A MIXED-RESPONSE INTELLIGENT TUTORING SYSTEM BASED ON LEARNING FROM DEMONSTRATION

A Dissertation

by

OMAR ÁLVAREZ XOCHIHUA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2012

Major Subject: Computer Science

A Mixed-response Intelligent Tutoring System Based on Learning from Demonstration

A MIXED-RESPONSE INTELLIGENT TUTORING SYSTEM

BASED ON LEARNING FROM DEMONSTRATION

A Dissertation

by

OMAR ÁLVAREZ XOCHIHUA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Co-Chairs of Committee,	Riccardo Bettati
	Lauren Cifuentes
Committee Members,	Richard Furuta
	Frank Shipman
Head of Department,	Duncan M. Walker

May 2012

Major Subject: Computer Science

ABSTRACT

A Mixed-response Intelligent Tutoring System Based on Learning from Demonstration.

(May 2012)

Omar Álvarez Xochihua, B.S., Universidad Autónoma de Baja California; M.A.I.,

Instituto Tecnológico y de Estudios Superiores de Monterrey

Co-Chairs of Advisory Committee: Dr. Riccardo Bettati
Dr. Lauren Cifuentes

Intelligent Tutoring Systems (ITS) have a significant educational impact on student's learning. However, researchers report time intensive interaction is needed between ITS developers and domain-experts to gather and represent domain knowledge. The challenge is augmented when the target domain is ill-defined. The primary problem resides in often using traditional approaches for gathering domain and tutoring experts' knowledge at design time and conventional methods for knowledge representation built for well-defined domains. Similar to evolving knowledge acquisition approaches used in other fields, we replace this restricted view of ITS knowledge learning merely at design time with an incremental approach that continues training the ITS during run time. We investigate a gradual knowledge learning approach through continuous instructor-student demonstrations. We present a Mixed-response Intelligent Tutoring System based on Learning from Demonstration that gathers and represents knowledge at run time. Furthermore, we implement two knowledge representation methods (Weighted Markov

Models and Weighted Context Free Grammars) and corresponding algorithms for building domain and tutoring knowledge-bases at run time.

We use students' solutions to *cybersecurity* exercises as the primary data source for our initial framework testing. Five experiments were conducted using various granularity levels for data representation, multiple datasets differing in content and size, and multiple experts to evaluate framework performance. Using our WCFG-based knowledge representation method in conjunction with a finer data representation granularity level, the implemented framework reached 97% *effectiveness* in providing correct feedback. The ITS demonstrated consistency when applied to multiple datasets and experts. Furthermore, on average, only 1.4 hours were needed by instructors to build the knowledge-base and required tutorial actions per exercise. Finally, the ITS framework showed suitable and consistent performance when applied to a second domain.

These results imply that ITS domain models for ill-defined domains can be gradually constructed, yet generate successful results with minimal effort from instructors and framework developers. We demonstrate that, in addition to providing an *effective* tutoring performance, an ITS framework can offer: *scalability* in data magnitude, *efficiency* in reducing human effort required for building a confident knowledge-base, *metacognition* in inferring its current knowledge, *robustness* in handling different pedagogical and tutoring criteria, and *portability* for multiple domain use.

DEDICATION

To my dear wife Olimpia, my fantastic sons Jared & Eber,

and especially to my unforgettable mother Silvia† and brother Saul†.

ACKNOWLEDGEMENTS

I would like to thank my research advisors, Dr. Riccardo Bettati and Dr. Lauren Cifuentes, for their guidance and support during my doctoral studies. I also thank the rest of my committee members, Dr. Richard Furuta and Dr. Frank Shipman, for their valuable and fruitful comments and feedback that helped me to improve my dissertation work.

I would like to thank my friends at Texas A&M University who helped me to conduct my research studies. Specially, I would like to express my appreciation to Rene Mercer for her always helpful advice, friendship, and constructive criticism in her multiple reviews of my dissertation. I also thank the participants involved in evaluating the functionality and performance of the implemented Intelligent Tutoring System framework.

Finally, I thank my wife, Olimpia, for her love and continuous support and encouragement during all these years. I also thank my sons, Jared and Eber, for living with us through this great experience and making this period of time more enjoyable. I also thank all my family members for their continuous support in my successfully accomplishing my doctoral studies.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

A. Motivation

Using computers in education has become a ubiquitous phenomenon in contemporary culture. New learning environments have been developed using various types of digital media (e.g., audio, video, and images) and computer network technology, thereby allowing delivery of more interactive and effective instruction to large numbers of students in an affordable and timely way. By using these technology affordances, students from different domains can learn basic concepts, apply those concepts through solving a diversity of problems, and even gain hands-on experience in their fields by interacting with virtual instruments at a distance. However, researchers agreed that it was not sufficient to make this instructional material merely accessible. Educational and Artificial Intelligence researchers saw the importance and feasibility of complementing this type of instruction with customized tutoring for every student (Nkambou, Bourdeau, & Mizoguchi, 2010). Empirical research has demonstrated that students perform twice as well when individual tutoring is used instead of a conventional group teaching approach (Bloom, 1984). Therefore, providing personalized guidance to students who learn through these new instructional environments, places Intelligent Tutoring Systems (ITSs) in a unique position to offer a comprehensive and meaningful educational service.

This dissertation follows the style of *International Journal of Artificial Intelligence in Education (IJAIED)*.

In order to customize the pedagogical process, ITSs have been complementing computerized learning environments. ITSs are computer-based experts that (1) provide personalized instruction, (2) monitor and assess student learning performance, and (3) provide feedback to students during the learning process (Psotka, Massey, & Mutter, 1998). ITS researchers use Artificial Intelligence (AI) techniques to gather and represent experts' domain knowledge and build effective and customized tutoring support for students during the problem solving process (Nkambou et al., 2010). The problem solving process can be summarized by the following set of activities: contextualize and understand the problem, identify possible problem solutions, select the best solution, implement the selected solution's steps, and evaluate solution outcomes and adapt the solution if needed (Jonassen, 1997).

Intelligent Tutoring Systems have been helping students learn how to solve problems in a variety of domains (e.g., geography, medical diagnosis, computer programming, mathematics, physics, genetics, law, and chemistry) since the early 1980's (Urban-Lurain, 1996). The types of problems found within these domains can be classified using a continuum based on the degree of structure inherent within the problem; this problem-structure continuum ranges from well-structured to ill-structured (Simon, Egidi, Viale, & Marris, 1992). The continuum identifies problems with single known solutions and clear solution paths (very well-structured) at one end, and those with several potentially acceptable solutions and lack of a defining solution path (ill-structured) at the other end (Jonassen, 1997). Within the ITS field, these two categories are usually referred to as well- and ill-defined.

