7-21-2010

Implementation of AutoTutor Lite

Lu Han

To the University Council:

The Thesis Committee for Lu Han certifies that this is the final approved version of the following electronic thesis: "Implementation of AutoTutor Lite."

Mohammed Yeasin, Ph.D.
Major Professor

We have read this thesis and recommend
Its acceptance:

David J. Russomanno, Ph.D.

Andrew McGregor Olney, Ph.D.

Accepted for the Graduate Council:

Karen D. Weddle-West, Ph.D.
Vice Provost for Graduate Programs

IMPLEMENTATION OF AUTOTUTOR LITE

By

Lu Han

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Major: Electrical & Computer Engineering

The University of Memphis

August 2010

Abstract

Han, Lu. M.S. The University of Memphis. August/2010. Implementation of AutoTutor Lite. Major Professor: Mohammed Yeasin.

The Intelligent Tutoring System (ITS) is a very efficient form of e-Learning, but most of the current existing ITSs usually require advanced computational resources and specialized client software installation. Thus, there is a need for an ITS that is accessible online and is less computationally demanding. The immediate objective of this thesis is to describe the implementation of an online tutoring system that requires fewer computational resources. This system is called AutoTutor Lite, which runs in a web browser. Another objective is to use the Learner's Characteristics Curves (LCC) as the evaluation method in AutoTutor Lite. By utilizing the semantic representation, the LCC technology is successfully integrated into AutoTutor Lite. In the final system test and evolution, AutoTutor Lite meets all the design requirements, and LCC plays an important role in the system.

Table of Contents

List of Figures

Chapter 1 Introduction

The concept of an Intelligent Tutoring System (ITS) was first proposed in 1970 in order to provide a highly efficient form of e-Learning (Carbonell, 1970). After decades of research, many successful ITSs have been designed and developed. Most of the existing ITSs produced combine well understood human learning principles with computer technology to achieve maximum learning. Typically, ITSs start as research projects in a university or institute for academic research purposes, so the nature of such systems and implementation methods produces highly effective learning systems, but also introduces difficulties when scaling up. Furthermore those systems usually require advanced computational resources and specialized client software installation. For example, AutoTutor (Graesser, et al., 2004), an exemplar of ITS, has a specialized client installation and requires intensive computation for the backend language space. All of these reasons prevent people from using ITS widely and also influence the growth of e-Learning markets. However, new technologies like Rich Internet Applications (RIAs) make it possible to solve these problems. It enables researchers and developers to create new ITSs in a simpler and more productive way. Therefore, the goal of this study is to address the implementation of a lightweight and more efficient version of an ITS, namely AutoTutor Lite.

Another problem that exists is that most ITSs cannot easily take advantage of the learner's previous contributions or tutoring history. For example, when the learner constantly repeats previous answers, the tutoring systems have difficulty providing accurate feedback. The Learner's Characteristic Curves

(LCC) (Hu & Martindale, 2008) technology is used to evaluate the learner's performance by tracking and comparing the learner's previous and current contribution. Therefore, another objective in this study is to utilize LCC as the evaluation method in the AutoTutor Lite in order to provide better performance.

The following two questions/issues are explored in the thesis:

1. Is it possible to develop an ITS deliverable online using the latest Rich Internet Application (RIA) technology?
2. How can LCC be used as a simple student model for such a version of ITS, namely AutoTutor Lite?

This study provides a solution to the above questions. Specifically, it is primarily focused on technical details; hence the final outcome of the study is the implementation of a prototype ITS developed on the Flash platform, namely AutoTutor Lite, that answers the above questions. At the same time, the LCC technology is used in AutoTutor Lite to evaluate the learner's performance. When the system was tested on pre-defined test case scripts, its behavior and response were as expected.

Since an ITS is not a traditional computer application, many factors beyond technology affect the system performance. For example, there are many content related parameters like learning thresholds and dialogue turn limits. A good pre-defined learning threshold can give the learner proper challenges and encouragements. Therefore, those parameters should be well-defined according to the content, knowledge level of the learner, and learning environment in order to challenge and encourage the learner properly. In the implementation of

AutoTutor Lite, those parameters are defined just for the system functionality test. Therefore future experimentation is needed to setup those parameters in order to tune up the system performance.

Another issue that needs to be discussed further is the LCC technology, as it is a powerful indicator of the learner's performance in AutoTutor Lite. Additionally, I haven't made complete use of the LCC outputs. Some AutoTutor data and further experiments are needed to take full advantage of the LCC technology.

Chapter 2 Literature Review

In this chapter, a discussion about the ITS and its related theories and technologies will be made. Also the AutoTutor, one successful exemplar ITS from which AutoTutor Lite borrows features, will be reviewed.

ITS: Intelligent Tutoring System

Ong and Ramachandran (2003) defined the Intelligent Tutoring System as “A virtual training assistant that captures the subject matter and teaching expertise of experienced trainers provides a captivating new option” (p. 2). It has been studied and researched for decades by people from different backgrounds, such as education, psychology, and computer science.

Intelligent tutoring systems date from the Artificial Intelligence (AI) research in the late 1950’s. Some famous researchers such as Alan Turing, John McCarthy, and Marvin Minsky attempted to make a computer act and respond like a human (Turing, 1950). Based on the rapid evolution of computer technology - especially the AI technology - researchers found that by utilizing the power of computers, machines can emulate human thinking. With further research, the machine could perform any task human associated with human thought, especially instruction.

Benjamin Bloom (1984) defined the “two sigma problem”. In his experiments, he observed that average students who received one-on-one tutoring performed two standard deviations better than students who received traditional classroom instruction. In other words, the experiments indicated the importance of individual instruction and tutors to the students' learning process, particularly in problem-solving domains. Based on the research and development of AI, people tried to capture the effective behaviors and responses of human tutors and create an optimal form of e-Learning using computer AI technologies in tutoring, teaching, and training (Carbonell, 1970).

ITS is used in Computer-Based Training (CBT) and Computer-Based Learning (CBL). Some researchers found that compared to traditional human tutors, ITS not only inherited features from e-Learning, but also provided better assistance and more apparent intervention under certain conditions (Merrill, Reiser, Ranney, & Trafton, 1992). ITS is now widely used in traditional education, military training, and industry training. For example the Carnegie Learning Algebra Tutor (Anderson J. R., 1992) is one of the most widely used ITS in school. If, under certain conditions, a carefully designed ITS can be as effective as human tutors, it will have substantial benefits for society (Corbett, Koedinger, & Anderson, 1997).

Natural language is one of the best choices when the system communicates with the user. It is also more efficient for knowledge presenting and authoring. Therefore, when an ITS processes the information and prepares feedback or instruction, it may use natural language. Additionally, domain knowledge can be provided to the ITS in natural language.

Here are some existing ITSs developed in recent years:

1. Stottler Henke Associates, Incorporated (SHAI) developed a simulation-based ITS for U.S. Navy officer tactical training. This ITS is used as part of the Tactical Action Officer (TAO) training program in high-level tactical skills (Stottler, 2000).
2. Carnegie Learning developed a suite of ITS-based "cognitive tutors" in secondary-level math subjects (Anderson, Corbett, Koedinger, & Pelletier, 1995).
3. AutoTutor, developed by University of Memphis, helps students learn by holding a conversation in natural language. It appears as an animated agent that acts as a dialog partner with the learner (Graesser, Chipman, Haynes, & Olney, 2005).

In terms of system behavior, all of the above ITSs are similar, even though they may focus on different topics or different fields. In terms of technology or architecture, those systems were implemented in different ways because there is no constraint on what kinds of technologies developers may use to build an ITS.

Autotutor: An Successful Exemplar of ITS

In the study, AutoTutor is used as an exemplar ITS. All the discussions and research on ITS are primarily based on the AutoTutor.

With one or more talking heads, AutoTutor acts as an agent that simulates a human tutor by making dialogues with a learner in natural language (Graesser, Jackson, & McDaniel, 2007). It is an intelligent tutoring system developed by researchers at the Institute for Intelligent Systems at the University of Memphis. Currently there are several versions of AutoTutor for different topics: The computer literacy version is designed to help students learn basic computer literacy topics covered in an introductory course (e.g., hardware, operating systems, and the Internet) (Graesser, VanLehn, Rose, Jordan, & Harter, 2001). The conceptual physics version is designed to help students learn Newtonian physics (VanLehn et al., 2007). Critical Thinking is a version of AutoTutor that aspires to teach scientific inquiry skills to students (Graesser et al., 2008). ARIES(Acquiring Research Investigative and Evaluative Skills) is a newer version of AutoTutor which teaches scientific inquiry by a trialogue among two animated agents and the human learner. It is an extension of the previous Critical Thinking AutoTutor (Millis et al., 2006).

AutoTutor works by having a conversation with the learner. AutoTutor appears as an animated agent that acts as a dialog partner to the learner. The animated agent delivers AutoTutor's dialog moves with synthesized speech, intonation, facial expressions, and gestures. Students are encouraged to articulate lengthy answers that exhibit deep reasoning, rather than to recite small bits of shallow knowledge. For some topics, there are graphical displays and animations.

In the interface module, AutoTutor includes one or two agents according to the versions, which are created in Curious Labs Poser 4 and are controlled by the Microsoft® speech engine. In the later version, ARIES, there are two agents created by Media Semantics, Inc. For the expert or domain module, AutoTutor puts all problems and answers into a particular domain which is organized in a curriculum script. For each problem, the curriculum script contains an ideal answer, expected good answers, misconceptions, anticipated question-answer pairs, a list of important concepts, and all problem-related dialog moves. The problem-related dialog moves currently being used by AutoTutor are Hint, Prompt, Prompt Completion, Pump, Assertion, Summary, and Feedback amongst others. Here is an example conversation between the AutoTutor and a relatively verbose, knowledgeable student about Computer literacy (Graesser, Wiemer-Hastings, Wiemer-Hastings, Kreuz, & TRG, 1999).

