2011

Intelligent Tutoring System Authoring Tools for Non-Programmers

Shrenik Devasani
Iowa State University

Intelligent tutoring system authoring tools for non-programmers

by

Shrenik Devasani

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Co-majors: Human Computer Interaction; Computer Science

Program of Study Committee:
Stephen B. Gilbert, Co-major Professor
Leslie Miller, Co-major Professor
Craig Ogilvie

Iowa State University

Ames, Iowa

2011

DEDICATED

I would like to dedicate this thesis to my parents, Ram Manohar and Bharathi, and my sister Shravya.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

An intelligent tutoring system (ITS) is a software application that tries to replicate the performance of a human tutor by supporting the theory of "learning by doing" and providing customized instruction to a student while performing a task within a problem domain such as mathematics, medical diagnosis, or even game play. ITSs have been shown to improve the performance of a student in wide range of domains. Despite their benefits, ITSs have not seen widespread use due to the complexity involved in their development. Developing an ITS from scratch requires expertise in several fields including computer science, cognitive psychology and artificial intelligence. In order to decrease the skill threshold required to build ITSs, several authoring tools have been developed.

In this thesis, I document several contributions to the field of intelligent tutoring in the form of extensions to an existing ITS authoring tool, research studies on authoring tool paradigms and the design of authoring tools for non-programmers in two complex domains – natural language processing and 3D game environments.

The Extensible Problem Specific Tutor (xPST) is an authoring tool that helps rapidly develop model-tracing like tutors on existing interfaces such as webpages. xPST's language was made more expressive with the introduction of new checktypes required for answer checking in problems belonging to domains such as geometry and statistics. A web-based authoring (WAT) tool was developed for the purpose of tutor management and deployment and to promote non-programmer authoring of ITSs. The WAT was used in a comparison study between two authoring tool paradigms – GUI based and text based, in two different problem domains – statistics and geometry.

User-programming of natural language processing (NLP) in ITSs is not common with authoring toolkits. Existing NLP techniques do not offer sufficient power to non-programmers and the NLP is left to expert developers or machine learning algorithms. We attempted to address this challenge by developing a domain-independent authoring tool, ConceptGrid that is intended to help non-programmers develop ITSs that perform natural language processing. ConceptGrid has been integrated into xPST. When templates created using ConceptGrid were tested, they approached the accuracy of human instructors in scoring student responses.

3D game environments belong to another domain for which authoring tools are uncommon. Authoring game-based tutors is challenging due to the inherent domain complexity and dynamic nature of the environment. We attempt to address this challenge through the design of authoring tool that is intended to help non-programmers develop game-based ITSs.

CHAPTER 1. INTRODUCTION

1.1 Overview

An intelligent tutoring system (ITS) is a software application that tries to replicate the performance of a human tutor by supporting the theory of "learning by doing" and providing personalized feedback and customized instruction to a student or a trainee while performing a task within a problem domain such as mathematics, medical diagnosis, or even game play.

1.1.1 Components of an Intelligent Tutoring System

There are four components in an intelligent tutoring system that represent tutoring and communication knowledge (Woolf, 2008): a domain module, a student module, a tutoring module and communication module.

The domain module is the backbone of an ITS and consists of the domain knowledge which represents experts' behavior and how they perform in the domain. It can include definitions, skills and processes required to perform a task. The student module consists of knowledge that represents the student's misconceptions, skill levels, behavior and mastery of the domain. It also specifies how the tutor must reason about the student's knowledge. The tutoring module consists of knowledge that represents teaching strategies. The communication module consists of knowledge that represents methods for communication between the student and the tutoring system. Examples include graphical user interfaces, avatars, and conversational dialogue mechanisms.

1.1.2 Classification of Intelligent Tutoring Systems

Intelligent tutoring systems can be classified into 3 groups:

1. Model-tracing tutors
2. Constraint-based tutors
3. Example-tracing tutors

1.1.2.1 Model-tracing tutors

Model-tracing tutors are based on the ACT-R theory and architecture of cognition (Anderson & Lebiere, 1998). According to the ACT-R theory, human knowledge can divided into two kinds of representations – declarative (consisting of facts) and procedural (consisting of productions). Procedural knowledge is formed from declarative knowledge.

A model-tracing tutor is associated with an expert model that comprises of production rules that represent domain knowledge. Model-tracing tutors employ a process called "model tracing" where the expert model is used to trace the student's actions. A model-tracing tutor can offer feedback for every step taken by the student while solving a problem. It identifies errors when a step taken by the student either matches a production rule that represents an incorrect step, or fails to match any rule.

Model-tracing tutors have been successfully implemented in several domains including college-level physics (Gertner & VanLehn, 2000; Shelby et al., 2001) and high school algebra (K.R. Koedinger, J.R. Anderson, W.H. Hadley, & M.A. Mark, 1997).

1.1.2.2 Constraint-based tutors

Constraint-based tutors (Mitrovic, Mayo, Suraweera, & Martin, 2001) employ the constraint-based modeling approach of student modeling. It is based on Ohlsson's theory of learning from performance errors (Ohlsson, 1996). Knowledge is represented in the form of constraints, rather than problem-solving paths. A constraint consists of three components – a relevance condition that describes when the constraint is applicable, a satisfaction condition that specifies additional tests and a feedback message associated with the constraint. A constraint-based tutor is interested in the current state that the student is in, rather than what the student has done thus far. As long as the student does not enter an incorrect state, he or she is free to do as desired. States that are pedagogically equivalent are grouped together into equivalence classes. All states in an equivalence class trigger the same instructional actions by the tutor. Constraint-based modeling requires a higher level of abstraction resulting in smaller domain models, which in turn reduces authoring effort.

Examples of constraint-based tutors include the SQL-Tutor, an ITS that supports students learning to write SQL queries (Mitrovic & Ohlsson, 1999) and the KERMIT, an ITS that teaches database design (Suraweera & Mitrovic, 2002).

1.1.2.3 Example-tracing tutors

Example-tracing tutors evaluate student behavior by comparing it with examples of correct and incorrect problem-solving behavior (V. Aleven, B. M. McLaren, J. Sewall, & K. R. Koedinger, 2009). Example-tracing tutors are capable of providing step-by-step guidance. They can tutor on complex problems consisting of multiple correct strategies by maintaining multiple correct interpretations of student behavior. Though example-tracing tutors are easier

to build than model-tracing tutors, it has been shown that the two types can be behaviorally indistinguishable.

Authoring tools like CTAT and xPST can be used to develop example-tracing tutors.

1.1.3 Benefits of Intelligent Tutoring Systems

Intelligent tutoring systems have been shown to improve the performance of a student in wide range of domains. Beal et al performed a controlled evaluation of an interactive on-line tutoring system for high school mathematics (Beal, Walles, Arroyo, & Woolf, 2007). They showed that students who received on-line tutoring showed improvement while students who received regular classroom instruction showed no pre- to post-test improvement. The AutoTutor, an ITS that simulate a human tutor by holding a conversation with the student in natural language has shown to achieve learning gains of approximately 0.8 sigma (Graesser, Chipman, Haynes, & Olney, 2005). PAT, an algebra tutor was used in a large scale experiment where 470 students in experimental classes using the tutor outperformed students in comparison classes by 15% on standardized tests (K.R. Koedinger, et al., 1997).

1.1.4 Challenges of Authoring Intelligent Tutoring Systems

Developing effective ITSs requires a good understanding of the thought processes involved both while teaching, and while learning. It has been observed that it takes 100 hours of development time to create 1 hour of instruction (Woolf & Cunningham, 1987). Building intelligent tutoring systems requires expertise in several fields such as computer science, artificial intelligence, cognitive psychology and interface design. This requirement makes building ITSs from scratch, a time consuming and challenging process and increases the

costs of creating the ITS, limiting the number of ITSs that can be created within the various domains.

1.2 Authoring Tools for Intelligent Tutoring Systems

The goal of an ITS authoring tool is to simply the process of building ITSs and decrease the skill threshold for building them. Also, they enable the rapid prototyping of ITS designs (T. Murray, 1999). Achieving these goals will help non-programmers and users who lack computational thinking build ITSs.

Computational thinking can be defined as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" (Wing, 2006).

Several authoring tools have been developed that are targeted specifically for non-programmers (Blessing, Gilbert, Ourada, & Ritter, 2007; Koedinger, Aleven, Heffernan, McLaren, & Hockenberry, 2004). When an authoring tool is intended to be used by non-programmers, it is essential to manage the trade-off that exists between ease of use and power, and scaffold computational thinking in the authoring tool for non-computational thinkers.

In this section, I discuss about two authoring tools: Cognitive Tutor Authoring Tools (CTAT) and the Extensible Problem Specific Tutor (xPST).

1.2.1 Cognitive Tutor Authoring Tools (CTAT)

The Cognitive Tutor Authoring Tools, or CTAT (Koedinger, Aleven, & Heffernan, 2003), is a tool suite that enables an instructor to add learning by doing to online courses.

CTAT supports the creation of two types of tutors: example-tracing tutors , which can be created without programming but require problem-specific authoring, and cognitive tutors, which require AI programming to build a cognitive model of student problem solving but support tutoring across a range of problems.

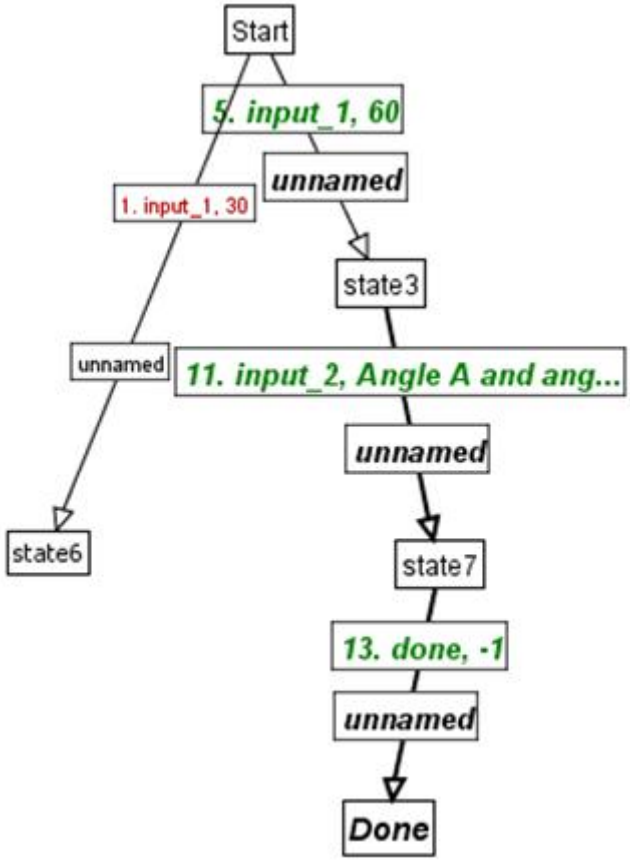


Figure 1.1 – Behavior graph in CTAT

A CTAT tutor-author uses an interface builder to create an interface for the tutor. Once the interface is ready, an example-tracing tutor can be developed by creating a directed, acyclic graph called the "behavior graph" that represents the acceptable ways of solving a problem. The links in the graph represent problem-solving actions, and the nodes represent

problem-solving states. A behavior graph can contain incorrect solution paths that reflect incorrect problem solving behavior by the student.

1.2.2 Extensible Problem Specific Tutor (xPST)

The Extensible Problem Specific Tutor, or xPST (Blessing, Gilbert, Blankenship, & Sanghvi, 2009), is an ITS authoring tool that helps rapidly develop example-tracing tutors on existing interfaces such as webpages. Building tutors on existing interfaces reduces tutor-development time, and allows the interface to be separable from the tutoring component.

The xPST System consists of three main components:

1. The xPST Engine
2. The Presentation Manager
3. The Web Authoring Tool

The xPST engine "eavesdrops" on the tutor interface and observes the student's actions on the interface. The Presentation Manager gives visual feedback to the student on the interface. The Web Authoring Tool helps tutor authors create and deploy tutors.

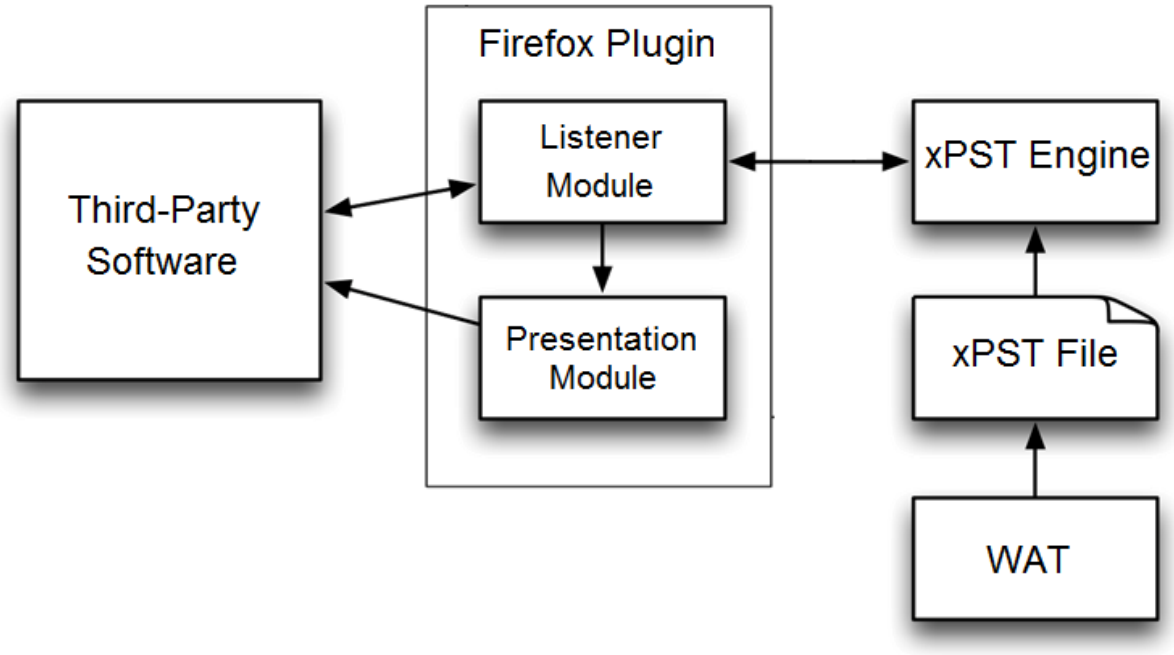


Figure 1.2 – xPST architecture

In order to create tutor, the author must create a cognitive model by writing an xPST file, using the Web Authoring Tool. This model represents the procedural knowledge necessary to correctly complete the steps involved in the task. The xPST file is a text file and uses a simple cognitive modeling language. Each xPST file must have a sequence section, a feedback section, and a mappings section.

In the sequence section, the author must mention the order in which steps must be completed by the learner. The sequence section supports different connecting words like "and", "or" and "then". For example, "stepA then stepB" corresponds to "Do stepA first and then do stepB".

In the feedback section, the author specifies the hints and just-in-time error messages (JITs) for each step name mentioned in the sequence section. The author must also specify the correct "answer" to each step. When the learner completes the corresponding step, it is

the equivalent of providing the correct answer to that step, and the tutor moves on to the next step in the sequence.

Each widget on the webpage has a unique ID, which can be seen using the xPST plugin, when the author clicks on the widget. The author must map these widgets to the step names used in the sequence section. This section helps the tutor know whether the learner has completed the required steps by eavesdropping on the learner's actions on the webpage.

1.3 Previous Work With xPST

The Extensible Problem Specific Tutor has been extended to support tutoring in 3D game environments and studies have been conducted to show xPST has been used by authors to build tutors on both web interfaces and game environments.

1.3.1 xPST Authoring Study

This study tested the ability of novice users of xPST to create tutors (S. Gilbert, Blessing, & Kodavali, 2009). The purpose of the study was to examine the learning curve of novice authors while learning to build tutors using xPST.

The study involved 10 participants. They were given access to an example tutor and a 44-minute video tutorial. Each participant was asked to build three tutors each for three different tasks, all related to searching for a particular library database (the ACM portal). A total of 26 tutors were developed by the 10 participants (some failed to complete all three tutors). The tutors were scored on a scale of 1 to 5, according to the rubric used in Blessing and Gilbert (2008). Eighteen tutors received a score of 4 or more. The time taken by the

participants to build the tutors was kept track of. The three tasks, on an average required 3.71 hours, 2.53 hours and 1.73 hours, respectively.

1.3.2 Torque xPST Driver

xPST was extended to support tutoring in 3D game environments developed using the game engine Torque (Kodavali, Gilbert, & Blessing, 2010). The Torque xPST driver acts as a bridge between the Torque game engine and the xPST engine to enable xPST tutoring in games. It does the job that the Firefox plugin does while tutoring on web interfaces. The driver eavesdrops on the game and captures events and actions done by the learner and sends them to the xPST engine.

Torque Game Engine Advanced (TGEA) (Lloyd, 2004), a commercial off-the-shelf game engine from GarageGames, was used as the simulation engine. TGEA supports scripting through TorqueScript, which is similar in syntax to JavaScript. It was used in developing both the driver and the game scenarios.

The Torque xPST driver comprises of two components – the Listener Module and the Presentation Module. The Listener Module captures events that occur in the game and sends them to the xPST engine. It also receives feedback that needs to presented to the learner from the xPST engine and sends it to the Presentation Module. The Presentation Module is involved in presenting the feedback it to the learner, which includes hints and just-in-time error messages. The communication between the xPST Engine and the Torque xPST driver happens through a “Dormin message”, which is a string in a specific format that contains various attributes that describe the current state of the task, the message to be communicated, and the action verb that determines the message’s course of action.

1.3.3 Torque xPST Authoring Study

This study tested the ability of novice users of xPST to create tutors in 3D games (S. B. Gilbert, Devasani, Kodavali, & Blessing, 2011). The purpose of the study was to examine the learning curve of novice authors while learning to build tutors for 3D games, using xPST.

The study involved 10 participants, selected based on a pre-survey, that included them only if they had a minimal amount of programming experience. They were given access to an example tutor and a 15-minute video tutorial. Each participant was asked to build two tutors each for two different tasks, online, over a two week period. The first task, titled "Target Acquisition", required the player to enter the proximity region of an enemy tower, start communication with his base, report his location, and then issue a "Fire" command. The second task, titled "Evacuate", was based on a scenario consisting of three cottages with a hostage in each of them. The player had search for the three cottages, and issue an "Evacuate" command to each of the three occupants. The "Target Acquisition" task required a minimum of three goalnodes, and the "Evacuate" task required a minimum of seven goalnodes.

A total of 20 tutors were developed by the 10 participants. The average time to complete Task A was 19.74 minutes, with a standard deviation of 9.16 minutes. The average time to complete Task B was 13.81 minutes with a standard deviation of 6.24 minutes. The results indicated that users with minimal programming experience could use xPST to create tutors for 3D game environments.

1.4 Research Questions

The research work described in this thesis attempts to answer the following questions:

1. How does an intelligent tutoring system authoring tool paradigm and the complexity of a problem domain affect the tutor-authoring process by non-programmers in comparison with programmers?
2. Can an authoring tool that uses a GUI to facilitate use by non-programmers enable the creation of a tutor that can accurately evaluate written textual responses as a human instructor would manually do?

1.5 Thesis Organization

This chapter provided an introduction to the field of intelligent tutoring systems and the challenges involved in authoring them. It has also described the Extensible Problem Specific Tutor (xPST) and the previous work related to it. In the rest of my thesis, I document several contributions to the field of intelligent tutoring in the form of extensions to an existing ITS authoring tool, research studies on authoring tool paradigms and the design of authoring tools in two complex domains – natural language processing and 3D game environments.