The types of problems students encounter within some domains may be very structured, well-defined problems. These types of problems are mainly characterized by having only one or a small number of correct answers. Well-defined problems also have a systematic solution path, as opposed to problems where solution paths vary widely and lead to various correct answers. For well-defined problems, which are commonly found in domains such as mathematics (Arroyo, Murray, Woolf, & Beal, 2003) and chemistry (Stevens & Soller, 2005), the domain knowledge can be often reduced to a small number of rules. Therefore, feedback use for well-defined domains should be constructed in order to help students memorize concepts, facts, or problem solution steps (Woolf, 2009). Hence, gathering instructors' knowledge and appropriate *tutorial actions* at ITS design time is a suitable and economical approach.

However, this approach is not sufficient for ill-defined problems because novel problem solutions may constantly arise based on contextual situations. Ill-defined problems are mainly characterized by their lack of systematic solution methods (Lynch, Ashley, Aleven, & Pinkwart, 2006). Two fields that exemplify ill-defined problems are law and architecture. These fields are referred to as ill-defined domains. Jonassen (1997) and Lynch et al. (2006) summarized a comprehensive list of characteristics attributed to problems within ill-defined domains. A compilation of these characteristics follows:

- initial steps to solve a problem can vary,
- multiple solutions and solution paths exist,
- right answers are context and time dependent,
- systematic solution methods do not exist, and

- new right answers will inevitably emerge.

Therefore, within ill-defined domains, the feedback provided by an ITS should be constructed in order to help students reason and make inferences based on the specific circumstances (Woolf, 2009).

Furthermore, within ill-defined domains, acquiring relevant and sufficient expert knowledge to design ITSs is extremely costly. Fournier-Viger, Nkambou, and Mephu-Nguifo (2008b) state that "for many ill-defined domains, the domain knowledge is hard to define explicitly" when developing an ITS. Thus, even when an expert, or set of experts, can provide a considerable amount of domain knowledge at design time, there is always the possibility of new points of view or evidence that may challenge previous conclusions.

Past ITS research, primarily within well-defined domains, has provided strong empirical evidence indicating improved learning outcomes by up to one or even two standard deviations when students are paired with an ITS (Lynch et al., 2006; Jonassen, 1997). However, the construction of these educationally powerful systems is still extremely costly and time consuming, principally in domains characterized by having ill-defined problems (Fournier-Viger, Nkambou, & Mephu-Nguifo, 2010).

Traditionally, ITS developers build the domain expertise of the ITS based on human expert knowledge. This domain expertise usually consists of a predefined set of solutions, solution paths, expected student misconceptions, and the required feedback, which are often predetermined by experts at ITS design time. Because of the significant time and cost currently necessary to build these tutoring environments, methods for

efficiently gathering and representing human expert knowledge within the ITS field remain highly researched topics.

This is of particular relevance within domains where students are expected to solve ill-defined problems that lack a clear structure and which are highly time and context dependent. Within these types of domains, because of the unpredictability of the entire set of problem solving steps, the ITS domain expertise cannot be based on a set of predefined solution paths, potential correct solutions, and a set of standard feedback responses. Hence, the framework of ITSs designed to assist students in addressing ill-defined problems must comprise a set of functional characteristics that allows an ITS to evolve and adapt to changing learning environments. Using these characteristics an ITS framework should progressively learn or infer novel solution paths, viable answers, and methods for advising students at run time. In addition, what may truly define the intelligence of an intelligent tutor is its *metacognitive* ability to autonomously determine its current knowledge and when that knowledge should be improved by external sources such as a human expert. Based on our review of the literature, ITS frameworks addressing the functionalities mentioned above do not exist, which begs the following question: What is the best way to construct such an ITS framework, and can that framework generate adequate feedback at a high level of effectiveness?

B. The Problem: ITS Knowledge Acquisition at Design Time

A challenging goal, for any ITS, is to learn enough adequate domain-specific knowledge and relevant feedback from human instructors or experts to effectively support students

in their learning. Within this document we will often refer to feedback provided by the ITS or human expert as a *tutorial action*. The acquisition of domain knowledge and development of tutorial actions has been a task primarily implemented at design time. ITS developers build this knowledge by interviewing experts regarding (a) meaningful, domain-specific problems, (b) the problem solving process, (c) possible problem solutions, (d) potential student misconceptions, and (e) the type of help instructors provide to support students in overcoming misconceptions. In addition, ITS developers observe and track experts as they move through the entire problem solving process. While this approach has proven effective for well-defined domains, it is for ill-defined domains for which these methods are not economically sound for gathering all of the domain knowledge and corresponding dynamic set of tutorial actions.

Customized feedback is the primary contribution the ITS makes to the learning process. However, for the ITS to effectively help students, ITS developers must consider where on the problem-structure continuum the domain falls for which the ITS is going to be utilized. The domain categorization on the continuum will determine the type of feedback students will need. This in turn will identify the best framework to use for constructing the ITS.

For an ITS to aid students in ill-defined problem solving, it is evident that gathering comprehensive domain knowledge during design time or inferring new knowledge and the adequate feedback at run time for unexpected situations is a difficult task (Amershi & Conati, 2009). Therefore, for ITSs intended for ill-defined domains, continuous instructor interaction, when requested by the intelligent tutor beyond design

time, should be considered. In this way, instructors will be able to monitor, evaluate, and contribute to the ITS-student tutoring process. However, this implies that the ITS should learn to identify novel students' solutions and autonomously determine whether its domain knowledge is sufficient. When unfamiliar situations are encountered by the ITS, the intervention by a human instructor with domain expertise would be required. This reduces the risk, which is common in systems with tutorial actions that are formulated at design time, of not being sufficiently student-centered.

Student-centered learning considers the needs of students as individuals and encourages them to have an active participation within the learning process (Jones, 2007). For instance, presenting the same material to every student without considering his or her specific learning needs (the one size fits all approach) is not student-centered (Woolf, 2009). By modeling student and domain knowledge, intelligent tutors have the capability to provide student-centered learning environments (Woolf, 2009). Within well-defined domains, ITSs use the predefined domain model to evaluate student performance in order to provide customized instruction and feedback. However, within ill-defined domains, ITSs should learn to identify unfamiliar situations and request the instructor's help when necessary in order to provide personalized feedback according to the characteristics of the solution provided by the student.

In this study, we present a mixed-response Intelligent Tutoring System framework based on Learning from Demonstration (LfD) techniques for which we will show to be applicable for well- and ill-defined problems. By having the capability of determining its current domain knowledge level, the intelligent tutor will be able to infer

its level of confidence before triggering a tutorial response for a student's feedback request. The ITS will either provide feedback to students when the knowledge confidence level is high, or ask the human instructor to direct the tutorial task; thereby training the ITS to handle similar situations in the future. The presented mixed-response ITS framework exemplifies an extended view of an intelligent tutor intended to sufficiently support student-centered learning within ill-defined domains by highlighting the importance of maintaining expert participation during run time and adding a much needed *metacognitive* feature. The success of this framework will illustrate that a metacognitive functionality can be implemented that will teach the ITS how to recognize and better apply its current knowledge as well as defer to the human expert when necessary.

