothesis that worked examples are an appropriate instruction method when using handwriting input interfaces that may not be able to support step-targeted feedback. Finally, an

important finding from this study was that students experienced better transfer of their learned skills to paper-based tasks in the handwriting condition than in the typing condition: the tutor predicted student performance on the post-test in the handwriting condition significantly better than in the typing condition. Predicting paper-based performance is critical for accurate assessment of student knowledge within curricula that use ITSs.

1.3.3. Learning in cognitive tutors with handwriting input

The Cognitive Tutor Study (Anthony, 2008) focused on the following research questions: (a) Do students experience differences in learning due to the modality in which they generate their answers? (b) Do the benefits due to the presence of worked examples sufficiently counteract the disadvantages of the lack of step-targeted feedback? (c) Do the results from the Math Input Study and the Lab Learning Study for the benefits of handwriting, in terms of time, user satisfaction, and improved transfer-to-paper, generalize to a more complex tutoring system and classroom environment? and (d) Do students experience less cognitive load, measured by self-report, when they use handwriting to solve problems than when they use typing? This study was an in vivo classroom study in which eight algebra classes taught by four different teachers at two high schools worked in a modified tutor lesson that enabled handwriting input. Students were approximately aged 13–17. Data from 76 students were usable; others had to be removed due to missing either the pre-test or post-test or spending too little time in the tutor during the study (e.g., less than 30 min). Classrooms were randomly assigned to use the control tutor or one of three modified tutors. We will focus here on the three main characteristics of each tutor: modality (handwriting or typing), worked examples (yes or no), and type of feedback (step-targeted or answer-only). The three modified tutors were: (1) Typing-Examples-StepFeedback, (2) Typing-Examples-AnswerFeedback, and (3) Handwriting-Examples-AnswerFeedback. No automatic recognition of the students' solutions was done; students typed their final answer and the system checked it for correctness. The control tutor (normal Cognitive Tutor) was classified as Typing-NoExamples-StepFeedback. The material in the lesson covered two- and three-step algebra equations, and contained problem types such as $ax+b=c$, $ax+b=cx+d$, and $a/x+b=c$, with integers, decimals and large numbers (greater than 1000). The study lasted for two to three classroom periods, after which all students in a class received the post-test at the same time. The Cognitive Tutor's automatic curriculum selection mechanism was used to provide students with problems appropriate to their skill level as the tutoring sessions progressed.

The Cognitive Tutor Study found that the usability benefits of handwriting input continue to hold, in terms of total time spent during the lesson: students were 20% faster in the handwriting condition than in the others. This study also found that worked examples added value to the normal Cognitive Tutor, even without handwriting, which is positive evidence that they can be instructionally helpful in this context. In addition, step-targeted feedback was important for student learning, and handwriting, while outperforming typing input without step-targeted feedback, did not outperform typing input with step-targeted feedback and examples. Therefore, the Cognitive Tutor Study determined that step-targeted feedback is important instructionally for students in the math tutoring domain, and so the technology needs to be improved to be able to provide more than just answer-level feedback.

1.4. Early results of handwriting recognition of students' math input

Prior work showed that recognition accuracy can be highly dependent on the user and on the task (cf. Read, 2007; Märgner and Abed, 2010). Therefore, we conducted some early explorations of handwriting recognition on the target domain (algebra and math) and population (middle and high school student learners) in order to understand the baseline performance we could expect.

1.4.1. The algebra learner corpus

As mentioned, our foundational studies did not include automatic recognition of students' handwriting during problem-solving. We used these studies as opportunities to collect a large corpus of handwriting data from students so that we could train a recognition engine to the target population and domain. The corpus we collected is called the "algebra learner corpus." It primarily consists of data from the Lab Learning Study, and is based on handwriting samples from 40 middle and high school students writing algebra equations. The corpus has been hand-segmented and hand-labeled. The corpus includes 16,191 symbols grouped into 1738 equations. Twenty-two unique symbols appear in the corpus: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, x, y, a, b, c, +, −, =, (,), ___,/} (Anthony et al., 2007a).

1.4.2. Establishing baseline recognition performance

We chose the Freehand Formula Entry System (FFES) as the recognition engine for our math tutor, based on initial evaluations of various freely available recognizers on the algebra learner corpus (see (Anthony, 2008) for more information). FFES (Smithies et al., 2001) uses nearest-neighbor classification based on a 48-dimensional feature space. FFES recognizes mathematical equations in a two-component process: character recognition (the California Interface Tools (CIT) character recognizer) (Smithies et al., 2001), and mathematical expression parsing (DRACULAE) (Zanibbi et al., 2002). Stroke grouping (character segmentation) is performed via an algorithm that finds the highest-confidence grouping of a set of m recently drawn strokes, where m is the maximum number of strokes in any symbol in the recognizer's symbol set. The authors of FFES reported single-symbol accuracies of 77% for eight users when the system was *not* specifically

trained to their handwriting (writer-independent), and rates as high as 95% for eight users when the system *was* specifically trained to their handwriting (writer-dependent) (Smithies et al., 2001).

To understand the baseline recognition accuracy we could expect in our target domain and population, we conducted a suite of similar experiments on the algebra learner corpus (Anthony et al., 2007b, 2008). We tested the recognizer both on single symbols and on full equations. Table 1 shows the comparison of FFES' performance on the algebra learner corpus to previously reported results of FFES. Writer-dependent tests were conducted individually for each of the 40 users in the algebra learner corpus and final results were averaged over all users. Accuracy on symbols was about 91%, while accuracy on equations was lower, only about 78%, due to errors in the stroke grouping step and accumulated chance of errors across all symbols in an equation.

For classroom use, writer-independent performance is required since the tutoring system cannot train its handwriting recognizer for each new student. In our writer-independent experiments on the algebra learner corpus, the number of samples per symbol per user included in the training set was varied, holding out some users' data for the test set and testing on both single symbols and equations. Recognition accuracy for single symbols leveled off around 83% after training on two samples per symbol per user. Accuracy on equations leveled off around 71% at the same point.

Note that we use a normalized (Levenshtein's, 1966) string distance, a measure of the number of edits (insertions, deletions, and substitutions) required to transform one string into another, to compute accuracy on equations rather than a binary yes/no score. The accuracy calculation is as follows:

$$1-\frac{(i+d+s)}{n}$$

In this equation, i is the minimum number of insertions, d is the minimum number of deletions, and s is the minimum number of substitutions required to transform the target string into the desired string, and n is the length of the correct string.

Thus, the accuracy for an equation recognized as "$3x+10=20$" that was actually written as "$3x-1=12$" would be $1-(1+0+3)/7$ or 43%.

These are the baseline results we are working from when adding context to the recognition process. Cursory comparisons to the previously discussed levels of recognition accuracy needed for user acceptance (91–97%) might seem to discourage incorporating handwriting input into ITSs. However, because user acceptance of recognition error is highly task-dependent, and because we can take away some of the focus of the user (student) on correcting system errors, the raw recognition accuracy numbers are not sufficient criteria to decide whether to proceed. We next describe our proposed interaction paradigm that pragmatically focuses the student on his or her own problem-solving errors rather than system recognition errors; and we show that we can improve recognition accuracy through the use of context.

2. Our pen input interaction paradigm

As mentioned, the result of the handwriting recognition process is not perfect. Recognition noise still occurs, even with the additional information provided by context, as we will discuss. Our goal is to minimize the impact of this noise on the student, allowing the student and the tutoring system to focus on task pragmatics, e.g., the problem to be solved, rather than on correcting system errors. This section describes the paradigm we use to accomplish this goal; this paradigm concentrates on identifying the step on which the student made the error, rather than on knowing exactly what the student wrote for every step. Students in our paradigm *type* their final answer rather than handwriting it, because typing a number for x is easy compared to typing the full solution, and it enables the system to unambiguously determine whether the student entered a correct final answer. The basis for this approach is the observation that students rarely get the final answer correct without solving the problem correctly, so we can ignore recognition noise that might otherwise occur on those solutions by skipping recognition for them.

Fig. 2 lays out the interactive process between a tutoring system (in green boxes) and a student (in blue rounded boxes). Decision points (in red diamonds) branch in places where the tutoring system must determine whether it has enough information to move the student on to the next problem. We present each phase of the process and

Table 1
Average baseline recognition accuracy of FFES for this target domain and population compared to prior results published on FFES for other populations. Blank cells were not reported in original FFES papers.

		FFES original corpus	Algebra learner corpus
Number of users in corpus		8	40
Writer-dependent, no context	**Accuracy per symbol**	95%	91%
	Accuracy per equation (string distance)	–	78%
Writer-independent, no context	**Accuracy per symbol**	77	83%
	Accuracy per equation (string distance)	–	71%

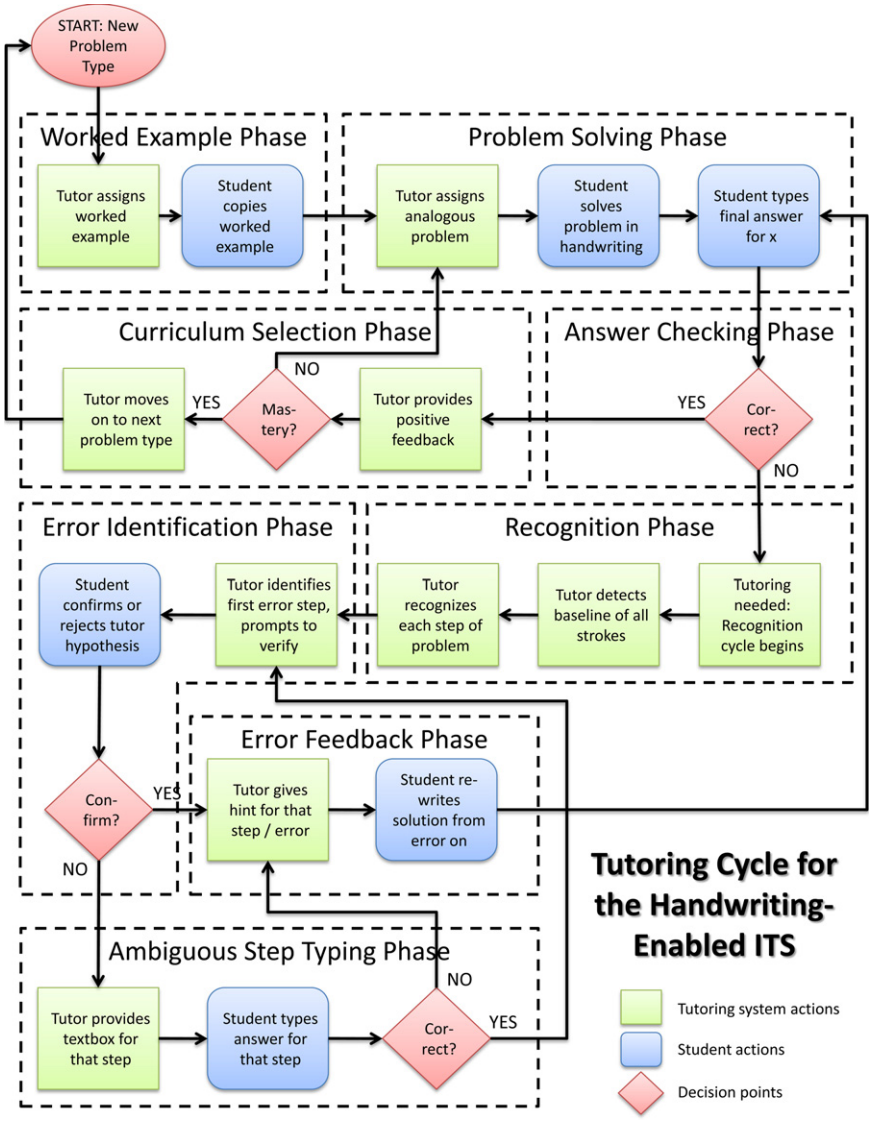


Fig. 2. The flowchart of our complete interaction paradigm for a handwriting-enabled ITS. The process begins at the upper left corner of the figure and follows the arrows, branching at certain decision points as tutoring and problem-solving occurs, until the student successfully enters the correct final answer. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

provide illustrations of how a tutoring system can implement it.