Chapter 2 describes a Web-based Authoring Tool (WAT) that allows easy creation of ITSs and their deployment on the web. It supports both learner management and tutor management on a single platform. In Chapter 3, I describe an evaluation of two intelligent tutoring system authoring tool paradigms: graphical user interface based and text based. Chapter 4 is a conference paper titled “Lattice-Based Approach to Building Templates for Natural Language Understanding in Intelligent Tutoring Systems” (Devasani, Aist, Blessing, & Gilbert, 2011). My contribution includes the design and development of the system, and writing major portions of the paper. Gregory Aist provided me with valuable suggestions,

especially for the user interface. Stephen Blessing provided me a corpus collected from one of his studies, which was used to evaluate and test the accuracy of my tool. Stephen Gilbert provided me with valuable suggestions and edited the paper. Chapter 5 is a workshop paper titled "Authoring Intelligent Tutoring Systems for 3D Game Environments" (Devasani, Gilbert, Shetty, Ramaswamy, & Blessing, 2011). My contribution includes the design and development of the framework, and writing major portions of the paper. Stephen Gilbert designed the user interface for the authoring tool. Suhas Shetty and Nandhini Ramaswamy made valuable improvements after identifying drawbacks in my design. Stephen Blessing provided valuable feedback and edited the paper. Chapter 6 summarizes the important features of this thesis and describes the future work involved with the tools I have developed. Appendix A describes various extensions to xPST that have developed in order to make it more expressive and powerful as a computational tool, while remaining useful to non-programmers. Appendix B contains the material relevant to the study described in Chapter 3.

CHAPTER 2. WEB-BASED AUTHORING TOOL

xPST is an ITS authoring tool that allows non-programmers to create tutors. Users without a technical background can find the tasks of management and deployment of tutors to be laborious. A web-based authoring tool that can take care of these issues is required and helps in more rapid creation of ITSs. A web-based authoring tool can prove to be extremely useful for the purposes of both maintenance and distribution of tutors that are developed using an intelligent tutoring system authoring tool. An efficient web-based authoring tool can manage all the resources and dependent files at a single location, thereby allowing the learner to concentrate on the task of building tutors and developing tutoring strategies.

2.1 Previous Work

An Authoring Tool was built to help authors develop tutors with xPST. It was a simple Integrated Development Environment (IDE) for authoring xPST tutors. It was designed to serve two purposes: 1) To provide a simple, easy to use graphical user interface (GUI) to author xPST files without installing additional software on the client computer 2) To provide a tool to log the time spent by the author while writing the xPST file. The tool also performs syntax checking and points out errors in the xPST file.

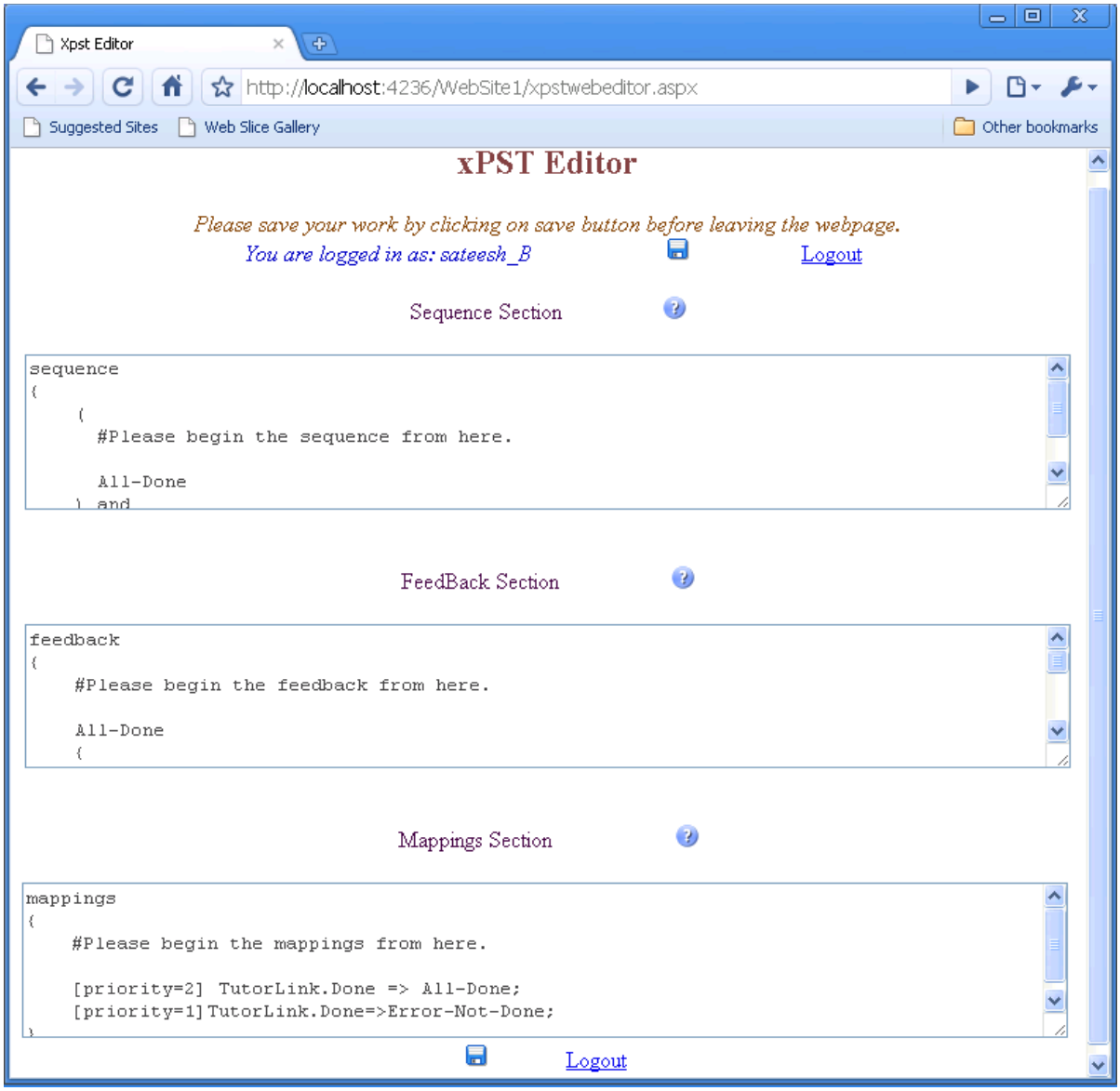


Figure 2.1 – Screenshot of the previous xPST web-based authoring tool

2.2 Current Implementation

The current implementation of the xPST Web-Based Authoring Tool (WAT) has implemented several improvements over the previous version. It supports both user management and tutor management on a single platform.

2.2.1 User Management

The Web-Based Authoring Tool supports user management in a simple manner. Users can create and manage own accounts on the WAT.

2.2.2 Tutor Management

Once registered, users can develop, manage, and deploy their web-based xPST tutors. Users can develop tutors over multiple sessions using the Tutor Editor, which is a part of the WAT. When a new tutor is created, the WAT creates four files associated with the tutor - .xpst file, .scenario file, .HTML file, and the .properties file, and links them. If needed, the user can download the four files associated with each tutor to his computer so that he can deploy them locally on his server.

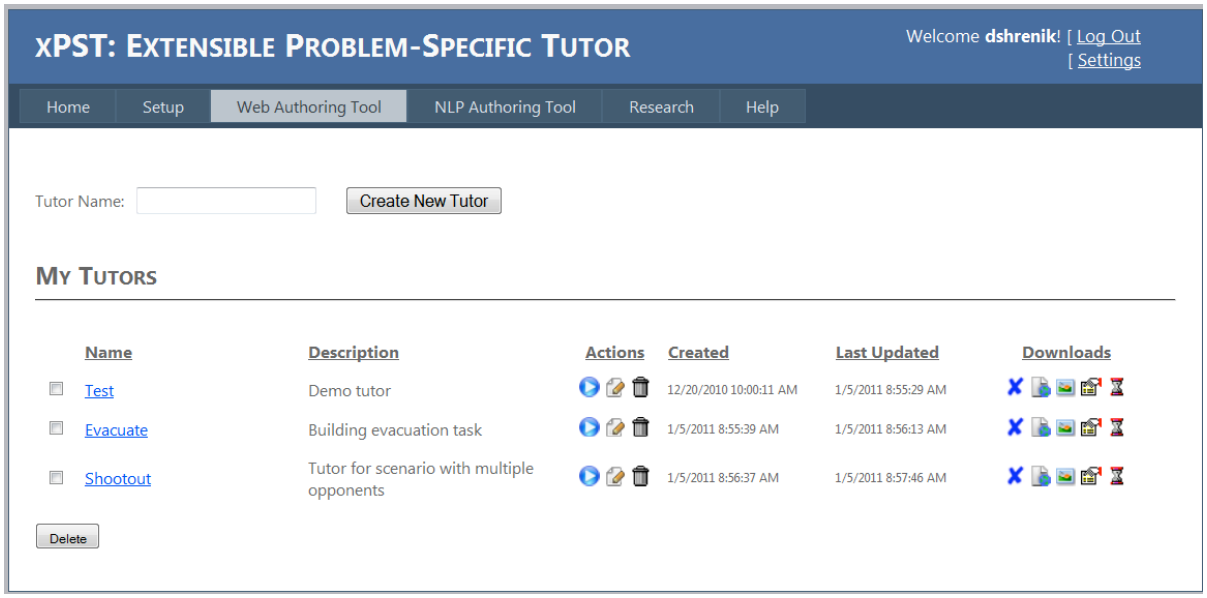


Figure 2.2 – Current xPST web-based authoring tool (WAT)

The following process is to be followed in order to create a tutor with the WAT:

1. Create the problem interface (webpage) if it does not already exist.
2. Login to the WAT
3. Create a new tutor
4. Provide the URL to the webpage that serves as the problem interface
5. Draft an xPST file using the Tutor Editor
6. WAT creates the necessary files, links and deploys them
7. Test the tutor by clicking “Run”

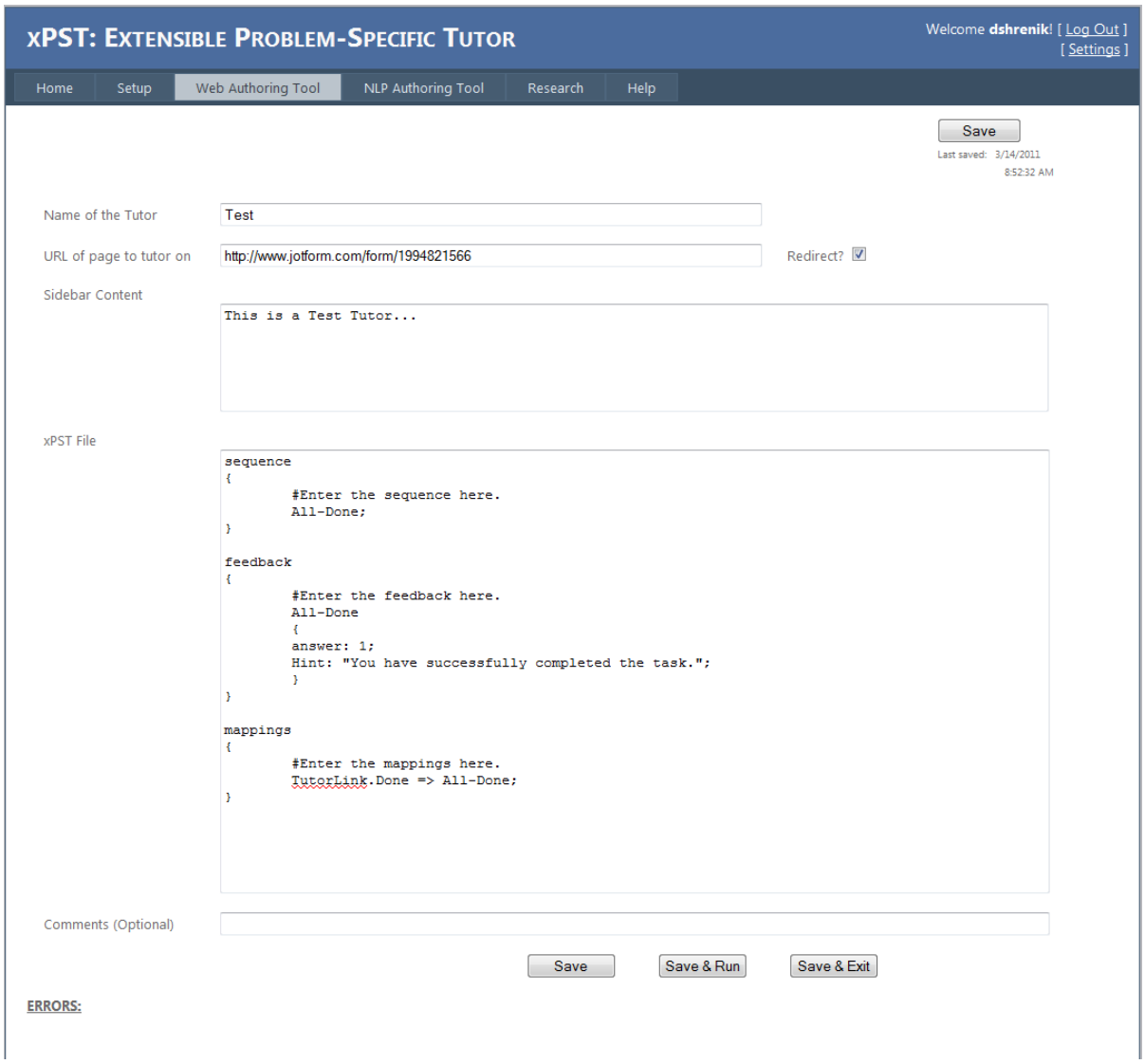


Figure 2.3 – WAT's Tutor Editor

2.2.3 Event Logging

The WAT supports logging of events that occur while an author develops a tutor. A separate log file is created and updated for each tutor that a user creates. This feature is useful for the purposes of data mining in research studies.

The following events are recorded by the WAT in the log file:

- TutorCreated – Time at which the tutor was created. This event is logged only once.
- TutorOpen – Time at which the tutor was opened. This event is logged when the tutor is created and every time the user clicks the 'Edit' button from the Tutor List.
- ManualSave – Time at which the tutors is saved manually. This event is logged every time the user clicks one of the "Save", "Save & Run", and "Save & Exit" buttons.
- AutoSave – The time at which the tutor was auto-saved. This happens every 20 seconds.
- xPSTEdit – This event is logged along with ManualSave or AutoSave, provided a change has been made to the xPST file since the last ManualSave or AutoSave (whichever occurred later).

The details of these events allow researchers to identify how long it takes to create tutors for a particular domain. Using the details of the above events, two important measures are calculated and recorded in the log file:

- xPSTTime – The total time spent by the user in editing the contents of the xPST file.

  $xPSTTime = \Sigma(X_i)$, where $X_i = value(i^{th}\ XpstEdit)$
- TotalTime – The total duration for which the Tutor Editor was open. This includes time spent in both editing the contents of the xPST file and testing the tutor.

TotalTime = $\Sigma(X_i) + \Sigma(Y_i)$, where X_i = value(i^{th} ManualSave), Y_i = value(i^{th} AutoSave)

2.2.4 Other Features

An important feature of the WAT is the syntax checker (this is an improved and more robust version of the syntax checker from the previous xPST authoring tool described in 2.1). It identifies syntax errors in the xPST file, and displays all the errors along with their line numbers and possible solutions at the bottom of the page. This feature is necessary to reduce tutor development time since early piloting of the interface showed that novice users spend most of their time correcting syntax errors.

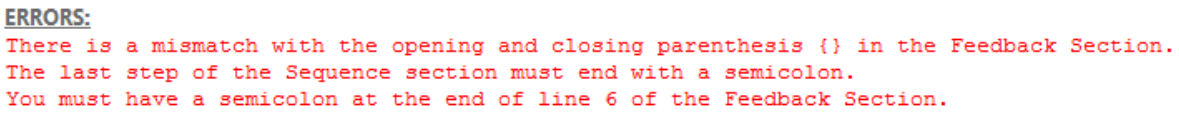


Figure 2.4 – Sample listing of errors in an xPST tutor

The WAT supports Versioning, so that previous versions of the tutors are not lost. Every time a tutor is manually saved, a new version is recorded, along with optional comments that the user might have specified that will be associated with that version.

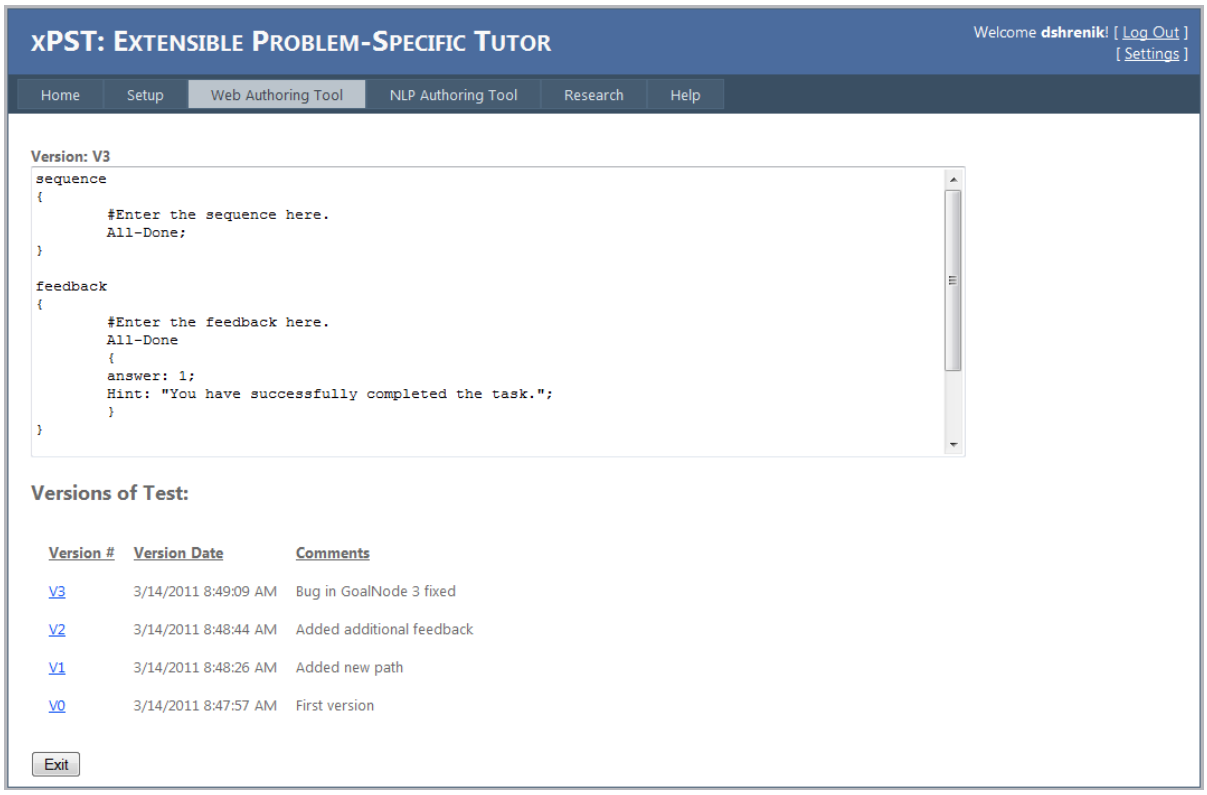


Figure 2.5 – Version management within the WAT

Also, the Tutor Editor automatically saves the xPST file periodically, every 20 seconds.

2.2.5 Testing

The WAT has been successfully used for two different cognitive modeling studies. The first was called "Evaluation of WebxPST: A Browser-Based Authoring Tool for Problem-Specific Tutors" (Blessing, Devasani, & Gilbert, 2011). A total of five tutor-authors, two course instructors, and three undergraduate students created a total of 74 tutors for statistics homework problems, using the WAT. The second study, titled "Evaluation of Two Authoring Tool Paradigms: GUI Based and Text Based" (see Chapter 3), had a total of

eight tutor-authors develop a total of 16 geometry and statistics tutors. Both the studies were conducted smoothly on the WAT and the tutor-authors reported no technical problems.

CHAPTER 3. EVALUATION OF TWO INTELLIGENT TUTORING SYSTEM AUTHORING TOOL PARADIGMS: GRAPHICAL USER INTERFACE-BASED AND TEXT-BASED

Manuscripts from this chapter will be submitted as a paper to the Eleventh International Conference on Intelligent Tutoring Systems. (2012).

