Using Novices to Scale Up Intelligent Tutoring Systems

Andrew M. Olney
University of Memphis
Memphis, TN
aolney@memphis.edu

ABSTRACT

Intelligent Tutoring Systems (ITS) simulate the behavior and pedagogy of human tutors. Several meta-analyses have found that ITS are generally as effective as human tutors at promoting learning. Unfortunately, ITS are extremely expensive to produce, with some groups estimating that it takes 100 hours of authoring time from AI experts, pedagogical experts, and domain experts to produce 1 hour of instruction. The expense of creating ITS seems to be the largest barrier to scaling up ITS for widespread adoption. Some groups have created specialized authoring tools to address this problem. However, these authoring tools still require experts to use them. We have developed an alternative approach that replaces authoring tools with a new learning environment and replaces experts with novices. In our approach, novices read static content like books and web pages together with a virtual student who proposes summaries, questions, concept maps, and predictions about the content being read. The virtual student combines AI with the corrections of previous human novices to continuously improve its summaries, questions, concept maps, and predictions. Our previous research has shown that by correcting errors that the virtual student makes, human novices learn the content better than reading alone. Moreover, by correcting errors, human novices implicitly author the content needed to create an ITS. Our current research has created the infrastructure to implement this approach in the real world and has evaluated this infrastructure by generating over three thousand tutoring modules from the Navy Electricity and Electronics Training Series. The tutoring modules run on a previously developed ITS for electronics that tutors students by holding a conversation in natural language.

ABOUT THE AUTHORS

Andrew M. Olney holds an M.S. and Ph.D. in Artificial Intelligence and Computer Science and presently serves as Associate Professor in both the Institute for Intelligent Systems and Department of Psychology at the University of Memphis. Dr. Olney has served as PI or Co-PI on over $10 million in e-learning grants funded by the Institute of Education Sciences, the National Science Foundation, Army Research Laboratory, and the Office of Naval Research. His research has focused on intelligent tutoring systems, which simulate the behavior and pedagogy of human tutors, and has particularly focused on conversational tutoring systems that interact in natural language. He was a Co-PI on the team that won first place in the Office of Naval Research STEM Grand Challenge and a Co-PI on the Army Research Lab's Generalized Intelligent Framework for Tutoring, which has released one edited book on intelligent tutoring systems every year for the past five years. Dr. Olney is the current editor of the Journal of Educational Data Mining and has over 100 publications in the fields of artificial intelligence, education, and psychology.
.

Using Novices to Scale Up Intelligent Tutoring Systems

Andrew M. Olney
University of Memphis
Memphis, Tennessee
aolney@memphis.edu

INTRODUCTION

Intelligent tutoring systems (ITS) are artificial intelligence-powered computer programs designed to implement the behavior and pedagogy of human tutors. Not only are ITS twice as effective at promoting learning than the previous generation of computer-based instruction, but they are also generally as effective as human tutors (Kulik & Fletcher, 2016). Although they have existed for decades, intelligent tutoring systems have, for the most part, seen only limited deployments in education and industry (cf. Ritter, Anderson, Koedinger, & Corbett, 2007). Given the effectiveness of ITS, their lack of general use is somewhat surprising. The superiority of tutoring over classroom instruction is well-known (Bloom, 1984). Presumably, the reason we still have classrooms is one of cost: it is simply too expensive to provide individualized instruction with human tutors. By comparison, ITS can scale for almost no cost. After all, ITS are simply computer programs and can be copied just as easily as any other computer program. So why is there an ITS scaling problem?

In the ITS community, the scaling problem has primarily been defined as a problem of authoring (Sottilare, Graesser, Hu, & Brawner, 2015). Although ITS are cheaper to duplicate than human tutors, it is generally believed to take a hundred hours or more of authoring effort to create one hour of instruction for an ITS (Aleven, McLaren, Sewall, & Koedinger, 2009; Corbett, 2002). Under this view, it is not the distribution cost that limits the deployment of ITS so much as the upfront development cost. Accordingly, authoring tools for ITS have been an active area of inquiry for several decades, and recent examples of authoring tools are quite effective, reducing the hours of authoring per hour of instruction ratio from approximately 100/1 to 15/1 (Aleven et al., 2016).

However, the users of these authoring tools are typically experts. Building an ITS requires multiple types of expertise, including programming, artificial intelligence, instructional design, and subject matter expertise. The primary objective of most ITS authoring tools is to mitigate expertise in programming and artificial intelligence (Aleven et al., 2016; Murray, 1999). This is a clear improvement over not having authoring tools, but it also calls into question the larger issue of expertise in ITS development. One might argue that the field has traded one scarce resource (human tutors) for an even scarcer resource: a convergence of experts in programming, artificial intelligence, instructional design, and the subject matter in question. After all, a human tutor training program may take a few days to a few weeks, but the expertise to make an ITS takes many years—and a team of PhDs.

We have previously proposed and designed a system called BrainTrust that authors ITS without experts and without authoring tools (Olney & Cade, 2015). In our approach, novices read static content like books and web pages together with a virtual student who proposes summaries, questions, concept maps, and predictions about the content being read. The virtual student combines AI with the corrections of previous human novices to continuously improve its summaries, questions, concept maps, and predictions. Our previous research has shown that by correcting errors that the virtual student makes, human novices learn the content better than reading alone. Moreover, by correcting errors, human novices implicitly author the content needed to create an ITS. We call this general approach authoring implicitly with a symmetric learning task (AISLT). In this paper, we describe an end-to-end implementation of our system and present results from using our system to generate an ITS from the Navy Electricity and Electronics Training Series (NEETS) textbooks.

In the following sections, we motivate our approach by briefly describing types of ITS and their corresponding authoring tools. We then focus on a particular type of ITS known as dialogue-based tutors (Nye, Graesser, & Hu, 2014), which are particularly suitable for our approach, and describe how we map authoring of dialogue-based tutors to activities in a secondary learning environment. The remaining sections describe the efforts of the present study, including converting the NEETS into web pages, extracting initial knowledge representation hypotheses from the

NEETS, the implementation of the client/server for AISLT, and the generation of ITS from the curated knowledge representations.