AutoTutor: When you turn on the computer, how is the operating system first activated and loaded into RAM?

When the computer is turned on a file is automatically booted up.

AutoTutor: Anything else?

This file contains the operating system.

AutoTutor: What else?

The operating system is then read into RAM and activated.

AutoTutor: Right. The CPU executes instructions in ROM, which help the operating system to start.

After being read to ROM the OS is activated.

AutoTutor: Right. Let's review what we've just gone over. The operating system must first be stored on the hard disk. When you turn on the computer, the CPU executes instructions stored in ROM. These instructions help boot the operating system. The operating system is then loaded from the hard disk into RAM.

More specifically speaking, AutoTutor is comprised of the following modules and utilities (Mathews, Jackson, Olney, Chipman, & Graesser, 2003).

1. Language Analysis module
2. Assessment module
3. Dialog Management module
4. Logging module
5. Avatar/Client module
6. Speech act classification module (SAC)
7. Latent Semantic Analysis (LSA) utility
8. Question answering utility

9. Curriculum script utility

10. Parser utility

11. Log utility

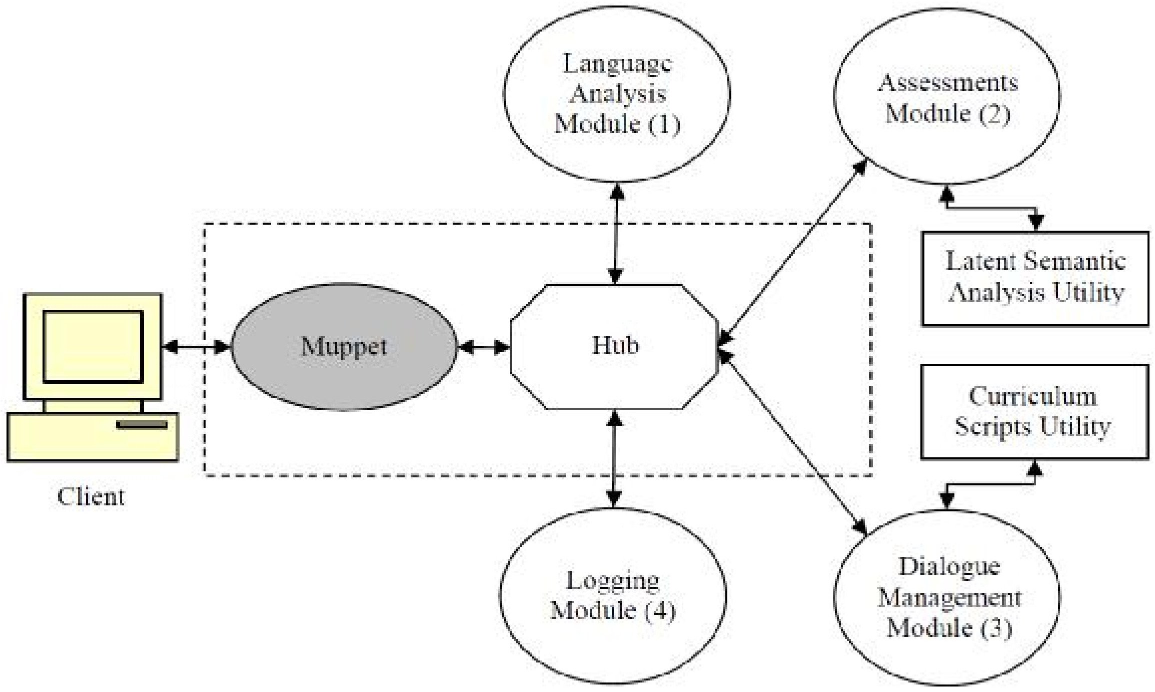

Language Analysis Module (1)
Assessments Module (2)
Latent Semantic Analysis Utility
Muppet
Hub
Curriculum Scripts Utility
Client
Logging Module (4)
Dialogue Management Module (3)

Figure 1. AutoTutor Basic Architecture *(Graesser, Chipman, Haynes, & Olney, 2005, p. 615)*

Figure 1 illustrates the basic architecture of AutoTutor and its modules and utilities. Among those modules and utilities, especially for the research in this thesis concerning AutoTutor Lite, the most relevant modules are Avatar/Client Module, Dialog Management module, Assessment Module and Latent Semantic Analysis (LSA) utility.

1. The Avatar/Client Module contains an animated agent on screen during the entire tutoring session for gestures, emotions, and voice delivery. It is created in Curious Labs Poser 4 and controlled by Microsoft Agent (Mathews et al.,

2003). The dialog during the tutoring session is synchronized with the agent’s emotions, gestures, and speech (Person et al., 2000).

2. The Dialog Management module, or Dialog Advancer Network (DAN) manages the conversation that occurs between a student and the AutoTutor. This module is comprised of a set of customized dialog pathways that is tailored to a particular student’s speech act categories. It enables AutoTutor to adapt each dialog move to the preceding student turn and respond appropriately. A pathway may include one or a combination of the following components: Discourse markers (e.g., "Okay" or "Moving on"), AutoTutor dialog moves (e.g., Positive Feedback, Pump, or Assertion), Answers to questions, or Canned expressions (e.g., "That's a good question, but I can't answer that right now"). For each topic in AutoTutor, knowledge is divided into several parts, called expectations (Graesser et al., 2005). AutoTutor guides the student in articulating the expectations through a number of dialogue moves. AutoTutor Dialog moves mainly contain pumps, hints, prompt and assertions and it follows a particular order: *Pump, Hint, Prompt, Assertion* then *Pump, Hint, Prompt* and *Assertion* (Graesser et al., 2004). As long as the student satisfies the expectations, AutoTutor will exit the cycle.

There are two types of pumps in the Dialog moves. The first, *specific Pumps,* are associated with the specified dialog topic (like "What else do you think about XXX?"). They are used to encourage the students to type more. The second pumps are *General Pumps* (like “what else?”) which are used to get the student to do more talking as well as specific pumps.

Hints and *Prompts* are for the student to fill in missing words. Hints and prompts are carefully selected in the expert module to produce content in the answers that fill in missing content words, phrases, and propositions. For example, a *Hint* to get the student to articulate the expectation “The magnitudes of the forces exerted by the two objects on each other are equal” might be “What about the forces exerted by the vehicles on each other?” This *Hint* would ideally elicit the answer “The magnitudes of the forces are equal.” A prompt to get the student to say “equal” would be “What are the magnitudes of the forces of the two vehicles on each other?” AutoTutor adaptively selects those *Hints* and *Prompts* that fill missing constituents and thereby achieves pattern completion (Graesser et al., 2004).

Assertions are used when students cannot fill in the content of an expectation after multiple conversational turns. AutoTutor will simply express the expectation as an *Assertion*.

AutoTutor ends up generating a high proportion of *Pumps* and *Hints* for articulate students with high knowledge but more *Prompts* and *Assertions* for low verbal, low knowledge students. This is because students with high knowledge tend to talk more by *Pumps* and *Hints*, while students with low knowledge tend to learn more through *Assertions* (Jackson, Mathews, Lin, Olney, & Graesser, 2003). After a multi-turn dialog move, all the expectations will be covered, and the main question is answered and evaluated (Graesser, Jeon, Yang, & Cai, 2007).

3. The Assessment Module uses Latent Semantic Analysis (LSA) utility to

assess students’ contributions and evaluate students’ performances. AutoTutor’s Assessment module compares the material in the curriculum script to students’ contributions using LSA, which measures the conceptual similarity of the two text sources (Mathews et al., 2003).

LSA: Latent Semantic Analysis

Before moving on to the next discussion, a brief review of Latent Semantic Analysis talked about previously will be given.

Landauer, Foltz and Laham (1998) first defined the Latent Semantic Analysis as "a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text” (p. 259). It was originally introduced for information retrieval (Deerwester, Dumais, Furnas, & Landauer, 1990).

Based on the analysis of a large collection of corpus, Latent Semantic Analysis (LSA) uses points in a very high dimensional semantic space to represent a single word and any set of words-such as sentences, paragraphs, or essays- either taken from the original corpus or new. One of the key aspects of LSA is the vector representation. In other words, any single word or collection of words in the semantic space can be represented as a vector in the LSA space, and they can also be considered as the points in the high dimensional semantic space. A lot of natural language analysis can be done based on the vector representation. Landauer, Foltz and Laham (1998) also mentioned that "Word and passage meaning representations derived by LSA have been found capable of simulating a variety of human cognitive phenomena, ranging from acquisition of recognition vocabulary to sentence-word semantic priming and judgments of essay quality" (p. 260).

One important concept in LSA is the term “weight”, which will be used in later discussion. It represents the relative frequency of a term in the corpus of text. Higher frequency terms are often function words, which primarily have a syntactic rather than semantic role in language, while those words with lower frequency are usually content words which play the semantic roles in language (Riordan & Jones, 2009). Therefore, in the LSA, lower frequency terms usually have a higher weight while higher frequency terms usually have a lower weight. For example, in the sentence “I am a student”, “I”, “am” and “a”, which are used more in daily life, would appear more in corpus of text, while “student” appears less in the corpus. The weights of “I”, “am” and “a” from LSA space will be lower than the weight of “student”. This is more representative of how these words used in daily life.