Shrenik Devasani, Stephen Gilbert, Stephen Blessing

3.1 Abstract

We describe an evaluation of two intelligent tutoring system authoring tool paradigms, graphical user interface-based and text-based in two domains, statistics and geometry. We conducted a study with 16 tutor-authors divided into 2 groups (programmers and non-programmers). Our results showed that the GUI-based approach provides a much lower bar for entry when compared to the text-based approach. However, the difference in tutor-authoring time between the two approaches reduces as the tutor-authors gain experience using the respective authoring tools.

3.2 Introduction

Authoring an intelligent tutoring system from scratch is a challenging task since it requires expertise in several fields including cognitive science, computer science and pedagogy. An authoring tool tries to lower the skill threshold required for developing ITSs and also enable their rapid development (T. Murray, 1999).

When an authoring tool is being designed, there are several design trade-offs involved because many of the design decisions that lead to an authoring tool result from conflicting trade-offs. For example, increasing the power of an authoring tool might come at the cost of its ease of use. In this paper, we try to evaluate the effects of the trade-offs involved in the design of GUI-based and text-based authoring tools.

Studies have been conducted to identify the advantages and disadvantages of visual programming (Whitley, 1997). Visual representations have shown to be beneficial when the size or complexity of the problem grows (Day, 1988; Polich & Schwartz, 1974). On the other hand, visual representations use space-saving techniques which cause low screen density, when compared to textual representations, and therefore will not be practical for large problems (Whitley, 1997).

For the experiment, we chose CTAT (Koedinger, et al., 2003) and the Extensible Problem Specific Tutor, or xPST (Blessing, et al., 2009) as examples of GUI-based and text-based authoring tools respectively. Both CTAT and xPST can be used to develop example tracing tutors (V. Aleven, B. McLaren, J. Sewall, & K. R. Koedinger, 2009), for both single strategy problems and multiple strategy problems. We chose two problem domains of varying complexity, from an authoring point of view – statistics and geometry. Statistics problems are sequential in nature and generally have a single solution path. Problems in geometry could have multiple strategies and therefore multiple solution paths.

3.2.1 Cognitive Tutor Authoring Tools (CTAT)

CTAT supports the development of example-tracing tutors through a technique called "programming by demonstration" (Nevill-Manning, 1993). Once the interface for a problem

has been built, the tutor-author proceeds by demonstrating all possible solution paths to the problem. The demonstration automatically creates a directed, acyclic graph called the "behavior graph", which represents the acceptable ways of solving a problem. The feedback associated with the individual states in the problem is added to the tutor through CTAT's GUI.

Though CTAT is capable of supporting the development of both cognitive tutors and example-tracing tutors, we consider only the example-tracing tutor development feature of CTAT in this study.

3.2.1 Extensible Problem Specific Tutor (xPST)

xPST is an ITS authoring tool that helps rapidly develop example-tracing tutors on existing interfaces such as webpages. An example-tracing tutor is built using xPST by writing a text file that describes the order in which individual steps in the problem are to be completed by the learner and the answers and feedback associated with each step.

xPST's syntax was designed such that it was simple enough for non-programmers to use, and also powerful enough for experienced users to build tutors rapidly.

3.3 Methods

The experiment involved 2 independent variables – authoring tool paradigm (text-based or GUI-based) and problem domain (statistics or geometry). Programming level (programmer or non-programmer) was the moderating variable.

3.3.1 Participants

Participants were recruited through an email advertisement, fliers on campus and word-of-mouth. Sixteen participants completed the study successfully. Each participant had to take a pre-survey that asked them questions about their experience with computer programming. A participant was classified as a "programmer" if he or she had taken two or more programming courses and as a "non-programmer", otherwise. The participants were divided into eight groups formed from all possible combinations of programming level (programmer or non-programmer), authoring tool paradigm (text-based or GUI-based) and problem domain (statistics or geometry). Before starting the study, the participants were asked to complete and sign an informed consent form online.

3.3.2 Materials

Each participant was given the task of building three tutors using a specific authoring tool (CTAT or xPST) for a specific domain (statistics or geometry). All three problems were of the same complexity, with their solutions having six subgoals or steps.

The participants were provided with the link to the study webpage that had all the resources and material required to complete their tasks. The study webpage contained a brief introduction the field of intelligent tutoring systems. The training material on the webpage included a video tutorial that gave a demo of the step-by-step procedure to be followed while creating a tutor, for a sample problem as well as a text tutorial. We estimate the total training time to be about 1-2 hours.

CTAT tutor-authors used Remote Desktop to log in remotely to a Microsoft Windows computer which had CTAT v2.10.0 and Adobe Flash Player 10 pre-installed. xPST tutor-

authors logged in to the xPST Web-based Authoring Tool (WAT, described in Chapter 2) using Mozilla Firefox 3 or higher. The problems for which the tutors were to be built were predefined and the interfaces for the problems were provided to the participants. CTAT tutor-authors could access the problem interfaces (.swf files) on the remote machine provided to them. xPST tutor-authors were provided with the links to the webpages that contained the problem interfaces.

3.3.3 Procedures

Each participant was asked to create three tutors, as if he or she was a teacher for that subject preparing homework problems for his or her students. Instructions were provided for each problem that included the problems' solutions and the types of feedback the tutor must give for each problem. The tutors created by the participant were meant to monitor each step in the corresponding problem. For each problem, the tutor had to provide exactly one hint. Also, for each tutor overall, there was one message that gives feedback for a specific error that a student might make.

The entire study was conducted online. The participants were allowed to complete their tasks over multiple sessions at their own pace, over a two-week period. After successful completion of their tasks, the participants received a compensation of $40 in cash and a chance to win $149 in cash through a lottery.

3.4 Results

We had a total of eight groups with two participants each. Each participant built a total of three tutors, leading to the creation of 48 tutors in all.

3.4.1 Model Analysis

Each tutor was scored on two criteria – "Solution Path" and "Error-Specific Feedback." A tutor was given a score of 1 under "Solution Path" if it correctly provided tutoring for all possible solution paths in the problem (including providing hints on each step), 0.5 if it correctly provided tutoring for one of the possible solution paths and 0 if it provided completely incorrect tutoring. A tutor received a score of 1 under "Error-Specific Feedback" if it correctly provided the required error-specific feedback and 0, otherwise.

The cumulative scores have been shown in Table 3.1. Since a group had two participants who built three tutors each, the maximum score possible is six. The model analysis shows that all the tutor-authors who were classified as programmers built tutors that provided accurate tutoring. Tutor-authors who were classified as non-programmers built tutors that displayed accurate tutoring behavior for statistics problems, but slightly less accurate behavior for geometry problems.

Table 3.1 – Scoring of tutors (maximum possible score is 6)

Authoring Tool	Problem Domain	Programmer / Non-programmer	Solution Path	Error-Specific Feedback
xPST	Statistics	Programmer	6	6
		Non-Programmer	6	6
	Geometry	Programmer	6	6
		Non-Programmer	4.5	5
CTAT	Statistics	Programmer	6	6
		Non-Programmer	6	6
	Geometry	Programmer	6	6

	Non-Programmer	4	6

3.4.2 Timing Data

The total time spent in creating each tutor was logged separately. The time spent in creating an xPST tutor was calculated as described in 2.2.3. The CTAT logger logs the time and date when the tutor-author interacts with the GUI. This total time measure for both CTAT and xPST includes the time spent in authoring the tutor as well as testing it.

Figure 3.1 shows the histogram of the time spent in minutes by the xPST tutor-authors, in building tutors for all three problems.

Figure 3.2 shows the histogram of the time spent in minutes by the CTAT tutor-authors, in building tutors for all three problems. The log file for the second tutor created by one of the participants (P11) was unavailable.

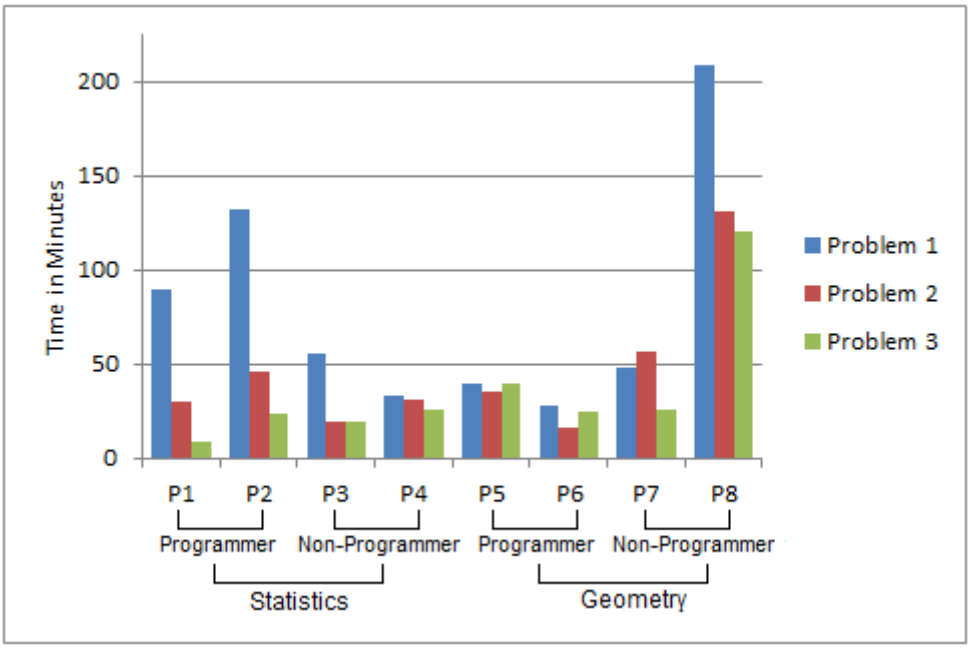


Figure 3.1 – Time spent by xPST tutor-authors

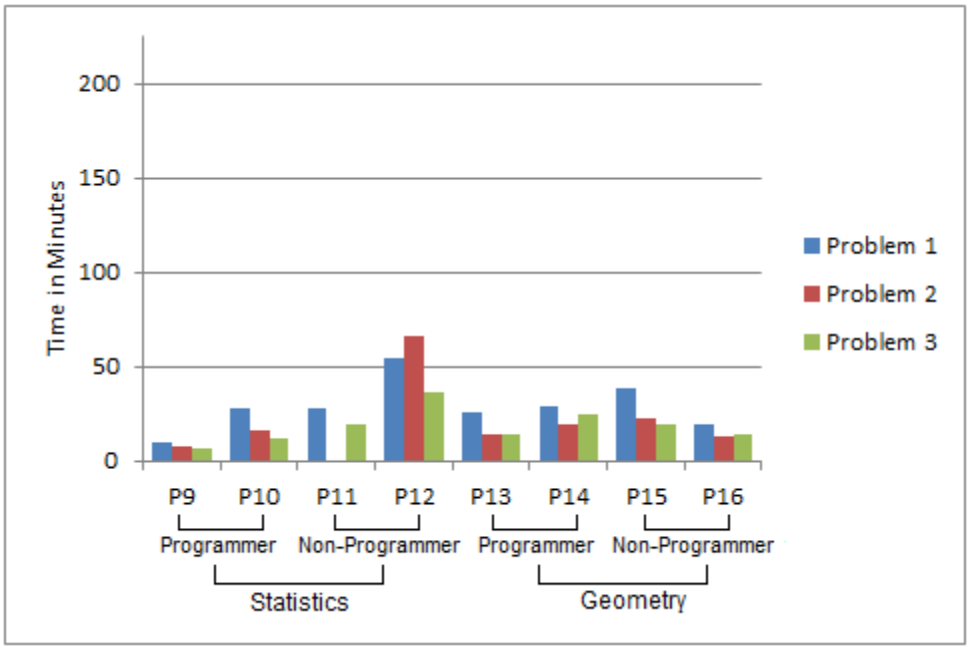


Figure 3.2 – Time spent by CTAT authors

Figure 3.3 shows the learning curve for the tutor-authors. It is interesting to see that after the tutor-authors gained experience building three tutors, the average time required in creating a tutor by the groups xPST–Statistics (19 min), CTAT–Statistics (18.75 min) and CTAT–Geometry (18 min) were almost equal. However, the average time required in creating a tutor for the third geometry problem using xPST (52 min) was much higher than the average time required using CTAT (18 min). Geometry problems involve multiple solution strategies. The results suggest that subtle ordering of steps is more convenient in CTAT because of CTAT's visual representation of the strategies on the behavior graph.

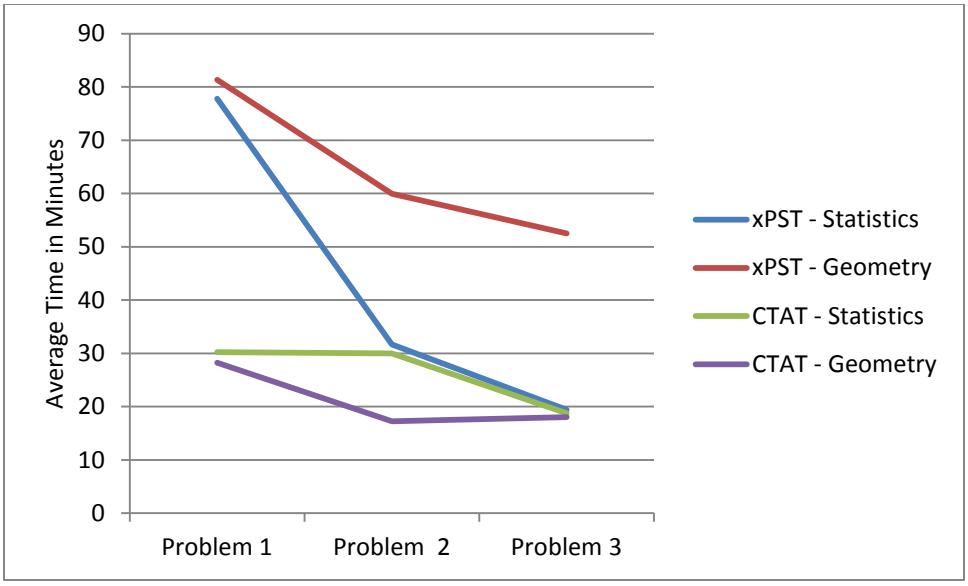


Figure 3.3 – Average time versus problem number

3.4.3 Exit Questionnaire Data

All 16 participants answered a short questionnaire (though it was optional). The questionnaire asked them for their feedback about the authoring tool they had used to create tutors. They were asked to rate the ease of use and power of the authoring tool on a Likert scale of 1 to 5, and to answer open-ended questions about the tool's strengths, weaknesses, and suggestions for improvement. One common theme that emerged from the open-ended questions was that both xPST and CTAT are easy to author once understanding how they work. Some quotations are included below that illustrate these:

"Very easy to use once you get a feel for the syntax." – P7 (xPST – Non-Programmer)

"Once we understand how to create the tutor then the tool is very simple to use." – P14 (CTAT – Programmer)

The average ratings from the exit questionnaire have been summarized in Table 3.2. Both xPST and CTAT were rated slightly higher by programmers for their ease of use when compared to non-programmers.

Table 3.2 – Average ratings by participants on a Likert scale of 1 to 5

Authoring Tool	Programmer / Non-programmer	Ease of Use	Power
xPST	Programmer	4.25	3.50
	Non-Programmer	3.50	3.25
CTAT	Programmer	4.75	4.00
	Non-Programmer	3.50	4.00

3.5 Discussion and Future Work

We used a between-subjects experimental design to prevent order effects and contamination as the participants moved between authoring systems and domains. We felt that the comparisons between levels of the independent variables would be compromised having if the variables were to be within-subject. We cannot claim that our results are statistically significant, likely because of the small sample size.

The graphical user interface-based approach allows for easier learning initially and a lower bar for entry. However, once a tutor is built, it can be time consuming to edit. One of the advantages of the text-based approach is that debugging and editing an existing tutor may be easier since the entire code is available to the tutor-author at one glance. We conclude by proposing a hybrid authoring tool that exploits the synergy between the graphical user interface-based paradigm and the text-based paradigm. Much like common integrated development environments (IDEs) such as Adobe Dreamweaver and Microsoft Visual

Studio, the ideal authoring tool would have a “design” tab which tutor-creation through a GUI and a “source” tab that supports the editing of the tutor directly through code. We expect such a tool would cater to programmers and non-programmers, experienced and non-experienced.

CHAPTER 4. LATTICE-BASED APPROACH TO BUILDING TEMPLATES FOR NATURAL LANGUAGE UNDERSTANDING IN INTELLIGENT TUTORING SYSTEMS

Research described in this chapter was published in the Fifteenth Conference on Artificial Intelligence in Education, Auckland. (2011).

Shrenik Devasani, Gregory Aist, Stephen Blessing, Stephen Gilbert

4.1 Abstract

We describe a domain-independent authoring tool, ConceptGrid, which is intended to help non-programmers develop intelligent tutoring systems (ITSs) that perform natural language processing. The approach involves the use of a lattice-style table-driven interface to build templates that describe a set of required concepts that are meant to be a part of a student's response to a question, and a set of incorrect concepts that reflect incorrect understanding by the student. The tool also helps provide customized just-in-time feedback based on the concepts present or absent in the student's response. This tool has been integrated and tested with a browser-based ITS authoring tool called xPST.

4.2 Introduction

Interpreting textual responses from students by an Intelligent Tutoring System (ITS) is essential if it can come close to matching the performance of a human tutor, even in domains such as Statistics and Physics, since the use of language makes the learning process

more natural. Natural language has the advantage of being easy to use for the student, as opposed to learning new formalisms.

Over the past decade, studies have been conducted that confirm the importance of using language in both traditional learning environments and in intelligent tutoring systems. Chi et al. (Chi, Bassok, Lewis, Reimann, & Glaser, 1989; Chi, De Leeuw, Chiu, & LaVancher, 1994) have showed that eliciting self-explanations enhances deeper learning and understanding of a coherent body of knowledge that generalizes better to new problems. Aleven et al. (Aleven, Koedinger, & Cross, 1999) conducted studies with the PACT Geometry Tutor in which students who provided explanations to solution steps showed greater understanding in the post-test, compared to students who did not provide explanations.

Many ITSs have successfully incorporated natural language processing. The CIRCSIM Tutor (Glass, 2001) is a language based ITS for medical students that uses word matching and finite state machines to process students' natural language input. Rus et al. (Rus & Graesser, 2006) have described an approach of evaluating answers by modeling it as a textual entailment problem. Intelligent tutoring systems such as the AutoTutor (Graesser et al., 2000) and Summary Street (Steinhart, 2001) use Latent Semantic Analysis (LSA) (Landauer, Foltz, & Laham, 1998) to evaluate student answers, a technique that uses statistical computation and is based on the idea that the aggregate of all the word contexts in which a word appears determines the similarity of meaning of words to each other. The problem with LSA is that it does not encode word order and it cannot always recognize negation. Another problem with LSA is that it scores students' responses only based on how

well it matches the ideal answer, and cannot point out what exactly is wrong with an incorrect response.

Though ITSs today use a variety of techniques to provide support for natural language understanding, user-programming of NLP in ITSs is not common with authoring toolkits. The various techniques described here do not give sufficient power to non-programmers as the NLP is left to expert developers or to machine learning algorithms, and the user is more likely to focus on tutoring strategies. Our approach addresses these issues.

4.3 The ConceptGrid Approach

ConceptGrid is intended to be used by tutor-authors with little or no programming experience. The most crucial aspect about developing an authoring tool that can be used by non-programmers is managing the trade-off between its ease of use and its expressive power. Keeping this in mind, ConceptGrid has been designed such that its ease of use and expressiveness lie between that of simple word matching approaches and complex approaches such as those that use complex machine learning algorithms.