AUTHORING INTELLIGENT TUTORING SYSTEMS

Given the approximately 50 years of research on ITS, variability in ITS structure and strategies is fairly high. However, there is substantial agreement in the field about what an ITS should do in terms of adapting to an individual student (Kulik & Fletcher, 2016). ITS adaptivity can be thought of as having two levels: an outer loop of problem selection and an inner loop of problem steps (VanLehn, 2006). Traditional computer-based instruction systems, which are less adaptive than ITS, only have an outer loop. As a result, they provide feedback only on student solutions. In contrast, ITS have an outer loop and an inner loop, where the inner loop gives students feedback on each step of the solution. This ability to give immediate feedback on student steps is consistent with theories of human tutoring (Olney, 2014). How a given ITS instantiates the inner and outer loops varies significantly in practice and has implications for authoring. For the sake of discussion, we focus on three popular ITS paradigms: model- or example-tracing tutors (Anderson, Corbett, Koedinger, & Pelletier, 1995; Aleven et al., 2016), constraint-based tutors (Mitrovic, 2012), and dialogue-based tutors (Nye et al., 2014).

Model- or example-tracing tutors are path-oriented and make a direct comparison to an expert solution. Historically, these tutors are based on the Adaptive Control of Thought-Rational (ACT-R) cognitive architecture, which represents procedural knowledge as production rules (Anderson et al., 1995; Anderson & Lebiere, 1998). Given an initial state (i.e., a dictionary of name-value pairs), a single matching production rule will generate a new state. Because multiple production rules may match a given state, the application of all production rules, iteratively, produces a large (potentially infinite) tree rooted in the initial state. Thus, for performance reasons, the implicit paths created by production rule application may be generated at runtime, pre-computed, or constrained by student input (Heffernan, Koedinger, & Razzaq, 2008). The process of model-tracing matches student input to expert knowledge (as corresponding production rules) by applying production rules to the current state to generate a frontier of possible states and then searching for the student input in that frontier (cf. depth- or breadth-first search). Because the student may make a mistake inconsistent with expert knowledge, model-tracing tutors typically include so-called "buggy production rules" to model these mistakes. Without these buggy rules, model-tracing would be unable to interpret erroneous student inputs, because they would be outside what the expert model could represent. From an authoring standpoint, model-tracing requires cognitive modeling and AI expertise to express all possible solutions to a problem in terms of production rules as well as buggy rules for common errors (Blessing, Gilbert, Ourada, & Ritter, 2009). Authors must also create instructional messages tied to the cognitive model, including remediation messages for the application of buggy rules and hints for when students get stuck. Recent work has tried to automate authoring model-tracing tutors by using inductive logic programming (Li, Matsuda, Cohen, & Koedinger, 2015; Matsuda, Cohen, & Koedinger, 2015), but this work still requires programming and AI expertise to bootstrap the system and correct the resulting production rules.

Example-tracing tutors follow the path-oriented, direct-comparison approach of model-tracing tutors but without production rules. Instead, example-tracing tutors record solution paths (and buggy paths) demonstrated by experts and then generalize them to allow for divergent behavior (Aleven et al., 2009; Aleven et al., 2016). As with model-tracing, these paths require authoring appropriate error messages and next-step hints. Additionally, authoring example-tracing tutors usually involves additional annotations to the solution paths to allow additional flexibility in interpreting student input, a process known as generalizing the solution paths. Generalization addresses the fundamental difference between model-tracing and example-tracing, which is that model-tracing generates solution paths by recombining production rules in various ways, while example-tracing (without generalization) has rigid prescripted solution paths. Generalization requires authoring of additional annotations to solution paths specifying whether steps can be done in alternate orders, skipped, repeated, or parameterized such that a range of student inputs could match a step (Aleven et al., 2009). In theory, the production rules created for model-tracing can be reused across multiple problems in a domain, while example-tracing solution paths are specific to the problems they are authored for as well as for isomorphic problems. However, despite the potential reuse of production rules in model-tracing tutors, example-tracing tutors are arguably more cost-effective. Notably, example-tracing tutors created with the CTAT authoring tool obviate the programming and AI expertise needed to create model-tracing tutors (Aleven et al., 2009).

Constraint-based tutors (Mitrovic, 2012) are constraint-oriented and make an indirect comparison to an expert solution. Although constraints may be defined in various ways (cf. Ohlsson & Mitrovic, 2007), the definition that

most distinguishes them from path-based approaches is as a test for a current state of affairs. Under this definition, constraints can be thought of as the necessary and sufficient conditions to define the solution set, i.e., the set of all possible solutions to a problem. In other words, solutions that do not violate any constraints are correct, and the violation of a constraint implies a solution is incorrect. Constraints do not specify a path or set of paths; they define a space of correct solutions. Each constraint is defined in terms of a relevance condition and a satisfaction condition (Ohlsson, 1992). The relevance/satisfaction relationship is similar to the *if/then* logic of a production rule, except that instead of the *then* portion being an action (e.g., *if* problem statement is a⁄b + c⁄d *then* **multiply** a⁄b by d⁄d and c⁄d by b⁄b), the *then* portion defines what must be satisfied or true in a student solution (e.g., *if* problem statement is a⁄b + c⁄d, and student solution is (a + c)⁄n *then* **b=d=n**. Because constraints specify what must be true of a solution rather than a particular solution or the steps leading to a solution, they are particularly well-suited for ill-defined tasks. Alternative definitions of constraints include path constraints, which are constraints on solution step order; semantic constraints, which are constraints that reference an external criterion; and syntactic constraints, which reflect what must be true at the step level (e.g., age must be a non-negative number and constant throughout the problem). These alternative constraints allow constraint-based tutors to match the behavior of path-oriented tutors more closely. Because authoring constraint-based tutors requires significant expertise in programming and AI, specialized authoring tools like ASPIRE have been developed (Mitrovic et al., 2009). However, although ASPIRE does not require programming, it does require manual construction of an ontology, which arguably requires some AI expertise.

The third major ITS paradigm is dialogue-based tutors (Nye et al., 2014), which are constraint-oriented and make a direct comparison to an expert solution. Although dialogue-based tutors have a long history, perhaps the most influential dialogue-based tutor of the last two decades is AutoTutor (Graesser, Chipman, Haynes, & Olney, 2005), which has influenced over two dozen other ITS (Nye et al., 2014). Due to space constraints, the following discussion focusses on AutoTutor as representative of dialogue-based tutors, though differences exist amongst various systems. Rather than use production rules or constraints to model knowledge and then annotate these knowledge representations with language, e.g., error remediation when a student violates a constraint, AutoTutor represents knowledge using language and interacts with students using language. In other words, in dialogue-based ITS like AutoTutor, everything is based on dialogue and language. The structure of the dialogue is based on observations of human tutoring (Person, Graesser, Magliano, & Kreuz, 1994; Graesser, Person, & Magliano, 1995; Olney, Graesser, & Person, 2010), and follow what Graesser and colleagues refer to as the 5-step tutoring frame:

1. Tutor poses the problem
2. Student attempts to answer
3. Tutor provides brief evaluation and feedback
4. Student and tutor have a multi-turn dialogue to improve the answer
5. Tutor assesses whether student understands the answer

The fourth step of this frame is where most of the rich tutoring interaction occurs, and this rich interaction has been described by Graesser and colleagues as Expectation and Misconception Tailored (EMT) dialogue. An expectation is part of the ideal answer to the problem or question posed by the tutor. Commonly, the ideal answer has multiple such expectations, each represented by a sentence, such that the ideal answer as a whole is represented by a multi-sentence paragraph. During EMT dialogue, the tutor works with the student to improve their answer to bring it closer to the ideal answer. To achieve this goal, tutors use various dialogue strategies like asking leading questions, providing feedback to student answers, providing examples, requesting clarifications, rearticulating solutions, and making metacognitive comments, e.g., “This is how to remember ...”