Another concept that is used in later discussion is the nearest neighbor term. After processing a large collection of corpora, LSA can represent the words used in them. Any single term or set of terms like phrases, sentences or paragraphs - either taken from the original corpora or new - could be represented by very high dimensional vectors. Therefore, the similarity between different terms can be measured by the cosine value of the vectors, and the nearest neighbor term of the target term can be defined as the term with related vector

has the highest cosine value with the target term related vector.

The goal of this thesis is to focus on the utilization of LSA data in LCC and AutoTutor Lite, so the review of LSA is stopped here. More information about LSA can be found from the references.

Chapter 3 AutoTutor Lite Architecture

The architecture of AutoTutor Lite follows the general architecture of ITS but also has some differences. Usually an ITS architecture includes four modules: the *interface module*, the *expert module* or *domain module*, the *tutor module*, and the *student module* (Martha Campbell Polson, 1988).

The *interface module* is used by students to interact with the ITS. In most cases, it is a graphical user interface. Sometimes this module will play a multi-media simulation of what the students are learning (e.g., Physics topic: Throw a pumpkin) to illustrate the domain knowledge. In AutoTutor Lite, Flash UI components are mainly used as interface module. It is used to present knowledge to the user and receive the user's response. Therefore, the communication between the user and the system is handled in this module.

The *expert module* or *domain module* contains the domain knowledge. Domain knowledge is the material that represents expertise in the problem domain the ITS is teaching. It should have an abundance of specific and detailed knowledge derived from people who have years of experience in a particular domain (Martha Campbell Polson, 1988). In AutoTutor Lite, the domain knowledge is stored on the host server in XML format. The Content Handler Module in AutoTutor Lite is responsible for managing the domain knowledge.

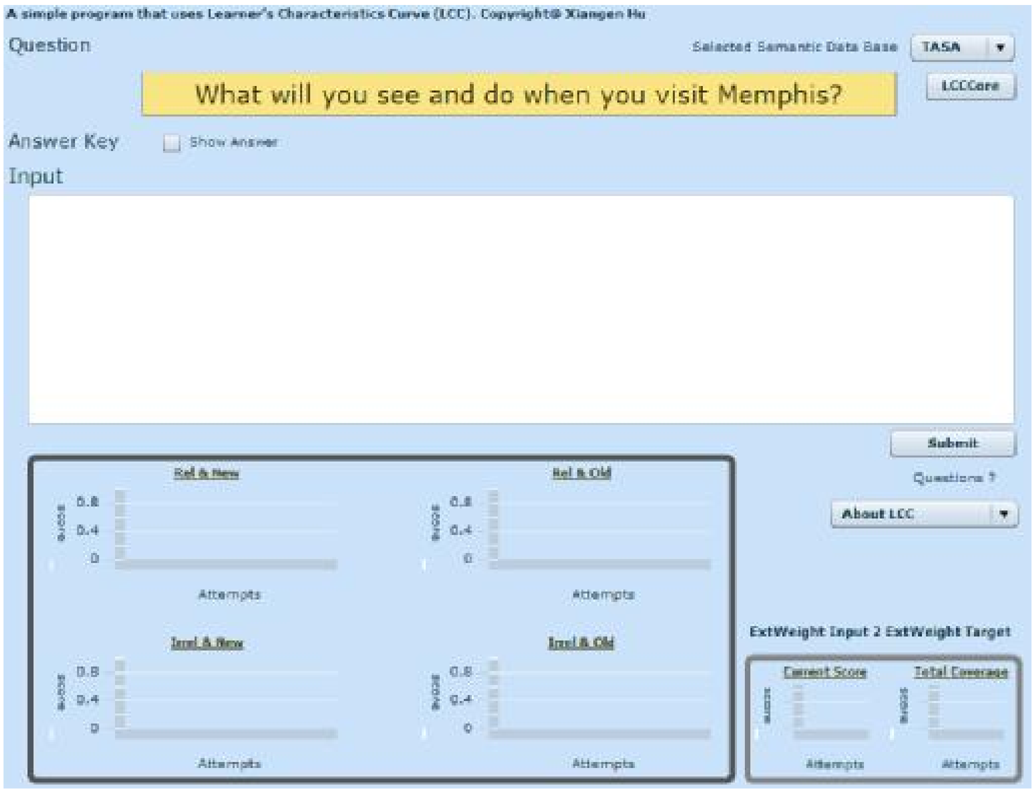
The *tutor module* sends the corresponding feedback to the students and the next action to the interface module after receiving the information about the mismatches, just like what a human tutor would do in such situations. In AutoTutor Lite, the design of the tutor module is adopted from AutoTutor. By referring to the pre-defined reaction rules, the *tutor module* in AutoTutor Lite will send out feedback and next action.

The *student module* is one of the most important modules in an ITS. The performance of the whole system relies heavily on this module. It contains the descriptions of students' knowledge and responses. According to Clancey and Soloway (1990), its basic responsibility is to deliver to the *tutor module* "an interpretation of a piece of behavior in terms of the various sequences of production rules that might have produced that piece of behavior" (p. 34). During the tutoring process, mismatches between students' responses and knowledge and the experts' pre-provided responses and knowledge are used to decide the next action of the system. In AutoTutor Lite, the LCC technology is used in the student module to evaluate the student's responses, which are semantically represented. Therefore, the LCC Analysis module and Evaluation module are working together in AutoTutor Lite as the *student module*.

The following discussion is focused on the design of the *student module* around the LCC, Semantic Representation, and the LCC in Evaluation.

LCC: Learner's Characteristic Curve

Consider a typical scenario in human tutoring where a tutor asks the student to answer a question that requires an elaborate verbal answer. Most often, students may not have the complete answer. What would be a reasonable way for an ITS to react to a sequence of incomplete answers? We consider two types of tutoring strategies: proactive tutoring and reactive tutoring. In proactive tutoring, the tutor will give the student some instructional feedback or suggestions in order to guide the student to find the ideal answer. That is what AutoTutor does. In reactive tutoring, the tutor only provides the evaluation result of a student's answer, such as “good” or “bad”. The Learner's Characteristic Curves (LCC) (Hu & Martindale, 2008) is a technology for reactive tutoring. It can provide an evaluation feedback based on a student's answer and the ideal answer. Though LCC presents the strategy of reactive tutoring, it can also be used in active tutoring to evaluate a student's answer and help the tutoring system provide more accurate instructional feedback. Figure 2 is the screenshot of LCC.


A simple program that uses Learner's Characteristics Curve (LCC). Copyright© Xiangen Hu

What will you see and do when you visit Memphis?

Selected Semantic Data Base: TASA

Figure 2. Screenshot of Standalone LCC Demo

The concept behind LCC is as follows:

1. Assuming there are three data sets at the beginning - expected contribution, learner’s current contribution, and learner’s previous contribution:
    a. The expected contribution data set contains the instructional content or expected answers.
    b. The learner’s current contribution data set only contains the learner’s input after the last submission.
    c. The learner’s previous contribution data set contains the learner’s cumulative inputs through and including the last submission. At the beginning of the tutoring, it may be empty since no input has been provided.
2. Represent the contributions by semantic vectors. (Keywords/Weighted Keywords/Extended Weighted Keywords semantic representation will be discussed in the later chapter.)
3. Find the overlaps of the three sets of data by using the matching method to compute the similarity among the three data sets. (There are several ways to compute the similarity which will be discussed in later chapters).

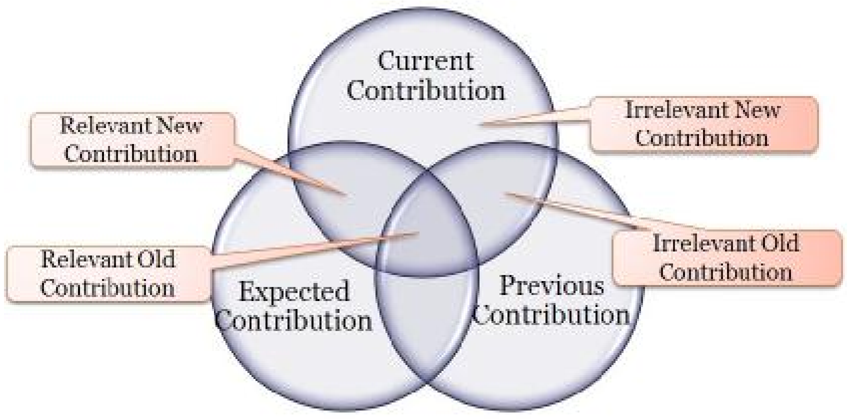

Current Contribution
Relevant New Contribution
Irrelevant New Contribution
Relevant Old Contribution
Irrelevant Old Contribution
Expected Contribution
Previous Contribution

Figure 3. LCC RN, RO, IN and IO Demonstration

Therefore, like in Figure 3 each current contribution can be categorized as in the Table 1:

Table 1

LCC RN, RO, IN and IO Table

	Old	New
Relevant	R-O	R-N
Irrelevant	IR-O	IR-N

Because the current contribution is divided into 4 sets, only 3 of the sets are independent. In other words, if any 3 sets of the contribution are given, the missing set could be recovered.

4. By analyzing the learner's current contribution turn by turn, 4 sets of data will be available: Relevant and New, Relevant and Old, Irrelevant and New, Irrelevant and Old. Visualizing the 4 sets of data, the following 4 curves, which are the so called Learner's Characteristics Curves, could be drawn.