The tutor-author develops the natural language understanding component for a tutor by breaking down the expected response to a question into specific concepts. The author then builds templates that describe a set of required concepts (that are meant to be a part of student's response to a question) and a set of incorrect concepts (that reflect incorrect understanding by the student). Every template is mapped to a single user-defined concept name. Since a student can describe a single concept in various forms, several templates can be used to describe different representations of a single concept, in order to recognize and

provide feedback to a wider range of student responses (both correct and incorrect). Thus, there is a one-to-many relationship between concepts and templates.

A template consists of one or more atomic checktypes, or check functions, that evaluate a student's input. These particular atomic checktypes are based on well-known algorithms and distance measures (Hamming, 1950; Levenshtein, 1966; Porter, 1980). The word "atomic" refers to the fact that these checktypes can be applied to a single word only. The set of atomic checktypes have been described in Table 4.1.

Apart from these atomic checktypes, we have two more checktypes that help make the template more expressive: Any(n_1, n_2) and Not(n, 'direction', word_list). The checktype "Any" matches any sequence of words that is at least n_1 words long and at most n_2 words. It helps account for words that are not explicitly accounted for using the other checktypes. The "Not" checktype takes care of negation. It makes sure that the n words appearing to the left or right (specified by 'direction') of the word following the checktype do not match the words mentioned in "word_list".

The checktypes Synonym and Stemmer can be nested within other atomic checktypes to make them more powerful. Levenshtein(Synonym('interface'),1), for example, captures the idea that any synonym of the word "interface" is fine, even if it has a spelling mistake.

When the student misses out on a subset of the required concepts, or mentions a subset of incorrect concepts, customized feedback can be given that points out the issue.

Table 4.1 – Atomic checktypes used in designing a template.

Checktype	Description
Exact(word_list)	Returns true if a literal character-by-character word match with any of the words in word_list is found
Almost(word_list)	Returns true if a literal match, after ignoring vowels, with any of the words in word_list is found
Levenshtein(n, word_list)	Returns true if the least Levenshtein distance between a word in word_list and matched word is <= n
Hamming(n, word_list)	Returns true if the least Hamming distance between a word in word_list and matched word is <= n
Soundex(word_list)	Returns true if a Soundex match with any of the words in word_list is found
Synonym(word_list)	Returns true if an exact match with any of the words in word_list or its synonyms from WordNet (Fellbaum, 1998) is found
Stemmer(word_list)	Returns true if a literal match with the stem of the matched word, with any of the words in word_list is found (uses Porter Stemmer)

4.4 The ConceptGrid Interface

The web-based interface is designed to allow the tutor-author to create templates that describe both required and incorrect concepts, and mention the customized just-in-time feedback that needs to be given to the students.

To simplify the process of constructing templates, we have a lattice-style table-driven interface for entering the template's checktypes and the corresponding parameters (Figure 4.1). A new template is created either by entering the dimensions of the table or by entering a sample response, from which a table is created and initialized. The table consists of a sequence of multi-level drop-down menus that represent the checktypes. The multiple levels help the author nest different checktypes. Each drop-down menu is associated with a specific number of textboxes that store the parameters associated with it. Each drop-down menu has several textboxes below it that store the contents of the parameter "word_list" associated with the corresponding checktype. The contingent approach of having the parameters dependent on the specific checktype provides a mild form of just-in-time authoring help. The user can navigate through the table just like a numerical spreadsheet and add or delete new rows and columns.

There are two sets of templates; the first describes required concepts and the second describes incorrect ones. Multiple templates can be mapped onto a single concept. Consider the following question in a statistics problem: "Based on your results, what do you conclude about the conditions of the music?" Let us assume that the correct answer to the question is "Reject the null hypothesis. There is a significant difference in memory recall between the rock music and no music conditions."

Some of the concepts that can be defined for the sample response mentioned above are described in Table 4.2.

The tutor-author then can design a ternary truth table called the Feedback Table (Figure 4.2) where he or she can enter the feedback that is to be given to the students, based on the truth values of the concepts: true – concept present (green check), false – concept absent (red X), or don't care (yellow dash). The author enters the values of the truth table through tri-state checkboxes. Feedback can be entered for both the absence of required concepts and presence of incorrect ones.

The Feedback Table helps provide feedback in a simple manner for seemingly complicated issues, such as an inconsistent statement (the last row of the Feedback Table in Figure 4.2) in the example discussed. Through its GUI, the Feedback Table achieves the computational equivalence of providing customized feedback by checking for truth conditions of the concepts using relational operators and conditional programming.

There is a provision to create user-defined variables that can be used while building checktypes or mentioning the feedback. This approach helps re-use templates for similar questions. The author can also enter a set of stop words that will be filtered out from the student's response prior to being processed.

Once the templates are designed and the feedback tables are filled, the author can test the templates with sample student responses. The output of the test mentions if the student's response has matched the required concepts. If a match is not found, then it displays the feedback associated with that response. It also displays the truth values of all the concepts defined by the author.

Table 4.2 – Examples of concepts. Conclusion-Correct and Conclusion-Incorrect look at the holistic response and the rest look at the sub-components of the response.

Concept Name	Description
Rejection-Correct	Matches responses that correctly mention whether the null hypothesis has to be rejected or not
Rejection-Incorrect	Matches responses that incorrectly mention whether the null hypothesis has to be rejected or not
Significance-Correct	Matches responses that correctly mention the significance of the result of the statistical test
Significance-Incorrect	Matches responses that incorrectly mention the significance of the result of the statistical test
Ind-Variable-Mention	Matches responses that explicitly mention the independent variable (e.g. type of music)
Dep-Variable-Mention	Matches responses that explicitly mention the dependent variable (e.g. memory recall)
Conclusion-Correct	Matches responses that have the correct conclusion of the statistical test
Conclusion-Incorrect	Matches responses that have the incorrect conclusion of the statistical test

4.5 Algorithm and Implementation

The implicit sequencing in the lattice approach means that the resulting complex checktypes are finite parsers. That is, progress through the lattice corresponds to progress left-to-right in processing the input.

The templates are represented internally as and-or trees. The algorithm involves a combination of recursion and memoization to efficiently process the input. Since the algorithm might need to backtrack many times, memoization helps speed up the processing by having function calls avoid repeating the calculation of results for previously processed inputs.

Our tool has been integrated with the Extensible Problem Specific Tutor (xPST) - an open source authoring tool that is intended to enable non-programmers to create ITSs on existing websites and software (Blessing, et al., 2009). Though xPST is a text-based authoring tool, its syntax is not very-code like. ConceptGrid has been customized to generate "code" that is compatible with xPST's syntax, based on the author's templates and Feedback Table, which can be then be inserted into any xPST file.

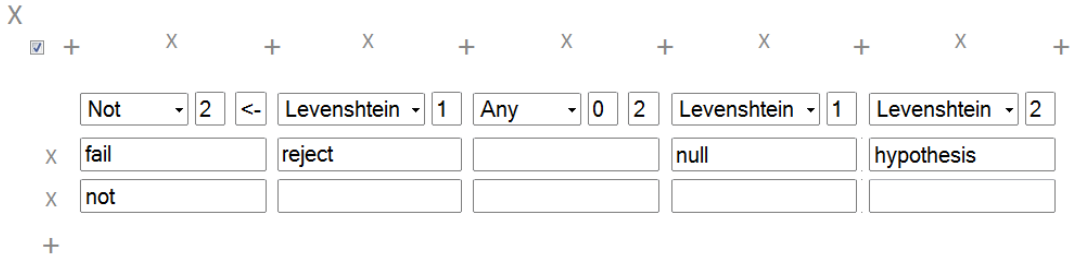


Figure 4.1 – The lattice-style table-driven interface of ConceptGrid. The template represents the concept "Rejection-Correct", described in Table 4.2.

	Rejection Correct	Rejection Incorrect	Significance Correct	Significance Incorrect	Conclusion Correct	Conclusion Incorrect	Feedback
x	▭	▭	▭	✓	▭	▭	It does not look like you have correctly mentioned the significance of the result.
x	✗	✗	▭	▭	▭	▭	You have not mentioned if the null hypothesis has to be accepted or rejected.
x	✓	▭	▭	✓	▭	▭	Your statements of rejection and significance are not consistent with each other.
+							

Figure 4.2 – Feedback Table

4.6 Results: The xSTAT Project

The research question for this paper is whether ConceptGrid could enable an instructor to create a tutor that would score students' free response answers as accurately as he or she manually did. At this point, the question is purely a feasibility issue: can it be done with the ConceptGrid tool? We tested this issue as a part of the xSTAT project at University of Tampa, dedicated to developing an intelligent homework helper for statistics students (Maass & Blessing, 2011).

For the xSTAT effort, six authors (3 instructors and 3 undergraduates) created multiple tutors each for college level statistics problems. The problems contained real-world scenarios with actual data, followed up by several questions for the student to answer. Each of the problems had a question at the end that asked students to enter the conclusion of the statistics test. To assess these problems, 6 were chosen out of the total pool of 74 and given to students as homework problems. All problems were solved on-line using a standard web browser. Half of the students received feedback on their answers via the xPST intelligent tutor (i.e., answers were marked as either correct or incorrect, and hints and just-in-time

messages were displayed), and half did not (i.e., these students simply filled out the web-based form). It is worth noting that these tutors were created without ConceptGrid, so that authors had to explicitly enter the "xPST code" that represents the templates without a graphical user interface. Also, in the absence of visualization through the Feedback Table, subsets of missing and incorrect concepts had to be explicitly mentioned. This non-lattice approach was not very usable by non-programmers. This difficulty motivated the creation of the ConceptGrid lattice approach, which is computationally equivalent and designed to be much more usable by non-programmers.

In all, 41 students solved a total of 233 instances of the six problems across the homework. We built a corpus after collecting all student responses to the end question (both those with tutoring and without). The corpus had 554 unique responses to this final conclusion question across the six homework problems. This corpus includes multiple incorrect responses by the same student to the same problem if they were in the tutored condition.

These responses were manually scored by an instructor and a teaching assistant based on the presence or absence of the concepts defined in Table 2. Then, one of the authors attempted to use ConceptGrid to produce templates that would score the 554 responses similar to those manual scores. The result of that work contained a total of 10 templates common to all six problems, to cover all concepts, except “Ind-Variable-Mention” and “Dep-Variable-Mention”. The concepts “Ind-Variable-Mention” and “Dep-Variable-Mention” required a template each that was unique to each of the six problems. In all, there were 22 templates across all six problems. A template, on an average consisted of 4 checktypes.

Since the manner in which a template tries to match a student's response – a sequence of words is comparable to the manner in which a regular expression matches a string, it might seem that the results have a lot of false negatives. But, since this approach tries to "understand" responses by looking for smaller concepts and key phrases with the help of checktypes rather than literal word matching, it is much more expressive. The results in Table 3, where we report the number of false positives, false negatives and the accuracy, the fraction of correct classifications, confirm this observation. The last column shows the values of Cohen's Unweighted Kappa (Cohen, 1960), which is a measure of the degree to which the human grader and ConceptGrid concur in their respective classifications.

Table 4.3 – Results of the classification of 554 student responses using ConceptGrid

Concept	False Positives	False Negatives	Accuracy	Kappa
Rejection-Correct	1	34	0.9368	0.8657
Rejection-Incorrect	6	5	0.9801	0.9217
Significance-Correct	1	7	0.9856	0.9662
Significance-Incorrect	12	1	0.9765	0.8890
Ind-Variable-Mention	1	3	0.9928	0.9853
Dep-Variable-Mention	4	3	0.9874	0.9733
Conclusion-Correct	0	24	0.9567	0.8614
Conclusion-Incorrect	6	0	0.9892	0.9727

4.7 Conclusions and Future Work

We have described ConceptGrid, a tool that is intended to help non-programmers develop ITSs that perform natural language processing. It has been integrated into an ITS authoring tool called xPST. We tested it as a part of the xSTAT project and were able to approach the accuracy of human instructors in scoring student responses.

We would like to conduct an empirical evaluation study that helps demonstrate that the ConceptGrid tool, a part of xPST, is actually feasible for non-programmers to use on a variety of tasks, as we have done for xPST's core authoring tool (S. Gilbert, et al., 2009). The study will also help provide an insight into the time required by a tutor-author to develop templates for particular question types.

Currently, ConceptGrid does not support a dialogue between the student and tutor as do CIRCSIM (Evens et al., 2001) and AutoTutor (Graesser, et al., 2005). It only evaluates student responses and gives just-in-time feedback. To support more extensive knowledge-construction dialogues, ConceptGrid responses would need to provide information required by the dialogue manager.

Our current approach is non-structural, i.e., it is focused on words and numerical analysis, rather than grammar and logic. The advantage with this approach is that it is simple for non-programmers to use, and is very effective in domains such as statistics where the student responses are expected to follow a general pattern. In addition, the ConceptGrid approach is domain-independent, one of its biggest advantages.

ConceptGrid could be extended to be structural as well, but that achievement might come at the cost of usability by non-programmers. To include structural matching, either the templates could nest by invoking other templates, or the atomic checktypes could include

some checktypes that invoked structural matching. For nested concepts, we could define a concept and then use it within more complex concepts, in the following manner.

GreaterThan(X,Y) = X – "bigger" or "more" or "greater" – "than" – Y

WellFormedConclusion = GreaterThan("weight of the log", "weight of the twig")

This way, the framework can be extended to more powerful natural language processing using a similar approach to the processing that context-free grammars allow. Alternately, the set of ConceptGrid atomic checktypes could be extended to enable structurally-oriented checktypes that would match a nonterminal from a context-free grammar, such as an NP with "twig" as the head in a syntactically oriented grammar, or match the semantics of a section of the utterance.

4.8 Acknowledgement

This work is done with the support of the U.S. Air Force Office of Scientific Research.

4.9 Appendix

ConceptGrid has been integrated within the web-based authoring Tool (WAT) described in Chapter 3. The WAT now supports the management of ConceptGrids by users. Also, the WAT logs events that occur while an author develops ConceptGrids. This data will be useful for the purpose of data mining in research studies.

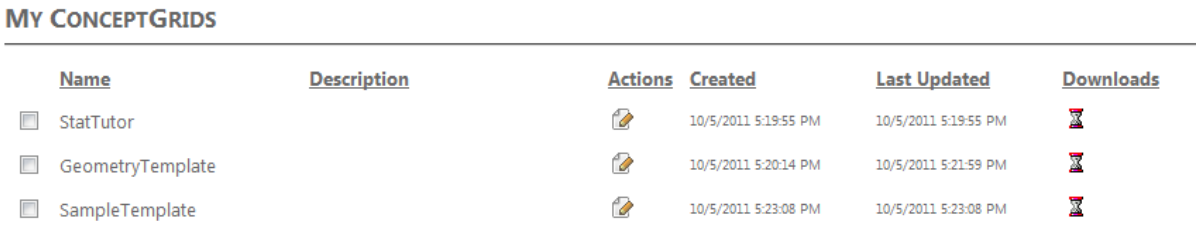


Figure 4.3 – ConceptGrid Management on WAT

CHAPTER 5. AUTHORING INTELLIGENT TUTORING SYSTEMS FOR 3D GAME ENVIRONMENTS

Research described in this chapter was published in the Authoring Simulation and Game-based Intelligent Tutoring Workshop at the Fifteenth Conference on Artificial Intelligence in Education, Auckland. (2011).

Shrenik Devasani, Stephen Gilbert, Suhas Shetty, Nandhini Ramaswamy, Stephen Blessing

5.1 Abstract

We describe the design of an authoring tool that is intended to help non-programmers develop game-based intelligent tutoring systems. The tutor-building approach involves the creation of states, both atomic and complex, that help model physical and cognitive states respectively. The tutor-author specifies the parameters associated with each entity in the scenario for each state. The resulting tutor dynamically updates the learner model and automatically assesses the performance of the learner and provides customized feedback.

5.2 Introduction

Game-based intelligent tutoring uses the synergy that exists between digital games and intelligent tutoring systems (ITSs) by combining the high degree of exploration and autonomy that characterizes digital games and the dynamic adaptive instruction supported by ITSs (Thomas & Young, 2009). Recent efforts that use such an approach include an environment used by middle schoolers to learn science concepts (Rowe, Shores, Mott, &

Lester, 2010) and another used by soldiers to learn language and culture issues (Johnson, 2009).

The last few decades have seen a change in the nature of teaching in modern universities (Laurillard, 1993). Research has shown that active engagement in the learning process by students leads to better learning, higher retention of information and development of skills such as logical thinking and independent decision making. Tutoring using games might provide these benefits as they manage to maintain the user's attention with a feeling of immersion within a simulated environment. Games are a form of reward- based learning and encourage active learning. There has been an increased interest by learning scientists in incorporating games and gaming principles into teaching and learning (Kirriemuir & McFarlane, 2004).

ITS researchers have begun exploring how games can be used in intelligent tutors. In some domains, games may be the only possible means of simulating and practicing real world problems. Simulation games are being used extensively in the military for teaching pilots to fly as well as for training on combat scenarios that would otherwise be extremely dangerous and expensive to train in the field (R.H. Stottler & Vinkavich, 2000). With the use of simulated environments, aggressive game play can help players relax and balance their aggression (Bensley & Van Eenwyk, 2001).

Several game-based ITSs have been developed and customized for military training (W. R. Murray, 2006; E. Remolina, S. Ramachandran, R. Stottler, & W. R. Howse, 2004). Yet, authoring tools that allow non-programmers to develop them are uncommon. Authoring of game-based tutors is challenging due to the inherent domain complexity, the dynamic nature of the environment, the different kinds of feedback required, and the interactions

between various non-player entities in the game. In this current work we explore an addition to a tutoring architecture that we have created to enable that architecture to not only tutor in such 3D environments but also to allow for the easy authoring of scenarios and instruction within those environments by non-programmers.

5.3 Previous Work

The Extensible Problem Specific Tutor (xPST) is an open source ITS authoring tool that supports tutoring within game-engine based synthetic environments (S. B. Gilbert, et al., 2011). It uses a simple modeling language that promotes authoring by non-programmers. The tutor-author must list the sequence of steps to be performed by the trainee and then describe the feedback associated with each step. We conducted a study that demonstrated that users with minimal programming experience can use xPST to create basic tutors for 3D game environments.

The drawback of this prior approach is that the model created by the tutor-author consisted of goalnodes, or steps to be performed by the learner, rather than states in the game. Though the use of goalnodes makes the tutor-building process very simple and usable by non-programmers, we found that goalnodes by themselves do not have sufficient power to encapsulate all information required to model a dynamic environment. Due to the existence of non-player entities and events that can happen without the trainee's knowledge, the game's state can change even without any action being performed by the trainee. The goal of the present work is to eliminate this weakness with the prior approach and to design a method for creating a tutor within 3D environments that allows for the full range of possibilities that exist within such environments, but is still not overly burdensome on the tutor-author.

5.4 Design

In the present work, we are designing a system that will support soldiers practicing realistic squad-level scenarios. One such scenario that we will use as an example here, is approaching and entering a building that may contain a hostage, an insurgent, and a bomb that needs to be defused.

We have conceptualized the tutoring model associated with such a scenario as composed of a collection of author-defined states. A state can be atomic or complex. An atomic state represents the physical state of the real world (game scenario with all its entities) at a given instance of time. It is described by the values of the properties of the entities in the scenario (Figure 5.1). Every property can take a special value called "don't care" (DC). When a property is assigned this value, it does not come into play when two states are being compared. A complex state is described as a pattern of one or more atomic states, defined using regular expressions. It can be used to represent a cognitive state and also helps model events that happen over time. For example, a trainee enters the complex state "S1 (S2|S3)* S4" when he enters S1, moves to S2 or S3, zero or more times and then finally enters S4. In the example scenario, this might correspond to starting at a location outside the building, approaching the building along a low wall, and then finally entering the building. Although the requirement of regular expressions in the modeling of a complex state makes the system less usable by non-programmers, the power that they offer justifies the trade-off.

Every state (both atomic and complex) is classified as one of these five types: "Start State" (represents the state when the scenario is loaded), "Goal State" (represents a state where the objective of the scenario is achieved), "Failed State" (represents a state where the game is lost, and there is no way back for the learner), "IntermediateCorrect" (represents a

correct intermediate state) and “IntermediateIncorrect” (represents an incorrect intermediate state).