2.1. Worked example phase

To facilitate learning (cf., Pashler et al., 2007), when introducing a new problem type (for example, solving equations of a certain structural form such as $ax+b=c$), the tutoring system first presents a worked example (e.g., an example for which the full step-by-step solution is provided, Clark et al., 2006) to the student. Students are likely to have already received some group-based classroom instruction on the topics to be presented in the tutor. In Fig. 3(a), the student sees the same equation on the top of the screen as is solved for him or her on the lefthand side of the screen. The student is instructed by the software to study the example and copy it step by step, self-explaining (Chi et al., 1989) the rationale behind each operation as he or she goes along. No recognition is necessary in the Worked Example Phase. The student copies the example, successfully types the final answer on the bottom of the screen, and is allowed to move on, shown in Fig. 3(b). (See (Pashler et al., 2007; Salden et al., 2010) for a discussion of worked examples as an instructional paradigm.)

2.2. Problem solving phase

The tutoring system then assigns the student a randomly generated problem of the same surface form for the student to solve independently. The worked example remains onscreen for the student to refer to until the

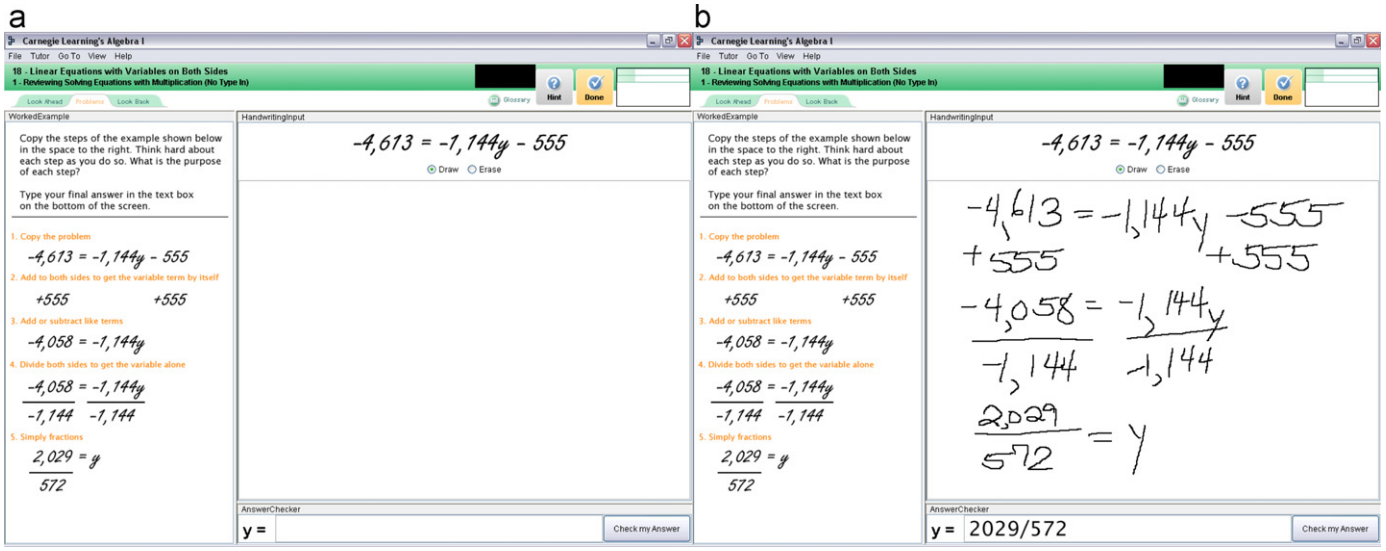


Fig. 3. Worked example phase. (a) The student receives a new problem type, and is asked to copy the example on the left. (b) The student successfully copies the example and types the final answer.

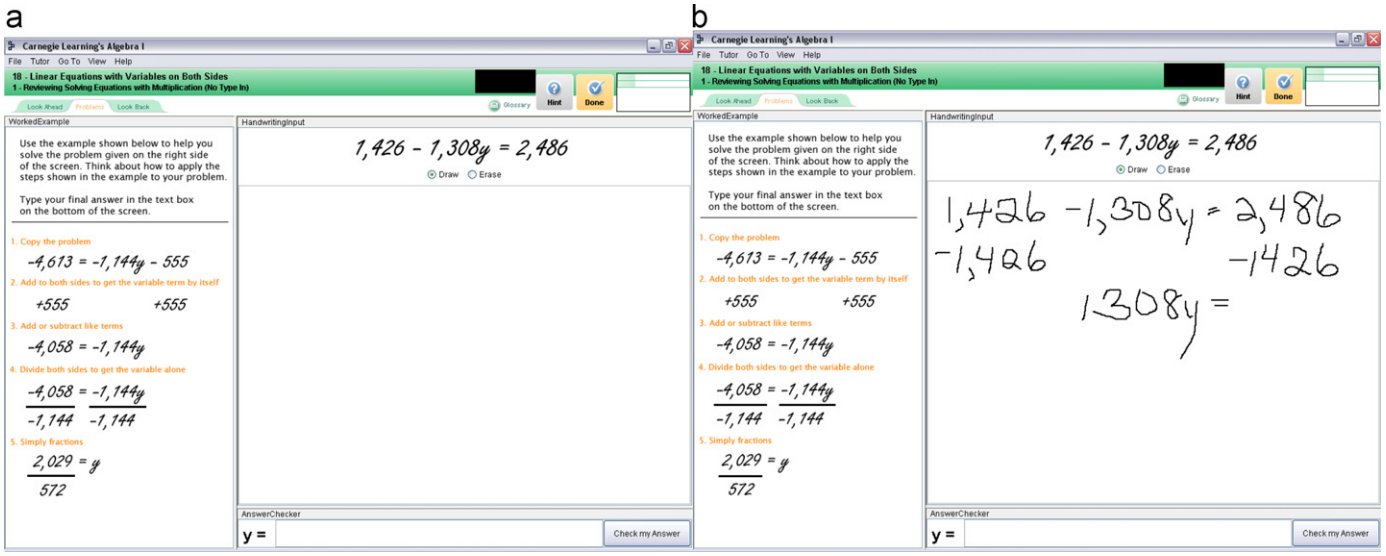


Fig. 4. Problem solving phase. (a) The student is assigned a new problem, similar in form to the example on the left (previously copied). (b) The student is in the process of solving the problem via handwriting input.

student has solved a few problems successfully, as a type of scaffolding, which is then removed as the student becomes more confident and is able to solve problems without referring to the example. Fig. 4(a) shows the newly assigned problem. Fig. 4(b) shows the student in the process of solving the problem in the handwriting-enabled tutor. As we will see, the student is making an error in the second step.

2.3. Answer checking phase

Once the student finishes solving the problem, the tutoring system requires him or her to type the final answer in the textbox at the bottom of the screen. This is done because it removes ambiguity in determining whether the student has been successful and can move on, or if tutoring is needed; also, typing the final step is simple. In Fig. 5, the student has not successfully solved the problem. He or she has made the common student error of dropping the negative sign that should be in front of "1308y" (we will discuss common student errors in the next section). Because the final answer is wrong, the tutoring system is able to tell with 100% confidence that the student has made an error on this problem, so recognition must begin. If the student had been correct, the system would have gone directly to the Curriculum Selection Phase (Section 2.8).

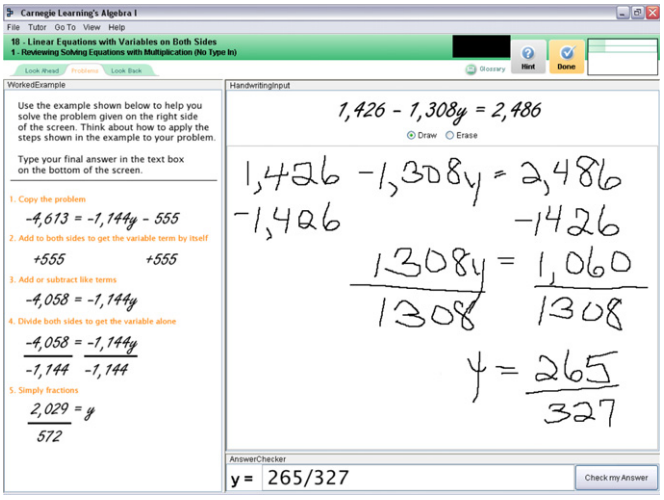


Fig. 5. Answer checking phase. The student completes the problem and types in his or her final answer. However, the student has made an error in dropping the negative sign from "−1308y" after step 1.