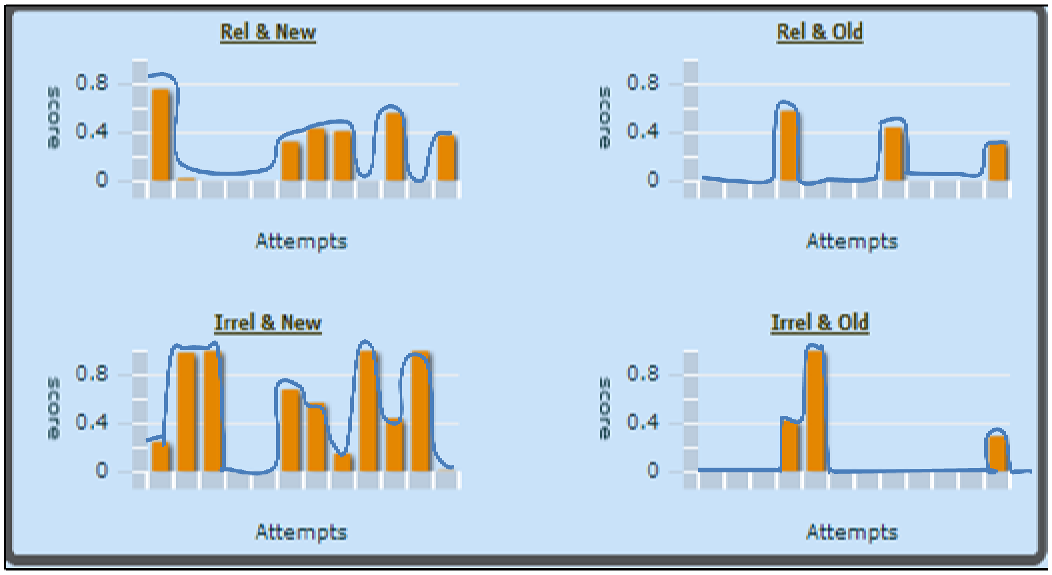


Figure 4. LCC Curve Demonstration

5. Based on the characteristics curves in Figure 4, provide feedback to the learner.

It is understandable that a human tutor would offer positive feedback when a student is constantly providing relevant and new (RN) contributions. Furthermore, if a student is actively constructing relevant answers, one would see a non-decreasing value for the (RN) in a sequence of responses. In the same fashion, other cells can be used as an indication of a student's knowledge. For example, an increasing value for the (IN) would indicate a lack of relevant knowledge.

Semantic Representation

As stated in the discussion above, one important pre-requisite in LCC is the Semantic Representation . In LCC there are three ways to semantically represent the contributions: keyword representation, weighted keyword representation, and extended weighted keyword representation (Hu, Cai, Wiemer-Hastings, Graesser, & McNamara, 2007). The pumpkin problem example from AutoTutor will be used in the following semantic representation to illustrate the differences among each semantic representation.

The following processing will be applied to each term in Current Contribution, Previous Contribution, and Expected Contribution before they are semantically represented.

1. Assuming all the terms from the contributions exist in the same semantic space. Each term is indexed by an integer i, where i ranges from 1 to the total number of terms in the semantic space, say $\mathcal{N}$.

2. Equivalently, each term (ith) can be represented as an $\mathcal{N}$-dimensional vector such that all the elements of the vector are zero, except the ith element.

3. The semantic representation for a collection of terms like phrases, sentences, and paragraphs will be the sum of vectors for each term.

Keyword Representation is the simplest semantic representation. It only considers the existence of the term. The value of the ith element in the corresponding vector for each term is 1.

Example: The pumpkin will move the same.

Terms in keyword representation (Duplicates are removed):

the, pumpkin, will, move, same

In semantic space the vector representation will be the sum of all term vectors, like the vector in Equation 1.

$$\bar{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ldots \\ 1 \\ 0 \\ 1 \\ \ldots \\ 0 \\ 1 \\ 0 \\ \ldots \end{pmatrix} \qquad \textit{Equation (1)}$$

Weighted Keyword Representation will consider the weight information for each term in addition to the existence of the term. Each term is weighted by a value between 0 and 1. The value is determined by the relative frequency of each term in the LSA space. The higher the frequency of a term is, the lower the value is. In the implementation, the weight information is coming form TASA (Touch-stone Applied Science Associates) LSA space. In other words, the nonzero elements in the vector representation in the semantic space are real numbers between 0 and 1 instead of having 1 for the nonzero elements in Table 2, as they are in "Keyword Representation".

Example: The pumpkin will move the same.

Weighted Keyword representation (Duplicates are removed):

Table 2

Weighted Keyword Representation Example

Word	Weight
the	0.008165
pumpkin	0.67533
will	0.131253
move	0.267978
same	0.172617

In LCC semantic space, the vector representation will be the sum all existing term vector, and the value for non zero term dimensions will be the weight for the corresponding terms, like the vector in Equation 2:

$$\bar{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ldots \\ 0.008165 \\ 0.67533 \\ 0 \\ 0 \\ 0.131253 \\ \ldots \\ \ldots \\ 0.267978 \\ 0.172617 \\ \ldots \\ 0 \\ 0 \\ 0 \\ \ldots \end{pmatrix}$$

Equation (2)

Extended Weighted Keyword Representation is based on weighted keyword representation. In addition to Weighted Keyword Representation, Extended Weighted Keyword Representation also represents the information of nearest neighbor terms and their weights of the existing terms. In the implementation, the number of nearest neighbor terms is up 9. This semantic representation can represent much richer information than the previous two. In AutoTutor Lite implementation, the Extended Weighted Keyword Representation is the default semantic representation.

Example: The pumpkin will move the same.

Extened Weighted Keyword representation (Duplicates are removed):

Table 3

Extended Weighted Keyword Representation Example

Word	Weight	Word	Weight	Word	Weight	Word	Weight	Word	Weight
the	0.008165	pumpkin	0.67533	will	0.131253	move	0.488872	same	0.172617
of	0.625410	vegetable	0.448989	continue	0.352125	vacate	0.934212	alike	0.421355
this	0.571235	pie	0.569672	future	0.315445	shift	0.486437	similar	0.310399
a	0.596546	squash	0.602953			walk	0.330017	like	0.119999
and	0.570214	seed	0.518115			leave	0.302977	opposite	0.382836
in	0.600245	orange	0.469565			motion	0.413654	equal	0.363444
		halloween	0.658093			run	0.276726	different	0.191179
		patch	0.519182			stop	0.294419	exact	0.442184
		head	0.236682			haul	0.613292	thing	0.23982
						go	0.177455	alike	0.421355

In LCC semantic space, the vector representation will be the sum of all existing term vectors plus their neighbor term vector, and the value for nonzero elements in the vector will be the weight value of each corresponding term, like in Equation 3

$$\bar{t} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \ldots \\ 0.008165 \\ 0.67533 \\ 0.131253 \\ 0.267978 \\ 0.172617 \\ \ldots \\ 0 \\ 0.177455 \\ 0.330017 \\ \ldots \\ 0 \\ 0 \\ 0.519182 \\ \ldots \end{pmatrix} \qquad \textit{Equation (3)}$$

Evaluation of Learner's Contribution using LCC

The basic idea of evaluation of learner's contribution in LCC is to compute the similarity among the learner's current contribution, previous contribution, and expected contribution based on semantic representations. According to the three types of Semantic Representation, there are up to nine different evaluation methods, including cross semantic representation matching. However, the Keyword Representation doesn't contain any weight information although 1 could be considered as the weight value for elements in the vector representation. In order to reduce the time required for computation, it should not be used for matching with other two semantic representations. Therefore, only five matching types are used in the evaluation: keyword to keyword matching, weighted keyword to weighted keyword matching, weighted keyword to extended weight keyword matching, extended weighted keyword to weighted keyword matching, and extended weighted keyword to extended weighted keyword matching. All five matching types are listed in Table 4.

Table 4

Semantic Representation Matching Matrix

	Keyword Representation	Weighted Keyword Representation	Ext-Weighted Keyword Representation
Keyword Representation	O	N/A	N/A
Weighted Keyword Representation	N/A	O	O
Ext-Weighted Keyword Representation	N/A	O	O

Keyword Matching is a matching method that only considers the shared keywords between two semantic representations. It is a traditional string matching method. The following 3 steps are used to decompose the current contribution into 8 parts (New, Old, Rel, Irr, RO, RN, IO, and IN) and calculate the LCC scores:

1. Assuming the Current Contribution and Expected Contribution are already in form of keyword representation, when keyword matching is used in LCC, first find the shared keywords from Current Contribution and Expected Contribution, which form Relevant Contribution. The rest of the keywords from the Current Contribution are the Irrelevant Contribution when compared to the Expected Contribution.