The full complexity of EMT dialogue has not yet been addressed by any dialogue-based tutor. Most of these ITS use just 11 of the 34 “dialogue moves” found in human tutoring (Olney et al., 2010), problem statement, pump, repetition, encouragement, hint, prompt, assertion, positive/neutral/negative feedback, and summary, and of these, only hint-prompt-assertion is used to implement EMT dialogue. EMT dialogue proceeds as follows. First, the tutor assesses whether any expectations have not been covered by the student’s answer. This assessment is done using a statistical technique called Latent Semantic Analysis (LSA; Landauer, McNamara, Dennis, & Kintsch, 2007). In brief, LSA compares student answers to expectations by first converting both into vectors and then calculating the angle between the vectors. If the angle is close to one, the answer and expectation are almost identical. Typically, a threshold is set between zero and one to capture whether a student’s answer is close enough to the expectation to be considered correct. Although LSA is not very precise, e.g., it is insensitive to word order, it has the advantages of being robust to noise and does not require labeled data to train, only a collection of documents in the domain. AutoTutor uses LSA to check

whether an expectation has been covered and select the next expectation (Graesser et al., 2005). Once an expectation has been selected, AutoTutor uses the hint-prompt-assertion strategy to get the student to articulate the expectation. The hint-prompt-assertion strategy tries to get the student to do as much of the explaining as possible by starting with a hint but backing off to a prompt if the hint is not successful at covering the expectation, and further backing off to an assertion (a paraphrase of the expectation) if the prompt is not successful. For example, the expectation "The force of gravity pulls the balls downward," might have as a hint, "How does the Earth's gravity affect objects?" and a prompt, "Gravity pulls objects in a direction that is _____?" After each student answer, AutoTutor provides positive/neutral/negative feedback, and if the student covers the expectation, AutoTutor either moves to the next expectation or moves to the summary if all expectations are covered. Dialogue-based tutors are constraint-oriented and make and direct comparison to an expert solution: covering each expectation is a constraint, and the expectations themselves are part of an expert answer.

The requirements for authoring a dialogue-based ITS like AutoTutor are relatively simple. Authors must create a problem statement and corresponding ideal answer, break the ideal answer into expectations, and then create multiple (usually two) hints, prompts, and assertions for each expectation, and write a summary (optionally identical to the ideal answer). Additionally, authors must collect documents in the domain suitable for creating an LSA space; typically, one or two textbooks are used, but it is also possible to use a general document collection and then cover domain-specific vocabulary by augmenting the corpus with selected Wikipedia pages (Riordan, Dale, Kreuz, & Olney, 2011). Because the inherent complexity of authoring dialogue-based ITS is low, few authoring tools have been developed. Early authoring tools were essentially forms guiding the user towards entering the information above and require no programming or AI expertise (Susarla, Adcock, Eck, Moreno, & Graesser, 2003). Notably, later authoring tools were more complex (Cai, Graesser, & Hu, 2015), e.g., requiring flowcharts to sequence conversations, but these authoring tools focus on trialogues between the user, a peer agent, and a tutoring agent, which is a conceptually distinct and more complicated task than authoring tutorial dialogues.

The major ITS paradigms discussed in this section have corresponding authoring tools to reduce the expertise required to create an ITS. The authoring tools for path-based ITS focus on eliminating the need for programming and AI expertise, and the authoring tools for constraint-based ITS focus on eliminating the need for programming and some (but not all) AI expertise. Dialogue-based ITS, in contrast, do not require any programming or AI expertise. However, what all the major ITS paradigms share, and what their authoring tools have not been able to eliminate, is the need for domain expertise. In each case, a domain expert is needed to pose problems, one or more solutions, and generate hints or other feedback messages.

AUTHORING IMPLICITLY WITH A SYMMETRIC LEARNING TASK (AISLT): BRAINTRUST

While eliminating or reducing the need for programming and AI expertise for ITS authoring is a significant achievement, it does not address the shortage of experts motivated to use these tools. The shortage of experts is a non-trivial problem. It requires many years of practice in a domain to become an expert (Ericsson, Krampe, & Tesch-Römer, 1993), and experts typically use that expertise for endeavors unrelated to authoring ITS content. Particularly in fields where new discoveries or changing educational standards alter the curriculum, experts are needed not just once but on an ongoing basis to update and curate the content.

We have previously proposed and designed an approach that addresses the shortage of motivated experts (Olney & Cade, 2015). This approach, which we call authoring implicitly with a symmetric learning task (AISLT; pronounced *eyelet*), considers the problems of expertise and motivation independently. Expertise is addressed by replacing experts with novices using a crowdsourcing strategy (cf. wisdom of the crowds). Using crowdsourcing, each novice's work is checked (and potentially edited) by other novices until confidence in correctness is high. This approach has been called iterative improvement in the human computation literature, where it has been used with novices to check and create ontologies (von Ahn, 2005; Cycorp, 2005). Motivation is addressed by embedding the authoring task in some other activity in which the novices are already engaged. This cloaking of the true task is sometimes referred to as implicit crowdsourcing; a common example is the reCAPTCHA where users select images containing cars or street signs in order to prove they are human, producing labeled data as a side effect (von Ahn, Maurer, McMillen, Abraham, & Blum, 2008). The motivation of our implicit authoring approach is highly similar to reCAPTCHA: authoring an ITS is a side effect of doing work the novices would already be doing, namely studying their class textbook.