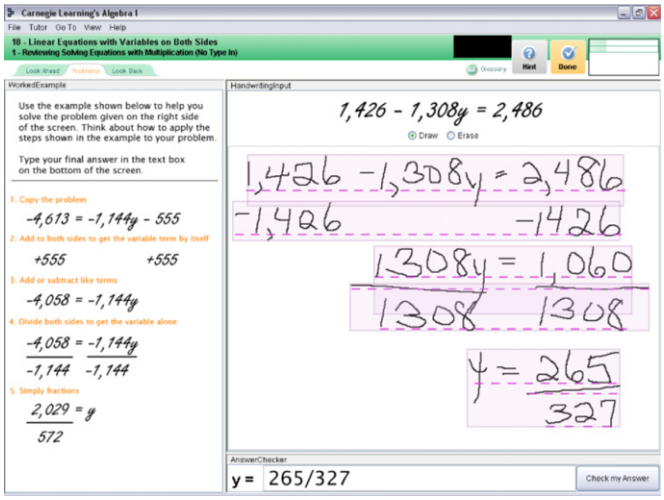


Fig. 6. Recognition phase. Next, the system launches the recognition process, by first extracting baselines for each problem-solving step and grouping strokes that make up individual steps together.

2.4. Recognition phase

Once the tutoring system has detected the need for tutoring, recognition of the student's problem solving solution commences. The first step of the recognition process is to extract baselines (or reference lines) of each line of input from the student's written input, e.g., using a mathematical expression parser such as DRACULAE (Zanibbi et al., 2002). Student problem-solving solutions are largely line-based, due to the step-by-step solution process taught in schools, making baseline extraction relatively reliable. The tutoring system then recognizes each step of the problem individually. DRACULAE (Zanibbi et al., 2002) can also provide mathematical parsing information to help the tutor group the right entities into mathematical relationships once each subgroup is recognized.

During recognition, we also use tutor context, for example the correct solution, to help narrow down the recognition space. Other potential pieces of context include the student model's expectation that this student will make an error, the software's knowledge about common student errors, and so on. Fig. 6 shows simulated baseline extraction results for the example solution shown above. We present the results of our system's recognition of real student problem-solving solutions, and how it improves with the use of context, in Section 3.3.

2.5. Error identification phase

After recognizing each step and representing it internally as a mathematical parse tree, the tutoring system attempts to identify the first step on which the student made an error (the "error step"). The subsequent steps would also be incorrect as errors propagate through the student's solution, but we attempt to find the *first* instance of a student error, because the tutoring opportunity is strongest here. To find the error, the tutoring system examines each hypothesized recognition result and compares it to the expected correct entry for that step (using normalized (Levenshtein's, 1966) string distance). Use of tutor context comes into play heavily here: the expected correct entry depends on previous steps the student has made (correctly), since there is often more than one way to solve a problem. A challenge for the tutoring system in performing the error identification step is that deviation from correct expected input may be due to two factors: (1) actual student problem-solving errors or (2) system recognition errors. We discuss the relative weights of each of these factors and our system's performance on this task in Section 3.3. Once the tutoring system identifies the hypothesized error step, it highlights this step for the student and prompts the student to validate the system's recognition hypothesis, shown in Fig. 7.

If the student indicates that the hypothesis about what he or she wrote for that step is *not correct*, the tutoring system will branch to the Ambiguous Step Typing Phase (Section 2.7). If the student agrees with the system's hypothesis, however, the tutoring system will know it has successfully identified not only the erroneous step, but *what the error actually is*. At this point, the tutoring system enters the Error Feedback Phase (Section 2.6). Note that the student may actually catch the mistake he or she has made and say "no" to the Error Identification prompt, intending to correct his or her input. The student will still be led to the Ambiguous Step Typing Phase (Section 2.7), where he or she can type the intended input. This example shows that our interaction paradigm provides a degree of robustness in allowing the student to self-correct.

2.6. Error feedback phase

In this phase, the tutor can provide detailed hints or other feedback for that step to help the student correct his

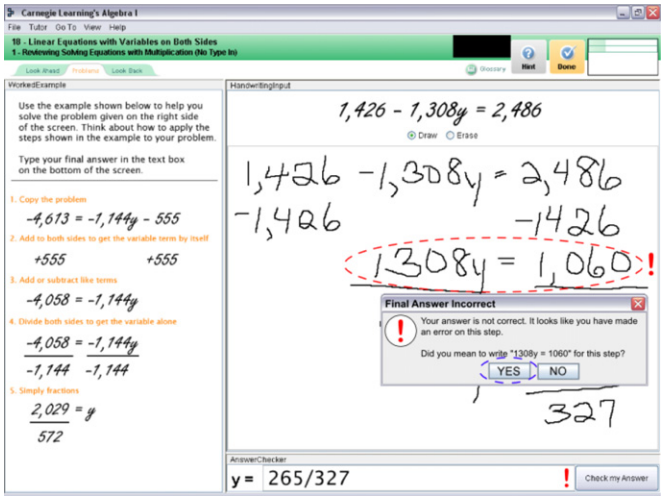


Fig. 7. Error identification phase. Here the system identifies the step on which it believes the student made the first error (correctly, in this case). It prompts the student to verify the system's hypothesis about the student's input in order to choose whether to tutor the student on this skill, or whether the error occurred elsewhere.

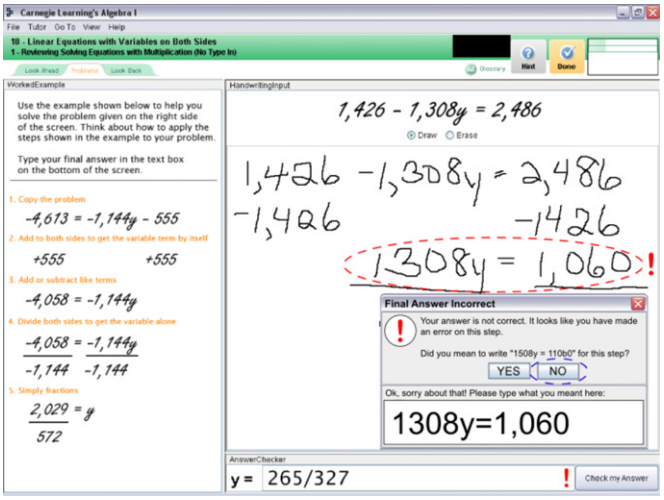


Fig. 8. Ambiguous step typing phase. The system is not able to verify its hypothesis about the student's input on the error step, and therefore it asks the student to type the step unambiguously.

or her error. The student rewrites his or her solution and types the new final answer, re-initiating the Answer Checking Phase (Section 2.3). This process will continue until the student enters the correct final answer. Cognitive Tutors guarantee that students eventually get all solutions correct on their own without excessive floundering (Koedinger and Corbett, 2006).

2.7. Ambiguous step typing phase

If the system finds that its hypothesis about what the expected error step says is incorrect, only then does the system engage the student in recognition correction interaction. It elicits the student's intent via unambiguous typing of this step, shown in Fig. 8. Doing so enables the tutoring system to determine with 100% confidence whether or not the student actually made an error on this step. Asking the student to handwrite his or her solution again could be problematic, as repeated input in the same modality tends to deviate more from the trained models as the user's frustration grows (Oviatt et al., 1998). If the step is actually correct based on the stored problem solution, the tutoring system will revisit the Error Identification Phase (Section 2.5), armed with new certainty about the steps prior to the first hypothesized error step. (If the step the student types actually contains an error, the tutoring system will branch to the Error Feedback Phase and provide a hint or feedback for the step to help the student correct his or her error (Section 2.6)).

2.8. Curriculum selection phase

Once the problem has been successfully solved, and the correct final answer typed (shown in Fig. 9), the tutoring system will enter the Curriculum Selection Phase. In this

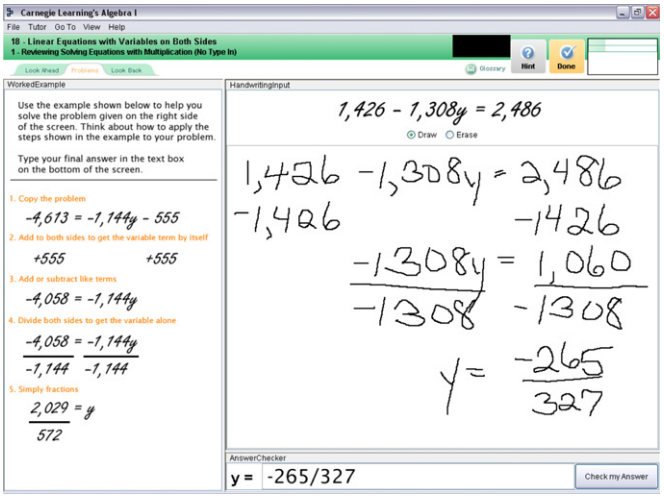


Fig. 9. Curriculum selection phase. The student completes the problem correctly, including typing the correct final answer. The system accepts this solution and chooses the next appropriate problem for the student.

phase, just as in typical Cognitive Tutors, the system determines whether or not the student has attained mastery of all the skills being practiced in the problem type. If the student does have high enough mastery, the tutoring system will assign a new problem type, returning to the beginning of this process at the Worked Example Phase (Section 2.1). If not, the student will receive more problems of the same structure to solve in the Problem Solving Phase (Section 2.2).

2.9. Remarks on design for a handwriting-enabled tutoring system

We have presented the interactive process by which a handwriting-enabled tutoring system can successfully tutor

Table 2
The most commonly encountered student errors from the Lab Learning Study data, their frequencies and rates in the 73 problems with errors, and a description of the error. Note that a problem solution can have more than one error.

"Error" type	Freq.	Rate (%)	Description
Arithmetic error	24	33	The student makes a simple arithmetic error, such as indicating that 7×6 equals 56.
Negative sign dropped	12	16	The student divides by a negative number and does not include the negative sign in the result.
Example mirroring or copying	12	16	The student writes numbers in the problem that are mirrored from the example provided, rather than from the problem to be solved.
Using a wrong number	12	16	The student applies an otherwise correct operation using an incorrect number.
Transcription error	5	7	The student copies the problem incorrectly or incorrectly copies a number from one step to another.
Performing different operations on each side	5	7	The student, for example, subtracts a number from one side but divides on the other side.
Using different numbers on each side	2	3	The student applies the same operation to both sides but uses different operands on each side.
Operating on terms rather than sides	2	3	The student applies an operation to two terms on the same side, ignoring the equals sign and the other side of the equation.
Reciprocal confusion	2	3	The student multiplies by the numerator rather than the denominator to remove a fraction.

the student by iteratively reducing the ambiguity of the student's input. In most cases, the recognition process will, in the face of recognition noise, recognize the student's input well enough to identify the error step, but in the cases in which the tutoring system has low confidence or makes mistakes, the paradigm we presented will enable the student to successfully complete the problem-solving process with a minimum of system-oriented interruptions. There are many ways we deal with input errors in this design: (1) ignoring them if the student types the correct final answer or if they occur after a system-identified error step, (2) confirming the system's recognition of the student's input at the point it believes the student made an error, and (3) finally asking the student to type an individual step that the system needs to understand is correct or not in order to provide appropriate tutoring.