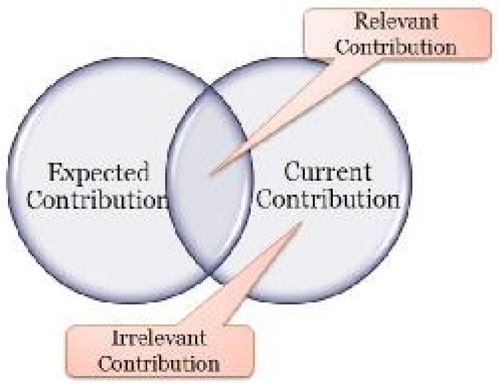


Figure 5. LCC Rel and Irr Contribution

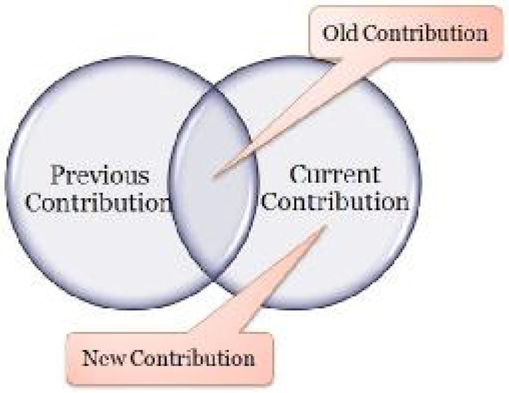


Figure 6. LCC New and Old Contribution

2. Apply the *Keyword Matching* to Relevant Contribution and Previous Contribution, Irrelevant Contribution and Previous Contribution. The matched keywords will be the RO Contribution and RN Contribution while the unmatched keywords will be the IO Contribution and IN Contribution. As a result, the learner's Current Contribution will be decomposed into RN, RO, IN and IO using LCC technology.

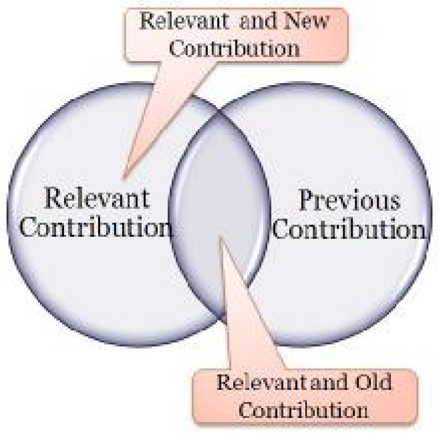


Figure 7. LCC R-N and R-O Contribution

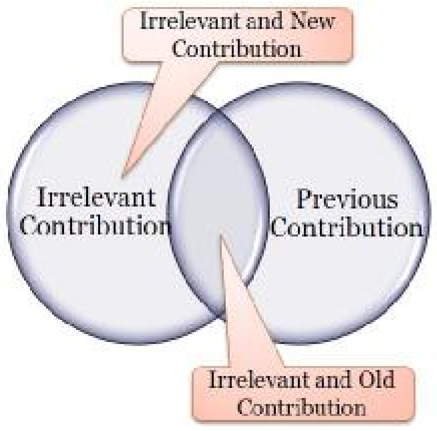

Irrelevant and New Contribution
Irrelevant Contribution
Previous Contribution
Irrelevant and Old Contribution

Figure 8. LCC I-N and I-O Contribution

3. Finally, compute the score for each contribution. Because the keyword matching simply considers the existence of keywords, the number of keywords in each contribution (RO, RN, IO, and IN) will be used as scores in each contribution of LCC.

Weighted Keyword Matching and Ext-Weighted Keyword Matching are more complex than *Keyword Matching* in terms of LCC score computation since the weight information is considered. Some vector computation is needed.

The same steps as in *Keyword Matching* will be used to find the overlap of the contributions, the only difference being that the Current Contribution, the Expected Contribution and the Previous Contribution are in the form of *Weighted Keyword Representation* or *Ext-Weighted Keyword Representation*. Through the same procedure the current contribution will be decomposed into eight sets of contribution: New, Old, Rel, Irr, RO, RN, IO, and IN. Since in the semantic space those 8 collections of terms can be considered as 8 vectors, vector representations will be used to calculate the LCC scores. Here are the definitions of the LCC scores:

RO Score:

$$\cos(Rel, Old) = \frac{Rel \cdot Old}{\sqrt{(Rel \cdot Rel) * (Old \cdot Old)}}$$

Equation (4)

RN Score:

$$\cos(Rel, New) = \frac{Rel \cdot New}{\sqrt{(Rel \cdot Rel) * (New \cdot New)}}$$

Equation (5)

IO Score:

$$\cos(Irr, Old) = \frac{Irr \cdot Old}{\sqrt{(Irr \cdot Irr) * (Old \cdot Old)}}$$

Equation (6)

IN Score:

$$\cos(Irr, New) = \frac{Irr \cdot New}{\sqrt{(Irr \cdot Irr) * (New \cdot New)}}$$

Equation (7)

According to the principle of the dot product, the LCC score definitions can be simplified as follows:

RO Score:

$$\cos(Rel, Old) = \frac{RO \cdot RO}{\sqrt{(Rel \cdot Rel) * (Old \cdot Old)}} \qquad \textit{Equation (8)}$$

RN Score:

$$\cos(Rel, New) = \frac{RN \cdot RN}{\sqrt{(Rel \cdot Rel) * (New \cdot New)}} \qquad \textit{Equation (9)}$$

IO Score:

$$\cos(Irr, Old) = \frac{IO \cdot IO}{\sqrt{(Irr \cdot Irr) * (Old \cdot Old)}} \qquad \textit{Equation (10)}$$

IN Score:

$$\cos(Irr, New) = \frac{IN \cdot IN}{\sqrt{(Irr \cdot Irr) * (New \cdot New)}} \qquad \textit{Equation (11)}$$

Here is a simple example script:

Question: What will you see and do when you visit Memphis?
*Target Answer**:** You will get BBQ Ribs from Corky's. You will be able to visit the Human Rights Museum. There is a great university in Memphis. You may also watch a basketball game in FedEx Forum. Downtown Memphis is also a fun place to be. You will never want to miss the Peabody Museum. The Peabody Hotel ducks will make you never forget your visit.*
*Previous Contribution**:** Visit Fedex forum. Eat fish and fried chicken. Try BBQ and ribs, stay near Mississippi River.*
*Current Contribution**:** Visit University of Memphis and Fedex institute of Technology.*

According to the definition equations, the LCC output scores are:

RN Score: 0.3461

RO Score: 0.4169

IN Score: 0.2245

IO Score: 0.0122

Chapter 4 Implementation of AutoTutor Lite

The following tools and technologies are used to develop AutoTutor Lite, which implements the LCC algorithm:

Development Tool: Adobe Flex builder 3 (Flex IDE based on Eclipse)

Development Language: ActionScript 3 and MXML

Webservice platform: Delphi

Web host: Windows server 2003 + IIS

Speech engine and agent: Products from Media Semantics, Inc

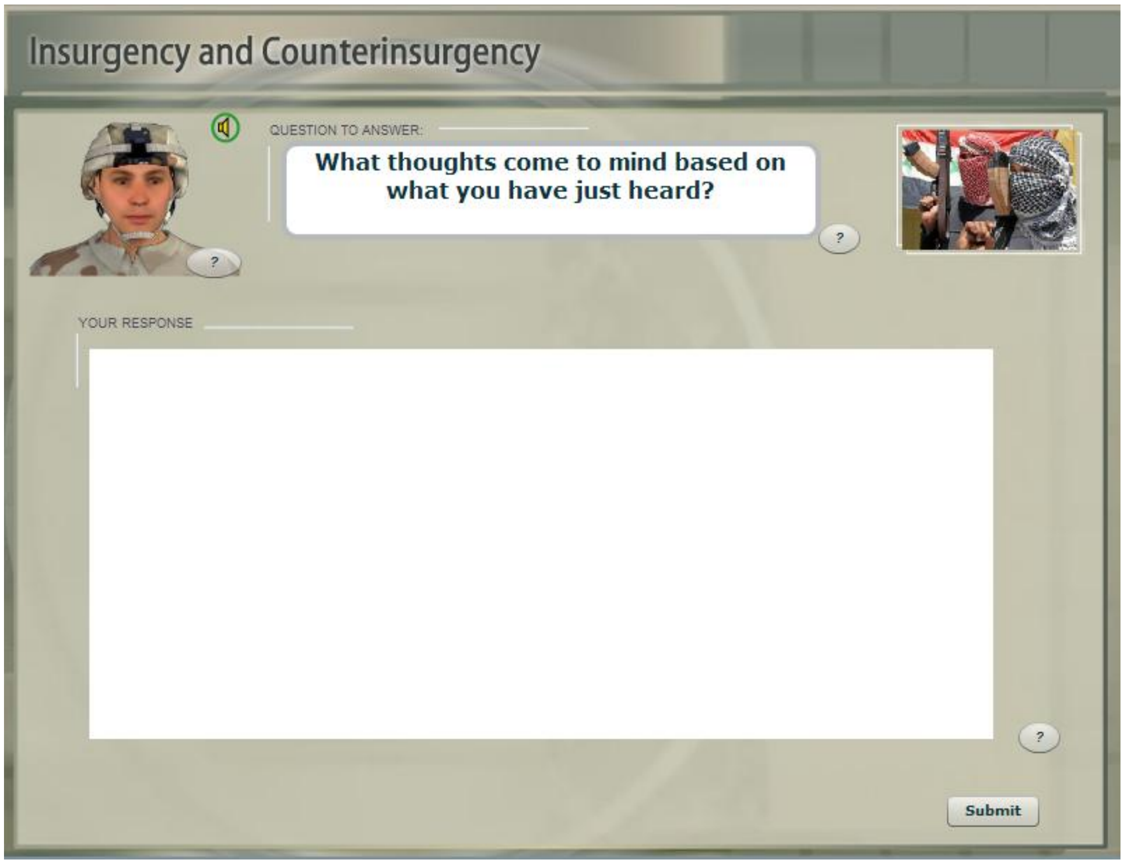
We chose the Flash platform as the project development platform for the following reasons:

1. Great cross-platform capability. As long as the browser has the Flash plug-in, a Flash file can be running on any system. Additionally, the Flash platform is dominant among all Rich Internet Application (RIA) platforms.
2. Strong capability for multimedia integration. Flash applications can easily integrate multimedia components like picture, voice, and speech agents. It also supports fantastic animations and an amazing look and feel when compared to traditional web applications

UI Description

Generally speaking, there are two UI stages in AutoTutor Lite: the *Seed Question State* and the *Expectation State*. Both states contain the speech agent and share the same theme style.