In addition to the five states described above, a tutor-author can define a “ResponseState” that can be tied to any of the five kinds of states. When the learner enters a particular state, the Response State that is tied to it describes the activities that will be performed automatically by non-player entities in the scenario. For example, when the learner approaches an enemy building, a ResponseState can send reinforcements by forcing existing insurgents to move to the entrance of the building, or spawn new insurgents.

There are three kinds of feedback associated with each state – hints, just-in-time messages and prompts. Hints are displayed when help is requested by the learner (Figure 2). Just-in-time messages are displayed when the learner makes a common mistake that the tutor recognizes. Prompts represent feedback that is neither requested nor based on incorrect events. They are displayed when the learner enters a particular state.

When the learner is in a complex state, he would also be in an atomic state. Also, it is possible for the current state of the game to match more than one complex state, when a larger complex state encapsulates smaller ones. For example, the complex state “(S1 S2)* S3” encapsulates the complex state “S2 S3”. Hence, the tutor-author is allowed to assign priorities to each state. Feedback associated with the state with a higher priority is presented first.

Time and speed are critical aspects in military training. The authoring tool allows the dynamic creation and updating of multiple counter variables, in order to track the time spent by the learner in performing specific activities. Counters can be created, started, paused or reset, as required, when the learner enters or leaves a particular state. The values of these

counters can be used while checking for specific conditions before giving just-in-time feedback. The values of these counters can also be used as pre-conditions for entering a state and post-conditions for leaving a state.

The tutor maintains a learner model by continuously tracking the skills of the learner. Based on the tasks involved in the scenario, the tutor is associated with a skill set, mapped to skill variables. Every state can be tied numerically to one or more skill variables. When the learner enters a particular state or performs an action, the values of the corresponding skill variables are updated accordingly. For example, a skill variable called "Accuracy" can be defined that keeps track of the percentage of accurate shots fired by the learner. Also, the number of hints or just-in-time error messages the learner receives in a particular state can affect the values of the skill variables.

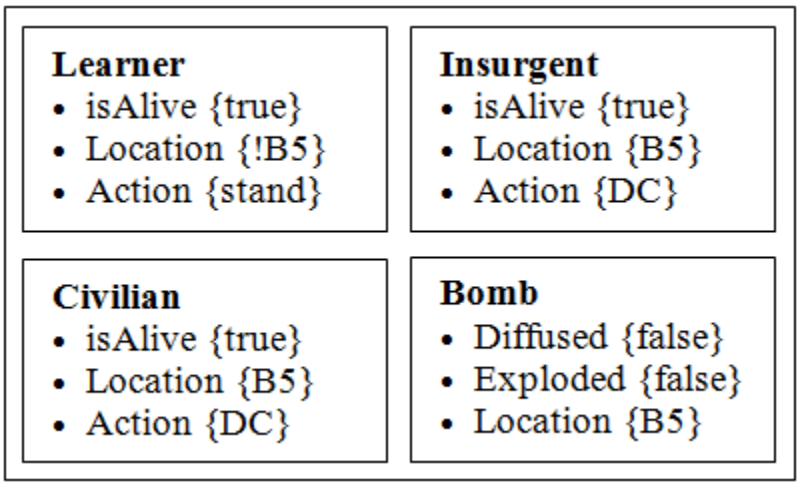


Figure 5.1 – An atomic state consisting of a learner, insurgent, civilian and a bomb.

Figure 5.2 – An example of a hint presented to the learner: Diffuse the bomb.

5.4.1 Simulation Engine: Virtual Battlespace 2

We are using Virtual Battlespace 2 (VBS2), a commercial-off-the-shelf, three-dimensional military simulator, based on the game engine Real Virtuality, to develop simulations. VBS2 offers realistic battlefield simulations and delivers a synthetic environment for training teams in arms operations and emergency response procedures. It provides user-defined mission scenarios and real-time scenario management facilities. A learner views the virtual environment from the first-person perspective and can move, interact, and operate just as he or she would, in real life.

Similar to the architecture described in our previous work at creating games-based ITSs [10], we have a Tutor Engine, a Listener module and a Presenter module. The Listener module keeps track of the current state of the game by querying the parameters of all the

entities in the scenario. This information is passed on to the Tutor Engine over the network, which matches the current state with author-defined states. The Tutor Engine then sends the appropriate tutoring feedback to the Listener module, which is then presented to the learner through the Presenter module.

Using the simulation engine, scenarios can be pre-defined and the tutor-author can concentrate solely on the task of tutor development and tutoring strategies. The biggest advantage of this architecture is that it can be extended to other game engines such as Unity and BigWorld, by modifying just the Listener and Presenter modules, and retaining the core Tutor Engine.

5.5 Tutor Authoring Process

The first step is to create the scenario using the VBS2 Mission Editor, which is a component of VBS2. This involves placing entities in a scenario and giving them unique IDs. The names of the properties of each entity in the scenario are defined, along with the possible values that they can take. Next, locations in the scenario that would be of interest from a tutoring point of view, such as buildings, are defined.

Once the scenario is ready, the tutor-author can start defining states and design the tutoring strategy. Consider the simple example scenario, “ClearBuilding”, consisting of the learner located outside Building 5 (B5) and an insurgent, a civilian and a bomb, all located in B5. In order to successfully complete the mission, the learner must do the following:

1. Run towards the nearest wall
2. Approach B5, staying close to the wall at all times
    a. Crouch, if near a window
    b. Enter the door
3. Kill the insurgent
4. Evacuate the building

5. Defuse the bomb

Table 5.1 describes some of the states that could be defined while designing the tutoring model for a tutor for the above scenario.

Table 5.1 – Possible atomic states for the scenario "ClearBuilding"

Entity	Property	Start	NearWindow	InsurgentDead	Diffused	PlayerDead
	StateType	Start	Intermediate	Intermediate	Goal	Failed
Learner	IsAlive	true	true	true	true	false
	Location	!B5	window	DC	DC	DC
	Action	DC	DC	DC	DC	DC
Insurgent	IsAlive	true	true	false	false	DC
	Location	B5	DC	DC	DC	DC
	Action	DC	DC	DC	DC	DC
Civilian	IsAlive	true	true	true	true	DC
	Location	B5	B5	DC	!B5	DC
	Action	DC	DC	DC	DC	DC
Bomb	Diffused	false	false	false	true	DC
	Exploded	false	false	false	false	DC
	Location	B5	DC	DC	DC	DC

Apart from the atomic states defined in Table 5.1, we can define a complex state "LearnerConfused", represented by the regular expression "(Start NearWindow){3, }". This state represents the learner being in a confused state of mind in which he continuously

switches between the states Start and NearWindow, three or more times. Here, the complex state “LearnerConfused” can be assigned a higher priority than the atomic state “NearWindow”.

When the tutor is deployed, the tutor engine continuously queries the parameters of all the entities in the scenario using the unique IDs given to them during the scenario building process. The engine tries to match the current state in the game with an author-defined atomic state. The frequency at which the current state is updated depends on the number of entities in the scenario and the number of states defined by the tutor-author. The tutor engine maintains a stack that stores all the atomic states visited by the learner in last in, first out order, called “History”. When it observes a state transition, it adds the current state to the History. Though the engine continuously updates the current state of the game, it is added to the top of the stack only if it notices a state transition. The History helps recognize complex states that the learner might be in, by finding matches between the author-defined complex states (which are regular expressions) and the components of the History that end with the stack top. The time spent by the learner in a state is also recorded.

Figure 5.3 – Just-in-time feedback in the state "NearWindow", when the learner fails to crouch down: You must crouch down when near a window.

6.6 Tutor Authoring Process

We have described the design of an authoring tool that helps a tutor-author think intuitively while creating a tutor for a synthetic environment. The approach helps model complex tutoring strategies, especially when several non-player entities and objects exist in the scenario. It is interesting to note that the tutor-author need not define state transitions (unlike finite-state automata), since the current state of the game is continuously updated and matched with author-defined states. The ability of a tutor-author to define a property value of an entity as a "don't care" helps alleviate a possible explosion in the number of states, many

of which might be unobservable to both the learner and the tutor-author. Also, though the number of complex states in a realistic environment might reach infinite proportions, the tutor-author needs to define only a subset of the possible states for which he or she wishes to provide appropriate feedback.

Once the authoring tool is completely developed, we would like to conduct an empirical evaluation study, similar to the one described in (S. B. Gilbert, et al., 2011) that demonstrates that the tool is actually feasible for non-programmers to build tutors for game scenarios.

In (Devasani, Gilbert, et al., 2011), as a proof of concept, we described how xPST has been adapted to provide support for tutoring on web interfaces, based in part on real-time physiological data. This can be extended to game-based tutors as well, where stress is a major factor that affects a learner's performance. Variables associated physiological signals such as the electrocardiogram signal, the heart rate signal and the heart rate variability signal can be used as parameters while defining states in the cognitive model of a tutor.

We would like to further simplify the task of building tutors by employing a technique known as "programming by demonstration" (Nevill-Manning, 1993). The tutor-author can build states by demonstrating the task. At any given instance of time during the demonstration, the tutor-author can record the atomic state at that instance, rather than explicitly listing out the values of the parameters, and then specify the feedback associated with that state. The set of complex states can be defined explicitly at a later point in time. This approach will help save a considerable amount of time by giving the tutor-author a head start, compared to having to start from scratch.

When multiple learners are involved in a scenario performing collaborative tasks, a Trainer Observation System can prove to be very handy. It can provide the trainer with useful insight through auto-generated statistics and data visualization features that reflect the performance of the learners, both as a team, and as individuals.

We also plan to design a Learning Management System (LMS) that stores a collection of scenarios and numerical values of the skill variables required for those tasks. As a learner completes tasks in a scenario, the LMS can offer further scenarios according to the learner's strengths and weaknesses, as reflected in the learner model.

5.7 Appendix

Research described in this section was published in the Twentieth Conference on Behavior Representation in Modeling and Simulation, Sundance. (2011).

Stephen Gilbert, Shrenik Devasani, Sateesh Kodavali, Stephen Blessing

5.7.1 Background

The military has used simulated environments and computer assisted instruction since the 1950s. Most recently, warfighters participate in live, virtual, and constructive training missions, which means the some fighters are in a field or urban practice site with BB guns or laser rifles ("live"), some are in simulators of Humvees or aircraft cockpits ("virtual") and some are playing serious games with virtual environments against computer-generated enemies ("constructive"). (Gorman, 1991)

There has been much study of how much simulation fidelity is required for good training transfer (Andrews, Carroll, & Bell, 1995; Castner et al., 2007), and whether the

simulation can induce a sense of presence, or immersion (Dede, 2009; Lessiter, Freeman, Koegh, & Davidoff, 2001; Stanney, 2002). Clark Aldrich continues to be enthusiastic about their potential for training (Aldrich, 2009). In some domains, simulation games may be the only possible means of simulating and practicing real world problems. Simulations are being used extensively in the military for teaching pilots to fly as well as for training on combat scenarios that would otherwise be extremely dangerous and expensive to train in the field (R. H. Stottler, 2000).

However, we suggest that the future of effective training lies not in the fidelity of the synthetic environment and virtual entities, but in the relevance of the feedback received by the learner. The ideal training environment (see Figure 5.4) will offer real-time adaptive training that offers personalized feedback and scenario customization based not only on trainees' behavior in the scenario but also on their skill sets upon entering the training and on their personal profiles, e.g., information about their personalities and their physiological responsiveness to stress, both of which affect performance (Beilock, 2010). Adapting training based on both performance and the trainee's stress response is critical to accurate personalized feedback.

5.7.2 Personalized Adaptive Training

The idea of personalized adaptive training embodies two concepts from the learning sciences. The first is adaptive testing or tailored testing, used, for example, by the Educational Testing Service on standardized tests such as the GRE to offer students harder questions when they answer correctly and easier questions when they choose incorrectly

(Thissen & Mislevy, 2000). The key principle is that the system adapts itself based on the learner's performance.

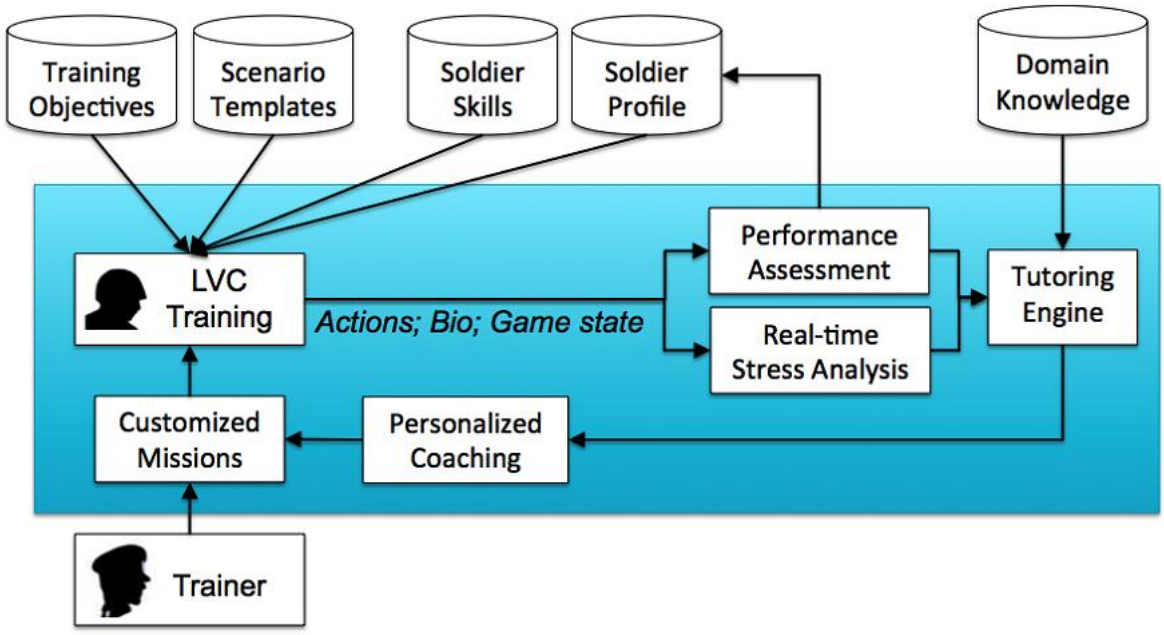


Figure 5.4 – The vision for the future of personalized adaptive training. The live, virtual and constructive training experience is determined by the training objectives, the soldier's previous skills, and the soldier's personality and stress resilience profile.

A system that not only tracks a learner's performance but also attempts to offer the learner useful feedback based on his or her mistakes is called an intelligent tutoring system (ITS). This is the second characteristic of the personalized adaptive training approach. ITSs have a cognitive model of an expert's domain knowledge that is used to identify patterns of learner behavior (model tracing) and give appropriate hints or corrective feedback. A cognitive model of algebra, for example, would know that students frequently forget the negative sign when solving equations, and would be able to say appropriately, "You might check to see if you've forgotten a negative sign somewhere…" Similarly, the Nintendo Wii

Fit game system could be considered a simplistic ITS, since it tracks your skills and offers feedback such as "You're leaning too far to the left."

ITSs have been demonstrated to have effective in a variety of school knowledge domains, such as algebra, geometry, and economics (Anderson, Conrad, & Corbett, 1989; K. R. Koedinger, J. R. Anderson, W. H. Hadley, & M. A. Mark, 1997; Ritter, Kulikowich, Lei, McGuire, & Morgan, 2007; VanLehn et al., 2005), resulting in up to a 30% improvement in standardized test scores (Franklin & Graesser, 1996) and learning time reductions (Corbett, 2001).

5.7.3 Challenge: Easily Creating an ITS for a Synthetic Environment

ITSs have also been created and customized for a variety of military synthetic environments (SEs) (W. R. Murray, 2006; E. Remolina, S. Ramachandran, R. H. Stottler, & W. R. Howse, 2004; R. H. Stottler, 2000; R. H. Stottler, Fu, Ramachandran, & Jackson, 2001) and Livak, et al created a more generalized tutoring approach using Unreal Tournament (Livak, Heffernan, & Mover, 2004). But what is still missing is 1) a more generic ITS authoring tool that could easily create ITSs for multiple SEs using modular abstraction from the SEs themselves, 2) an ITS protocol that leverages physiological data, and 3) an easy-to-use authoring tool for ITSs that could be used by military trainers with no programming experience.

CHAPTER 6. SUMMARY AND FUTURE WORK

This chapter summarizes the research work done related to the questions raised in 1.4, and proposes future work related to xPST and ConceptGrid.

The research related to the first question, "How does an intelligent tutoring system authoring tool paradigm and the complexity of a problem domain affect the tutor-authoring process by non-programmers in comparison with programmers?", involved an evaluation study of two authoring tools, CTAT (GUI-based) and xPST (text-based). Statistics and geometry were chosen as the two problem domains. A total of 16 participants were divided into eight groups formed from all possible combinations of programming level (programmer or non-programmer), authoring tool paradigm (text-based or GUI-based) and problem domain (statistics or geometry). Each participant authored a total of three tutors. The results showed that the GUI-based approach provided a lower bar for entry in comparison with the text-based approach. However, the difference in tutor-authoring time between the two approaches reduced as the tutor-authors gained experience using the respective authoring tools. After the tutor-authors gained experience building three tutors, the average time required in creating a tutor for a statistics problem for both CTAT (18.75 min) and xPST (19 min) were almost equal. However, the average time required in creating a tutor for the third geometry problem using xPST (52 min) was much higher than the average time required using CTAT (18 min). Geometry problems involve multiple solution strategies. The results suggest that subtle ordering of steps, required in complex domains is more convenient with a GUI-based authoring tool.

We addressed the second research questions, “Can an authoring tool that uses a GUI to facilitate use by non-programmers enable the creation of a tutor that can accurately evaluate written textual responses as a human instructor would manually do?” by developing a domain independent authoring tool, ConceptGrid, which is uses a lattice-style table driven interface that is intended to facilitate use by non-programmers. The approach involves the use of the interface to build templates that describe a set of required concepts that are meant to be a part of a student’s response to a question, and a set of incorrect concepts that reflect incorrect understanding by the student. We tested ConceptGrid by having a tutor-author use ConceptGrid to produce 22 templates that were meant to score responses to six open ended questions about conclusions to statistical tests. When these templates were tested against a corpus of 554 unique responses to the six questions, they approached the accuracy of a human instructor who manually graded the responses.

It will be interesting to design and evaluate a hybrid authoring tool that exploits the synergy between the graphical user interface-based paradigm and the text-based paradigm by having a “design” tab for the purpose of tutor development through a GUI and a “source” tab that helps edit a tutor directly through code, like in most IDEs.

We plan to do an empirical evaluation study of ConceptGrid.by having non-programmer participants develop ConceptGrids for open-ended questions. Such a study will help provide insight into ConceptGrid’s usability by non-programmers.

APPENDIX A. EXTENSIONS MADE TO XPST

This appendix describes the extensions that have been made to xPST in order to offer tutor-authors more power and also to improve the tutoring experience of students. For a detailed documentation of xPST and its syntax, please visit http://code.google.com/p/xpst/.

A.1 Extensions to xPST's Language

xPST's language has been made more expressive with the introduction of new checktypes, or evaluation functions, that are used by a tutor-author to check the validity of students' answers. The previous set of checktypes was not powerful enough to build tutors for problems in domains such as geometry and statistics.