In the next sections, we describe in more detail the methods our prototype system actually uses to implement this paradigm and to recognize written problem-solving solutions and identify the error steps, and how well it performs at doing so on real-world data.

3. Evaluation of our pen input interaction paradigm

We begin by describing the types of student errors that occur in real-world problem-solving datasets and the recognizer test corpus we created to include representative, common errors. This corpus, which we will call the "generated error corpus," was then used to test the recognizer with context to determine how well the system would do on identifying student errors and recovering from them.

3.1. Types of student errors

In this section we introduce the types of problem-solving errors we saw students make. The initial categorization was made on the data from the Lab Learning Study. Understanding what types of errors students make allows us to test the context-enabled recognition process on real examples of errors to help ensure the interaction paradigm we developed would work in real use.

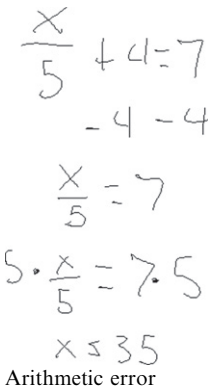
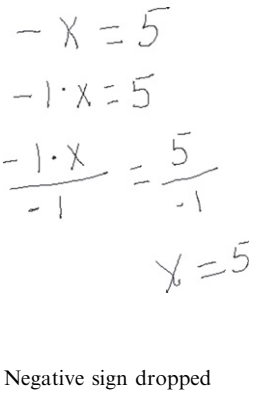
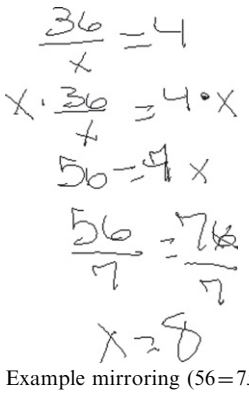
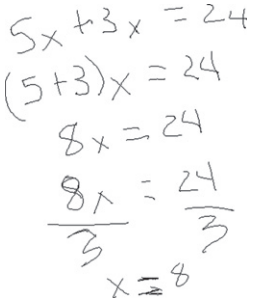
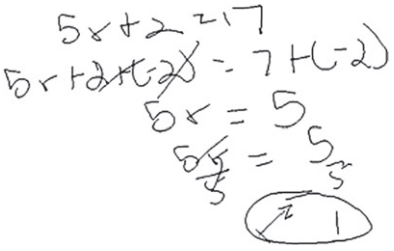
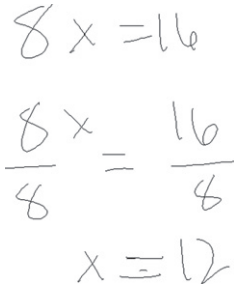
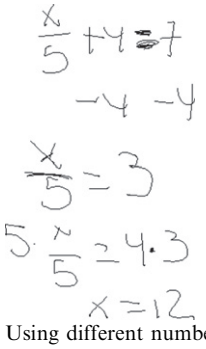
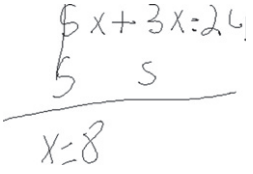
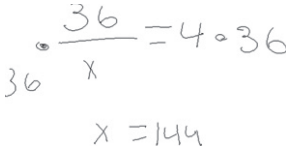
3.1.1. Common error types

Table 2 briefly presents a description of each of the common problem-solving error types and their frequencies in our Lab Learning dataset. Table 3 gives examples of actual handwritten student problem-solving solutions that illustrate the main error types. Note that the emphasis of these categories is on how the system can interpret the student's input in order to score it relative to the correct input, and the categories may not necessarily correspond to the student's intent. Providing help in these circumstances may need to be more broadly worded to cover multiple possible learner misconceptions that manifest themselves in the same expressed error. This same challenge occurs in the existing typing-based tutor when providing hints for these types of errors.

3.1.2. Other error types

We also saw a large quantity of errors of other types that we deemed not in scope for this work. Usually this decision was based on the fact that the "errors" were not problem-solving errors but rather a result of the student inputting information that was not on task or not following the directions. Table 4 briefly presents these "error" types and their frequencies in our Lab Learning dataset. The tutoring system can address these errors, when the entered answer is incorrect, by requiring the student to show (more of) his or her work. Note that we do not intend to imply that these "errors" are not important to learning and pedagogy, but rather that addressing them would require more sophisticated intelligence in the system than this work aims to support. Furthermore, in some cases, such as "not showing work," the student would still be able to use this system successfully to enter his or her final answer, but would need to write the problem-solving solution in the interface to receive help to fix his or her answer if incorrect.

Table 3
Illustrative examples of the main student-made error types found in our Lab Learning Study.

 Arithmetic error	 Negative sign dropped	 Example mirroring ($56=7x$ was from the example)
 Using a wrong number	 Transcription error	 Performing different operations on each side
 Using different numbers on each side	 Operating on terms rather than sides	 Reciprocal confusion

3.2. The generated error corpus

To build a test corpus of problem-solving examples that included instances of errors, we chose the five most common errors students made from the types of student errors in the Lab Learning Study. These five types were: "arithmetic error," "negative sign dropped," "example mirroring," "using a wrong number," and "performing a different operation on each side." We included both a correct and incorrect version of the problem solution for each test case in order to reveal how recognition noise might interfere with recognition of correct problems. The first five test cases involved just one of the five chosen error types, and the sixth one was a more challenging incorrect problem involving two of the errors: the "negative sign dropped" and the "using a wrong number" error types. All of the test cases were examples of a real student's error on a specific problem from the Lab Learning dataset. Because we did not have an example of every problem solution with the same error from every student writer, we created additional test cases ourselves by randomly selecting samples from each student from the algebra learner corpus to fit each test case and laying them out spatially on a virtual canvas. This method ensured we would have a breadth of handwriting samples from different students to test the recognition thoroughly. In total, 4800 sample instances of these problems were created from the algebra learner corpus: 10 examples per each of 40 students per each of six problems both with and without the problem-solving error: $10 \times 40 \times 6 \times 2 = 4800$ total test problem instances.

3.3. Results of the evaluation of the improved recognizer + tutor

In this section we describe the results of our evaluation of the improved recognizer when using context information from the tutoring system. We compare the accuracy of

Table 4
Error types which were deemed out of scope for this work from the Lab Learning Study data, their frequencies and rates in the 73 problems with errors, and a description of the error. Note that a problem solution can have more than one error.

"Error" type	Freq.	Rate (%)	Description
Not showing work	37	51	The student simply writes the final answer without showing the process
Incomplete solution	25	34	The student submits a partial solution without the line "x=X" (e.g., cannot determine their final answer)
Expressed answer via "plug and chug"	12	16	The student writes the original equation with the value of x in the place of the variable, for example, writing $5\times3+3\times3=24$ as the solution to $5x+3x=24$
Off-task writing	18	25	The student scribbles or writes messages unrelated to the problem-solving process in the input space

this improved recognizer against the default recognizer without using tutor knowledge. Recall the two-pronged approach we use to minimize the impact of recognition errors on the student's learning experience. First, task pragmatics are used: the student types the final answer and the system determines whether there's been a student error or not. This information helps the system choose whether or not to launch the "Recognition Phase" of the interaction paradigm. Then, if recognition is launched, it uses context from the tutor itself. We describe in the next section what types of tutor information are available and how they help to refine the recognizer's hypotheses and improve recognition overall, enabling the tutor to more confidently identify the student's error and intervene appropriately. Because we focus on identifying student errors rather than the complete recognition of student problem-solving solutions, the raw recognition accuracy itself is less critical. We then describe how the use of task pragmatics contributes to the success of our tutoring paradigm.

3.3.1. Improving recognition accuracy via context

To improve handwriting recognition accuracy and decrease the negative impact of recognition errors on the student, we incorporate domain-specific context information into the recognition process, thereby constraining the recognition hypotheses to contextually-relevant ones and yielding higher accuracies. As Koerich et al. (2003) noted, smaller vocabularies (lexicons) tend to be more accurately recognized than open lexicons. Domain-specific recognition typically uses smaller vocabulary sizes and specific grammar rules, yielding higher accuracies in its particular context, and so it can be used in cases where domain-general recognition may not yet be suitably accurate.

In the tutoring domain, much context is available for free. For instance, the tutoring program assigns the problems for the student to solve, so it knows what the student should be writing (the correct answer). From the model of student knowledge, it knows the probability that this student will get the correct answer, i.e., based on prior opportunities to practice the same skill(s) applicable to the current step. From years of learning science research into difficulty factors assessment (Tabachneck et al., 1995), the system knows the most common misconceptions (known in intelligent tutoring literature as "bugs") that students might demonstrate. For the prototype work presented in this paper, we used the information from the tutor about *all possible* problem-solving steps that would be correct for the next step.

The tutor context information was used to further refine the recognition hypothesis once the strokes had already been fed into the stroke grouper and character recognizer, rather than as a top-down constraint on what the recognizer would consider as potential hypotheses. The method we used to incorporate the tutor's context information into the recognition process comes from collaborative informational retrieval and is known as "ranking fusion" (Renda and Straccia, 2003). The recognition hypothesis n-best list can just as easily be considered as a rank-list, and the tutor's context information provides a list of potential correct problem-solving results for the current step, each of which is treated as equally likely by our prototype. For simplicity, because the recognizer recognizes one character at a time and its n-best lists include the hypotheses for only one symbol at a time, we put each element of the set of correct next problem-solving steps into a "bag of words" (Buscaldi and Rosso, 2007) (actually, characters in our case) and sort them by symbol frequency to obtain ranks for each potential symbol that might be in the recognizer's n-best list. We then use a technique called "average-rank sort" (Borda, 1781) to sort the tutor rank-list and the recognizer rank-list into one list. This list replaces the recognizer's n-best list for the current symbol it is recognizing; by considering the tutor information, the position of various symbols may be changed in the list, altering the recognizer's top hypothesis. For example, if the recognizer's n-best list's top two elements are "s" and "5" but the tutor's bag of symbols did not include "s", the average-rank sort will drop "s" to much lower in the list, bubbling up "5". For further details on this process, see (Anthony, 2008).