C. A Modern View of an Intelligent Tutoring System

We envision that modern ITS frameworks, primarily those addressing ill-defined domains, should be based on a set of functional characteristics that allow them to dynamically teach and learn while in use. An intelligent tutoring environment should have the capability to identify knowledge gaps within its knowledge-base and request human help to lead the tutoring process when necessary and learn from it. We propose a tutoring environment in which instructors have a more active role; allowing them to continuously train the intelligent tutor and customize the tutoring knowledge-base based on individual instructional perspectives. Next, we describe how we define the envisioned

ITS framework by the following functional characteristics: *effectiveness*, *scalability*, *efficiency*, *portability*, *metacognition* and *robustness*.

1. Effectiveness

We define ITS *effectiveness* as the system's capability to properly learn and use the set of demonstrations obtained from student-instructor interactions. The intelligent tutor should be able to construct or reconstruct its knowledge-base based on these demonstrations. The knowledge-base generated from demonstrations should be represented in such a way that identical or highly similar student problem solving situations are classified correctly and appropriate tutorial actions are generated. In other words, an ITS should have the ability to successfully provide adequate tutorial actions according to the learned knowledge (Woolf, 2009).

2. Scalability

One of the primary objectives of the use of ITSs (and many other educational technologies) is to scale up the ability of a single instructor (or a small number thereof) to effectively reach out to and teach large numbers of students. Therefore, any technology supporting scalability should do so along the following lines: (1) scalability in terms of numbers of students; (2) scalability in terms of student diversity; and (3) scalability in terms of adaptability to instructors' pedagogical perspectives. In particular, for an ITS, this means having the ability to build and continuously refine its knowledge-base in order to provide adequate help to a large number of students despite their diverse

backgrounds. Furthermore, ITSs should maintain or increase its performance proportionally as it is applied to larger situations. Hence, the ITS knowledge representation structures should be capable of accommodating that growth and maintaining an effective system functionality to meet tutorial needs regardless of the size or diversity of the knowledge obtained (Viccari, Jaques, & Verdin, 2008).

3. Efficiency

We define ITS *efficiency* as the system's capability to quickly and easily learn from demonstrations obtained from student-instructor interactions. As stated previously, a primary challenge in building ITSs is the time required to gather the domain knowledge and corresponding tutorial actions. This is because of the large amount of time required by ITS designers to gather and represent knowledge from domain experts. ITS authoring systems are intended to reduce or eliminate the need for designer and domain expert interaction by allowing instructors to build the ITS knowledge individually. The efficiency of ITS authoring systems is often measured by this amount of time and effort required by instructors to build an ITS; principally at design time (Lynch et al., 2006). However, we are measuring the efficiency of the proposed ITS framework by considering the amount of time required to build the ITS domain knowledge and tutorial actions at run time as opposed to merely at design time.

4. Portability

We consider *portability* (*domain agnosticism*) to be a core characteristic of a modern ITS framework. That is, it should be able to be used across learning domains with minimal implementation effort. The use of flexible knowledge representation structures is a key element for ITS frameworks to be domain independent. In fact, one of the primary tenets of this work is that ITSs can be designed in an a priori domain agnostic fashion with domain knowledge added at run time. Portability is considered within the ITS field but not often addressed (Nkambou, et al., 2010). Most of the current ITS frameworks for ill-defined problems have been implemented and tested with only a target domain in mind.

5. Metacognition

In order to continuously improve its own knowledge-base, a modern ITS should have the capability to know what it does and does not know, and to thereby determine its competency to address previously unknown student problem solving situations. In our research, this is referred to as *metacognition*. Furthermore, having metacognitive skills implies that the ITS will know when to best use its current knowledge and –most importantly– when to best invoke strategies for learning and extending its knowledge-base (Fournier-Viger et al., 2008b). Adding metacognitive capabilities to an ITS enables it to assess its current domain knowledge level, to grow and adapt, and to remain active and useful over time (Chernova & Veloso, 2009).

6. Robustness

Finally, when training an ITS at design time, experts communicate the set of correct and incorrect solutions and solution paths in a systematic way. In addition, ITS developers often obtain the domain knowledge from only one or a few experts with equal or similar instructional perspectives (Woolf, 2009). This ITS implementation environment results in building and testing the ITS knowledge-base in a controlled fashion. For an ITS framework in which knowledge is built at run time, however, the system grows its domain expertise on-line from student and instructor input. Particular attention must be paid such that the learning process leads to an effective set of rules despite different student backgrounds, sequences of training examples, and differences in instructors' pedagogical perspectives. In addition, the ITS should be able to handle errors or instructor differences while it is being trained and maintain acceptable system performance despite anomalous or inconsistent inputs. We call this characteristic the *robustness* of the ITS.

D. Hypotheses of the Study

The purpose of this study is to determine if an ITS using a mixed-response framework addressing the aforementioned functional characteristics (e.g., *effectiveness*, *scalability*, *efficiency*, *portability*, *metacognition* and *robustness*) can build comprehensive domain knowledge and appropriate tutorial actions based on demonstrations of human instructor-student interactions. This study also seeks to determine whether the ITS can effectively estimate its knowledge confidence level in order to initiate interaction with

students and provide adequate feedback based on its learned knowledge, or trigger a help request asking human tutors to lead the tutoring process. Our study is focused on the development of a mixed-response ITS framework using exercises developed from the Network Security (*cybersecurity*) domain. Problems addressed within this domain include well- and ill-defined characteristics. Based on the previous discussion, our study evaluates the following hypotheses:

1. Hypothesis 1 (effectiveness)

The ITS's domain knowledge and tutorial actions can gradually be built at run time from logs of instructor-student interactions by using Learning from Demonstration techniques.

We present how the proposed mixed-response ITS framework can "learn" solution patterns produced by students and ask the human instructor to provide feedback when solution paths unknown to the ITS arise, thus progressively improving its ability to apply tutorial actions and ultimately automating instruction. We demonstrate how LfD can be used to derive a policy to autonomously classify student activities and respond with tutorial actions in an effective way.

2. Hypothesis 2 (metacognition)

The proposed mixed-response ITS framework can automatically determine its knowledge confidence level by using a confidence-based approach to Learning from Demonstration

in order to autonomously decide whether to interact with the student or make a request for an instructor response.

We demonstrate that by associating the ITS's previously learned solutions with new student activities, we can compute the confidence level that the ITS has in the policy. By using these confidence levels, our ITS framework can autonomously decide whether to interact with the students as long as the confidence level is high, or make a request for and learn from the instructor's response whenever the confidence level is not sufficiently high, thereby implementing the aforementioned metacognitive functional characteristic.