Seed Question state is the first UI state displayed to the user. It displays a seed question, which can be considered the overall question for the topic. It also contains a response Panel and a submit button for the user to enter responses.



What thoughts come to mind based on what you have just heard?

Figure 9. Screenshot of Seed Question State

In Figure 9, the seed question in the seed question panel is “What thoughts come to mind based on what you have just heard?” The user is to type his response in the response panel.

The *Expectation state* contains more visualized components than the first Seed Question State. It has the following components:

a) Seed Question Display Panel

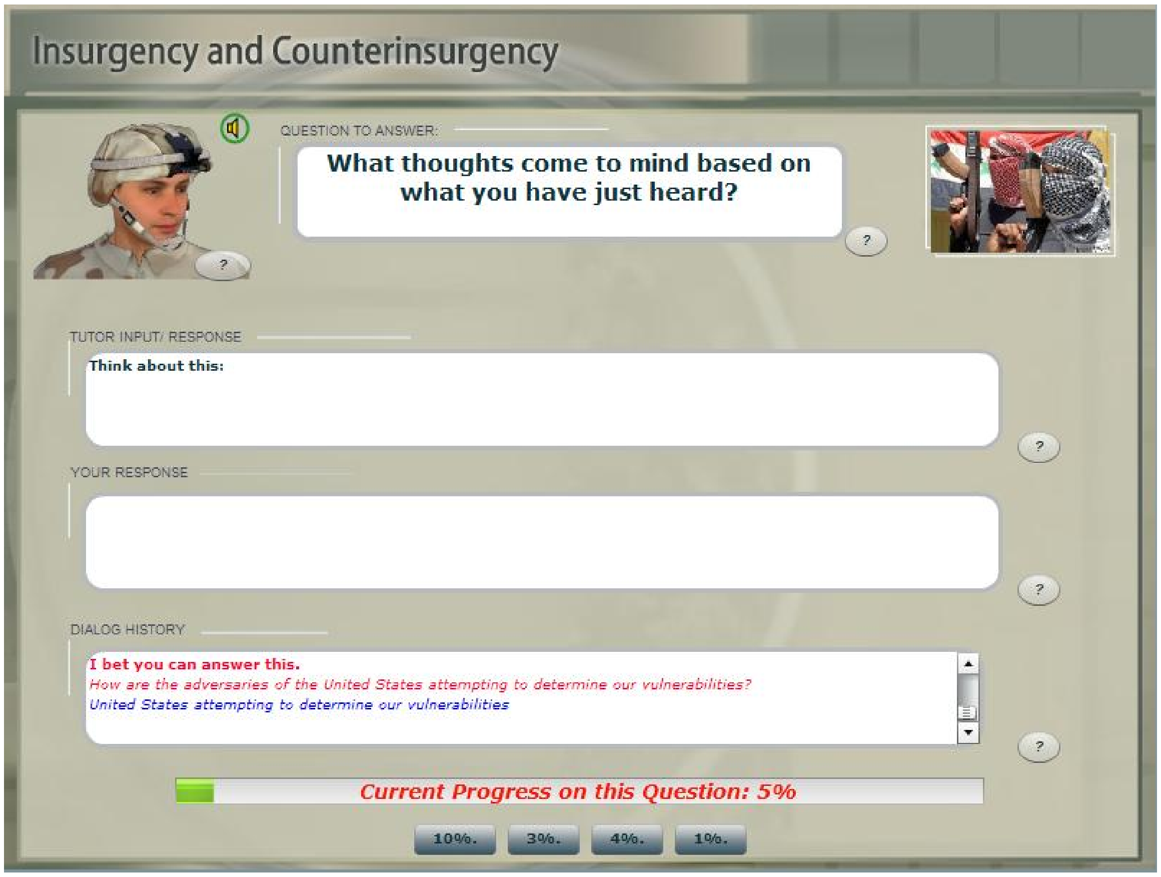
b) Tutor Dialogue Panel

c) History Panel

d) User Response Panel

e) Score Panel

The Seed Question Display Panel and User Response Panel are the same as in the Seed Question State. The Tutor Dialogue Panel displays the tutor’s dialogue to the user including Hints, Feedbacks, and Assessments. In other words, this panel maintains the dialogue with the users. The History Panel contains all history interaction data, including both tutor and user data. The Score Panel shows the user’s coverage scores, which represent how much the user’s contribution covers the target answer. There are four independent expectation/subtopic coverage scores and an overall score. The independent expectation coverage score is the measurement of the specified expectation, while the overall coverage score is the average of all the expectation coverage scores.



What thoughts come to mind based on what you have just heard?

Think about this:

I bet you can answer this.
How are the adversaries of the United States attempting to determine our vulnerabilities?
United States attempting to determine our vulnerabilities

Current Progress on this Question: 5%

Figure 10. Screenshot of Expectation State

In the Figure 10, the AutoTutor Lite is in the Expectation state. After a couple of dialog moves, the History panel displays the dialog history between AutoTutor Lite and the user. The Tutor Dialog Panel will display the next hint or prompt. In the Score Panel, the overall coverage score is 5%, which is about the average of the 4 expectation coverage scores (10%, 3%, 4% and 1%).

Basic Architecture

The following diagram illustrates the high level architecture of AutoTutor Lite.

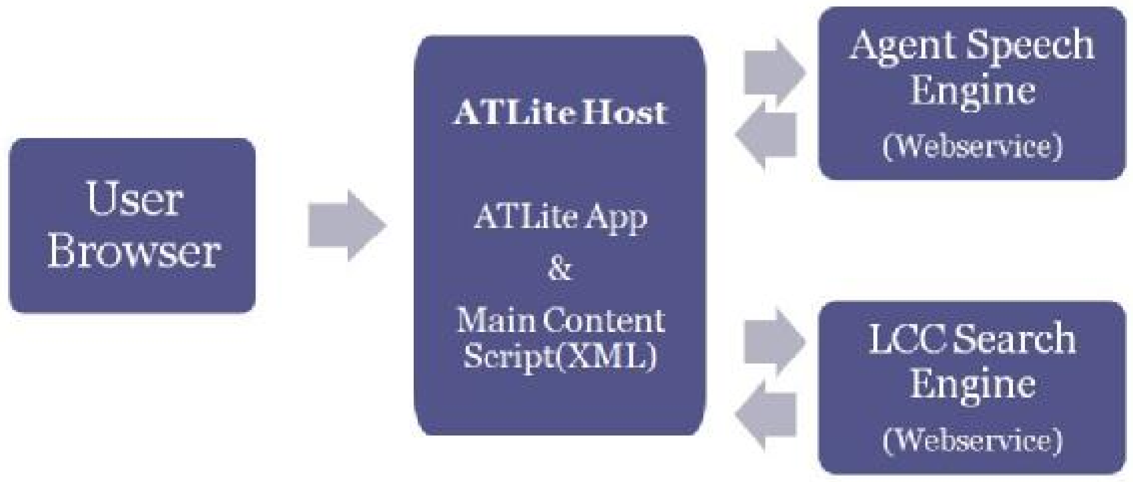


Figure 11. AutoTutor Lite High Level Architecture

The compiled Flash file is hosted on the web server. Users can access the AutoTutor Lite application via a web browser. All of the content scripts are stored on the host server in XML format. During the interaction between the user and AutoTutor Lite, AutoTutor Lite will call the Agent Speech Engine Webservice for speech data and the LCC Search Engine Webservice for semantic data.

In the project, there are three main folders and one MXML file under the root folder of the source.

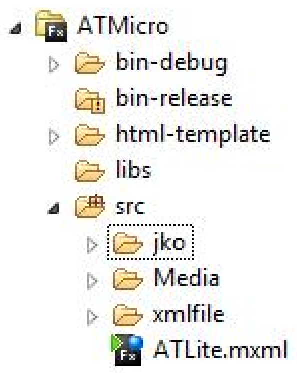


Figure 12. AutoTutor Lite Project Structure

In Figure 12, the *jko* folder contains all the visualized and non-visualized components and classes. The *Media* folder contains all the media files such as pictures and icons. The *xmlfiles* contains all the data files, which are in the format of XML. The *ATLite.mxml* file is the main canvas, which is the container for all other visualized components.