Our specific AISLT implementation, BrainTrust, enhances assigned reading activities with a peer agent who tries to learn with the human student. As the human student reads, the peer agent enacts various reading comprehension strategies based on the reading, and the human student corrects the peer agent. We specifically chose the four reading comprehension strategies of reciprocal teaching (Palincsar & Brown, 1984), because they not only help students better understand the text (National Institute of Child Health and Human Development, 2000) but also reflect the work needed to create a dialogue-based ITS (Olney & Cade, 2015). In reciprocal teaching, teacher and students take turns reading the text while generating questions, clarifying concepts, summarizing, and making predictions about what's coming up next. Each participant comments on and contributes to the questions, summaries, etc. produced by other participants. The work done executing reciprocal teaching strategies overlaps with the work done for authoring dialogue-based ITS. In terms of our AISLT framework, reciprocal teaching strategies are a symmetric learning task with respect to dialogue-based ITS. Table 1 compares reciprocal teaching strategies, their realizations in BrainTrust, and the authoring tasks they address.

Table 1. Alignment of reciprocal teaching (RT) with BrainTrust and authoring activities

RT Activity	BrainTrust Activity	Authoring Task
Summarizing	Gist	Problem Statement Creation
Asking Questions	Question Generation	
Clarifying Concepts	Concept Mapping	Expectation & Question Creation
Predicting	Predicting	

BrainTrust is inspired by reciprocal teaching but adapts it to fit a 1-to-1 interaction where the roles aren't symmetric (the peer agent is always the enactor, corrected by the human student). During a BrainTrust session, the human student first reads a text block (between 250 words and a page) and then corrects the peer agent as the peer agent enacts the activities as ordered in Table 1. First, the gist activity summarizes what the text is about in a single word or phrase. For example, the peer agent might say, "I think this is about atoms, is that right?" and display this text on the screen, giving the human student the ability to edit the word *atoms*. This high-level summary is useful for the authoring of tutor openings and problem statements for a text block like, "What can you say about atoms here?" Next, during question generation, the peer agent generates questions about the text, and students may edit the question and also give an answer. Question generation currently has no role in authoring, but it could potentially be used to validate questions generated from the concept map were created without errors. Perhaps the most key activity in BrainTrust is concept mapping. The concept map operationalizes the reciprocal teaching strategy of clarifying concepts (i.e., as opposed to verbally) and can also be viewed as a low-level summary. For authoring, concept maps can be used to automatically author tutor expectations and questions using AI and natural language processing techniques (Olney, Person, & Graesser, 2012; Olney, Graesser, & Person, 2012). Such questions range from relatively shallow questions targeting a definition or property, e.g., "What are molecules made of?" as well as deeper causal questions, e.g., "Why are ions formed?" Finally, the prediction activity has the peer agent guess what is coming up next. In summary, concept maps can be used to author expectations, assertions (identical or paraphrased expectations), hints, and prompts, and the other activities serve to filter and validate this content. Gists can be used to author openings or problem statements. The other activities have no authoring purpose in the current system but are kept to increase motivation (Olney & Cade, 2015).

Previous work on BrainTrust focused on design (Olney & Cade, 2015), with the goal of implementing a design that could both help students learn while reading and also implicitly author an ITS. This is a key property of the AISLT approach that distinguishes it from schemes like reCAPTCHA: implicit authoring and learning are happening simultaneously. The methodology used in that work centered on mock-up interfaces, i.e., interfaces that presented the same scripted materials to all participants. Results from that work suggest that BrainTrust can help students learn more than reading alone and that students are able to correct peer agent errors, demonstrating that AISLT approach is feasible. Our current research has created the infrastructure to implement this approach in the real world (see Figure 1) and has evaluated this infrastructure by generating knowledge representations and tutoring modules from approximately five thousand pages of books from the Navy Electricity and Electronics Training Series. The automatically authored tutoring modules run on a dialogue-based ITS for electronics. The remaining sections of this paper describe this effort in three phases: the bootstrap system that creates the initial peer agent knowledge representation hypotheses and dialogue, the improvement system that presents peer agent hypotheses to the user and saves user modifications, and the generation system that uses the curated peer agent knowledge representations to

author tutoring modules for a dialogue-based ITS. We report evaluations on the bootstrap system and the generation of tutoring modules only; evaluation of the human computation system remains for future work.

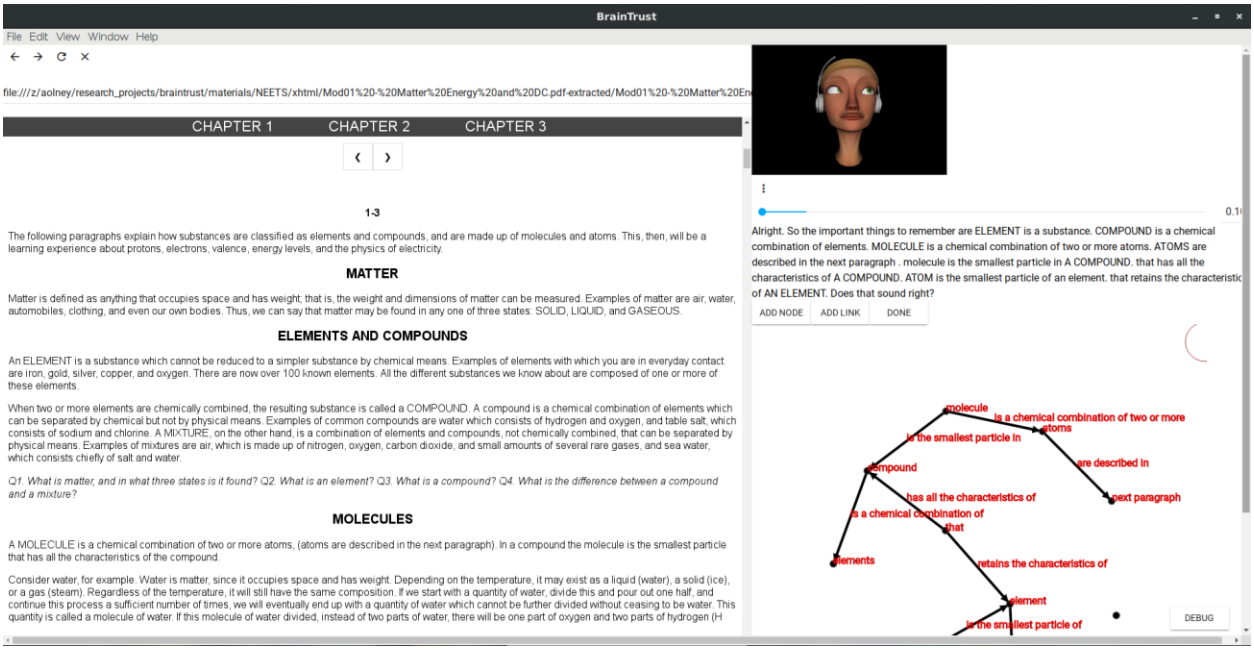

Figure 1: BrainTrust during a concept mapping activity

BOOTSTRAP SYSTEM

The design of BrainTrust presupposes that the learning materials are already online in a machine-readable format (e.g., HTML). This is a reasonable assumption as most online textbooks provide machine-readable access for screen readers used by the visually impaired. Because the present study used legacy materials in PDF form, and such legacy materials may present complications for years to come, we briefly describe the process of making these materials machine-readable.