3. Hypothesis 3 (efficiency, scalability, and robustness)

The proposed mixed-response ITS framework based on Learning from Demonstration can be implemented by using data representation methods that support three functional characteristics envisioned for modern ITS: efficiency, scalability, and robustness.

We describe the methods used for representing the ITS knowledge-base and how these methods support the implementation of the functional characteristics envisioned for a modern ITS framework. These methods were implemented in a batch and online ITS learning fashion and their performances were compared. Specifically, we show how efficiency, scalability, and robustness are highly influenced by the selected data representation method.

4. Hypothesis 4 (portability)

The proposed mixed-response ITS framework based on Learning from Demonstration can be implemented by using data representation methods that can provide a transferable framework to different content domains.

We describe how the methods used for representing the ITS knowledge-base and tutorial actions support the implementation of the proposed ITS framework in two different domains. *Cybersecurity* (configuration of network security devices), a domain considered as ill-defined, and *databases* (specification of SQL-queries), a domain considered more structured along the problem-structure continuum. Also, we illustrate several common characteristics found in logged data obtained from students solving problems within both well- and ill-defined domains.

E. Relevant Contributions

Recent studies regarding ITSs supporting ill-defined problem solving have typically focused on the ITS knowledge acquisition from experts at design time. Our research expands this method by integrating the instructor into the tutoring process (mixed-response approach) of problem solving, allowing an ITS to learn from this integration at run time.

This research is of significance to the domains of ITSs and educational technology. Our mixed-response approach extends the work of the ITS field by introducing a twofold approach to designing ITS frameworks by: (1) presenting a method for naturally integrating instructors into the tutoring process; and (2),

continuously improving an ITS's domain knowledge by learning from students' and instructors' interactions with the ITS. By allowing instructors to monitor, evaluate, and contribute to the ITS-student tutoring process, we support intelligent tutor developers in addressing ill-defined domains where knowledge is constantly changing.

The use of students' data to generate the ITS's knowledge-base allows for the identification of unexpected situations, as well as contextualization of the domain knowledge to specific audiences. By using this framework, well-defined problems can also be considered in order to filter noisy or inaccurate demonstrations. Educators who explore the advantages of such an approach will be able to analyze past learning experiences and identify unsuitable demonstrations. By adding two interactive modes to support cognitive processes: instructor-student, and intelligent tutor-student, we leave outliers and pedagogically interesting situations to the human instructor to handle and routine situations to the ITS.

We also introduce a novel view of an ITS framework addressing ill-defined domains. The proposed framework consists of six functional characteristics. We envision these future tutoring systems to be equipped with: *effectiveness* and *efficiency* in learning and using the instructor-students demonstrations; *scalability* supporting the expansion of their knowledge-base; *portability* based on a framework capable to be used in different domains; *metacognitive* skills to understand their existing knowledge level; and, *robustness* that allows them to handle different student backgrounds and instructors' pedagogical perspectives. Our final contribution, is a new and effective method to represent an ITS's knowledge-base consisting of sequential data. The

proposed knowledge representation method is primarily designed for ill-defined domains.

F. Dissertation Structure

The remainder of the dissertation is organized as follows. In Chapter II we give an overview of ITS knowledge learning methods, LfD, and approaches similar to the mixed-response approach. Then, in Chapter III we outline the implementation methods used for the proposed framework and the ill-defined domain (*cybersecurity*) used to test the ITS framework. Also, we describe the approaches (Weighted Markov Model and Weighted Context Free Grammar) used for knowledge representation. In Chapter IV we present the procedures and methods used to evaluate the mixed-response ITS framework. We also describe the evaluation results. Finally, we present conclusions regarding the framework's functionality and future work in Chapter V.

CHAPTER II

RELATED WORK

A. Intelligent Tutoring Systems

Intelligent Tutoring Systems are computer-based tutors that, after learning domain and tutoring knowledge from experts, can provide customized instruction to students. Once ITSs are developed, they can provide one-on-one instruction similar to human instructors with relatively lower cost and total flexibility in time and location (Lu, 2006).

The term Intelligent Tutoring Systems was first coined by Sleeman and Brown (1982) as an evolving term of Intelligent Computer-Aided Instruction (ICAI). The term ITS was quickly adopted by the ITS research community to clearly differentiate ITSs from Computer-Aided Instruction (CAI). The latter often referred to the general use of computers in education (Nwana, 1990), whereas, an ITS makes use of artificial intelligence to provide customized instruction. A set of distinct features that distinguish ITSs from CAI systems have been identified as illustrated in Table 1 (Woolf, 2009): generativity, student modeling, expert modeling, mixed-initiative, interactive learning, instructional modeling, and self-improving. Woolf (2009) presents a recompilation of these seven features and a set of ITSs for which they are exemplified. However, she concludes that few ITSs contain all of these features, and current knowledge of intelligent tutors' abilities to address each feature ranges from simple to sophisticated, emphasizing that more research on artificial intelligence, cognitive science, and education is required to address all of them.

Table 1
Artificial intelligence features of intelligent tutors

Features of Intelligent Tutor	Description of Feature
Generativity	The ability to generate appropriate problems, hints, and help customized to student learning needs.
Student modeling	The ability to represent and reason about a student's current knowledge and learning needs and to respond by providing instruction.
Expert modeling	A representation and way to reason about expert performance in the domain and the implied capability to respond by providing instruction.
Mixed-Initiative	The ability to initiate interactions with a student as well as to interpret and respond usefully to student-initiated interactions.
Interactive learning	Learning activities that require authentic student engagement and are appropriately contextualized and domain-relevant.
Instructional modeling	The ability to change teaching mode based on inferences about a student's learning.
Self-improving	A system's ability to monitor, evaluate, and improve its own teaching performance based on its experience with previous students.

This table was obtained from "Building Intelligent Interactive Tutors" by Beverly P. Woolf (MacMillan Publishing, 2009).

These distinguishing features mainly emphasize that an ITS should: (a) effectively and efficiently generate customized help to student learning needs based on a correct representation and reasoning of the learned students' and instructors' knowledge; (b) consist of domain-relevant activities and an interactive learning environment; (c) have the ability to initiate or respond to interactions with students; and (d), be able to self-improve its teaching performance based on observed students' activities.