The new checktypes incorporated into the xPST framework have been described below:

- Abs("a") - Checks if the absolute value of the entered answer equals "a". e.g.: JIT {v == Abs("1")}: "Cannot be +/-1";
- Round("a") - Checks if the rounded entered answer equals "a". e.g.: answer: Round("a");
- Floor("a") - Checks if the floor of the entered answer equals "a". e.g.: answer: Floor("a");
- Ceil("a") - Checks if the ceiling of the entered answer equals "a". e.g.: answer: Ceil("a");

- IsRange("[]","a","b") - Checks if the entered answer is inside the interval ["a","b"]. The other variants of this checktype are Range("()", "a", "b"), Range("[)", "a", "b") and Range("(]", "a", "b").

  e.g.: answer: IsRange("[)","a","b");

- IsNotRange("[]","a","b") - Checks if the entered answer is outside the interval ["a","b"]. The other variants of this checktype are Range("()", "a", "b"), Range("[)", "a", "b") and Range("(]", "a", "b").

  e.g.: answer: IsNotRange("()","a","b");

- IsAbsRange("[]","a","b") - Checks if the absolute value of the entered answer is inside the interval ["a","b"]. The other variants of this checktype are IsAbsRange("()", "a", "b"), IsAbsRange("[)", "a", "b") and IsAbsRange("(]", "a", "b").

  e.g.: answer: IsRange("[)","a","b");

- IsNotAbsRange("[]","a","b") - Checks if the absolute value of the entered answer is outside the interval ["a","b"]. The other variants of this checktype are IsNotAbsRange ("()", "a", "b"), IsNotAbsRange("[)", "a", "b") and IsNotAbsRange("(]", "a", "b").

  e.g.: answer: IsNotRange("()","a","b");

Adapting xPST to support tutoring on math problems, particularly statistics, required the creation of several functions used for answer-checking. These new checktypes were used in the development of xSTAT (Maass & Blessing, 2011), an intelligent homework helper for students solving college level statistics problems.

A.2 Tutoring with Physiological Data

xPST has been extended to provide support for tutoring based on physiological data. This extension has been developed as a proof of concept, for web interfaces. This can be extended to game-based tutors as well. An xPST tutor-author can offer customized just-in-time feedback based on the values of physiological signals related to stress and arousal. We modeled our physiological simulation based on a physiological monitoring device called FlexComp that returns the electrocardiogram signal (EKG), the heart-rate signal (the heart rate in beats per minute) and the heart rate variability signal (measures how the heart rate varies over time). When the values of these signals are combined with the value of the blood pressure, this data can be useful for identifying trainee stress, e.g. when approaching the end of the competition time of a time-based test. When a trainee is attempting to complete a scenario, tasks that cause high stress could be identified, and the trainee could be asked to practice more of such tasks. Also, trainees who are identified as high-responders to stress might be assigned to less stressful duties longer term.

This extension of xPST for tutoring with physiological data is still in its early stages. We have demonstrated the use of heart rate signals while tutoring on a timed web-based college statistics assignment, giving different kinds of just-in-time feedback depending on whether time is running out and depending on whether stress levels are higher when learners made mistakes. This extension work will continue for an ongoing project at Iowa State led by Nir Keren and Warren Franke analyzing the effect of stress on firefighters during immersive CAVE (Cave Automated Virtual Environment)-based learning.

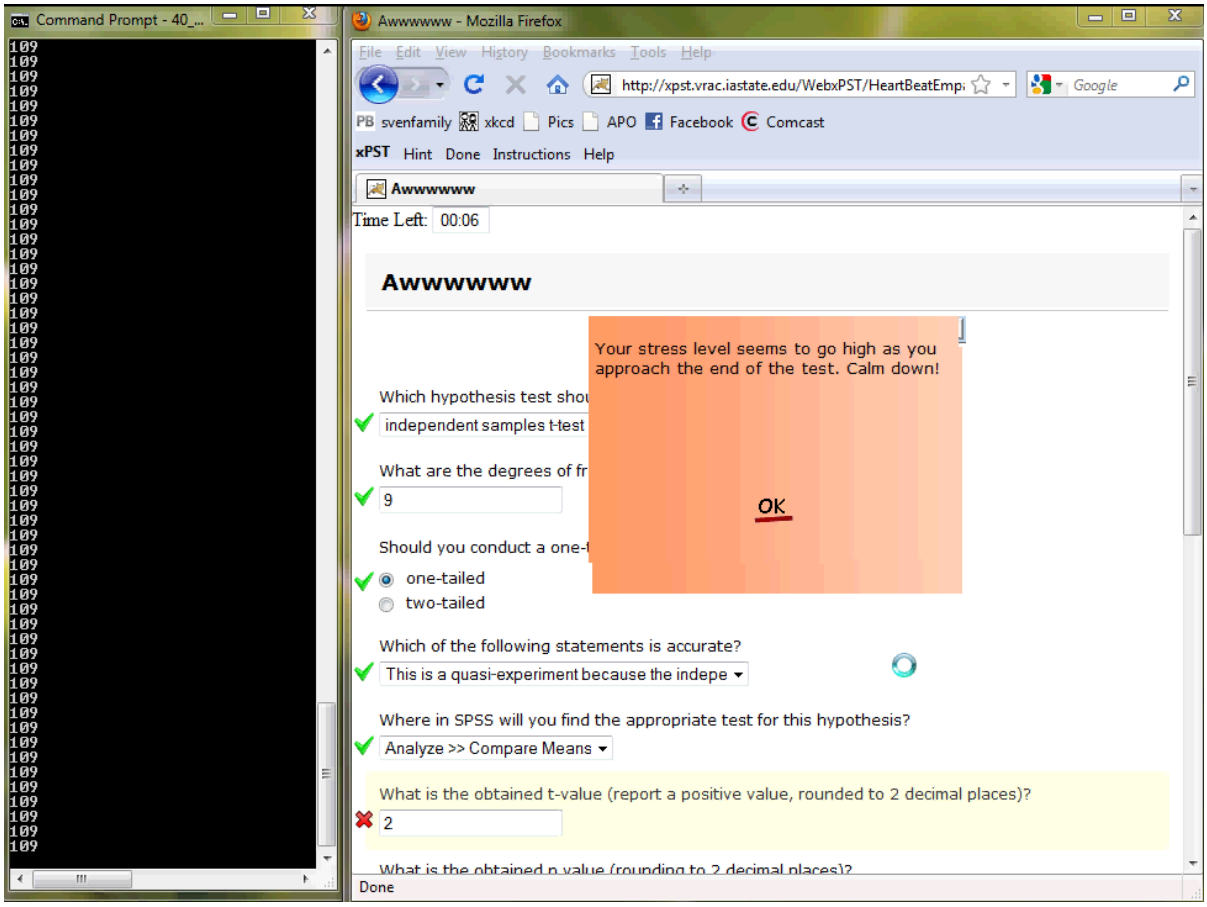


Figure A.1 – Screenshot of an xPST prototype giving feedback based on heart rate during a statistics problem. "Stress" in this prototype is used loosely.

A.3 Visual Feedback

Tutors built with the previous versions of xPST could not implicitly provide positive and negative feedback to the learners. The tutors built with xPST were incapable of providing positive feedback, and negative feedback was possible only if the tutor-author explicitly defined just-in-time error messages.

The current extension provides visual feedback, both positive and negative. When a subgoal (Blessing, et al., 2009) is completed, a green check mark is placed next to the widget

tied to the subgoal, and when an incorrect answer is provided to a goalnode, a red X-mark is (see Figure A.1) placed next to the widget tied to the goalnode.

APPENDIX B. EVALUATION OF TWO INTELLIGENT TUTORING SYSTEM AUTHORING TOOL PARADIGMS

This appendix contains the material relevant to the study described in Chapter 3

Volunteers needed for Research Study

We are conducting an evaluation study of 2 authoring tool paradigms that help non-programmers build Intelligent Tutoring Systems. Activities are to be done online, at your own convenience within a two-week period. Activities typically require approximately 6 hours.

Your participation is completely voluntary. All of the information you provide will strictly be kept confidential and reported in summary form only. No individual will be identified, nor will his name be attached to any data. At the end of the project, researchers will destroy any personal identifying information.

Compensation will be provided to participants in this study in the form of $40 and participation in a raffle for $149 (the price of an iPod Nano).

Limited programming experience is required, e.g. editing HTML or doing SPSS scripting. You must an adult, aged 18 or older. To volunteer, please contact Shrenik Devasani at shrenik@iastate.edu .

Figure B.1 – Advertisement for recruiting participants

1. What is/was your undergraduate major?

2. How many undergraduate programming courses have you taken?

3. On a scale of 1-5, to what extent would you describe yourself as technically proficient or a "techie?"
(1 = "not techie at all", 5 = "extremely techie")

4. Have you ever edited HTML or something similar, e.g. SPSS scripting or Outlook filter rules? Please answer "Yes" or "No". If "Yes", describe your experience in 1-2 sentences.

5. Do you have any programming experience? Please answer "Yes" or "No". If "Yes", describe your experience in 1-2 sentences.

6. Do you have any background in Geometry and Statistics? Please answer "Yes" or "No". If "Yes", describe your experience in 1-2 sentences.

Please forward this mail to anyone age 18 or older (outside ISU included) who might want to participate.

Figure B.2 – Pre-survey form

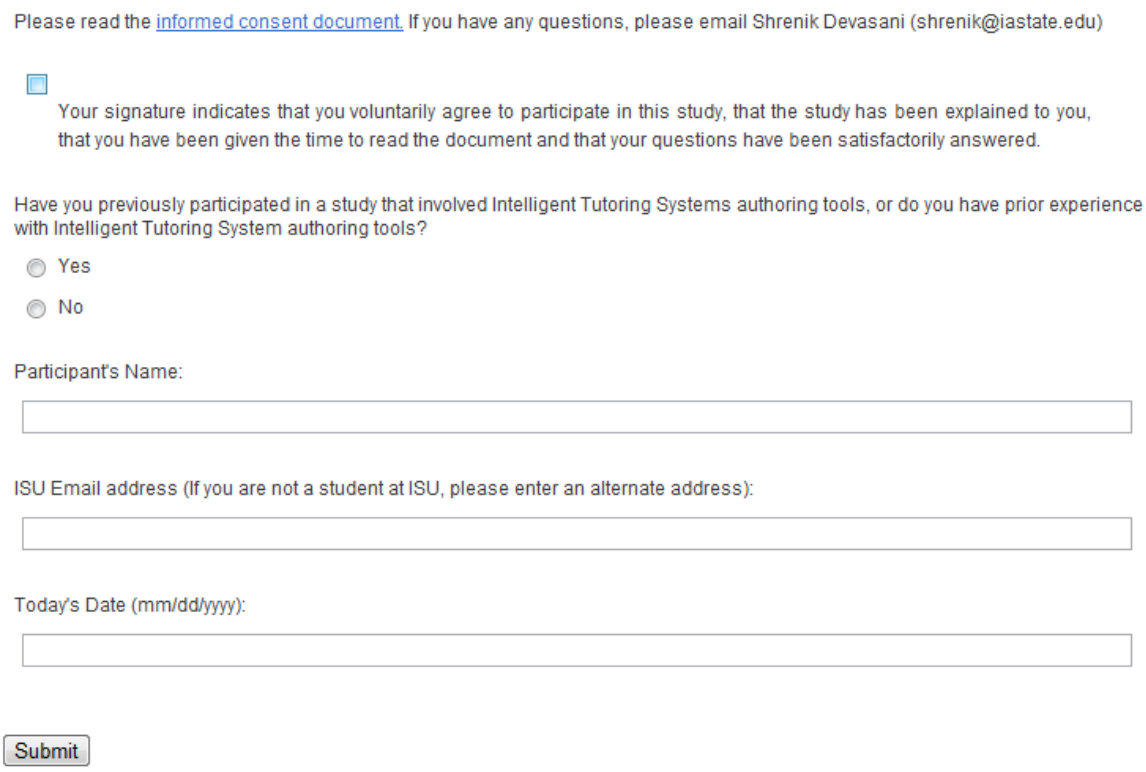


Figure B.3 – Informed consent document

Please enter your ISU Email address (If you are not a student at ISU, please enter an alternate address):

1. On a scale of 1-5, how would you rate the ease of use of the authoring tool you used? (5 – "Very simple to use", 1 – "Extremely complicated")

	Extremely complicated 1	2	3	4	Very simple to use 5
Ease of use	○	○	○	○	○

2. On a scale 1-5, how would you rate the power (flexibility and ability to customize the tutoring experience) of the authoring tool you used? (5- "Very powerful", 1 – "Not powerful at all")

	Not powerful at all 1	2	3	4	Very powerful 5
Power	○	○	○	○	○

3. Describe the top 2 strengths and top 2 weaknesses of the authoring tool you used.

4. How successful do you think the tutor you created will be, from a student's point of view?

5. On a scale of 1-5, how seriously did you take the task of creating tutors? (5 – "Very serious", 1 – "Not serious at all")

	Not serious at all 1	2	3	4	Very serious 5
Power	○	○	○	○	○

6. Any other comments?

Submit

Figure B.4 – Exit questionnaire

Virtual Reality Applications Center: Human Computer Interaction

Cognitive Modeling Authoring Study

Stephen Gilbert, Ph.D, Shrenik Devasani (IRB #10-547)

Introduction

An Intelligent Tutoring System (ITS) is a software application that that provides personalized instruction and customized feedback to students as he or she completes tasks within a problem domain such as statistics, medical diagnosis and even game play.

Creating ITSs from scratch is time consuming and requires expertise in several fields - artificial intelligence, cognitive psychology and computer science, to name a few. Several authoring tools have been developed that help users without a technical background build ITSs. The Extensible Problem Specific Tutor (xPST) is one such open source authoring tool that enables creation of ITSs on existing websites or software.

The goal in doing this study is to learn about the process of authoring cognitive models for problems in geometry using xPST.

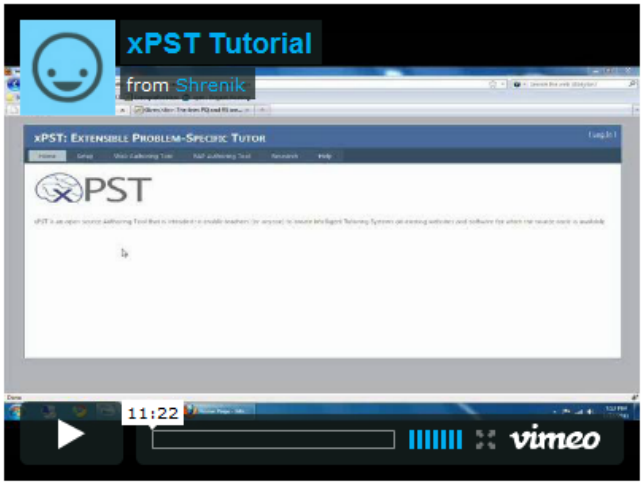
What Do You Need?

- Internet
- Mozilla Firefox 3.0 or above
- xPST plugin - download from here.

How it's Done?

The video tutorial below gives you a demo of the step-by-step procedure to be followed while creating a tutor using xPST, for a sample problem. (For better clarity, please right click on this link and save the video to your computer and play it from there.)

You can access the xPST file for the example shown in the tutorial here.



Please read these instructions about writing an xPST file.

Figure B.5 – Study webpage for xPST tutor-authors (1/2)

Your Deliverables

In this study, you will create tutors for 3 geometry homework problems as if you are a geometry teacher preparing homework problems for your students. The table below gives you the URLs of the 3 problem interfaces, and the instructions that include the solutions to the problems and the types of feedback your tutors should give for each problem.

The tutor will be monitoring each step of a problem. For each step in a problem, your tutor must provide exactly 1 hint. Also, for each tutor overall, there will be 1 message that gives feedback for a specific error that a student might make (Please make sure you read the instructions about writing an xPST file. It explains how to provide error specific feedback).

A tutor is said to be complete when it runs without mistakes and provides appropriate hints and error-specific feedback.

For each problem, there is a "problem interface" that you will use. Here are the links to the interfaces of each of the 3 problems (you will need these URLs while building the tutors) and the instructions for the problems, as the instructor:

Problem	Instructions
Problem 1	Problem 1 Instructions
Problem 2	Problem 2 Instructions
Problem 3	Problem 3 Instructions

You Do It!

Please plan about 6 hours for this activity.

- Login to the Web Authoring Tool using the username and password provided to you.
- Enter a name for your tutor without spaces and click 'Create New Tutor'.
- In the 'Edit Tutor' page, enter the URL of the problem interface. The URLs are available in the table in the 'Your Deliverables' section.
- Give a brief description of your tutor.
- Edit the content of your xPST file.
- Click on 'Save & Run' to save and test your tutor.
- Repeat the previous two steps until your tutor runs as expected, without any errors.
- To help with accurate data about how people use the Web Authoring Tool, **please logout** when you are not working on the task and go idle for more than 2 minutes.

Guidlines and Suggestions

It has been found that almost 99% of the errors that happen in your initial attempts will be syntactical errors. So whenever you face an error first please check the syntax in your three sections of the xpst file. Although the system points out your errors, it is not yet 100% foolproof.

- Your tutor name must not contain any spaces.
- In your xPST file, do not use spaces in the step names that you define.
- In your xPST file, make sure your that the opening and closing parantheses and curly brackets match.
- You can start this activity and then stop and come back to it later, even on a different computer; just be sure to click 'Save'.
- To help with accurate data about how people use the Web Authoring Tool, **please logout** when you are not working on the task and go idle for more than 2 minutes.

Tech Support and Questions

Please direct all your questions (either technical or regarding the study) to Shrenik Devasani (shrenik@iastate.edu) at any point in the study.

Done with Tasks?

The time for working on the tasks will be two weeks.

Once you feel you are done with the tasks you can email Shrenik (shrenik@iastate.edu) informing him about your completion of the tasks. Shrenik will make sure he can see your models on the server and will send you an email with instructions on taking the end survey and receiving your payment or research credits.

Figure B.6 – Study webpage for xPST tutor-authors (2/2)

Virtual Reality Applications Center: Human Computer Interaction

Cognitive Modeling Authoring Study

Stephen Gilbert, Ph.D, Shrenik Devasani (IRB #10-547)

Introduction

An Intelligent Tutoring System (ITS) is a software application that that provides personalized instruction and customized feedback to students as he or she completes tasks within a problem domain such as statistics, medical diagnosis and even game play.

Creating ITSs from scratch is time consuming and requires expertise in several fields - artificial intelligence, cognitive psychology and computer science, to name a few. Several authoring tools have been developed that help users without a technical background build ITSs. CTAT is one such open source authoring tool that enables creation of ITSs on Flash interfaces, among others.

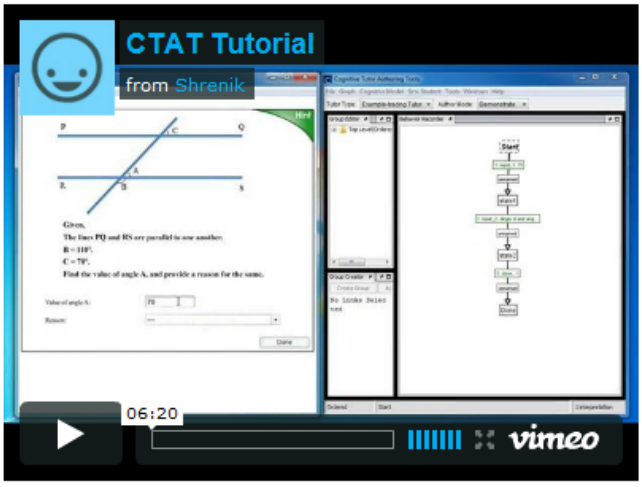
The goal in doing this study is to learn about the process of authoring cognitive models for problems in geometry using CTAT.

What Do You Need?