Legacy Format Conversion

The Navy Electricity and Electronics Training Series (NEETS) (Navy, 1998) is a series of self-study manuals on electronics for various Navy ratings. The NEETS contain 24 manuals consisting of 4976 pages total. The PDF-formatted NEETS contain text in a variety of fonts and styles, as well as images and diagrams. Thus, to make the NEETS usable by BrainTrust requires conversion of the PDFs to HTML in a way that preserves the semantics of the document elements. For example, the NEETS contain in-text questions and answers, so these must be distinguished from main body text for the purposes of parsing and knowledge extraction. Typical HTML would add these document semantics using different kinds of tags, e.g., *<p>* for main body text, or class attributes to identify non-body elements. Unfortunately, off the shelf PDF conversion tools do not add these document semantics, so a custom approach is required. In the present study, we used the following procedure, though alternative procedures are no doubt possible. First, the PDFs were converted to non-annotated HTML using the Apache Tika toolkit (Apache Foundation, 2017). In this first-pass HTML, every text element was annotated with a *<p>* tag, every image was annotated with a *<img>* tag, and every page was annotated with a * * tag. Next, a custom parser was manually developed to extract the semantic structure of the document from the first-pass HTML. The parser used custom logic based on the structure of the NEETS. A full description of the parser is outside the scope of the present discussion, but it categorized pages into types (e.g., table of contents, main text, and index), headers into types (e.g., chapter, section, and subsection), and page components into types (e.g., image caption, learning objective, or paragraph) as it traversed the raw HTML returned by Tika. The resulting semantic structure was stored as a JSON object and used to generate a new HTML file for each NEET with a structure consistent with its document semantics. For example, *<p>* tags were replaced with more appropriate tags and those tags were annotated with class attributes like *<h3 class="subsection">*. Top level navigation and pages links were also generated at this stage. Finally, a cascading style sheet was manually created to

match the format of the NEETS as closely as possible. The left-hand side of Figure 1 shows an example NEET transformed by this process. The full transformed NEETS are available at https://olney.ai/neets-web/.

Knowledge Extraction

Our previous work used semantic parsing combined with a predefined set of key terms (e.g., from a glossary or index), together with a link ontology to extract concept maps from text (Olney, Cade, & Williams, 2011; Olney, Person, & Graesser, 2012; Olney, Graesser, & Person, 2012). However, in a general AISLT system, a predefined set of key terms cannot be assumed. Additionally, the link ontology used in previous work would require the AISLT users to have some understanding of AI to use effectively. Thus, we developed a new methodology avoiding key terms and a link ontology and compared to a baseline methodology using key terms.

Procedure

NEETS main body text was parsed at the chapter level using syntactic and discourse parsing with coreference resolution (Surdeanu, Hicks, & Valenzuela-Escarcega, 2015). Coreference resolution outputs chains of words or phrases that refer to the same thing. For example, *she*, *her*, and *Mary* might be a coreference chain crossing several sentences about Mary. Discourse parsing segments the text into elementary discourse units (i.e., a main or subordinate clause) and annotates these units with relationships like *contrast* or *elaboration*.

The two key tasks of the bootstrap AI are identifying the gist and concept map triples of each page (see Table 1). The bootstrap AI selected a gist for each page by selecting the coreference chain with the greatest length on the page, removing stopwords (common words). The criterion of using the longest chain ensures that the gist is the most centrally connected phrase on the page. The triples for each page were selected using two different procedures involving coreference chains. The first procedure (KEY) selected all sentences containing terms having coreference chains of length two or more and which had a term in the chain that was a key term (key terms in the NEETS are marked by uppercase formatting) such that each sentence had at least two such chains. Note that this use of key terms for selection of sentences is different from previous use of key terms for defining nodes in a concept map. The second procedure (CLOZE) follows previous work on generating cloze practice items (Olney, Pavlik, & Maass, 2017), which selects all sentences containing terms having coreference chains of length two or more such that each sentence had three such chains, excluding stopwords. After both procedures selected sentences on a page, those sentences were broken into elementary discourse units that were then converted into triples using concept map generation. Concept map generation used a relaxed version of previously described methods (Olney et al., 2011; Olney, Graesser, & Person, 2012) in the sense that no key terms were assumed, no link ontology was used, and a single triple was generated from each elementary discourse unit. A full description of this process is outside the scope of the present discussion, but the basic process was 1) use the subject plus modifiers as the start node, 2) use the main verb and any following prepositions as the edge, and 3) use the remaining text as the end node. For example, the elementary discourse unit, "the compass needle will move in the opposite direction," will become the triple <*compass needle*> - <*move in*> - <*opposite direction*>. Our evaluation of the bootstrap AI focused on gists and triples, but we briefly note that for human computation purposes, questions can be generated from these triples using existing methods (Olney, Graesser, & Person, 2012), or in cases like the NEETS where in-text question/answer pairs are also present, those question/answer pairs can be used as well. Finally, predictions were simply the gists of following pages.

Results & Discussion

The primary metrics for evaluating the bootstrap AI are successful creation of gists and successful creation of triples. Of the 3328 NEETS pages containing content (i.e., not table of contents, index, etc.), all had successful creation of gists. Triple creation differed dramatically between KEY and CLOZE procedures as shown in Table 2.

Table 2. Triple generation of KEY and CLOZE procedures in bootstrap AI

Procedure	Success	Median*	Q1*	Q3*
KEY	.328	1	1	2
CLOZE	.961	5	3	8

Note. Q1= 25th percentile; Q3=75th percentile
*Calculated for success cases only

The CLOZE procedure was able to recover at least one triple on nearly all pages, while the KEY procedure was only able to recover that many on a third of the pages. In the middle 50% of those successful cases, the CLOZE procedure yielded 3-8 triples compared to the KEY procedure's 1-2 triples. The difference between the two procedures appears to be that the KEY procedure's reliance on key terms specified by the NEETS is too restrictive—more restrictive than the CLOZE procedure requiring an additional chain per sentence. These results strongly suggest that key terms are not needed to get good coverage, and they additionally suggest that key terms may sometimes be insufficient for this task.