In this research we complement and enhance the currently identified ITS features with our six proposed characteristics (i.e., *effectiveness*, *scalability*, *efficiency*, *portability*, *metacognition* and *robustness*). For instance, we emphasize the significance of the ITS *effectively* and *efficiently* learning and using its knowledge. However, for ill-defined domains, we suggest that this characteristic can only be accomplished through continuous participation from human-instructors within the tutoring process. We visualize *robust* expert models adapting to varying instructor pedagogical perspectives, handling inconsistencies in input data, and equipped with a *scalability* feature for continuous knowledge growth accommodating new learning needs. Finally, even though an ITS can self-improve its teaching performance based on observed student experiences, the intelligent tutor should be designed with *metacognitive* skills to evaluate its knowledge level in order to trigger a correct tutoring response. For example, if the ITS knowledge confidence level is high based on previously learned domain knowledge and observations of previous students, then the ITS will provide feedback to the student. Otherwise, the intelligent tutor will request help from a human expert in order to answer students' tutoring requests and learn from observed instructor-student interaction.

VanLehn (2006) introduced a new description of how tutoring systems behave. He emphasized that, even though ITSs differ widely in their task domain and the implementation of the intelligence features mentioned above, ITSs tutoring behaviors are quite similar. He describes tutoring systems as having two loops: the *outer loop* and the *inner loop*. The outer loop decides which task the student should do next based on the student's background or prior student performance. A task is considered an activity or

exercise intended to get students to learn specific domain content or skills. The inner loop is executed within a task. This loop is executed once for each step taken by the student in the solution of a task. A step is represented by the actions the student performs while solving a problem. Within the inner loop the intelligent tutor has a closer tutoring interaction with students by analyzing students' steps, giving different levels of feedback (i.e., minimal, detailed, or error specific feedback), hints on the next step, and assessing students' knowledge. VanLehn (2006) emphasizes the relevance of the inner loop functionality; he classifies systems without an inner loop as CAI systems and those with one as an ITS. Hence, the focus of this research is on the enhancement of the inner-loop functionality, which we refer to as the *tutoring loop*.

B. The Architecture Supporting an ITS

When master instructors teach their students, they tend to use already acquired domain knowledge and teaching strategies. Based on observation of a student's learning behavior, instructors decide what knowledge he or she is missing and the best method to convey that knowledge (e.g., through a hint, question, example, or any other teaching strategy). Simultaneously, instructors estimate and consider student differences with regards to knowledge background, abilities, and learning styles.

ITSs should represent how people teach and learn. Knowledge about the domain, students, and teaching strategies should be represented and stored in the ITS's backend. This information should be efficiently conveyed through a user interface (frontend). This

knowledge is commonly organized within an ITS in four modules: *domain*, *student*, *teaching*, and *interface* or *communication*.

Domain-knowledge, commonly known as *domain-model*, represents the knowledge acquired from observing how experts perform and solve problems in the domain.

Student-knowledge, commonly known as *student-model*, represents a student's level of mastery of the domain and contextual information (e.g., possible misconceptions, time spent on problems, help requested, and correct answers). This module allows the ITS to deliver customized instruction.

Tutorial-knowledge, commonly known as *teaching-model* or *pedagogical-model*, represents teaching or tutoring strategies and includes methods for feedback reasoning (e.g., what type of feedback to give, when and how to give feedback, and appropriate amount of feedback).

Communication-module represents methods for monitoring student activity and for communication between student and the ITS (e.g., text-based dialogues, animated agents, natural language communication, interactive screens, and multiple-choice or open-ended questions).

These modules interact with each other before a tutoring response is produced (Woolf, 2009). The modules' interactions are referred to as the *tutoring loop*. Figure 1 illustrates the four modules present in an ITS and their interrelationship representing the tutoring loop.

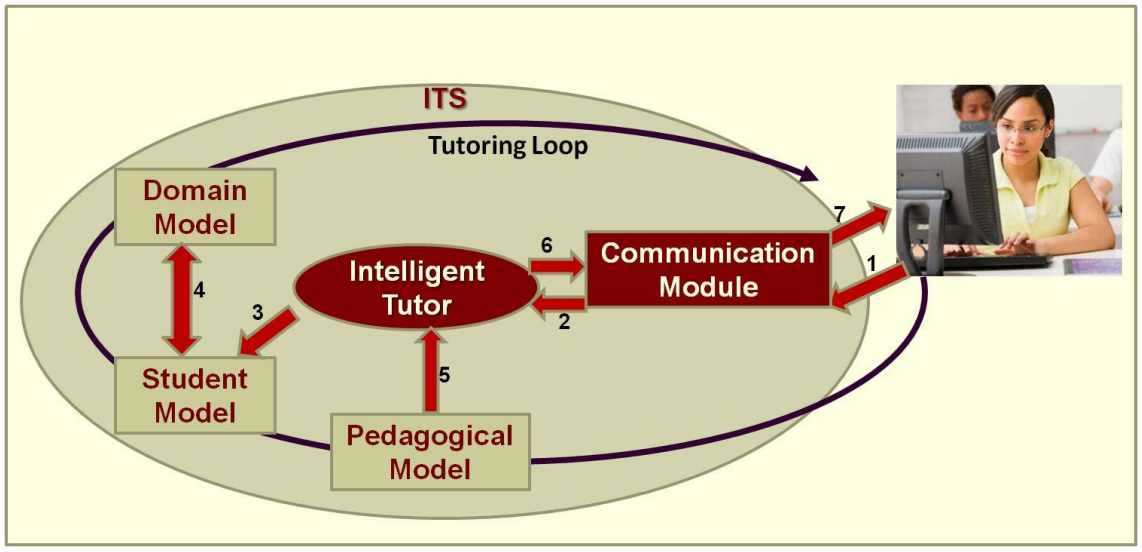


Figure 1. Intelligent tutoring loop. (1) The ITS monitors and logs the student problem solving performance; (2) The intelligent tutor receives the student activity and (3) builds or updates the student model; (4) The ITS is continuously comparing the student problem solving state and the learned domain knowledge; (5) If the ITS detects a student misconception or the student asks for help, the ITS selects the tutorial action based on the current student state; and, (6,7) The ITS interacts with the student providing the selected feedback.

In this research we are principally interested in the construction of the *domain-model*. Specifically, we are interested in developing *effective* and *efficient* methods for gathering and representing domain knowledge for ITSs aiding students learning in ill-defined domains. These methods are based on the premise that, within domains addressing ill-defined problems, the participation of human-instructors is a vital and constant necessity.

C. Knowledge-base Modeling Approaches

Two of the most frequently used approaches for constructing the *domain-model* of Intelligent Tutoring Systems are *Model-tracing* (Anderson, 1996) and *Constraint-based*

Modeling (Ohlsson, 1994). In this section we describe the theory and rationale behind these two approaches, as well as their limitations when it comes to the implementation of ITSs within ill-defined domains.

1. The Model-tracing Approach

The *Model-tracing* approach is based on the ACT-R (Adaptive Character of Thought-Rational) theory (Anderson, 1996). This theory assumes that a person performs a task based on a set of discrete operations. *Model-tracing Tutors* implement the ACT-R approach typically by representing task models (problem solving steps) using a set of *production rules* (e.g., cognitive tutors). A production rule is commonly represented by *if-then* statements (e.g., IF <condition> THEN <conclusion> or IF <condition> THEN <action>). Task models are usually built from cognitive task analysis.