- Internet
- Adobe Flash Player 10 (already installed on the machine given to you)
- CTAT 2.10 (already installed on the machine given to you)

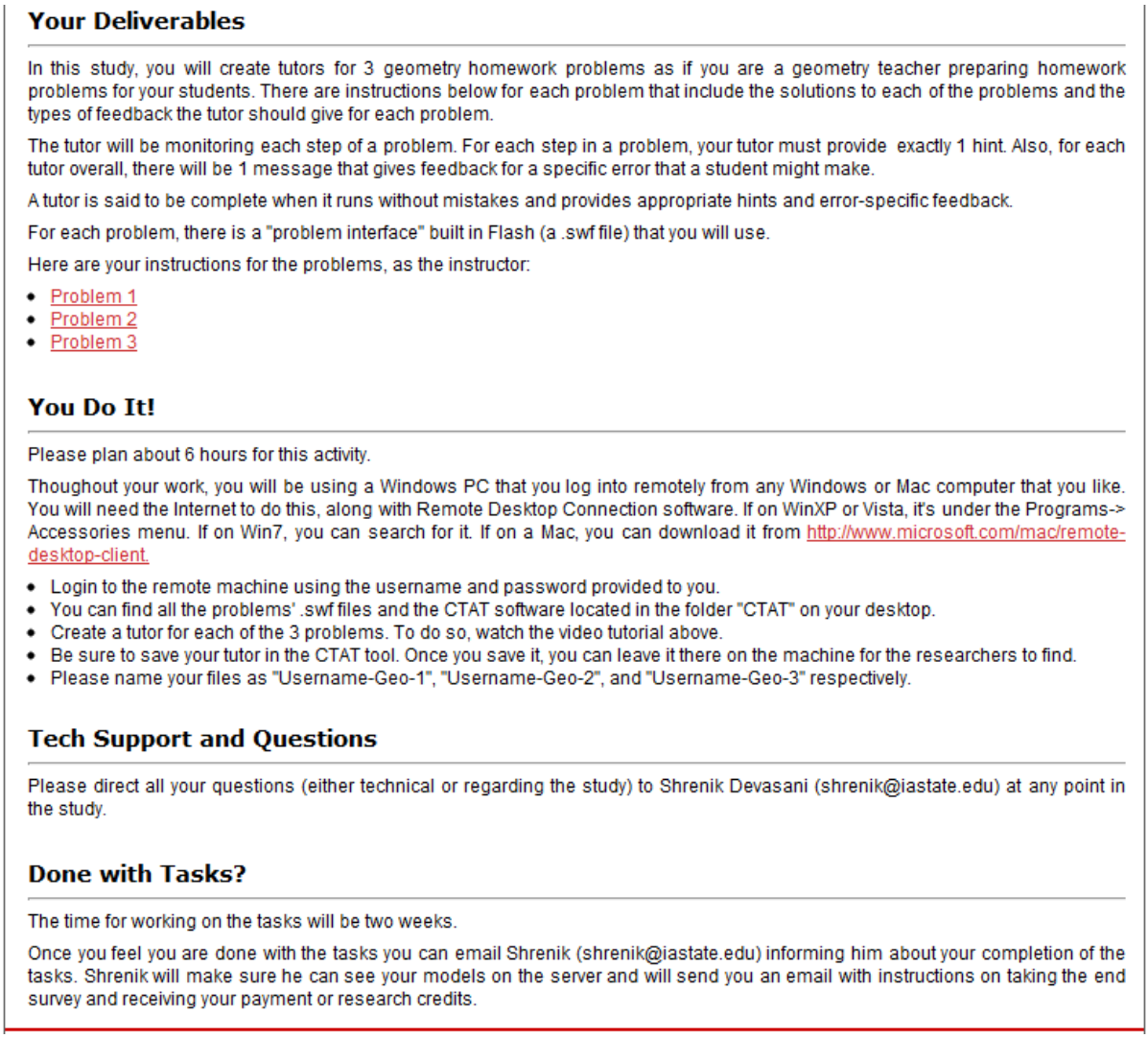
How it's Done?

The video tutorial below gives you a demo of the step-by-step procedure to be followed while creating a tutor using CTAT, for a sample problem. (For better clarity, please right click on this link and save the video to your computer and play it from there.)



Please read these instructions about creating a tutor with CTAT.

Figure B.7 – Study webpage for CTAT tutor-authors (1/2)



Your Deliverables

In this study, you will create tutors for 3 geometry homework problems as if you are a geometry teacher preparing homework problems for your students. There are instructions below for each problem that include the solutions to each of the problems and the types of feedback the tutor should give for each problem.

The tutor will be monitoring each step of a problem. For each step in a problem, your tutor must provide exactly 1 hint. Also, for each tutor overall, there will be 1 message that gives feedback for a specific error that a student might make.

A tutor is said to be complete when it runs without mistakes and provides appropriate hints and error-specific feedback.

For each problem, there is a "problem interface" built in Flash (a .swf file) that you will use.

Here are your instructions for the problems, as the instructor:

- Problem 1
- Problem 2
- Problem 3

You Do It!

Please plan about 6 hours for this activity.

Thoughout your work, you will be using a Windows PC that you log into remotely from any Windows or Mac computer that you like. You will need the Internet to do this, along with Remote Desktop Connection software. If on WinXP or Vista, it's under the Programs-> Accessories menu. If on Win7, you can search for it. If on a Mac, you can download it from http://www.microsoft.com/mac/remote-desktop-client.

- Login to the remote machine using the username and password provided to you.
- You can find all the problems' .swf files and the CTAT software located in the folder "CTAT" on your desktop.
- Create a tutor for each of the 3 problems. To do so, watch the video tutorial above.
- Be sure to save your tutor in the CTAT tool. Once you save it, you can leave it there on the machine for the researchers to find.
- Please name your files as "Username-Geo-1", "Username-Geo-2", and "Username-Geo-3" respectively.

Tech Support and Questions

Please direct all your questions (either technical or regarding the study) to Shrenik Devasani (shrenik@iastate.edu) at any point in the study.

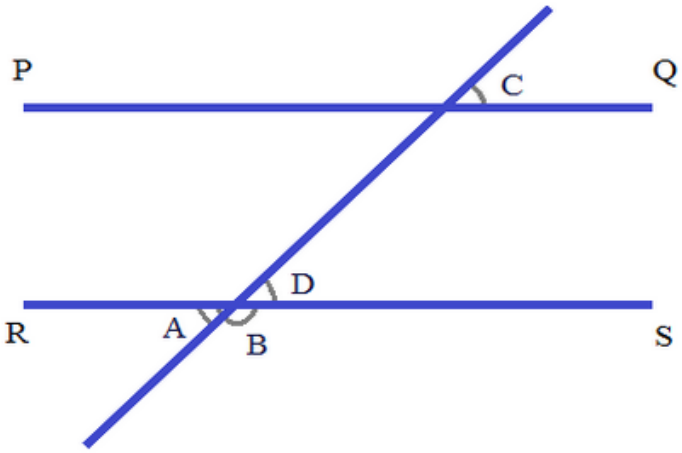
Done with Tasks?

The time for working on the tasks will be two weeks.

Once you feel you are done with the tasks you can email Shrenik (shrenik@iastate.edu) informing him about your completion of the tasks. Shrenik will make sure he can see your models on the server and will send you an email with instructions on taking the end survey and receiving your payment or research credits.

Figure B.8 – Study webpage for CTAT tutor-authors (2/2)

Geometry Problem 1



Given,
The lines PQ and RS are parallel to one another.
D = 70°.

Find the value of any 1 unknown angle (A, B, or C), and provide a reason.

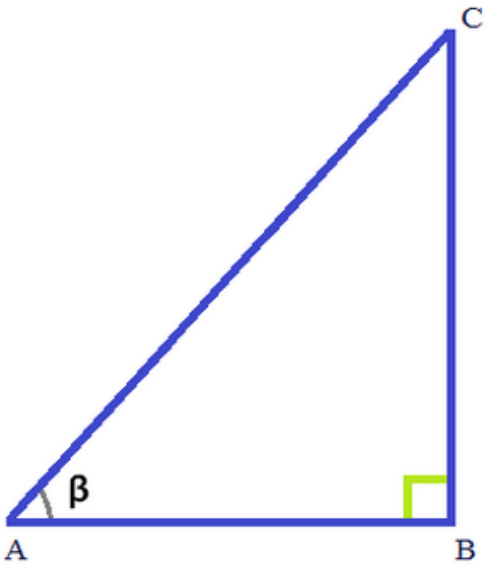
Value of angle A: Reason:

Value of angle B: Reason:

Value of angle C: Reason:

Figure B.9 – Geometry Problem 1

Geometry Problem 2



Given,
AB = 3, BC = 4, AC = 5.

Apply a trigonometric function to &beta such that the answer is in terms of AB and BC.

$\square\ \beta = \frac{\square}{\square}$

Figure B.10 – Geometry Problem 2

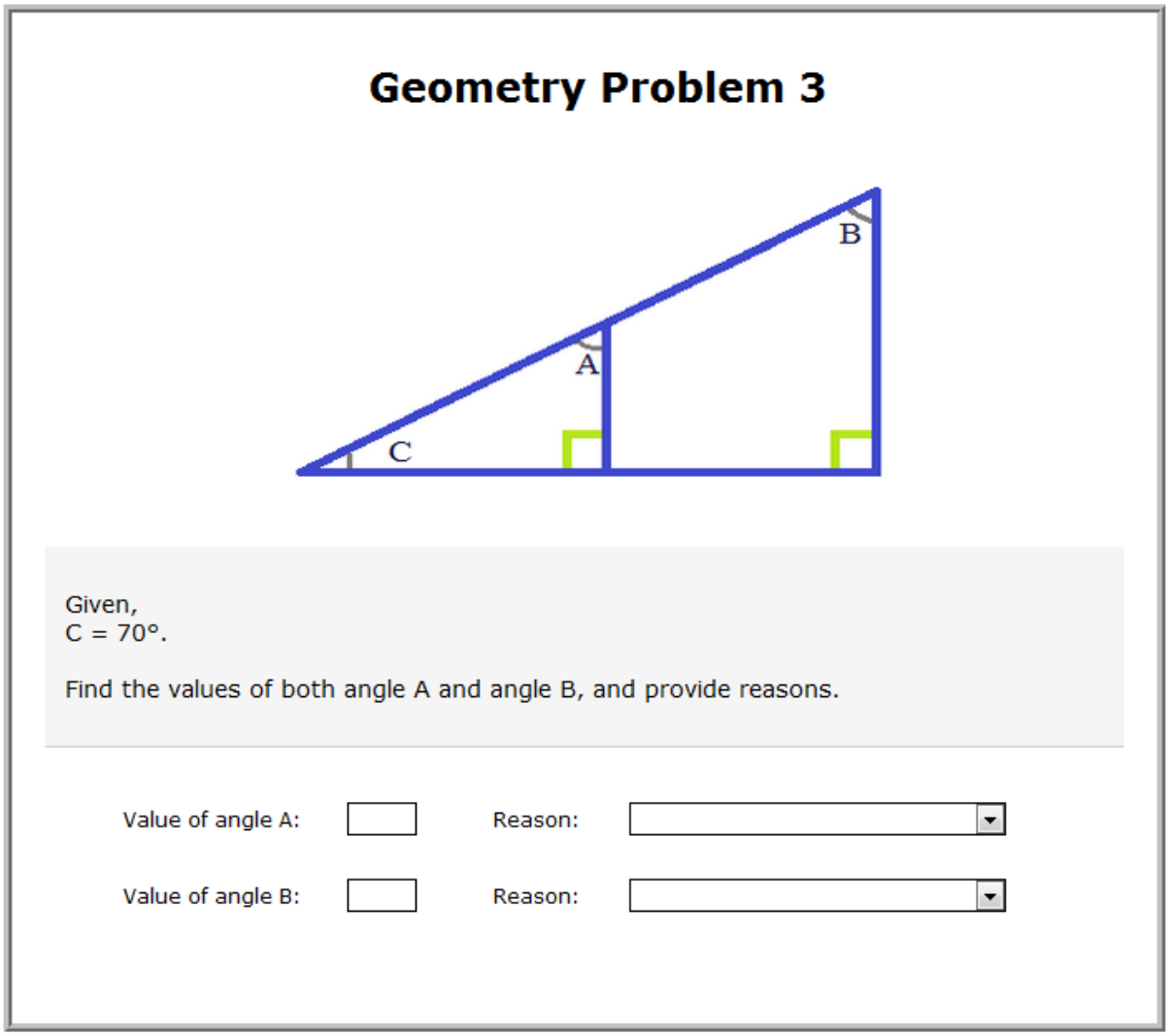


Figure B.11 – Geometry Problem 3

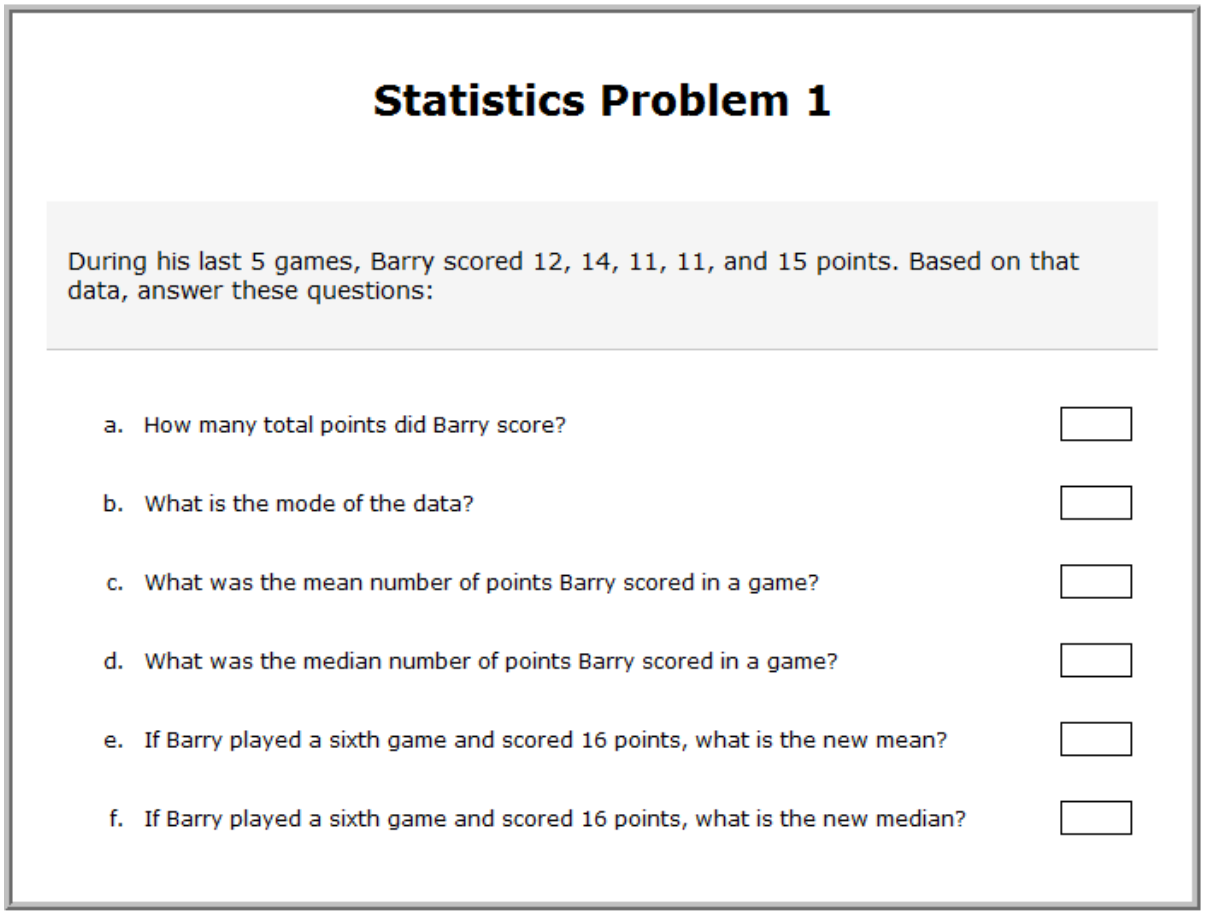


Figure B.12 – Statistics Problem 1

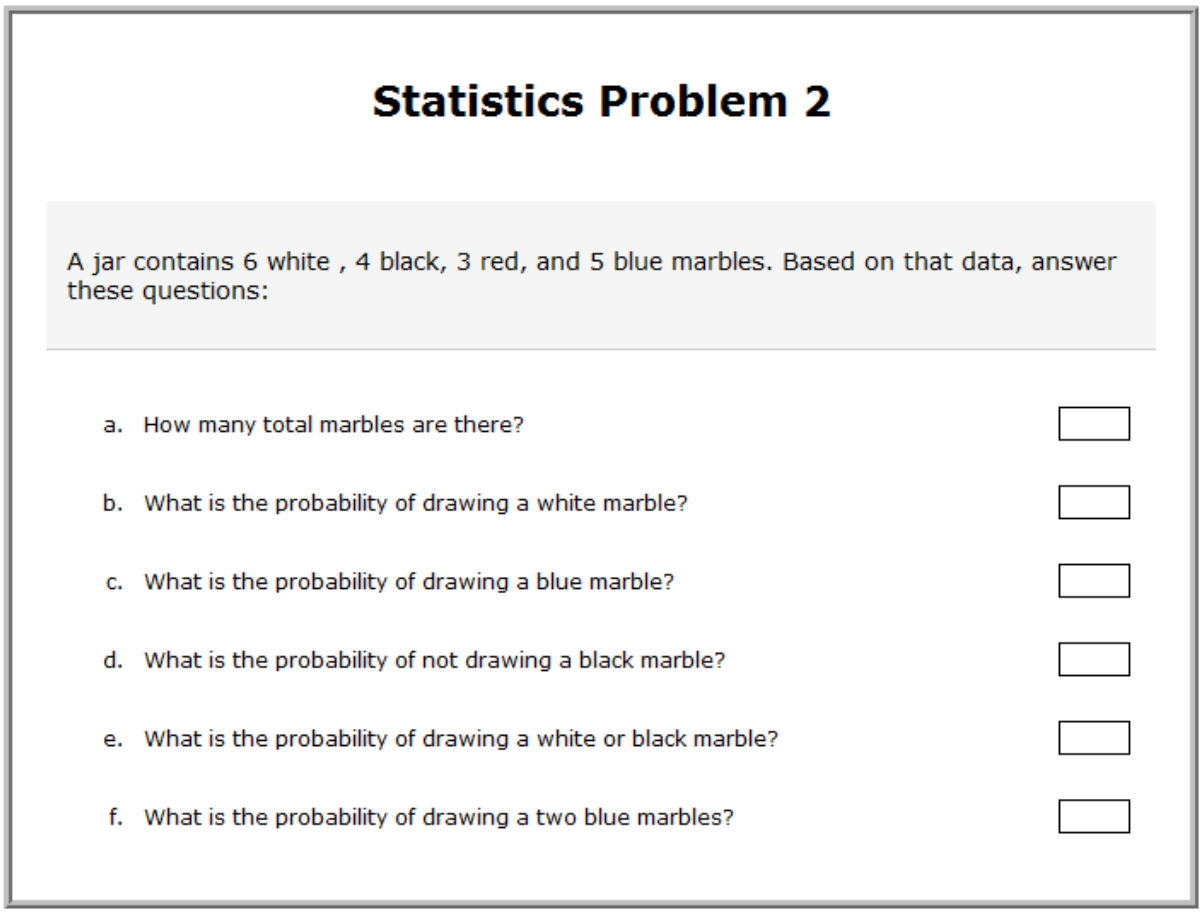


Figure B.13 – Statistics Problem 2

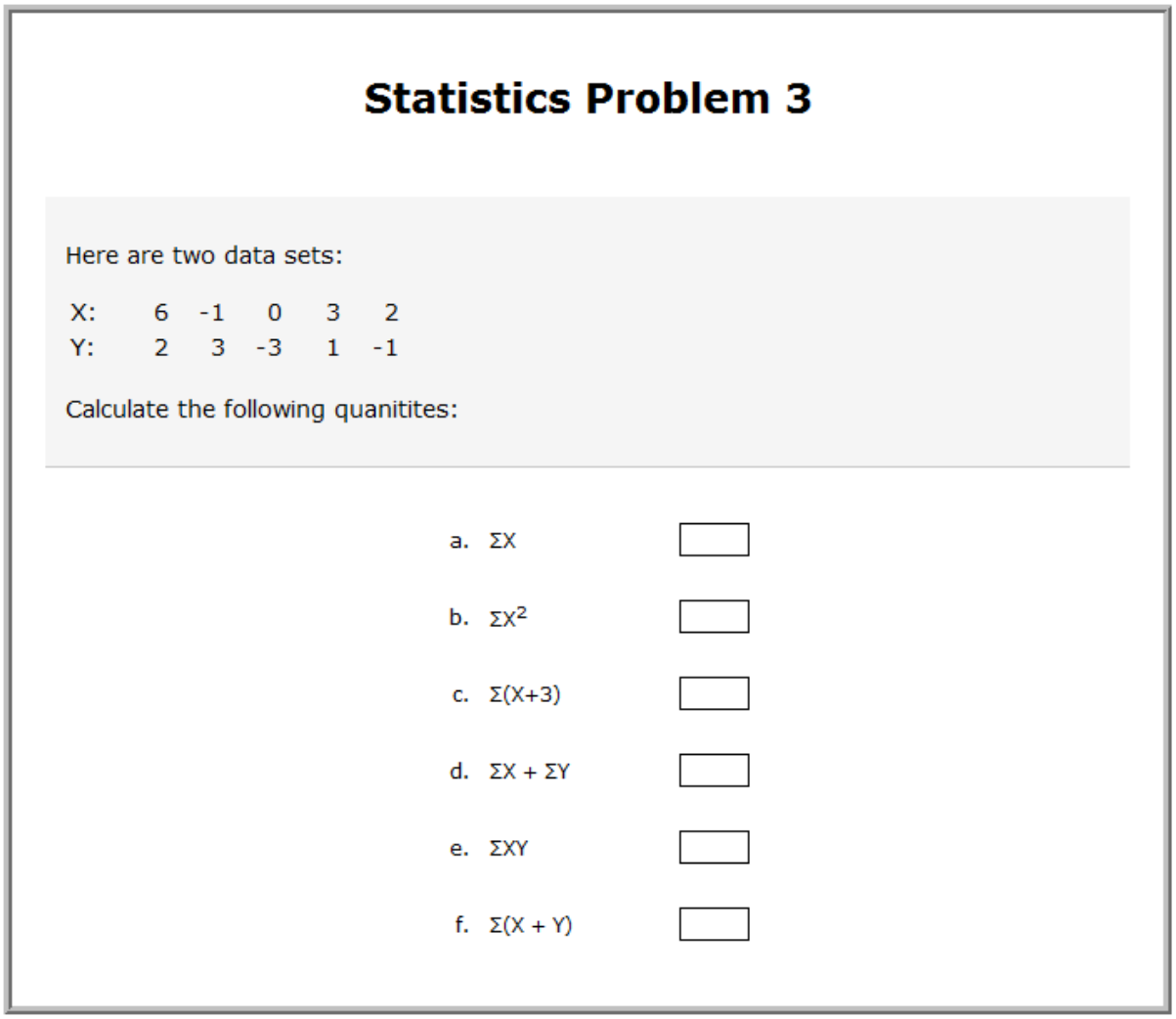


Figure B.14 – Statistics Problem 3

Instructions to Create an xPST File:

Each xPST file must have a sequence section, a feedback section, and a mappings section.