The combined tutor+recognizer was tested on the generated error corpus, using writer-independent five-fold cross-validation so that each student's samples were part of the test set once and the training set four times. We repeated this experiment iteratively with 11 different weights for the tutor ranks (W_T) and the recognizer ranks (W_R), ranging from 0.00 to 1.00, which affects the average-rank sort step of the combination tutor, to find the best settings. (The recognizer weight W_R was always equal to $1-W_T$.) The best improvement over the recognizer alone was seen for the (W_T=0.40, W_R=0.60) pair, with an

accuracy of 72.5% (error=27.5%), although results for several of the nearby weights were quite similar. Performance of the recognizer alone (W_T=0.00, W_R=1.00) is 66.3% (error=33.7%). These results, summarized in Table 5, show an 18% reduction in recognition error when the system is augmented with the tutor's context information provided by our prototype, a difference found to be significant using a paired-samples t-test ($t(18{,}720)=27.997$, $p<0.0001$). (Note that values of W_T in the range of 0.20 to 0.60 also produce accuracy results that are better than the recognizer alone (Anthony, 2008).) Recall that we used a normalized (Levenshtein's, 1966) string distance, a measure of the number of edits (insertions, deletions, and substitutions) required to transform one string into another, to compute accuracy on equations rather than a binary yes/no score. Use of context also increases the number of equations gotten 100% correct by the recognizer, from 26% to 39%, or a 50% increase in fully correct equations in the generated error corpus.

The without-context equation accuracy on the generated error corpus (66.3%) is lower than that on the algebra learner corpus cited in Section 1.4.2 (71.3%), most likely due to the characteristics of the test set. The average length of the equations and expressions from the generated error corpus was shorter than in the algebra learner corpus, and although the accuracy is normalized by equation length, shorter expressions do tend to have higher error rates. For other comments, see (Anthony, 2008).

3.3.2. Identifying student errors

With the recognizer more finely-tuned to the target population and domain, we next focus on using this improved recognizer to identify the step on which the student made an error, as specified by our interaction paradigm.

When a student has entered a final answer that does not match the tutor's expected answer (or is not mathematically equivalent to it), the tutoring system launches the Recognition Phase (Section 2.4) with the goal of identifying the first step on which the student made an error. To determine on what step(s) the student made an error, the recognizer can use the string distance between the recognition hypothesis of what the student actually wrote and the tutor's information about what the step should say if it is correctly solved. If the string distance is greater than a chosen threshold, the step is likely incorrect.

We tested the combined recognizer+tutor's ability to perform this task on the generated error corpus. The performance on this task is imperfect: the performance achieved on identifying the first error at (W_T=0.40, W_R=0.60) was about 42%. A challenge to this task is that, once a student makes an error, the error will tend to propagate through the problem, causing the subsequent steps to diverge more sharply from the expected input. Still, we show an improvement over chance; since the problems have between three to five steps, chance at identifying the error step correctly on the first try is 25%.

If the system gets the error step identification wrong, the process can repeat. Through a combination of verifying the system's hypothesis in the Error Feedback Phase (Section 2.6) and soliciting explicit unambiguous input from the student in the Ambiguous Step Typing Phase (Section 2.7), we enable the system to provide tutoring at the step where it is most likely needed by the student. In the next section we discuss the anticipated impact these interruptions will have on the student's learning experience, based on the frequency and types of errors students make.

As we have seen, there is room for improvement in the system's ability to identify the student's error step on the first try. However, even with the system's current performance, the anticipated impact on the student's learning experience is low, as we discuss in the next section.

3.3.3. Anticipated impact on student experience

To guage the success of our approach in realizing the interaction paradigm we defined, it is not sufficient to simply compute the handwriting recognition accuracy, even with context. Instead, we stress the anticipated impact on the student's experience. How often is the student asked to confirm a step that he or she actually made correctly, because the system did not successfully identify the error step on the first try? We consider the important factors in this section.

As described in the previous section, when the student's final answer is incorrect and tutoring commences, the system may not find the first step on which the student made an error in the problem on its first try. Each time the system gets the identification wrong, it launches an

Table 5
Average recognition accuracy of FFES for this target domain and population on the generated error corpus when tutor context is used during recognition. This table shows data for the best-performing weights for tutor and recognizer of (W_T=0.40, W_R=0.60).

Number of users in corpus		**Generated error corpus** 40
Writer-independent, no context	**Accuracy per equation (string distance)**	66%
	Equations 100% correct	26%
Writer-independent, with context	**Accuracy per equation (string distance)**	73%
	Equations 100% correct	39%

Ambiguous Step Typing Phase, and the student types his or her step so the system can eliminate ambiguity. These steps are unnecessary and extraneous, possibly reducing the benefits for learning of using the handwriting input in the first place. We can use the error identification performance results to estimate how often the system will have to *unnecessarily* intervene with the student in this way.

The expected average number of unnecessary steps per problem that a student will have to type on error problems ($E(n)$) is given by the following equation:

$$E(n) = \sum_{n=0}^{\infty} ne^{(n+1)}$$

In this equation, n is the number of unnecessary steps per problem and e is the probability of the system incorrectly identifying the proper step where the student error is (error rate), given by one minus the success rate. For the value 0.50 of e, the sum converges to 1.0, meaning that with a 50% success rate (50% error rate) at identifying which step the student error is on, the student will have to do an average of one extra step per problem. In other words, the student would correct the system unnecessarily once per problem on which an error occurs. Furthermore, the worst case is that the system chooses the first error *last* when intervening with the student, meaning that the maximum number of unnecessary interventions will be equal to the number of steps in the problem. (Recall that the student does not have to do any extra steps on problems without errors, because he or she would have typed in the final answer correctly and would have been allowed to move on without the system needing to perform any recognition.)

In the corpus of student problem-solving solutions from the Lab Learning Study, there were 73 problems out of 500 that the students did not solve correctly on the first try, which corresponds to a 14% problem error rate in the corpus (i.e., corrections needed on 1 out of 7 problems). Therefore, the overall expected number of extra steps is the product of this rate and the expected number of extra steps on error problems alone. For the error identification success rate of 42% at (W_T=0.40, W_R=0.60) from the results reported in the previous section, the sum (with e=0.58=1.0−0.42) converges to 1.907, meaning that almost twice per problem *on which there was an error* students would have to enter an unnecessary step. Given that only 14% of problems have errors in our data, multiplying 1.907×0.14 yields approximately 0.267, meaning that *approximately one out of every four problems* overall would require unnecessary extra steps over the course of a lesson.

We have established that handwriting of equations is faster than typing, twice as fast in one study (Anthony et al., 2007a) and 20% faster in another (Anthony, 2008). Having to correct the system's recognition errors on one out of four problems (on average) would cut into that time benefit, at least by 25%. Even assuming conservatively that the added overhead of correcting recognition errors via typing costs the students twice as much time, one can expect to cut into the time benefit only by 50%. Thus, in the case in which students in handwriting were twice as fast, the students in handwriting would still be over 50% faster than students in typing for the same problems. Taken concretely, if a student takes two hours to complete a lesson in the typing modality, she would take one hour to complete it in handwriting, with no tutoring feedback or system error correction. With the addition of automatic recognition in the paradigm we have defined, and the need to sometimes correct the system's errors, students would still be able to complete the lesson successfully in 1 h and 30 min on average. Over many lessons, this time savings allows the students to move on to much more advanced material than their typing counterparts.

Furthermore, the timing benefits are present even if the student error rate is higher. For example, we might expect to see an increase in the proportion of student errors as the instructional domain becomes more complex, such as in a university math course rather than the high school course studied in this work. With the same e of 0.58 (error identification success rate of 42%), if students' rate of errors doubled to 28% of the problems, we expect students would be interrupted about every other problem; if student errors increased to 50% of the problems, we expect students would be interrupted on almost every problem. If we can improve error identification success rate up to 61% (e=0.39), students can make errors on up to 50% of the problems and still be interrupted on only one out of every four problems. Finally, even with the original student error rate of 14%, as long as the system's success rate for error identification is at least 35% (e=0.65), there is still an estimated time savings for the students in handwriting.

3.4. Remarks on evaluation of the paradigm

The baseline student experience to which we are comparing is a typing and menu-based interaction, which is cumbersome for performing math and creates cognitive load, fluency and transfer problems. With the improvements to recognition accuracy, including the use of context and training to the target domain and population, and the focus of the interaction on tutoring student errors rather than full knowledge of the student's solution, we can raise the performance of the handwriting interface to minimize the need for students to correct recognition errors. *Even with the occasional recognition correction*, the benefits of handwriting input for math (over typing) are still present: usability, speed, user preference, and transfer to paper.

4. Related work

4.1. Other alternative interfaces for intelligent tutoring systems

Besides the work presented in this paper, other tutoring systems have explored more natural interfaces, such as natural

language processing of typed input (Aleven et al., 2003; Freedman, 1999), spoken dialogs with conversational agents (Beal et al., 2005; Graesser et al., 2003; Litman and Silliman, 2004), and animated characters with gesture-based interfaces (Oviatt and Adams, 2000). However, most systems do still rely on standard WIMP interfaces. The prevalence of WIMP interfaces is due in part to the fact that the technology available to most students in the classroom has been limited to keyboard-and-mouse — this situation is changing however, as students receive PDAs, Tablet PCs or iPads in the classroom (Jackson, 2004; Valentino-Devries, 2010; Wood, 2002). In addition, research into handwriting recognition technology has not emphasized making recognizers easy to use and adapt for new domains by non-experts, and recognition systems are often inaccessible or opaque to anyone but the system's own developers.

4.2. Other handwriting interfaces for mathematics

Standard interfaces for entering mathematical equations into computers have focused heavily on keyboard- and mouse-based interfaces, especially on the desktop. Mathematics tools that use a typing interface often require the user to become an expert at a new programming language (e.g., Microsoft Mathematics, MapleSoft's Maple, The MathWorks' Matlab, and Wolfram Research's Mathematica). These programs have a large learning curve, even for mathematics experts, and therefore are not only difficult or inaccessible for many novices but also slow for experts to use. These computer interfaces are optimized for entering linear text (Smithies et al., 2001). Linear input methods might inhibit mathematical thinking and visualization, especially for some learning tasks, since mathematics often appears in higher dimensional layouts, enabling the representation of fractions, superscripts, subscripts, and other notations.

Mathematics interfaces that do not require users to linearize their input are called template-based editors, which force users to select pre-defined mathematical structure templates (e.g., fractions, superscripts, subscripts) from a menu or toolbar and then to fill in the templates with numbers and operators by typing on the keyboard. Users can construct a representation of higher-dimensional mathematics, but must do so in a top-down manner, making later structural changes difficult (Smithies et al., 2001). The most common such tool is the equation editor included in Microsoft Office (the equation editor is a simplified version of Design Science's MathType tool). Worthy of note is that Microsoft (2005) has an extension to the equation editor for the Tablet PC version of Windows that allows handwritten input. However, because it is not customizable by the end-user or an application developer, it cannot be easily adapted to new domains such as math learning, making it suboptimal for use in research into new handwriting recognition applications.