Production rules intend to describe, in a declarative and procedural way, the correct solution steps and frequent misconceptions of students. Production rules representing misconceptions are known as "buggy" rules and are usually associated with feedback actions. *Declarative production rules* represent propositional knowledge (e.g., knowing *that*) while *procedural production rules* represent skill or performance knowledge (e.g., knowing *how* and *when*) (Anderson, 1983). Typically, levels of dependency among production rules exist; hence, one production rule triggers one or more additional production rules. Normally, the complete set of production rules is provided by an expert at design time, whereas, a student's cognitive model is created while the student interacts with the system. The Model-tracing approach consists of

tracking the production rules applied by the student to generate the student's cognitive model.

An example of a set of correct and incorrect production rules addressing a configuration task within the cybersecurity domain is shown in Figure 2. These production rules are addressing a security requirement intended to prevent an internal computer network from accepting traffic considered to be spoofed or unsafe. In order to configure security devices to filter pre-specified traffic, the *iptables-configuration* command is used.

(a)
IF a security requirement is to deny spoofed packets
THEN the *iptables-configuration* denying spoofed packets should be first in the configuration sequence

(b)
IF the goal is to enter an *iptables-configuration* denying spoofed packets
AND spoofed IP addresses are known
THEN the *iptables-configuration* should REJECT spoofed IP addresses

(c)
IF the goal is to enter an *iptables-configuration* denying spoofed packets
AND spoofed IP addresses are known
THEN the *iptables-configuration* should DROP spoofed IP addresses

Figure 2. Examples of production rules for a packet filtering requirement. (a) *Declarative production rule* exemplifying a correct statement; (b) *Procedural production rule* exemplifying a correct action; (c) Procedural "buggy" rule exemplifying a common incorrect use of REJECT and DROP parameters.

This approach can easily be applied to less complex domains where the set of solution paths and common misconceptions are well known. In those cases, the production rules can be straightforwardly implemented. However, there are a few

limitations to using this approach. The cost of implementing the set of required production rules can be extremely high, specifically, when ITS designers try to address all potential student misconceptions. Anderson, Corbett, Koedinger, and Pelletier (1995) and Koedinger, Aleven, Heffernan, McLaren, and Hockenberry (2004) reported a significantly large amount of time required to produce a set of production rules. Anderson et al., (1995) reported a ratio of 10 hours of work to produce a single production rule. In addition, production rules tend to be tied to the design context and, within ill-defined domains, cognitive tasks (problem solving steps) are regularly hard to obtain and reduce to *if-then* statements (Woolf, 2009; Fournier-Viger et al., 2010). For instance, for the ill-defined *cybersecurity* domain addressed in this study, specifically for the security requirement addressed in Figure 2, a number of situations can cause students' misconceptions while building the spoofed packets *iptables-configuration* rule. In addition, an organization's security policy can consist of up to thousands of security requirements. This implies a multitude of possible rule constructions and rule sequences in order to implement the final configuration, which only adds to the time and cost for designing an ITS supporting this domain.

2. The Constraint-based Modeling Approach

This approach to building *domain-models* is founded on the rationale that effective tutoring can be achieved by examining the problem state at which a student arrived, regardless of the path taken (Ohlsson, 1994). Contrary to the *Model-tracing* approach, *Constraint-based Tutors* build the *domain-model* based only on declarative

knowledge representing correct solutions. Correct solutions are specified by a set of constraints representing domain principles. Each constraint principally consists of a *relevance condition*, a *satisfaction condition*, and a feedback action. The relevance condition describes the action or step the student is trying to accomplish and the satisfaction condition specifies the expected answers or actions. Whenever a satisfaction condition is violated based on the relevance condition, the ITS provides the student with the feedback action assigned to the violated constraint (Fournier-Viger et al., 2010). An example of a constraint consisting of the relevance condition and satisfaction condition using the same security requirement in Figure 2 is shown in Figure 3.

Relevance Condition:
The student is configuring a security requirement to deny spoofed packets and the spoofed IP addresses are known

Satisfaction Condition:
The *iptables-configuration* should REJECT spoofed IP addresses and be first in the configuration sequence

Figure 3. Example of a constraint representing a correct solution.

The focus of the *Constraint-based* approach is on assessing the student's current solution state instead of the entire solution path (how the student arrived there). This approach is advantageous for ill-defined domains in which there are no clear or ultimate strategies to solve a problem (Fournier-Viger et al., 2010). Therefore, the correctness of a student's solution can be based on whether or not the solution satisfies the set of predefined constraints. Another benefit to using this approach is the reduction in time required to build the *domain-model*. The amount of building time is reduced because the

approach does not take into account the representation of potential student misconceptions. In other words, the approach only considers the identification of constraints that should be fulfilled to accept a solution as correct. Mitrovic (1998) reported that on average, 1.1 hours were necessary for implementing and testing a constraint in the SQL-queries domain. This amount of time is significant when compared to the 10 hours reported by Anderson et al., (1995). However, the time required for the constraints generation is influenced by the intended domain. Therefore, the time reduction achieved by using this approach cannot be generalized for every domain. Hence, the cost in design time required for developing an ITS using this approach is still considered a necessary research topic.

Furthermore, this and the Model-tracing approach are in general used to build *knowledge-bases* only during the design phase (Lynch et al., 2006; Koedinger et al., 2004), which limits the size of the *domain-model* as a result. Finally, because of the many acceptable solutions and uncertainty of constraints supporting a solution, this approach has not been widely adopted for ITSs supporting students learning within ill-defined domains. However, there does exist some evidence of successful implementations of ITSs for ill-defined domains using this approach. For instance, the KERMIT/EER tutor used to teach how to design *entity-relationship* diagrams, a database modeling method (Mitrovic, Martin, & Suraweera, 2007). Yet, the way in which developers of the KERMIT/EER tutor handled the inestimable amount of acceptable solutions was accomplished by designing the system interface to restrict the range of possible students' solutions (Fournier-Viger et al., 2010), a significant limitation that

must be addressed. More examples of ITSs addressing ill-defined problem solving using conventional and novel knowledge representation approaches are presented in the next section.