> *In a classic Flex project, there are two types of program files: MXML files and ActionScript files. MXML files are mainly used on visualized components while ActionScript files are mainly used on non-visualized components or classes like logic classes and computation classes.*

AutoTutor Lite Logic Process

The Logic Process Flowchart of AutoTutor Lite is below:

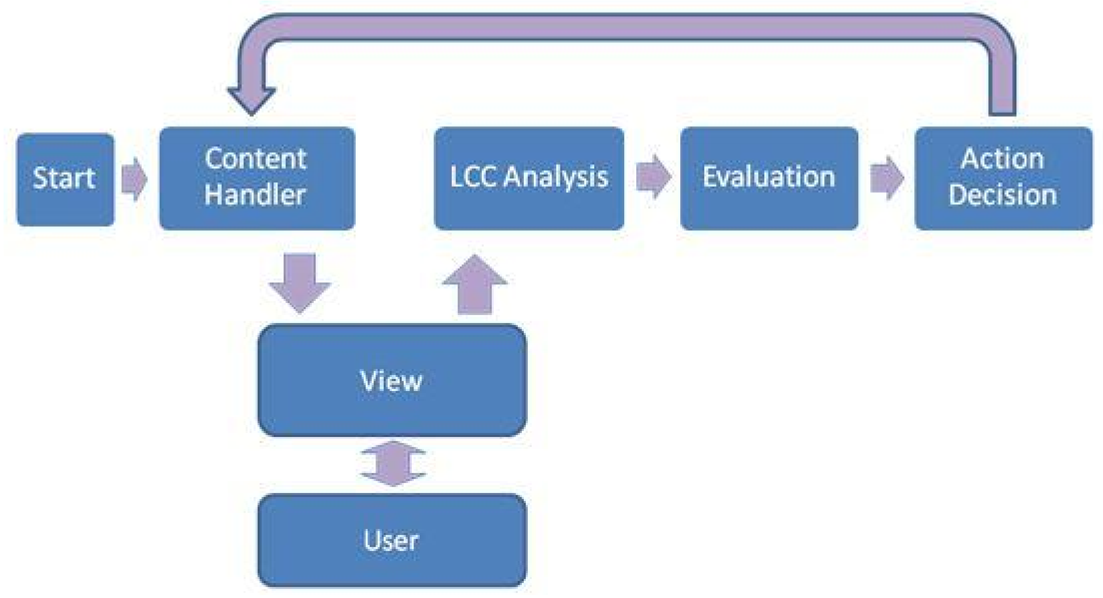


Figure 13. AutoTutor Lite Logic Process Flowchart

AutoTutor Lite includes five main modules: *Content Handler Module, View Module, LCC Analysis Module, Evaluation Module*, and *Action Decision Module*.

At the beginning, the *Content Handler Module* retrieves the seed question from the main content script, and the view module displays the question to the user. Next, the user enters the answer to the *View Module*. Then, the *LCC Analysis Module* will perform the LCC analysis by calling the LCC webservice after receiving the user's interaction data. The LCC analysis result is sent to the *Evaluation Module* for further evaluation based on some pre-defined thresholds. The final evaluation is sent to the *Action Decision Module*, in which tutor feedback and the next tutor move will be determined. Based on the next tutor move decision, the *Content Handler Module* will retrieve the corresponding script from the main content script and send it to the *View Module*. Now a full dialog turn is completed and the system is ready for the user's next input.

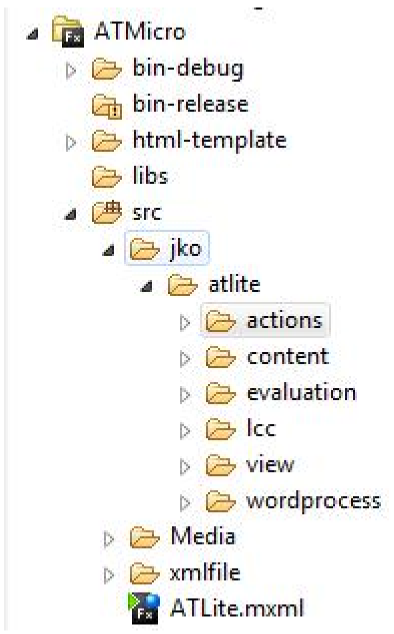


Figure 14. AutoTutor Lite Class Structure

In the above screenshot of the AutoTutor Lite development environment, there are 6 subfolders, which contain the major source codes of the project and correspond to the main modules of the system.

The *content* folder contains the *Content Handler Module*. This module will parse the content XML files and return the required content script to the *View Module*.

The *view* folder contains all the visualized components of the two UI states. It is the *View Module* and also the Interface Module in the general ITS architecture. It is on top of a basic canvas: *ATLite.mxml*. All tutor actions will be displayed on this module for the user. Specifically, the component *AvatarC* in the *view* folder loads the speech agent and the component *cbLib* is used to communicate with the speech engine webservice. The user’s interaction with the tutor system relies on the view module.

The *lcc* folder contains the two classes for the *LCC Analysis Module*, which will focus on the interaction data and perform LCC analysis. This module will send the user’s interaction data to the LCC webservice engine and get the semantic data. Based on the return data and LCC analysis, the *LCC Analysis Module* will pass the LCC score (RN, RO, IN and IO) to the *Evaluation Module*.

The *evaluation* folder contains the classes for the *Evaluation Module*. In AutoTutor Lite there are two scores involved in the *Evaluation Module*.

First is the LCC score, sent in by the *LCC Analysis Module*. The *Evaluation Module* will compare the LCC score (in the current version, only the RN score is used) with a pre-defined threshold, which is a mean (number between 0 and 1) with a standard deviation, for example 0.5±0.1. Therefore, the threshold divides the scope between 0 and 1 into 3 intervals:

1. Below the mean minus one standard deviation
2. Between the mean minus one standard deviation and the mean plus one standard deviation
3. Above the mean plus one standard deviation

Corresponding to the 3 intervals, three types of user performance status are defined: “Low Contribution”, “Moderate Contribution”, and “Current Expectation Covered”. Every time an LCC score arrives, the evaluation module will output the user’s performance type based on the threshold.

The second score is the Expectation Coverage Score which counts the cumulative user coverage for the expectation. If the Expectation Coverage Score is above the mean plus one standard deviation, the expectation will be marked as covered. The output of the user performance status is also “Current Expectation Covered”. Because there are multiple expectations in one topic, AutoTutor Lite launches the questions and hints for each expectation in default order. If the user’s contribution covers expectations other than the current targeted one, the *Evaluation Module* considers the user’s performance as “Other Expectation Covered”. If all the expectations are covered, the user performance status will be “All Expectation Covered”. This status information is sent to the *Action Decision Module*.

The *actions* folder contains the *Action Decision Module*. The primary component of this module is the pre-defined Tutor Navigation Rule XML. It defines tutor actions and user performance in the following format:

```
 <Tutor LastAction="TAHint">
        <Student Response="AllExpectationCovered">
                <Actions>
                        <Action>TAPositiveFeedback</Action>
                        <Action>TATRNS</Action>
                        <Action>TASummary</Action>
                </Actions>
        </Student>
        <Student Response="OtherExpectationCovered">
                <Actions>
                        <Action>TANeutralFeedback</Action>
                        <Action>TATRND</Action>
                <Action>TASummarizeNewlyCoveredExpectations</Action>
                        <Action>TATRNH</Action>
                        <Action>TAHint</Action>
                </Actions>
        </Student>
        <Student Response="CurrentExpectationCovered">
                <Actions>
                        <Action>TAPositiveFeedback</Action>
```

```
                    <Action>TATRNS</Action>
<Action>TASummarizeNewlyCoveredExpectations</Action>
                    <Action>TAChooseNewExpectation</Action>
                    <Action>TATRNH</Action>
                    <Action>TAHint</Action>
               </Actions>
          </Student>
          <Student Response="ModerateContribution">
               <Actions>
                    <Action>TANeutralFeedback</Action>
                    <Action>TATRNH</Action>
                    <Action>TAHint</Action>
               </Actions>
          </Student>
          <Student Response="LowContribution">
               <Actions>
                    <Action>TANegativeFeedback</Action>
                    <Action>TATRNH</Action>
                    <Action>TAHint</Action>
               </Actions>
          </Student>
</Tutor>
```

This Tutor Navigation Rule XML is inherited from AutoTutor, so most of the student responses and tutor action types are the same as AutoTutor. However in AutoTutor Lite some tutor action types are removed, such as prompts and pumps, in order to simplify AutoTutor Lite. Additionally, one more tutor action type called “Tutor Transition” is added, which is used after the AutoTutor Lite finishes feedback and before it asks for a new hint or prompt in order to make the tutor dialog move much smoother.

Each tutor action is decided by a previous tutor action and previous user response status. Therefore, there are two inputs and one output for the action module. The two input variables are the Tutor current action type and the user response type received from the evaluation module. The output is a collection of tutor actions. This module could be considered as a rule reading and searching module.

The *wordprocess* folder contains some semantic utility classes, such as a string cleanup utility and a regular expression utility. It will be used by other modules when some common string processes are needed.

Chapter 5 Results

The project is past the internal test (Alpha version). The majority of the code for the project is finished. The following links are for compiled versions for testing purposes:

AutoTutor Lite Debug Version

AutoTutor Lite Testing Version

The following script is used to test the AutoTutor Lite:

Exp 1: The adversaries the United States currently faces and is likely to face for many years to come are continuously and consciously evaluating our strengths and weaknesses, aiming to avoid our strengths and attack our vulnerabilities.
Exp 2: The United States Government should therefore constantly assess its effectiveness in using all instruments of national power (diplomatic, information, military, and economic), striving to learn and adapt more quickly and effectively than our adversaries.
Exp 3: Learning organizations may defeat insurgencies; armies that fail to learn and adapt quickly learning organizations do not.
Exp 4: Effective learning organizations encourage individuals to pay attention to the rapidly changing situations that characterize COIN campaigns rapid enemy innovation, shifting attitudes of local populations, local civilian leadership turmoil.