Unlike typing, writing math allows the use of paper-based mathematical notations simply and directly. It is therefore natural and convenient for users to communicate with computers via pen input (Blostein and Grbavec, 1996). Several research and commercial systems do exist that allow users to input and/or edit mathematical expressions via handwriting input. We have already discussed MathPad^2 (LaViola and Zeleznik, 2004), which is among the most robust and complex. In MathPad^2, users can write out mathematics or physics equations and the system animates the physical relationships given by these equations, for example, to animate a pendulum or oscillating sine curve. Another relevant example is the PenProof system (Jiang et al., 2010), which allows users to sketch and write geometry proofs (not necessarily equations) that the system automatically validates. Other systems such as MathPaper (Zeleznik et al., 2008), Algo-Sketch (Li et al., 2008) and xThink's MathJournal (2003) allow the sketching and writing of mathematics, but rely on in-context menus to allow users to perform manipulations. Littin's (1995) recognition and parsing system, the Natural Log system (Matsakis, 1999), FFES (Smithies et al., 2001), PenCalc (Chan and Yeung, 2001), inftyEditor (Suzuki et al., 2004), and JMathNotes and the related E-Chalk system (Tapia and Rojas, 2003, 2005) are simple equation entry and editing programs without the added benefit of sketching or graphing. Many of the earlier systems on this list are out of date and not maintained.

Most of the mentioned handwriting-based interfaces for math focus only on letting users *input* mathematics. They do not provide a structured approach to *learning* how to perform mathematical operations. There is at least one commercial software program that does use pen input for math for educational goals, albeit very simply: the AlphaCount iPhone app (2010), based on the $N multi-stroke pen gesture recognizer (Anthony and Wobbrock, 2010). Students practice entering numbers and counting objects onscreen via finger writing. Oviatt et al. (2006) have investigated the use of pen-based input for geometry learning, focusing on the cognitive load imposed by less familiar interfaces such as tablet computers vs. digital paper. MathBrush (Labahn et al., 2008) adds a pen-input layer onto an existing computer algebra system (CAS), motivated by educational pedagogy research into the best ways to introduce technology into the classroom. Newton's Pen (Lee et al., 2007) is a pen-top computer tutor for physics statics problems in which students fill out template worksheets on digital paper. However, in none of this work is there scaffolded, tailored feedback or a model of student learning, both of which are significant contributors to the advantage of using Cognitive Tutors (cf. Koedinger and Corbett, 2006).

4.3. Other recognition error recovery strategies

In systems with potential for errors, it often falls to the user to repair such errors. Mankoff and Abowd (1999) identified five approaches to error handling in recognition-based interfaces: (1) error reduction, (2) error discovery, (3) error correction, (4) validation of techniques, and (5)

toolkit-level support. Error repair has been studied extensively in speech recognition interfaces, but less so in handwriting interfaces. The approaches to error repair taken in this work are (1) error reduction – avoid making errors in the first place as much as possible (e.g., use of context), and (2) error discovery – attempt to find or reduce the system's errors before they are presented to the student (e.g., use of task pragmatics). Error correction techniques and their validation can be taken into account in future work. Many types of error correction techniques exist, and some work has been done in exploring their suitability for use with children. Read et al. (2003a) showed that students spend more time correcting errors when recognition is real-time (i.e., displayed as characters are being written) than when it occurs at the end of a discrete unit, such as a sentence or equation; however, the total number of errors made does not differ. The extra time spent repairing errors is extraneous to learning, so delaying recognition feedback until later seems wise in this domain with this target population, and is consistent with our paradigm.

A system developer may choose to provide explicit error recovery techniques, and may find it useful to know what types of errors one can expect from this population and in this domain. Besides domain-specific errors such as the algebra problem-solving errors we presented in Section 3.1, specific error types that are likely to occur in handwriting interfaces have been studied (Schomaker, 1994): one can expect to find (1) discrete noise, (2) badly formed shapes, (3) input that is legible by the human but not by the recognizer, (4) misspelled words, (5) canceled material, and (6) device-generated errors. Types of repair strategies undertaken by users when these errors occur are deletion, completion, insertion and overwriting (Hürst et al., 1998). Error types for children using handwriting input also include (Read et al., 2001b) spelling errors, construction errors (e.g., penmanship errors), and execution errors in using the handwriting device, on top of recognition errors. In the learning domain, of course, the possibility of the student making math errors is very real and must also be taken into account. Additionally, observational studies of children using pen-based input and handwriting recognizers have been undertaken which have helped to identify specific types of device-generated errors, including position-based errors such as when the stylus and pointer onscreen are not properly calibrated, or when the student writing goes off the page (Read et al., 2002).

5. Conclusions

We have presented the theoretical and practical aspects of an interaction paradigm for handwriting-based intelligent tutoring systems for mathematics. We based our design recommendations on foundational work establishing the affordances and benefits of handwriting input for mathematics (Anthony et al., 2005, 2007a, 2007b, 2008), and expect that these benefits will hold for a variety of domains. A key component to this expectation is the idea that a student in a learning environment should be able to separate the concepts he or she is learning from the interface he or she is using to perform the learning tasks. Robust and transferrable learning is a critical goal for education research (Koedinger et al., in press) and the interaction can help or hinder it. Fluent and natural interaction is the first step, and the second step is to ensure that correcting system recognition errors does not become central to the interaction, allowing students to focus on correcting their own learning errors and misconceptions.

5.1. Limitations and future directions

While we have presented compelling evidence that a handwriting-based intelligent tutor could be effective in enhancing the student learning experience, even with imperfect recognition, we have not yet tested the full prototype that realizes the presented interaction paradigm with students. Such a test seems worthwhile, especially given that newer recognition technologies may be available that would improve the results presented here even further. Another avenue for improvement is to use more of the context information that Cognitive Tutors provide, including knowledge of the specific student's likelihood of making an error on specific steps involving certain skills.

The generalization of the work presented here beyond algebra to other domains would be enlightening. So would a concrete realization of the anticipated impact of the presented interaction paradigm on the student's learning experience, namely, quantifying the degree to which students really do move on to new material faster as a result of using handwriting-based ITSs vs. typing ones, and measuring the long-term learning benefits of doing so.

Some of the components of the technical approach could be further investigated through explicit comparisons to other methods. For example, using a different baseline recognition engine, more of the tutor's context, and other ways to weight and combine the tutor's and recognizer's hypotheses during recognition all have potential to yield improvements. We have used average-rank sort here, but more sophisticated techniques such as Bayesian networks or Markov chaining might be promising. We have also incorporated the tutor's context information by merging it with the independently-calculated recognition result for a specific step, but using the context to prune the recognition space before the recognizer begins may be a viable alternative. Simplifying the tutor's context into a bag of words removes some of the information present in the tutor's context, such as bi-grams and tri-grams, and so options might be considered which retain this information and allow it to be used to improve results.

5.2. Implications for design of pen-based ITSs

This paper presented several techniques to improve handwriting recognition accuracy for use in the math

tutoring domain, including training on a set of handwriting data from the target population (e.g., junior high students rather than users in general) in order to enhance writer-independent accuracy and to reduce or remove the need for students to take classroom time to train the system before they begin learning. In addition, we used domain-context information to refine recognition results, by adding information from the tutoring system about the set of correct possible answers at each step of the problem-solving process. The domain-context information, though very simple in scope, significantly reduced recognition error by 18%. The use of task pragmatics, namely, that exact recognition is not needed when students are correct or once the system has identified the error step, is a further advantage. Taken together, the impact on the student was quantified and expressed in a general formula yielding an estimation, given current recognition performance, that students will have to correct the recognizer's errors on average on one out of every four problems. This formula can be used to estimate the impact on the student when better recognizers become available as technology and algorithms continue to advance.

Finally, based on all these results, we structured a tutoring interaction paradigm that we have outlined via mock-ups of the stages of interaction. Designers of intelligent tutoring systems for mathematics can use this interaction scenario to build on our proof-of-concept prototype and implement a tutoring system that can take advantage of the benefits of handwriting input, in spite of imperfect recognition.

A key implication for the design of such future systems is that allowing students to type their final answer removes all impact of recognition errors on correct problems. The system may well make several recognition errors on a correct solution, but these errors never propagate to the student and therefore do not interfere with student learning. In essence these recognition errors do not "count" and the expected recognition accuracy will be much higher than raw estimates, depending on the prevalence of student errors in the real-world. We have termed this concept *task pragmatics* and have illustrated a particular strategy for capitalizing upon this idea in this application; however, other strategies may help in this or other ambiguous situations. For example, an automated telephone system with speech recognition may not need to resolve all ambiguity in recognition in order to match the user's input to one of the accepted grammar strings (e.g., "Representative!").

Finally we close this article with an illustrative example of how this handwriting-based interaction paradigm for ITSs could work in another tutoring domain, for example, chemistry. Chemistry includes its own type of "equation solving" known as stoichiometry, a method of balancing the relative quantities of reactants and products in chemical reactions (see Fig. 10).

The interaction paradigm we presented could transfer remarkably well to such problem-solving activities. The

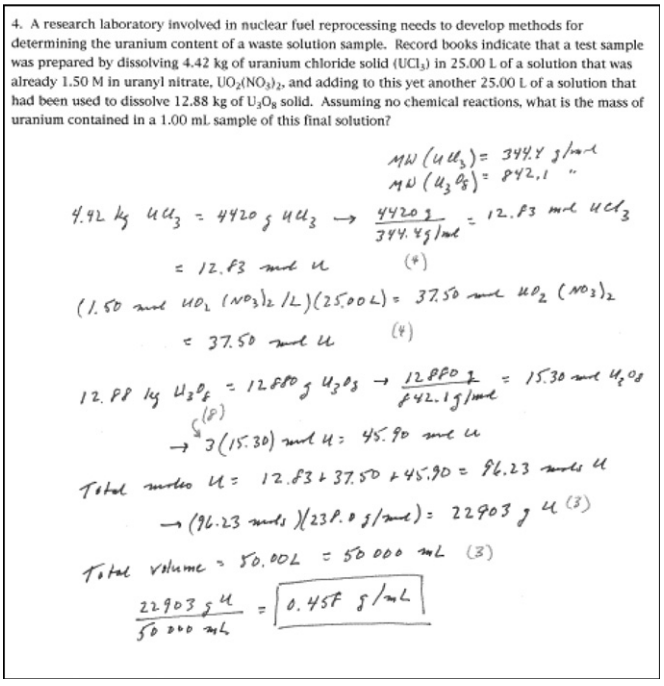


Fig. 10. An example of a stoichiometry problem.

recognition engine should be trained on the expanded symbol set that chemistry uses as compared to the simplified algebra symbol set covered by the algebra learner corpus, and the recognition accuracy might therefore decrease to some extent. However, a tutoring system that understands chemistry problems (such as the Cognitive Tutor for Stoichiometry, McLaren et al., 2008) would be able to provide the same context to the recognizer to improve accuracy and decrease the requirement for the student to correct many of the system's recognition errors. Typing the final answer (e.g., "0.458 g/ml" in the given example) is still possible, and easier than typing the entire problem due to the fractions and equation balancing operations that are performed in stoichiometry.