1. Enter the sequence section:

In the sequence section, we mention the order in which the steps must be completed by the learner. The sequence section supports different connecting words like "and", "or" and "then". For example, "stepA then stepB" corresponds to "do step A first and then do step B". When you use "then", the order matters. The "and" connector corresponds to "do step A and do step B". The order does not matter. "or" corresponds to "either do step A or do step B". Give your steps user-friendly names with no spaces, like "CalculateAngle". We can group steps within the sequence by using parentheses. Make sure that each of the opening and closing parenthesis match. The step called All-Done should always be the last step in the sequence. Your entire sequence ends with a semi-colon.

Example:

```
sequence
{
  (Step1 or Step2) then Step3 then (Step4 and Step5) then All-Done;
}
```

2. Enter the feedback section:

In the feedback section, we specify the hints and just-in-time error messages, or "JITs," for each step. We also specify the correct 'answer' to each step. When the learner completes the corresponding step, it is the equivalent of providing the correct answer to that step, and the tutor moves on to the next step in the sequence. The correct answer to each step has been mentioned on the study webpage. We can specify an error-specific JIT to check for a specific mistake made by a student. We refer to the value entered by the student using "{v}" and then use the "==" operator to compare it with the wrong answer. (Please see the example below). In this section, we must have a semicolon at the end of every line. It is necessary that the opening and closing curly brackets should match.

Example:

```
feedback
{
  Step1
  {
    answer: "Angle A and angle B are adjacent angles.";
    Hint: "Angle A and B are adjacent angles.";
    JIT{v=="Angle A and angle B are corresponding angles." }: "Are you sure that A and B are corresponding angles?";
  }
}
```

3. Enter the mappings section:

The last section is the mappings section. You need to know whether the learner has done the steps you're interested in, so you need to eavesdrop on the learner's actions on the webpage. Each widget on the webpage has a unique ID, which can be seen using the xPST plugin when we click on the widget. If your sidebar doesn't show this information, make sure your plugin is set to Authoring mode, which you change in your Add-ons options. We map these widgets IDs to the step names you used in your sequence.

The TutorLink.Done is a special mapping that signifies that all the tasks have been completed by the student.
We must have a semicolon at the end of every line.

Example:

```
mappings
{
  input_3 => Step1;
  input_4 => Step2;
  input_4 => Step3;
  TutorLink.Done => All-Done;
}
```

General Instructions:

- The editor saves the file periodically, but you can also save it manually as and when you wish to, by pressing the Save button.
- You can start this activity and then stop and come back to it later, even on a different computer; just be sure to click 'Save'.
- To help with accurate data about how people use the Web Authoring Tool, please logout when you are not working on the task and go idle for more than 2 minutes.

Figure B.15 – Instructions to create an xPST tutor (text tutorial)

Instructions to Create a CTAT Tutor:

1. Open the student interface and CTAT:

The first step in authoring an Example-tracing tutor is to open both the student interface and CTAT. Position the CTAT window and Flash student interface so that both windows are visible.

2. Create the "Start state":

The next step is to create the initial problem state, or "start state" by selecting Graph > Create Start State in the Behavior Recorder. When prompted for a start state name, enter "Start", and click OK. You should now see your start state in the Behavior Recorder as a new node titled "Start".

3. Demonstrate correct and alternate answers:

After creating a start state, demonstrate correct and alternate answers, as well as common errors, that students might make for this problem. Ensure that CTAT's Author Mode is set to "Demonstrate". Then demonstrate a correct solution to the problem. Make sure that you press the Enter key after entering a value in a text field. Press the Done button after completing the last step of a solution path. The Behavior Recorder will record the demonstrated behavior in the form of a "behavior graph". It records each demonstrated step as an 'edge'—the line connecting two nodes—in its graph. In the graph, actions are represented by edges, while states are represented by nodes. You can demonstrate an alternative solution path by clicking on the start state node in the behavior graph, and demonstrating the solution steps for that path.

In case you make a mistake while demonstrating a solution, you can delete the corresponding state by right clicking on the node and pressing the delete button.

4. Demonstrate incorrect steps:

You can demonstrate an incorrect step in the problem, so that the tutor will be able to recognize this incorrect step and display a message that gives feedback for a specific error that the student might make. In general, any student input that is not recognized by the tutor is marked as incorrect; but by defining incorrect steps in the graph, the tutor will be able to provide a customized error feedback message for the specified input. When demonstrating incorrect actions, you therefore need to mark them explicitly as incorrect behavior. So right-click the edge that was created for the incorrect input and select Change Action Type > Incorrect Action. At the prompt to edit the error message, enter an error message that the student will see when they perform this action.

5. Provide hints:

You can attach hints to the links that represent correct actions. These hints will be displayed when the student requests a hint. To define a hint message, click on a green action label displayed on an edge. From the pop-up context menu, select Edit Hint and Success Messages. Here you can define hints that will be shown to the student when they request a hint.

6. Save your tutor:

Finally, be sure to save your "graph" in the CTAT tool. Select File > Save Graph or use the keyboard shortcut Ctrl-S or Cmd-S to save the behavior graph. That graph contains your tutor. Once you save it, you can leave it there on the machine for the researchers to find.

7. Test your tutor:

To test your tutor, set CTAT's Author Mode to "Test Tutor". To test your tutor from the beginning, click the start state in the behavior graph; then interact with the student interface to check if everything works as expected.

Figure B.16 – Instructions to create a CTAT tutor (text tutorial)

BIBLIOGRAPHY

Aldrich, C. (2009). *The complete guide to simulations and serious games*. San Franscisco: Pfeiffer.

Aleven, V., Koedinger, K. R., & Cross, K. (1999). *Tutoring answer explanations fosters learning with understanding.* Paper presented at the Proceedings of Artificial Intelligence in Education, AIED '99.

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

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

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science, 13*, 467–505.

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

Andrews, D. H., Carroll, L. A., & Bell, H. H. (1995). The future of selective fidelity in training devices. *Educational Technology, 35*(6), 32-36.

Beal, C. R., Walles, R., Arroyo, I., & Woolf, B. P. (2007). On-line tutoring for math achievement testing: A controlled evaluation. *Journal of Interactive Online Learning, 6*(1), 43-55.

Beilock, S. (2010). *Choke: What the secrets of the brain reveal about getting It right when you have to*. NY, NY: Free Press.

Bensley, L., & Van Eenwyk, J. (2001). Video games and real-life aggression: Review of the literature. *Journal of Adolescent Health*.

Blessing, S. B., Devasani, S., & Gilbert, S. B. (2011). *Evaluation of WebxPST: A browser-based authoring tool for problem-specific tutors.* Paper presented at the Proceedings of the Fifteenth Conference on Artificial Intelligence in Education, Auckland.

Blessing, S. B., Gilbert, S., Blankenship, L., & Sanghvi, B. (2009). *From SDK to xPST: A new way to overlay a tutor on existing software*.

Blessing, S. B., Gilbert, S., Ourada, S., & Ritter, S. (2007). *Lowering the bar for creating model-tracing intelligent tutoring systems.* Paper presented at the Proceedings of the 13th International Conference on Artificial Intelligence in Education.

Castner, A. K., Chukhman, I. A., Colbert, E. J., Dale, M. E., Lewis, B. Y., & Zaret, D. R. (2007). *An agent-supported simulation framework for metric-aware dynamic fidelity modeling*. Paper presented at the Proceedings of the 2007 Spring Simulation Multiconference.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science, 13*(2), 145-182.

Chi, M. T. H., De Leeuw, N., Chiu, M. H., & LaVancher, C. (1994). Eliciting self-explanations improves understanding. *Cognitive science, 18*(3), 439-477.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*(1), 37-46.

Corbett, A. T. (2001). *Cognitive computer tutors: Solving the two-sigma problem.* Paper presented at the Conference of User Modeling, Sonthofen, Germany.

Day, R. S. (1988). Alternative representations. *The Psychology of Learning and Motivation: Advances in Research and Theory, 22*, 261-305.

Dede, C. (2009). Immersive interfaces for engagement and learning. *Science*, *323*(5910), 66-69.

Devasani, S., Aist, G., Blessing, S. B., & Gilbert, S. B. (2011). *Lattice-based approach to building templates for natural language understanding in intelligent tutoring systems.* Paper presented at the Proceedings of the Fifteenth Conference on Artificial Intelligence in Education, Auckland.

Devasani, S., Gilbert, S. B., Shetty, S., Ramaswamy, N., & Blessing, S. B. (2011). *Authoring intelligent tutoring systems for 3D game environments.* Paper presented at the Proceedings of the Authoring Simulation and Game-based Intelligent Tutoring Workshop at the Fifteenth Conference on Artificial Intelligence in Education, Auckland.

Evens, M. W., Chang, R. C., Lee, Y. H., Shim, L. S., Woo, C. W., Zhang, Y., et al. (2001). *CIRCSIM-Tutor: An intelligent tutoring system using natural language dialogue.* Paper presented at the Proceedings of the Midwest Artificial Intelligence and Cognitive Science Conference, Oxford, OH.

Fellbaum, C. (1998). *WordNet: An electronic lexical database*: The MIT press.

Franklin, S., & Graesser, A. (1996). *Is it an agent, or just a program? A taxonomy for autonomous agents* Paper presented at the Third International Workshop on Agent Theories, Architectures, and Languages.

Gertner, A., & VanLehn, K. (2000). *Andes: A coached problem solving environment for physics*.

Gilbert, S., Blessing, S. B., & Kodavali, S. (2009). *The Extensible Problem-Specific Tutor (xPST): Evaluation of an API for tutoring on existing interfaces.* Paper presented at the Artificial Intelligence in Education.

Gilbert, S. B., Devasani, S., Kodavali, S., & Blessing, S. B. (2011). *Easy authoring of intelligent tutoring systems for synthetic environments.* Paper presented at the Proceedings of the Twentieth Conference on Behavior Representation in Modeling and Simulation.

Glass, M. (2001). *Processing language input in the CIRCSIM-Tutor intelligent tutoring system.* Paper presented at the Artificial Intelligence in Education.

Gorman, P. F. (1991). The future of tactical engagement simulation. In D. Pace (Ed.), *Proceedings of the 1991 Summer Computer Simulation Conference* (pp. 1181-1186). Baltimore, MD: Society for Computer Simulation.

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

Graesser, A. C., Wiemer-Hastings, P., Wiemer-Hastings, K., Harter, D., Tutoring Research Group, T. R. G., & Person, N. (2000). Using latent semantic analysis to evaluate the contributions of students in AutoTutor. *Interactive Learning Environments, 8*(2), 129-147.

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System Technical Journal, 29*(2), 147-160.

Johnson, W. L. (2009). *A simulation-based approach to training operational cultural competence.* Paper presented at the Proceedings of ModSIM.

Kirriemuir, J., & McFarlane, A. (2004). Literature reviews in games and learning.

Kodavali, S., Gilbert, S., & Blessing, S. (2010). *Expansion of the xPST framework to enable non-programmers to create intelligent tutoring systems in 3D game environments*.

Koedinger, K. R., Aleven, V., Heffernan, N., McLaren, B., & Hockenberry, M. (2004). *Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration.* Paper presented at the Proceedings of Seventh International Conference on Intelligent Tutoring Systems, Berlin.

Koedinger, K. R., Aleven, V. A., & Heffernan, N. (2003). Toward a rapid development environment for cognitive tutors. *Artificial intelligence in education: Shaping the Future of Learning Through Intelligent Technologies, 97*, 455.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education, 8*(1), 30-43.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent Tutoring Goes to School in the Big City. 30-43.

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

Laurillard, D. (1993). *Rethinking university teaching: A framework for the effective use of educational technology*: Routledge.

Lessiter, J., Freeman, J., Koegh, E., & Davidoff, J. (2001). A cross-media presence questionnaire: The ITC-sense of presence inventory. *Presence: Teleoperators and Virtual Environments, 10*(3), 282-297.

Levenshtein, V. I. (1966). *Binary codes capable of correcting deletions, insertions, and reversals.* Paper presented at the Soviet Physics Doklady.

Livak, T., Heffernan, N. T., & Mover, D. (2004). *Using cognitive models for computer generated forces and human tutoring.* Paper presented at the 13th Annual Conference on (BRIMS) Behavior Representation in Modeling and SImulation, Arlington, VA.

Lloyd, J. (2004). The Torque Game Engine. *Game Devel. Mag, 11*(8), 8–9.

Maass, J., & Blessing, S. (2011). *xSTAT: An intelligent homework helper for students.* Paper presented at the Georgia Undergraduate Research in Psychology Conference.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-based tutors: A success story. *Engineering of Intelligent Systems*, 931-940.

Mitrovic, A., & Ohlsson, S. (1999). Evaluation of a constraint-based tutor for a database. *International Journal of Artificial Intelligence in Education, 10*(3-4), 238-256.

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education, 10*(1), 98-129.

Murray, W. R. (2006). *Intelligent tutoring systems for commercial games: The virtual combat training center tutor and simulation.* Paper presented at the The Second Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE), Marina del Rey, California.

Nevill-Manning, C. G. (1993). Programming by demonstration. *New Zealand Journal of Computing, 4*(2), 15-24.

Ohlsson, S. (1996). Learning from performance errors. *Psychological Review, 103*(2), 241.

Polich, J. M., & Schwartz, S. H. (1974). The effect of problem size on representation in deductive problem solving. *Memory & Cognition, 2*(4), 683-686.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program, 14*(3), 130-137.

Remolina, E., Ramachandran, S., Stottler, R., & Howse, W. R. (2004). *Intelligent simulation-based tutor for flight training.* Paper presented at the Proceedings of the Industry/Interservice, Training, Simulation & Educational Conference.

Remolina, E., Ramachandran, S., Stottler, R. H., & Howse, W. R. (2004). *Intelligent Simulation-Based Tutor for Flight Training.* Paper presented at the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC), Orlando, FL.

Ritter, S., Kulikowich, J., Lei, P., McGuire, C. L., & Morgan, P. (2007). What evidence matters? A randomized field trial of Cognitive Tutor Algebra I, pp. 13-20.

Rowe, J. P., Shores, L. R., Mott, B. W., & Lester, J. C. (2010). *Individual differences in gameplay and learning: A narrative-centered learning perspective.* Paper presented at the Proceedings of the Fifth International Conference on Foundations of Digital Games.

Rus, V., & Graesser, A. (2006). *Deeper natural language processing for evaluating student answers in intelligent tutoring systems.* Paper presented at the American Association for Artificial Intelligence.

Shelby, R., Schulze, K., Treacy, D., Wintersgill, M., VanLehn, K., & Weinstein, A. (2001). *An assessment of the Andes tutor.* Paper presented at the Proceedings of the 2001 Physics Education Research Conference.

Stanney, K. M. (2002). *Handbook of virtual environments*. Mahwah, NJ: Erlbaum.

Steinhart, D. J. (2001). *Summary Street: An intelligent tutoring system for improving student writing through the use of latent semantic analysis.* Unpublished Ph.D., University of Colorado, Boulder.

Stottler, R. H. (2000). *Tactical Action Officer Intelligent Tutoring System.* Paper presented at the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC), Orlando, FL.

Stottler, R. H., Fu, D., Ramachandran, S., & Jackson, T. (2001). *Applying a generic intelligent tutoring system authoring tool to specific military domains.* Paper presented at the Industry/Interservice, Training, Simulation & Education Conference (I/ITSEC), Orlando, FL.

Stottler, R. H., & Vinkavich, L. M. (2000). *Tactical Action Officer Intelligent Tutoring System (TAO ITS).* Paper presented at the Proceedings of the Industry/Interservice, Training, Simulation & Education Conference.

Suraweera, P., & Mitrovic, A. (2002). *KERMIT: A constraint-based tutor for database modeling*.

Thissen, D., & Mislevy, R. J. (2000). Testing algorithms. In H. Wainer (Ed.), *Computerized Adaptive Testing: A Primer*. Mahway, NJ: Lawrence Erlbaum Associates.

Thomas, J. M., & Young, R. M. (2009). *Towards a domain-independent framework to automate scaffolding of task-based learning in digital games.* Paper presented at the Proceedings of the 4th International Conference on Foundations of Digital Games.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., et al. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education, 15*(3), 147-204.

Whitley, K. N. (1997). Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages and Computing, 8*(1), 109-142.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

Woolf, B. P. (2008). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*: Morgan Kaufmann.

Woolf, B. P., & Cunningham, P. (1987). *Building a community memory for intelligent tutoring systems*.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my gratitude to those who helped me at various stages of my research work and with the writing of this thesis. Firstly, I would like to thank my major professor, Dr. Stephen Gilbert for his guidance and support throughout this research and for being a wonderful mentor. He helped me realize my true passion by introducing me to the world of Learning Sciences and Intelligent Tutoring Systems. I am grateful to him for having sent me to several conferences and workshops, which gave me tremendous exposure. I owe a lot to Dr. Stephen Blessing from the University of Tampa who collaborated with us on all our publications. His suggestions and feedback were invaluable. I would also like to thank my committee members, Dr. Leslie Miller and Dr. Craig Ogilvie, for their guidance, encouragement, and for being very patient with me. Dr. Gregory Aist, who is a co-author on one of my papers, gave me some great suggestions during the early stages of my research work. Steven Ourada, the lead architect of xPST provided me with much needed guidance and helped me get to speed. Finally, I would like to thank my peers who provided valuable assistance while completing my research work: Munish Gopal, Suhas Shetty, Nandhini Ramaswamy, Sateesh Kodavali, Mike Oren, Ankit Agrawal and Sugam Sharma.