D. Successful ITSs Supporting Learning in Ill-defined Domains

Most of the current ITSs are effectively and efficiently supporting students learning to solve well-defined problems, consequently having a tremendous positive impact on education. Empirical research reports one or even two standard deviations of improvement in students' learning outcomes for well-defined problem-solving as a result of using an ITS (Woolf, 2009). Given the characteristics of well-defined problems, developers of ITSs supporting learning for this type of problem solving are able to acquire and represent expert and student knowledge in a comprehensive way. Therefore, students are able to receive precise feedback for the majority of misconceptions that occur. There are also research studies describing successful implementations of ITSs designed for assisting students with addressing ill-defined problem solving as well (Fournier-Viger et al., 2010, Noronha & Fernandes 2005, Noronha, Galante, & Fernandes, 2005). However, for ill-defined problems, it is difficult to acquire and represent expert and student knowledge using conventional knowledge representation methods. During the past few years, a group of researchers have been working in this subfield of ITSs. They are researching the adaption of traditional tutoring models to ill-defined domains and developing novel methods for data modeling and feedback inference. This research is occurring in domains such as language learning, medical

diagnosis, legal reasoning, and database design (Aleven, Ashley, Lynch, & Pinkwart, 2008). Examples of systems using conventional and new approaches and the techniques used to generate and represent the domain knowledge are described next.

Walker, Ogan, Aleven, and Jones (2008) presented two adaptive support approaches for asynchronous online discussion in an ill-defined domain, intercultural competence: *individual support* and *peer moderator*. The ill-definedness of this domain resides in the nature of a forum where many valid arguments can be proposed, yet there is not a single right solution. Consequently, forums are open to a variety of discussion threads based on different participants' background, viewpoints, and interpretation of facts. The researchers implemented a decision tree model used to assess a discussion post and provide feedback. The decision tree was populated with a set of elements a correct post should consider (e.g., on-topic, correct facts, and good argumentation) and a set of predefined feedback messages (considering positive and constructive criticism for unsatisfactory posts). To identify these elements the researchers performed a quantitative and a qualitative analysis. Then feedback was developed by consulting an expert.

After the *knowledge-base* was populated and feedback generated, the two approaches were used for ITS interaction with students while they participated in the forum. The *individual support* approach assessed the students' posts by using a keyword analysis algorithm to identify which element in the tree was not satisfied. After that, the system provided feedback by randomly selecting a feedback message from the set addressing the missing element. The *peer moderator* approach considered students moderators from the class to assess the discussion posts and provide feedback.

Moderators selected a post from the board, and then were asked to assess the post by answering *yes* or *no* questions about which elements were correctly addressed. The system provided a feedback template consisting of potential positive and constructive criticism statements according to the unsatisfied elements. The students needed to fill the template with specific details they wanted to address from the post they were reviewing. Students' discussions and moderators' feedback were used as new demonstrations to update the ITS *knowledge-base*.

The results of the study indicated that the *peer moderator* feedback was better than the *individual support* feedback and was much more inconsistent in quality. The inconsistency was perhaps because the researchers used students as moderators and feedback providers in the *peer moderator* approach. Given that student moderators are learning at the same time as their peers, their interpretation of satisfied or unsatisfied elements might vary as they gain experience, thus influencing the type and quality of feedback they provide over time.

We agree with two of the premises of this study: (1) providing human support to perform tasks that an automated system is unable to do could definitely augment the system's effectiveness, and (2) the corpus of new input data from students using the system and moderators ratings and feedback might serve as new training data for implementing a comprehensive intelligent support. In this way, the ITS's intellect will continuously grow supporting the scalability characteristic envisioned for intelligent tutors within ill-defined domains. However, contrary to Walker et al., (2008), we consider the integration of instructors, not students, as human experts providing

feedback for novel situations and moderating the ITS performance. Integrating the instructor support to the *tutoring loop* will increase the consistency in the feedback quality.

CanadarmTutor is another example of and ITS supporting learning in an ill-defined domain (Fournier-Viger, Nkambou, & Mephu-Nguifo, 2009). This intelligent tutor helps astronauts learn how to operate a robotic arm in a 3D simulated environment in the International Space Station. The robotic arm is only visible through several cameras available within the spatial environment. Learners should determine an appropriate set of arm movements to put the arm into an intended final position and use the correct cameras to visualize their performance. Movements should avoid collisions with elements within the spatial environment as well as hazardous configurations. Even though the learning goal is to move the arm from an initial state to a final state, the learning task is considered ill-defined because there are a limitless number of solution paths (seven joints arm), there is no standard way to determine legal movements for shifting from one state to the optimal next one, and contextual characteristics of subspaces influencing the arm's manipulation (Fournier-Viger et al., 2010). The significance of analyzing this ITS's performance is that its knowledge representation and reasoning has been implemented and tested using several conventional and novel knowledge representation models. Therefore, empirical research regarding benefits and drawbacks of those approaches are available.

The first tested implementation was based on an *expert system* approach in which a *path-planner* was used (Kabanza, Nkambou, & Belghith, 2005). An expert system

approach allows the ITS to generate expert solutions. Then, the ITS compares the generated solutions with the student's solution; differences between the generated and student solutions are used to determine the required feedback. This approach allows the ITS to produce demonstrations of correct and incorrect motions and entire solution paths, as well as track students' solutions. However, some demonstrations generated by the ITS cannot be realistically performed by a human, and it is impossible to estimate student's knowledge gaps (Fournier-Viger et al., 2010).

The second knowledge representation implementation was based on a *Model-tracing* approach (Fournier-Viger, Nkambou, & Mayers, 2008a). A cognitive task analysis was generated from demonstrations of solution paths performed by real learners. Researchers divided the entire 3D learning space into 3D subspaces. The *declarative knowledge* was represented as relationships among the 3D subspaces, spatial modules or elements within subspaces, and the cameras used to visualize elements within subspaces. The *procedural knowledge* was represented as a sequence of cameras used to visualize the subspaces, calibration of camera parameters, and the set of subspaces that should be visited to reach the final state. Using the Model-tracing approach, the ITS was able to afford additional tutoring services such as providing learners access to use the declarative and procedural knowledge stored by the ITS, evaluating the learners' knowledge level, generating customized exercises based on the estimated learners knowledge level, and giving proactive feedback.

The knowledge regarding sequences of cameras and subspaces that might be used and visited to reach the intended location was practical to represent by using a task modeling approach.

To address the limitation that the Model-tracing approach could not overcome in representing arm movements by using explicit task models, the researchers implemented a third approach based on the automatic generation of *partial task models* from learners' demonstrations while completing an exercise (Fournier-Viger et al., 2009). First, they designed the data structure to store and organize data supporting the required granularity. The data structure consisted of a sequence of events saved in a database. Each event represented a set of actions performed by the learner (e.g., camera selection, camera adjustment, arm joint rotation). Researchers identified 112 different actions. The events within the database included sequences of actions differing in length and content. As a result, a dataset consisting of sequential data was generated from learners. Then, sequential pattern mining algorithms were used to identify frequent *sequence patterns* or subsequences. Subsequences were automatically annotated based on whether or not they were part of successful solution paths. Additional manual annotations could be easily added to detected patterns as well. Finally, the generated *knowledge-base* was used to identify learners' expertise regarding the arm manipulation and provide tutoring services. For instance, the tutor might propose the next possible arm rotation movement and allow the learner to explore the learned patterns. The one limitation identified for this third approach was the ITS's inability to offer any support to the learner for unexplored solution paths (Fournier-Viger et al., 2010).