This example is meant to illustrate concretely the considerations that are applicable when extending our proposed interaction paradigm to other domains. The most direct extensions involve domains such as chemistry in which the final answer consists of type-able text, but we expect that sketch-based recognition can be augmented as well by similar methods to build on this paradigm for domains such as geometry or physics in which solutions partially or wholly consist of diagrams. We look forward to the future of natural and usable ITSs that use handwriting input, spoken input, and even sketch, to allow students to work directly within their domain and experience deeper, robust learning that can transfer to learning and practice environments in the real world.

Acknowledgments

The authors thank Mark D. Gross, Jennifer Mankoff, and Tom M. Mitchell for comments on the dissertation that informed this paper, and the anonymous reviewers of this

article for helpful comments. The authors acknowledge Carnegie Learning, Inc., for permission to reproduce the screenshot of their algebra tutor software for this paper, and Paul Karol for permission to reproduce his stoichiometry answer key in this paper. The work presented in this paper was supported by grants from the National Science Foundation (NSF Award SBE-0354420) and the Pittsburgh Science of Learning Center. The first author was partially supported by an NSF Graduate Research Fellowship during the period of this work. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the NSF or PSLC.

References

Abowd, G.D., 1999. Software engineering issues for ubiquitous computing. In: Proceedings of International Conference on Software Engineering, ISCE'99. ACM, NY, pp. 75–84.

Aleven, V.A.W.M.M., Koedinger, K.R., Popescu, O.A., 2003. A tutorial dialog system to support self-explanation: evaluation and open questions. In: Proceedings of the International Conference on Artificial Intelligence in Education, pp. 39–46. IOS Press, Amsterdam, the Netherlands.

AlphaCount, 2010. AlphaCount. 〈http://alphacount.wordpress.com/〉, (accessed September 10, 2011).

Anderson, R.H., 1968. Syntax-directed Recognition of Hand-printed Two-dimensional Mathematics. PhD Thesis, Department of Engineering and Applied Mathematics, Harvard University.

Anthony, L., Yang, J., Koedinger, K.R., 2005. Evaluation of multimodal input for entering mathematical equations on the computer. In: Proceedings of the ACM Conference on Human Factors in Computing (CHI), pp. 1184–1187. ACM Press, NY, USA.

Anthony, L., Yang, J., Koedinger, K.R., 2007a. Benefits of handwritten input for students learning algebra equation solving. In: Proceedings of the International Conference on Artificial Intelligence in Education (AIEd), pp. 521–523. IOS Press, Amsterdamn, the Netherlands.

Anthony, L., Yang, J., Koedinger, K.R., 2007b. Adapting handwriting recognition for applications in algebra learning. In: Proceedings of the ACM Workshop on Educational Multimedia and Multimedia Education, pp. 47–56. ACM Press, NY, USA.

Anthony, L., Yang, J., Koedinger, K.R., 2008. Toward next-generation, intelligent tutors: Adding natural handwriting input. IEEE Multimedia 15 (3), 64–68.

Anthony, L. 2008. Developing Handwriting-based Intelligent Tutors to Enhance Mathematics Learning. PhD Thesis, Carnegie Mellon University, CMU-HCII-08-105.

Anthony, L., Wobbrock, J.O., 2010. A lightweight multistroke recognizer for user interface prototypes. In: Proceedings of Graphics Interface 2010, Gi '10. Canadian Information Processing Society, Toronto, Ont., Canada, pp. 245–252.

Awal, A.-M., Mouchere, H., Viard-Gaudin, C., 2010. The problem of handwritten mathematical expression recognition evaluation. In: 2010 12th International Conference on Frontiers in Handwriting Recognition. Kolkata, India, pp. 646–651.

Beal, C., Johnson, W.L., Rabrowski, R., Wu, S., 2005. Individualized feedback and simulation-based practice in the tactical language training system: an experimental evaluation. In: Proceedings of the Artificial Intelligence in Education Conference (AIEd), pp. 749–749. IOS Press, Amsterdam, the Netherlands.

Belaid, A., Haton, J.P., 1984. A syntactic approach for handwritten mathematical formula recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 6 (1), 105–111.

Blostein, D., Grbavec, A., 1996. Recognition of mathematical notation. In: Wang, P.S.P., Bunke, H. (Eds.), Handbook on Optical Character Recognition and Document Analysis. World Scientific Publishing Company, pp. 557–582.

Blostein, D., Lank, E., Rose, A., Zanibbi, R., 2002. User interfaces for on-line diagram recognition. In: Blostein, D., Kwon, Y.-B. (Eds.), Graphics Recognition Algorithms and Applications. Springer, Berlin Heidelberg, Berlin, pp. 92–103.

Borda, J.C., 1781. Memoire sur les elections au scrutin. In: Histoire de l'Academie Royal des Sciences.

Brown, R.M., 1964. On-line computer recognition of handprinted characters. IEEE Transactions on Electronic Computing 13, 750–752.

Buscaldi, D., Rosso, P., 2007. A bag-of-words based ranking method for the wikipedia question answering task. In: Peters, C., Clough, P., Gey, F.C., Karlgren, J., Magnini, B., Oard, D.W., de Rijke, M., Stempfhuber, M. (Eds.), Evaluation of Multilingual and Multi-modal Information Retrieval: Seventh Workshop of the Cross-Language Evaluation Forum, Revised Selected Papers; Lecture Notes in Computer Science 4730, pp. 550–553. Springer Berlin, Heidelberg.

Chan, K.-F., Yeung, D.-Y., 2000. Mathematical expression recognition: a survey. International Journal on Document Analysis and Recognition 3, 375–384.

Chan, K.-F., Yeung, D.-Y., 2001. PenCalc: a novel application of on-line mathematical expression recognition technology. In: Proceedings of Sixth International Conference on Document Analysis and Recognition. Seattle, WA, USA, pp. 774–778.

Cheema, S., LaViola J., 2010. Towards intelligent motion inferencing in mathematical sketching. In: Proceedings of the 2010 International Conference on Intelligent User Interfaces, pp. 289–292.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., Glaser, R., 1989. Self-explanations: how students study and use examples in learning to solve problems. Cognitive Science 13, 145–182.

Clark, R.C., Nguyen, F., Sweller, J., 2006. Efficiency in Learning: Evidence-Based Guidelines to Manage Cognitive Load. Pfeiffer, San Francisco, CA, USA.

Corbett, A.T., Koedinger, K.R., Anderson, J.R., 1997. Intelligent tutoring systems. In: Helander, M.G., Landauer, T.K., Prabhu, P.V. (Eds.), Handbook of Human-Computer Interaction. Elsevier Science B.V, Amsterdam, The Netherlands, pp. 849–874.

Dimitriadis, Y.A., Coronado, J.L., 1995. Towards an art based mathematical editor that uses online handwritten symbol recognition. Pattern Recognition 28 (6), 807–822.

Dimond, T.L., 1957. Devices for reading handwritten characters. In: Proceedings of the Eastern Joint Computing Conferences, pp. 232–237.

Elliott, E.S., Dweck, C.S., 1988. Goals: an approach to motivation and achievement. Journal of Personality and Social Psychology 54 (1), 5–12.

Fateman, R., Tokuyasu, T., Berman, B.P., Mitchell, N., 1996. Optical character recognition and parsing of typeset mathematics. Journal of Visual Communication and Image Representation 7 (1), 2–15.

Frankish, C., Hull, R., Morgan, P., 1995. Recognition accuracy and user acceptance of pen interfaces. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI), pp. 503–510.

Freedman, R., 1999. Atlas: a plan manager for mixed-initiative, multimodal dialogue. In: AAAI Workshop on Mixed-initiative Intelligence.

Goldberg, D., Goodisman, A., 1991. Stylus user interfaces for manipulating text. In: Proceedings of ACM Symposium on User Interface Software and Technology (UIST), pp. 127–135.

Glaser, R., 1976. Components of a psychology of instruction: toward a science of design. Review of Educational Research 46, 1–24.

Graesser, A., Moreno, K.N., Marineau, J.C., Adcock, A.B., Olney, A.M., Person, N.K., 2003. Autotutor improves deep learning of computer literacy: is it the dialog or the talking head? In: Proceedings of the International Conference on Artificial Intelligence in Education, pp. 47–54.

Guyon, I., Warwick, C., 1998. Handwriting as computer interface. In: Cole, R.A., Mariani, J., Uszkoreit, H., Varile, G.B., Zaenen, A., Zampolli, A. (Eds.), Survey of the State of the Art in Human Language Technology. Cambridge University Press, Boston, MA, pp. 78–83.

Hürst, W., Yang, J., Waibel, A., 1998. Error repair in human handwriting: an intelligent user interface for automatic on-line handwriting

recognition. In: Proceedings of the IEEE International Joint Symposia on Intelligence and Systems, pp. 389–395. IEEE.

Jackson, L., 2004. Laptops, handhelds,or tabletpcs? Education World. 〈http://www.education-world.com/a_tech/tech/tech198.shtmlS〉 (accessed 09.09.2011).

Jiang, Y., Tian, F., Wang, H., Zhang, X., Wang, X., Dai, G., 2010. Intelligent understanding of handwritten geometry theorem proving. In: Proceedings of the 15th International Conference on Intelligent User Interfaces, IUI '10. ACM, NY, USA, pp. 119–128.

LaLomia, M.J., 1994. User acceptance of handwritten recognition accuracy. In: Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI), p. 107.

Landay, J.A., Myers, B.A., 1995. Interactive sketching for the early stages of user interface design. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95). ACM Press/Addison-Wesley Publishing Co., NY, USA, pp. 43–50.

Koedinger, K.R., Corbett, A.T., 2006. Cognitive tutors: technology bringing learning science to the classroom. In: Sawyer, K. (Ed.), The Cambridge Handbook of the Learning Sciences. Cambridge University Press, pp. 61–78.

Koedinger, K.R., Corbett, A.C., Perfetti, C., accepted. The knowledge-learning-instruction (KLI) framework: bridging the science-practice chasm to enhance robust student learning. Cognitive Science. http://dx.doi.org/10.1111/j.1551-6709.2012.01245.x, in press.