IMPROVEMENT SYSTEM

User Interface

The user interface in Figure 1 was largely informed by the previous mock-up interface (Olney & Cade, 2015). However, the mock-up interface was created using Microsoft's Silverlight plugin technology, and browser plugins have since fallen out of favor with many browsers because of the security risks they create. Accordingly, a new interface was developed using JavaScript for client-side scripting. The user interface itself was written in the F# language and transpiled to JavaScript using the Fable compiler (Garcia-Caro, 2017). The interface is an Electron application (GitHub, 2016) with an embedded Chrome browser for displaying web pages. Electron is a cross-platform framework for developing desktop applications using web technologies and can run on essentially any platform that can run the Chrome browser. The new interface's embedded browser allows students to authenticate with web pages as needed and to recover the structure of those web pages through the embedded browser API. In other words, the embedded browser allows total access to any web page the user reads, even pages that require authentication and use an encrypted connection. Most UI elements (e.g., buttons, text boxes, etc.) use the React framework for single page applications (Facebook, 2018) except for the peer agent and the concept map editor. The peer agent uses ThreeJS for JavaScript-based animation (Cabello, 2018; Stickman Ventures, 2017) and MaryTTS for text to speech capability, running as a local or remote server (Schröder & Trouvain, 2003). The concept map editor is written in D3, a JavaScript library for visualizing and interacting with data (Bostock, 2018). D3 provides a fine grain level of control over the concept map, including custom graph layouts. Like any single page application, the interface changes based on the current state of the application. The left-hand side, which contains the embedded browser, is constant throughout so that the user can always refer to the text. The right-hand side only appears when the peer agent is being taught and displays a single reading comprehension activity at a time (see Table 1). During each of the activities, the peer agent articulates the content of the current task, followed by a brief pause, after which the text of the task appears below the agent and becomes editable. Each activity has a *Done* button; when the user finishes the activity and presses the button, the interface presents the next activity if any.

Human Computation

The human computation server is built on top of the Node.js JavaScript runtime environment (Node.js Foundation, 2018) and the Express web application framework (Holowaychuk, 2018). Express both supports an authentication/registration website for BrainTrust as well as a REST-ful API for human computation tasks. Because the server requires all users to be authenticated, users must register for an account using an email address or Google OAuth and log in before using BrainTrust. User registration information is stored in a MongoDB database (MongoDB, 2018) together with user attributes for ability. MongoDB is a document-oriented NoSQL database (i.e., not a relational database); BrainTrust uses MongoDB for users (authentication, ability) and for URLs (URL, human computation). The available set of activities that a peer agent can invoke is determined by the current URL in the embedded browser of the user interface and the current state of the human computation process for that URL, as stored in MongoDB. This design assumes that each URL is a suitable size unit of text for the purpose of correcting the peer agent. In the NEETS version of BrainTrust, each page of text has a unique URL. This may be a limitation, and future work may need to have methods of subdividing pages and storing them separately in the database. Each URL entry in MongoDB contains a history of all the activities completed by users on that URL, i.e., a human computation timeline. For example, if a user completes all the activities for a text shown in Table 1, their revised activities are stored with their user identifier and their ability rating at that moment in time. Ability is therefore allowed to change over time without compromising the ability assessment for a particular set of revised activities. The initial bootstrap AI entry is coded as its own user, e.g., AIv1, and the bootstrap AI is invoked whenever a requested URL has no existing set of activities. When the user interface makes a request for activities (i.e., for the peer agent to present), the human computation algorithm traverses the history of all activities for that URL and selects the optimal activities using the following

criteria. First, each activity revision is scored by the ability of the user who produced it. Second, activity revisions that exactly match across users are rescored as the multiplication of those user's abilities. Finally, the highest scored activity revision is returned to the user. In the case of triples, because our previous research suggests that users are better at noticing incorrect information than adding missing information, extra triples are intentionally added to the concept map to help ensure completeness.

INTELLIGENT TUTORING SYSTEM GENERATION

Procedure

Generation of a dialogue-based ITS from the human computation database is currently an off-line batch process. A JSON data dump from MongoDB corresponding to the optimal activities for each URL was used as the basis for generating dialogue-based ITS modules. Because our evaluation did not include any improvement from human computation, this database dump is equivalent to the output of the bootstrap AI. These data are transformed in various ways to create tutoring modules, as described next.

As previously discussed, only the gist and concept map activities are used for authoring. The gists, being single words or phrases, are used with natural language generation templates to produce openings and problem statements. For example, the gist, "energy levels of electrons," could produce the opening, "Let's talk about energy levels of electrons," or the problem statement, "What can you say about the energy levels of electrons?" The triples are used to generate everything else in the tutoring module. Each triple is converted into a natural language sentence to create a corresponding expectation. The concatenation of expectations becomes the ideal answer to the problem. To elicit each expectation, hints and prompts are created using natural language generation techniques (Olney, Graesser, & Person, 2012). Each expectation was parsed using a syntactic and semantic parser that assigned semantic role labels and predicates (Johansson & Nugues, 2008). Prompts are generated targeting the subject, object, and any semantic adjuncts of the parsed expectations. For example, the expectation, "the second shell contains 8 electrons when full," would yield the questions, "What contains 8 electrons when full?", "What does the second shell contain when full?", and "When does the second shell contain 8 electrons?" for subject, object, and adjunct targets respectively. Hints are somewhat simpler to generate because they make use of templates targeting the same information as prompts, e.g., "Tell me about the second shell." for the subject target. The tutorial dialogue generated from all these operations is formatted for the AutoTutor Conversation Engine (ACE) XML schema, at which point the finished module is uploadable to any ACE-conformant ITS. For the present study, resulting ACE tutoring modules were uploaded to ElectronixTutor (Graesser et al., 2018).

Results & Discussion

The primary metrics for evaluating ITS generation are the successful creation of minimally complete modules (i.e., at least one gist, triple, hint, and prompt was generated) and their properties. Ideally, modules will contain multiple triples each with multiple hints and prompts. Complete module creation again differed dramatically between KEY and CLOZE procedures as shown in Table 3.

Table 3. Tutoring module generation following KEY and CLOZE procedures in bootstrap AI

		Triple			Hint			Prompt		
Procedure	**Success**	**Median***	**Q1***	**Q3***	**Median***	**Q1***	**Q3***	**Median***	**Q1***	**Q3***
KEY	.315	1	1	2	4	3	7	3	2	5
CLOZE	.960	5	3	8	18	10	29	12	7	19

Note. Q1= 25th percentile; Q3=75th percentile
*Calculated for success cases only

Using the gist and triples from the CLOZE procedure, minimal complete modules were successfully created for nearly all pages, while the KEY procedure's gists and triples could only successfully generate minimal complete modules for about a third of the pages. These results are not surprising given the results of Table 2 because, without triples, it is impossible to generate a minimal complete module. However, these results further indicate that hints and prompts

can almost always be generated from a triple successfully, as the success rates for minimal complete modules are just slightly lower than the success rates for triples.