Three major test cases are used to test the AutoTutor Lite:

1. Completely correct input
2. Partially correct input
3. Completely incorrect input

Single case and random mixed cases are used during the test. For example, the following log XML is one of the test cases:

```
<assessment >
  <Round >
    <Tutor >undefined</Tutor>
    <TAActions>
      <TAAction>TAQuestion</TAAction>
    </TAActions>
  </Round>
  <Round>
    <newInput>constantly assess its effectiveness in using all instruments of national
power (diplomatic, information, military, and economic)</newInput>
    <STDResponse>OtherExpectationCovered</STDResponse>
    <TAActions>
      <TAAction>TAPositiveFeedback</TAAction>
      <TAAction>TAPositiveFeedback</TAAction>
      <TAAction>TAChooseNewExpectation</TAAction>
      <TAAction>TATRNH</TAAction>
      <TAAction>TAHint</TAAction>
    </TAActions>
    <newTutor >Good Answer! The United States Government should therefore
constantly assess its effectiveness in using all instruments of national power diplomatic
information military and economic striving to learn and adapt more quickly and effectively
than our adversaries What about this:What are the enemies of the United States are
constantly doing?</newTutor>
  </Round>
  <Round >
    <SemanticsScore>
      <ExpScoreLCC Exp="1">0.0117797595121844485</ExpScoreLCC>
      <ExpScoreLCC Exp="2">0.6586057879543086</ExpScoreLCC>
      <ExpScoreLCC Exp="3">0.05170962593895807</ExpScoreLCC>
      <ExpScoreLCC Exp="4">0.05129273393585995</ExpScoreLCC>
      <AveScore>20</AveScore>
    </SemanticsScore>
    <newInput>COIN campaigns rapid enemy innovation, shifting attitudes of local
populations, local civilian leadership turmoil.</newInput>
    <STDResponse >OtherExpectationCovered</STDResponse>
    <TAActions>
      <TAAction>TANeutralFeedback</TAAction>
      <TAAction>TATRND</TAAction>
      <TAAction>TATRND</TAAction>
      <TAAction>TATRNH</TAAction>
      <TAAction>TAHint</TAAction>
    </TAActions>
    <newTutor >Your contribution does not really answer the question that I was asking,
but it was not a bad answer. Let me summarize the answer for you. Effective learning
organizations encourage individuals to understand the operating environment paying
close attention to its rapidly changing nature  This is especially vital in COIN which may
be characterized by frequent rapid enemy innovation shifting attitudes of local
populations and local civilian leadership turmoil
Let's try this question: Why are adversaries of the United States constantly evaluating
the United States? </newTutor>
  </Round>
 </assessment>
```

Based on the internal tests, the AutoTutor Lite meets the expected requirements. The LCC algorithm plays an important role in the evaluation module. The overall performance is acceptable.

Chapter 6 Conclusion

In the study, an online tutoring system, namely AutoTutor Lite, is implemented based on the ITS theories and some recent computer technologies. This new tutoring system inherits features from ITS - an optimal form of e-Learning using advanced AI - and AutoTutor, a successful exemplar of ITS. Since AutoTutor Lite is developed on the most popular RIA platform - the Adobe Flash Platform - it eliminated the need for specialized client software installation required by most existing ITSs. It also features the LCC technology as the evaluation method, which is a new method to evaluate learners' performance by tracking their learning characteristic curves. Several types of semantic representation and matching methods, which are used for natural language processing in AutoTutor Lite, are also discussed in the study. According to the internal test and evaluation, the AutoTutor Lite meets the expected requirements, and its behavior and response followed the original design purpose.

The following future works are needed:

1. Optimize the thresholds for content. In the current version, the threshold is pre-defined only for the functionality test. It is not evaluated based on the topic or the user. This is also the first time the LCC technology is used in an ITS. New experiment data and previous training data are needed to optimize the thresholds in order to properly evaluate user performance.

2. Utilize other LCC scores besides RN. In the current version, only the RN score is used for evaluation, while other scores (RO, IN, IO) are also very important. Since each user's input is divided into four sets: (RN, RO, IN, and IO),

only three of the sets are independent. The goal will be to find the equation below:

$$Performance = \mathcal{F}(RN, RO, IN) \qquad \textit{Equation (12)}$$

Therefore, some data analysis will be applied to previous AutoTutor user data in order to find the relationship between user performance and all LCC scores (RN, RO, IN and IO).

References

Anderson, J. R. (1992). Intelligent tutoring and high school mathematics. *the Second International Conference on Intelligent Tutoring Systems* (pp. 1-10). Montreal, Canada: Springer-Verlag.

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

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

Carbonell, J. R. (1970). AI in CAI: artificial intelligence approach to computer assisted instruction. *IEEE Transactions on Man-Machine Systems, MMS-11* (4), 190–202.

Clancey, W. J., & Soloway, E. (1990). *Artificial intelligence and learning environments* (illustrated ed.). Boston, MA, USA: MIT Press, 1990.

Corbett, A. T., Koedinger, K. R., & Anderson, J. R. (1997). Intelligent tutoring systems. In M. Helander, T. K. Landauer, & P. V. Prabhu, *Handbook of human-computer interaction* (pp. 849-874). Pittsburgh, PA, USA: North Holland.

Deerwester, S., Dumais, S. T., Furnas, G. W., & Landauer, T. K. (1990). Indexing by latent semantic analysis. *Journal of the American Society For Information*, 391-407.

Graesser, A., Chipman, P., Haynes, B., & Olney, A. (2005). AutoTutor: an intelligent tutoring system with mixed-initiative dialogue. *IEEE Transactions in Education, 48*, 612–618.

Graesser, A., Jeon, M., Yang, Y., & Cai, Z. (2007). Discourse cohesion in text and tutorial dialogue. *Information Design Journal*, 199-213.

Graesser, A., Millis, K., Chipman, P., Cai, Z., Wallace, P., Britt, A., et al. (2008). A demonstration of ITSs that promote scientific inquiry skills: critical thinking tutor and ARIES. *Intelligent Tutoring Systems: 9th International Conference on Intelligent Tutoring Systems* (p. 3). Montreal, Canada: Springer.

Graesser, A., VanLehn, K., Rose, C., Jordan, P., & Harter, D. (2001). Intelligent tutoring systems with conversational dialogue. *AI Magazine*, 39-51.

Graesser, A. C., Jackson, G. T., & McDaniel, B. (2007). AutoTutor holds conversations with learners that are responsive to their cognitive and emotional states. *Educational Technology* (47), 19-22.

Graesser, A. C., Lu, S., Jackson, G. T., Mitchell, H. H., Ventura, M., Olney, A., et al. (2004). AutoTutor: a tutor with dialogue in natural language. *Behavior Research Methods, Instruments, & Computers* (36), 180-193.

Graesser, A. C., Wiemer-Hastings, K., Wiemer-Hastings, P., Kreuz, R., & TRG. (1999). AutoTutor: a simulation of a human tutor. *Cognitive Systems Research*, 35-51.

Hu, X., & Martindale, T. (2008). Enhance learning with ITS style interactions between learner and content. *The Interservice/Industry Training,* *Simulation & Education Conference (I/ITSEC)* (pp. 4-8). Orlando, FL, USA: I/ITSEC.

Hu, X., Cai, Z., Wiemer-Hastings, P., Graesser, A., & McNamara, D. (2007). Strengths, limitations, and extensions of LSA. In T. L. D. McNamara, *LSA: A Road to Meaning* (p. 401). Mahwah, NJ, USA: Erlbaum.

Jackson, T., Mathews, E., Lin, K.-I., Olney, A., & Graesser, A. (2003). Modeling student performance to enhance the pedagogy of AutoTutor. *User Modeling 2003* (p. 146). Johnstown, PA, USA: Springer Berlin/Heidelberg.

Landauer, T. K., Foltz, P. W., & Laham, D. (1998). An introduction to latent semantic analysis. *Discourse Processes, 25* (2,3), 259-284.

Martha Campbell Polson, J. J. (1988). *Foundations of intelligent tutoring systems.* Hillsdale, NJ, USA: Psychology Press.

Mathews, E. C., Jackson, G. T., Olney, A., Chipman, P., & Graesser, A. C. (2003). Achieving domain independence in AutoTutor. *The Seventh World Multiconference on Systemics, Cybernetics, and Informatics Proceedings: Computer Science and Engineerings I* (pp. 172-176). Orlando, FL, USA: World Multi-Conference on Systemics, Cybernetics and Informatics.

Merrill, D. C., Reiser, B. J., Ranney, M., & Trafton, J. G. (1992). Effective tutoring techniques: a comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences*, 277-305.

Millis, K., Magliano, J., Britt, A., Wiemer-Hastings, K., Halpern, D., & Graesser, A. (2006). *Acquiring research evaluative and investigative skills (ARIES) for* *scientific inquiry.* Northern Illinois University. DeKalb, IL, USA: Northern Illinois University.

Ong, J., & Ramachandran, S. (2003). *Intelligent Tutoring Systems: Using AI to Improve Training Performance and ROI.* San Mateo, CA, USA: Stottler Henke Associates Inc.

Person, N., Craig, S., Price, P., Hu, X., Gholson, B., Graesser, A., et al. (2000). Incorporating human-like conversational behaviors in AutoTutor. *Fourth International Conference on Autonomous Agents* (pp. 93-97). Barcelona, Spain: ACM Press.

Riordan, B., & Jones, M. N. (2009). Redundancy in perceptual and linguistic experience: exploring cohesion of semantic semantic categories in distributional and feature-based models of semantic representation. *Topics in Cognitive Sciene* .

Stottler, R. H. (2000). Tactical action officer intelligent tutoring system (TAO ITS). *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC) 2000* (p. 12). Orlando, FL, USA: I/ITSEC.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, *LIX*, 433-460.

VanLehn, K., Graesser, A., Jackson, G., Jordan, P., Olney, A., & Rose, C. (2007). When are tutorial dialogues more effective than reading? *Cognitive Science* (31), 3-62.