Koerich, A.L., Sabourin, R., Suen, C.Y., 2003. Large vocabulary off-line handwriting recognition: a survey. Pattern Analysis and Applications 6, 97–121.

Labahn, G., Lank, E., Marzouk, M.S., Bunt, A., MacLean, S., Tausky, D., 2008. MathBrush: a case study for pen-based interactive mathematics. In: Proceedings of the Fifth Eurographics Symposium on Sketch-Based Interfaces and Modeling, SBIM '08. Annecy, France, pp. 143–150.

Lapointe, A., Blostein, D., 2009. Issues in performance evaluation: a case study of math recognition. In: 10th International Conference on Document Analysis and Recognition. Barcelona, Spain, pp. 1355–1359.

LaViola, J.J., 2006. An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations. In: Proceedings of Eurographics Workshop on Sketch-Based Interfaces and Modeling, EG Workshop Series, pp. 157–164.

LaViola, J.J., Zeleznik, R., 2004. MathPad^2: a system for the creation and exploration of mathematical sketches. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004) 23 (3), 432–440.

Lee, W., de Silva, R., Peterson, E.J., Calfee, R.C., Stahovich, T.F., 2007. Newton's pen: a pen-based tutoring system for statics. In: Proceedings of the Fourth Eurographics Workshop on Sketch-based Interfaces and Modeling, SBIM '07. ACM, NY, USA, pp. 59–66.

Levenshtein, V.I., 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (8), 707–710.

Li, C., Miller, T.S., Zeleznik, R.C., Jr, J.J.L., 2008. AlgoSketch: algorithm sketching and interactive computation. In: Proceedings of the Fifth Eurographics Symposium on Sketch-based Interfaces and Modeling, SBIM '08. Annecy, France, pp. 175–182.

Lin, J., Newman, M.W., Hong, J.I., Landay, J.A., 2000. DENIM: finding a tighter fit between tools and practice for web site design. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '00). ACM, NY, USA, pp. 510–517.

Litman, D.J., Silliman, S., 2004. Itspoke: an intelligent tutoring spoken dialogue system. In: Proceedings of the Human Language Technology Conference: Fourth Meeting of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (Companion Proceedings), pp. 233–236.

Littin, R.H., 1995. Mathematical Expression Recognition: Parsing Pen/Tablet Input in Real-Time Using LR Techniques. Master's Thesis, University of Waikato, Hamilton, New Zealand.

Liu, C.-L., Yin, F., Wang, D.-H., Wang, Q.-F., 2010. Chinese handwriting recognition contest 2010. In: 2010 Chinese Conference on Pattern Recognition (CCPR). IEEE, p. 5.

MacKenzie, I.S., Chang, L., 1999. A performance comparison of two handwriting recognizers. Interacting with Computers 11, 283–297.

Mankoff, J., Abowd, G., 1999. Error Correction Techniques for Handwriting, Speech, and Other Ambiguous or Error Prone Systems. Technical Report GIT-GVU-99-18. Georgia Institute of Technology.

Märgner, V., Abed, H.E., 2010. ICFHR 2010 — Arabic handwriting recognition competition. In: 2010 12th International Conference on Frontiers in Handwriting Recognition. Kolkata, India, pp. 709–714.

Matsakis, N.E., 1999. Recognition of Handwritten Mathematical Expressions. Master's Thesis, Massachusetts Institute of Technology.

McLaren, B.M., Lim, S.-J., Koedinger, K.R., 2008. When is assistance helpful to learning? Results in combining worked examples and intelligent tutoring. In: Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (Eds.), Proceedings of the Ninth International Conference on Intelligent Tutoring Systems (ITS-08), Lecture Notes in Computer Science, 5091. Springer Berlin Heidelberg, Berlin, pp. 677–680.

Microsoft, 2005. Equation Pack for Windows XP Tablet PC Edition. 〈http://www.microsoft.com/windowsxp/downloads/tabletpc/educationpack/overview4.mspx〉, (accessed September 10, 2011).

Miller, E.G., Viola, P.A., 1998. Ambiguity and constraint in mathematical expression recognition. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 784–789. AAAI Press.

O'Connell, T., Li, C., Miller, T.S., Zeleznik, R.C., LaViola, J.J., 2009. A usability evaluation of AlgoSketch. In: Proceedings of the Sixth Eurographics Symposium on Sketch-based Interfaces and Modeling, SBIM '09. New Orleans, Louisiana, p. 149.

Oviatt, S., Adams, B., 2000. Designing and evaluating conversational interfaces with animated characters. In: Cassell, J., Sullivan, J., Prevost, S., Churchill, E. (Eds.), Embodied Conversational Agents. MIT Press, Cambridge, MA, USA, pp. 319–343.

Oviatt, S., Arthur, A., Cohen, J., 2006. Quiet interfaces that help students think. In: Proceedings of the ACM Symposium on User Interface Software and Technology, pp. 191–200.

Oviatt, S., MacEachern, M., Levow, G.-A., 1998. Predicting hyperarticulate speech during human-computer error resolution. Speech Communication 24 (2), 87–110 May 1998.

Pashler, H., Bain, P., Bottge, B., Graesser, A., Koedinger, K., McDaniel, M., Metcalfe, J., 2007. Organizing Instruction and Study to Improve Student Learning (NCER 2007–2004). National Center for Education Research, Institute of Education Sciences, U.S. Department of Education, Washington, DC.

Purcell, S.C., 1977. Understanding Hand-Printed Algebra for Computer Tutoring. Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Read, J.C., MacFarlane, S.J., Casey, C., 2000. Where's the 'm' on the keyboard, mummy? In Womens'. Engineering Society.

Read, J.C., MacFarlane, S.J., Casey, C., 2001a. Can natural language recognition technologies be used to enhance the learning experience of young children? Computers and Learning.

Read, J.C., MacFarlane, S.J., Casey, C., 2001b. Measuring the usability of text input methods for children. In: Proceedings of BCS British-HCI, pp. 559–572. Springer Verlag.

Read, J.C., 2002. Optimising text input for young children using computers for creative writing tasks. In: Proceedings of BCS British-HCI, pp. 503-519. Springer Verlag.

Read, J.C., MacFarlane, S.J., Casey, C., 2002. Pens behaving badly — usability of pens and graphics tablets for text entry with children. In: Adjunct Proceedings of the ACM Symposium on User Interface Software and Technology, pp. 21–22. ACM Press.

Read, J.C., MacFarlane, S.J., Casey, C., 2003a. A comparison of two on-line handwriting recognition methods for unconstrained text entry by children. In: Proceedings of BCS British-HCI, pp. 29–32. Research Press International, Bristol, UK.

Read, J.C., MacFarlane, S.J., Casey, C., 2003b. Good enough for what?: acceptance of handwriting recognition errors by child users. In: Proceedings of the 2003 conference on Interaction design and children, pp. 155–155, NY, USA. ACM Press.

Read, J., 2007. A study of the usability of handwriting recognition for text entry by children. Interacting with Computers 19, 57–69.

Renda, M.E., Straccia, U., 2003. Web metasearch: rank vs. score based rank aggregation methods. In: Proceedings of the ACM symposium on Applied computing, pp. 841–846, NY, USA. ACM.

Roschelle, J., Tatar, D., Chaudbury, S.R., Dimitriadis, Y., Patton, C., DiGiano, C., 2007. Ink, improvisation, and interactive engagement: learning with tablets. Computer 40, 42–48.

Salden, R.J.C.M., Koedinger, K.R., Renkl, A., Aleven, V., McLaren, B.M., 2010. Accounting for beneficial effects of worked examples in tutored problem solving. Educational Psychology Review 22 (4), 379–392.

Santos, P.J., Baltzer, A.J., Badre, A.N., Henneman, R.L., Miller, M.S., 1992. On handwriting recognition performance: some experimental results. In: Proceedings of the Human Factors Society 36th Annual Meeting, pp. 283–287.

Schomaker, L., 1994. User-interface aspects in recognizing connected-cursive handwriting. In: Proceedings of the IEE Colloquium on Handwriting and Pen-based Input, 8/1–8/3.

Smithies, S., Novins, K., Arvo, J., 2001. Equation entry and editing via handwriting and gesture recognition. Behaviour and Information Technology 20, 53–67.

Suzuki, M., Kanahori, T., Ohtake, N., Yamaguchi, K., 2004. An integrated OCR software for mathematical documents and its output with accessibility. In: Miesenberger, K., Klaus, J., Zagler, W.L., Burger, D. (Eds.), Computers Helping People with Special Needs. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp. 648–655.

Sweller, J., 1988. Cognitive load during problem solving: effects on learning. Cognitive Science 12, 257–285.

Tabachneck, H.J.M., Koedinger, K.R., Nathan, M.J., 1995. A cognitive analysis of the task demands of early algebra. In: Proceedings of the 17th Annual Conference of the Cognitive Science Society, pp. 397–402.

Tapia, E., Rojas, R., 2003. Recognition of on-line handwritten mathematical formulas in the e-chalk system. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR), pp. 980–984. IEEE.

Tapia, E., Rojas, R., 2005. Recognition of on-line handwritten mathematical expressions in the e-chalk system — an extension. In: Proceedings of the Eighth International Conference on Document Analysis and Recognition (ICDAR'05). Seoul, South Korea, pp. 1206–1210 Vol. 2.

Valentino-Devries, J., 2010. Using the ipad to connect:parents, therapists use apple tablet to communicate with special needs kids. Wall Street Journal 〈http://online.wsj.com/article/SB10001424052748703440004575547971877769154.htmlS〉(accessed 09.09.11).

Wood, C., 2002. Technologyandeducation. PC Magazine. 〈http://www.pcmag.com/article2/0,4149,15154,00.aspS〉 (accessed 09.09.11).

xThink, 2003. MathJournal. 〈http://www.xthink.com/MathJournal.html〉, (accessed 10.09.11).

Zanibbi, R., Blostein, D., Cordy, J.R., 2002. Recognizing mathematical expressions using tree transformation. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 1455–1467.

Zanibbi, R., Pillay, A., Mouchere, H., Viard-Gaudin, C., Blostein, D., 2011. Stroke-based performance metrics for handwritten mathematical expressions. In: Proceedings of the 2011 International Conference on Document Analysis and Recognition (ICDAR'11), Beijing, China, pp. 334–338.

Zeleznik, R., Miller, T., Li, C., LaViola, J.J., 2008. MathPaper: mathematical sketching with fluid support for interactive computation. In: Butz, A., Fisher, B., Krüger, A., Olivier, P., Christie, M. (Eds.), Smart Graphics. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp. 20–32.