In the middle 50% of successful complete modules, both the CLOZE and KEY procedures led to about two hints and prompts per triple, which is relatively ideal for dialogue-based tutors. However, because CLOZE procedure yielded 3-8 triples compared to the KEY procedure's 1-2 triples, these CLOZE procedure tutoring modules were much more fully developed in terms of dialogue, comparable in size to manually-authored tutoring modules.

CONCLUSIONS

While authoring tools for model-tracing, constraint-based, and dialogue-based tutors have attempted to reduce or eliminate the programming and AI expertise needed to create an ITS, they all require domain expertise to create ITS content. The AISLT approach provides a solution to this problem by replacing domain experts with novices who implicitly author an ITS while engaged in a symmetric learning task. The specific AISLT implementation described in this paper, BrainTrust, has novices implicitly author a dialogue-based tutor by performing reading comprehension exercises while reading their textbook. In this paper, we described the BrainTrust infrastructure and several evaluations of the infrastructure's bootstrap AI and ITS module generation. Our studies to date continue to support the AISLT approach to authoring but with several limitations and issues for future work. First, the current AISLT implementation, BrainTrust, is only applicable to dialogue-based tutors. This limitation stems from the close correspondence between BrainTrust's reading comprehension activities and the authoring tasks of a dialogue-based tutor. For a procedural domain requiring a path-oriented approach, it is unclear how the current reading comprehension activities would apply or what the alternative symmetric learning task would be. Similarly, it is unclear how the current reading comprehension activities could be used to author constraint-based tutors. Unless these limitations are overcome, an AISLT implementation like BrainTrust will not be effective for domains like mathematics and programming, though it will likely be effective for conceptual problem-solving domains where dialogue-based tutors have been built (see Nye et al., 2014, for a review). Another limitation is that it is still unknown whether the human computation process described here will iteratively improve knowledge to an acceptable state for generating tutoring modules, which is a key requirement for a successful AISLT implementation. Our previous work suggests that students are able to correct faulty knowledge, but it is possible that common misconceptions could creep in and contaminate tutoring module generation. To test this possibility would require several longitudinal trials in different subject areas. The major contribution of this paper was to present the complete BrainTrust infrastructure and show that it is ready to explore these questions within the broader AISLT framework.

ACKNOWLEDGEMENTS

This research was supported by the National Science Foundation (1352207), Institute of Education Sciences (R305C120001; R305A130030), Office of Naval Research (N00014-16-C-3027), and Army Research Laboratory (W911NF-12-2-0030). Any opinions, findings, and conclusions, or recommendations expressed in this paper are those of the author and do not represent the views of these organizations.

REFERENCES

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. R. (2009, April). A new paradigm for intelligent tutoring systems: Example-tracing tutors. *International Journal of Artificial Intelligence in Education*, *19*(2), 105–154.

Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S., ... Koedinger, K. R. (2016, Mar 01). Example-tracing tutors: Intelligent tutor development for non-programmers. *International Journal of Artificial Intelligence in Education*, *26*(1), 224–269.

Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive tutors: lessons learned. *The Journal of the Learning Sciences*, *4*(2), 167-207.

Anderson, J. R., & Lebiere, C. J. (1998). *The atomic components of thought*. Lawrence Erlbaum.

Apache Foundation. (2017). *Apache Tika.* Retrieved 2017-04-01, from https://tika.apache.org/

Blessing, S. B., Gilbert, S. B., Ourada, S., & Ritter, S. (2009, April). Authoring model-tracing cognitive tutors. *International Journal of Artificial Intelligence in Education*, *19*(2), 189–210.

Bloom, B. S. (1984, June). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, *13*(6), 4–16.

Bostock, M. (2018). *D3.js - Data-Driven Documents.* Retrieved 2018-06-10, from https://d3js.org/

Cabello, R. (2018). *three.js - Javascript 3d library.* Retrieved 2018-06-10, from https://threejs.org/

Cai, Z., Graesser, A., & Hu, X. (2015). ASAT: Autotutor script authoring tool. In R. Sottilare, A. Graesser, X. Hu, & K. Brawner (Eds.), *Design recommendations for intelligent tutoring systems* (Vol. 3, pp. 199–210). Orlando, FL: U.S. Army Research Laboratory.

Corbett, A. T. (2002, November). Cognitive Tutor Algebra I: Adaptive student modeling in widespread classroom use. In *Technology and Assessment: Thinking Ahead – Proceedings from a Workshop* (p. 5062). Washington, DC: National Academy Press.

Cycorp. (2005). *FACTory*. http://game.cyc.com/.

Ericsson, K. A., Krampe, R. T., & Tesch-R¨omer, C. (1993). The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, *100*(3), 363–4.

Facebook. (2018). *React - A JavaScript library for building user interfaces.* Retrieved 2018-06-10, from https://reactjs.org/index.html

Garcia-Caro, A. (2017). *Fable.* Retrieved 2018-06-10, from http://fable.io/

GitHub. (2016). *Electron: Build cross-platform desktop apps with JavaScript, HTML, and CSS.* Retrieved 2018-06-10, from https://electronjs.org/

Graesser, A. C., Chipman, P., Haynes, B., & Olney, A. M. (2005, November). AutoTutor: An intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions on Education*, *48*(4), 612- 618.

Graesser, A. C., Hu, X., Nye, B. D., VanLehn, K., Kumar, R., Heffernan, C., ... Baer, W. (2018, Apr 16). ElectronixTutor: an intelligent tutoring system with multiple learning resources for electronics. *International Journal of STEM Education*, *5*(1), 15.

Graesser, A. C., Person, N. K., & Magliano, J. P. (1995). Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology*, *9*, 1-28.

Heffernan, N. T., Koedinger, K. R., & Razzaq, L. (2008, April). Expanding the model-tracing architecture: A 3rd generation intelligent tutor for algebra symbolization. *International Journal of Artificial Intelligence in Education*, *18*(2), 153–178.

Holowaychuk, T. (2018). *Express - Node.js web application framework.* Retrieved 2018-06-10, from https:// expressjs.com/

Johansson, R., & Nugues, P. (2008). Dependency-based syntactic-semantic analysis with PropBank and NomBank. In *CoNLL '08: Proceedings of the Twelfth Conference on Computational Natural Language Learning* (pp. 183–187). Morristown, NJ: Association for Computational Linguistics.

Kulik, J. A., & Fletcher, J. D. (2016). Effectiveness of intelligent tutoring systems. *Review of Educational Research*, *86*(1), 42-78.

Landauer, T. K., McNamara, D. S., Dennis, S., & Kintsch, W. (Eds.). (2007). *Handbook of latent semantic analysis*. Mahwah, New Jersey: Lawrence Erlbaum.

Li, N., Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2015). Integrating representation learning and skill learning in a human-like intelligent agent. *Artificial Intelligence*, *219*, 67 - 91.

Matsuda, N., Cohen, W. W., & Koedinger, K. R. (2015, Mar 01). Teaching the teacher: Tutoring SimStudent leads to more effective cognitive tutor authoring. *International Journal of Artificial Intelligence in Education*, *25*(1), 1–34.

Mitrovic, A. (2012). Fifteen years of constraint-based tutors: what we have achieved and where we are going. *User modeling and user-adapted interaction*, *22*(1-2), 39–72.

Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., & Mcguigan, N. (2009, April). ASPIRE: An authoring system and deployment environment for constraint-based tutors. *International Journal of Artificial Intelligence in Education*, *19*(2), 155–188.

MongoDB. (2018). *MongoDB.* Retrieved 2018-06-10, from https://www.mongodb.com/index

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, *10*, 98–129.

National Institute of Child Health and Human Development. (2000). *Report of the National Reading Panel. Teaching children to read: An evidence-based assessment of the scientific research literature on reading and its implications for reading instruction*. Washington, DC: U.S. Government Printing Office.

Navy, U. (Ed.). (1998). *Navy electricity and electronics training series.* Naval Education and Training Professional Development and Technology Center.

Node.js Foundation. (2018). *Node.js.* Retrieved 2018-06-10, from https://nodejs.org/en/

Nye, B. D., Graesser, A. C., & Hu, X. (2014). AutoTutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education*, *24*(4), 427–469.

Ohlsson, S. (1992). Constraint-based student modeling. *International Journal of Artificial Intelligence in Education*, *3*(4), 429–447.

Ohlsson, S., & Mitrovic, A. (2007). Fidelity and efficiency of knowledge representations for intelligent tutoring systems. *Technology, Instruction, Cognition and Learning (TICL)*, *5*(2-3-4), 101–132.

Olney, A. M. (2014). Scaffolding made visible. In R. Sottilare, A. Graesser, X. Hu, & B. Goldberg (Eds.), *Design recommendations for intelligent tutoring systems: Instructional management* (Vol. 2, pp. 327–340). Orlando, FL: Army Research Laboratory.

Olney, A. M., Cade, W., & Williams, C. (2011, June). Generating concept map exercises from textbooks. In *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications* (pp. 111–119). Portland, Oregon: Association for Computational Linguistics.

Olney, A. M., & Cade, W. L. (2015). Authoring intelligent tutoring systems using human computation: Designing for intrinsic motivation. In D. D. Schmorrow & C. M. Fidopiastis (Eds.), *Foundations of augmented cognition* (Vol. 9183, p. 628-639). Springer International Publishing.

Olney, A. M., Graesser, A. C., & Person, N. K. (2010). Tutorial dialog in natural language. In R. Nkambou, J. Bourdeau, & R. Mizoguchi (Eds.), *Advances in intelligent tutoring systems* (Vol. 308, p. 181-206). Berlin: Springer-Verlag.

Olney, A. M., Graesser, A. C., & Person, N. K. (2012). Question generation from concept maps. *Dialogue and Discourse*, *3*(2), 75–99.

Olney, A. M., Pavlik Jr., P. J., & Maass, J. K. (2017). Improving Reading Comprehension with Automatically Generated Cloze Item Practice. In E. André, R. Baker, X. Hu, M. M. T. Rodrigo, & B. du Boulay (Eds.), *Artificial Intelligence in Education* (pp. 262–273). Springer

Olney, A. M., Person, N. K., & Graesser, A. C. (2012). Guru: Designing a conversational expert intelligent tutoring system. In P. McCarthy, C. Boonthum-Denecke, & T. Lamkin (Eds.), *Cross-disciplinary advances in applied natural language processing: Issues and approaches* (p. 156-171). Hershey, PA: IGI Global.

Palincsar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and comprehension-monitoring activities. *Cognition and Instruction*, *1*, 117-175.

Person, N. K., Graesser, A. C., Magliano, J. P., & Kreuz, R. J. (1994). Inferring what the student knows in one-to-one tutoring: the role of student questions and answers. *Learning and individual differences*, *6*(2), 205-229.

Riordan, M. A., Dale, R., Kreuz, R. J., & Olney, A. (2011). Evidence for alignment in a computer-mediated text-only environment. In L. Carlson, C. Hoelscher, & T. F. Shipley (Eds.), *Proceedings of the 33rd Annual Meeting of the Cognitive Science Society* (p. 2411-2416). Austin, TX: Cognitive Science Society.

Ritter, S., Anderson, J. R., Koedinger, K. R., & Corbett, A. (2007). Cognitive Tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review*, *14*(2), 249–255.

Schröder, M., & Trouvain, J. (2003, Oct 01). The German text-to-speech synthesis system MARY: A tool for research, development and teaching. *International Journal of Speech Technology*, *6*(4), 365–377.

Sottilare, R. A., Graesser, A. C., Hu, X., & Brawner, K. (Eds.). (2015). *Design recommendations for intelligent tutoring systems: Authoring tools & expert modeling techniques* (Vol. 3). Orlando, FL: U.S. Army Research Laboratory. (Available at: https://gifttutoring.org/documents/)

Stickman Ventures. (2017). *Stickman Ventures - Ginger WebGL Morph Demo*. Retrieved 2018-06-10, from https://sv-ginger.appspot.com/

Surdeanu, M., Hicks, T., & Valenzuela-Escarcega, M. A. (2015, June). Two practical rhetorical structure theory parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations* (pp. 1–5). Denver, Colorado: Association for Computational Linguistics.

Susarla, S., Adcock, A. B., Eck, R. N. V., Moreno, K. N., & Graesser, A. C. (2003). Development and evaluation of a lesson authoring tool for AutoTutor. In V. Aleven et al. (Eds.), *AIED2003 Supplemental Proceedings* (p. 378-387). Sydney, Australia: University of Sydney School of Information Technologies.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intelligence in Education*, *16*(3), 227–265.

von Ahn, L. (2005). *Human computation* (Doctoral Thesis). Carnegie Mellon University. (UMI Order Number: AAI3205378)

von Ahn, L., Maurer, B., McMillen, C., Abraham, D., & Blum, M. (2008). reCAPTCHA: Human-based character recognition via web security measures. *Science*, *321*(5895), 1465–1468.