Based on both observed advantages and disadvantages of each prior knowledge representation approach, the researchers implemented a final hybrid approach. A preliminary evaluation was conducted using a small sample of learners who had previous experience using earlier versions of the *CanadarmTutor*. Participants reported a more comprehensive tutoring environment. The developers of the *CanadarmTutor* also presented a compilation of successful ITSs using conventional as well as original knowledge representation approaches (Fournier-Viger et al., 2010). However, they suggest further research using hybrid models to implement ITSs supporting learning within ill-defined domains. They believe that using hybrid approaches will help ITS designs complement tutoring services and overcome downsides to each approach. In addition, further investigation on effective domain-general approaches to represent domain knowledge was recommended (Fournier-Viger et al., 2010).

Even though previous studies describe methods for adapting conventional knowledge representation approaches to implement intelligent tutors supporting learning in ill-defined domains; and despite the recommendation of hybrid approaches as an excellent method for building these types of systems, Machine Learning (ML) and Data Mining (DM) techniques are clearly successful in identifying the unexpected situations present in ill-defined problems and generating effective and efficient tutoring. Furthermore, effective data structures representing the gathered knowledge, different from those used in conventional approaches (e.g., *production-rules*, *constrain conditions*), should be defined. Finally, user (learners and instructors) participation is vital to obtaining relevant domain knowledge. Learners continuously provide

unexpected solution paths based on their unique backgrounds and contexts and who better to scaffold students in these situations than the human expert themselves? However, remarkably little attention has been given to the integration of the human tutor to the tutoring process during the use of the ITS. We take advantage of active instructor participation within the tutoring loop as a consistent feedback source to novel situations.

Our research is based on the integration of these three elements: (1) implementation of machine learning techniques for data collection and knowledge representation; (2) design of efficient and effective data structures to represent the learned knowledge; and (3), collection of knowledge from ITS learner and instructor demonstrations. We show how the integration of these three features successfully addresses the six functional characteristics (e.g., *effectiveness*, *scalability*, *efficiency*, *portability*, *metacognition* and *robustness*) envisioned for novel ITS frameworks supporting learning within ill-defined domains. We conclude this chapter describing ITSs building domain knowledge from data consisting of students' and instructors' demonstrations and we describe an alternative approach based on ML techniques.

E. Building Domain Knowledge from User Data

Building the domain knowledge and tutoring strategies from both histories of student activities and the feedback provided by the human instructor is an advantageous approach. Often times, instructors detect unexpected problem solving situations and students' misconceptions at run time. The instructor recognizes that the student has entered an unexpected problem solving state, uses his/her expertise to analyze the

unexpected state, and provides the consequent feedback to the student. Therefore, monitoring student activity and instructor feedback is a meaningful source for detecting and recording new problem solving situations within ill-defined domains.

1. Students Data-based Knowledge Learning

Generating domain knowledge from student action data is a new method that affords identification of common and unexpected students' solution sates. Recently, similar to one of the versions of the *CanadarmTutor* mentioned above, a number of ITS developers have been considering building ITS *knowledge-bases* through observation of students' activity (Amershi & Conati, 2009; Nkambou, Mephu-Nguifo, & Fournier-Viger, 2008; Jarvis, Nuzzo-Jones, & Heffernan, 2004; McLaren, Koedinger, Schneider, Harrer, & Bollen, 2004; Blessing, 1997). Research using this approach has shown meaningful evidence regarding the building of domain knowledge and tutorial actions from data (*data-based models*) instead of relying on recalled direct experience from experts.

Nkambou, Mephu-Nguifo, Couturier, and Fournier-Viger (2007) for example, describe an ITS system that builds a behavior graph (BG) by inferring frequent action sequences from student activity. Sequential pattern mining is used to identify frequent action sequences, and then association among patterns is obtained (Agrawal & Srikant, 1995). Behavior graphs are used within the ITS field to represent all the possible correct and incorrect paths a student can take while solving a problem. BG nodes are commonly annotated with instructional or feedback messages. Similarly, McLaren et al. (2004) describe Bootstrapping Novice Data (BND), which aids the authoring of ITSs by directly

leveraging student interaction log data. The BND approach records all the actions of students in a log file, which is then used to create a behavior graph. Also, Bernardini and Conati (2010) present a data-based framework to train the ITS. They identify common interaction behaviors from logged students' data within an exploratory learning environment. In such environments students learn while freely experiencing the environment; this type of interaction makes it hard to predict the many user behavior possibilities. A similar approach to learning domain knowledge from students' data can be found in Mostafavi & Barnes (2010) and Johnson & Barnes (2010).

In contrast to the approaches mentioned above, we propose to integrate observations from both students and instructors. This is particularly important when the ITS's *knowledge-base* is not good enough to associate current tutorial actions to a new or unexpected student behavior such as those present within ill-defined domains.

2. Students-instructor Interaction Data-based Knowledge Learning

By monitoring students' exploration of the learning environment and the actions they take to solve a problem, relevant data is automatically collected and clustered. Afterward, in off-line mode, clusters including similar solution paths are labeled as correct or incorrect. Then, clusters labeled as incorrect are annotated with the required feedback by experts. Finally, the gathered domain and tutoring knowledge is used to classify students' new solution paths and scaffold them in their problem solving process.

However, remarkably little attention has been given to the integration of the human tutor to the tutoring process. Only a few research proposals emphasize the

importance of considering the human tutor intervention within the tutoring process (Segedy, Sulcer, & Biswas, 2010; Johnson & Barnes, 2010). These researchers propose human tutor participation based on instructors monitoring student activity through a "control panel." We suggest a more flexible interaction approach, where the ITS and human tutor interact and contribute to the tutoring task what each does best. We refer to this approach as mixed-response interaction. The implementation of this approach considers a metacognitive capability that allows the ITS to interact with students when its knowledge confidence level is high. Otherwise, the instructor takes control of the tutoring task and the ITS learns from the instructor-student interaction based on Machine Learning techniques.

Similar approaches considering a continuous participation of experts in the knowledge-base generation are present in other type of intelligent systems. For instance, within computer-based design environments an evolutionary approach for knowledge-base generation was proposed by Fischer, McCall, Ostwald, Reeves, and Shipman (1994). Design domains are considered highly ill-defined because they are open to the users' creativity while building artifacts or abstract objects (Fournier-Viger et al., 2010). Fischer (1997, p.9) emphasizes that knowledge acquired at design time from experts "will never be complete because domain knowledge is tacit" and that additional knowledge is always activated when practitioners face new situations. The proposed approach considers three steps of information acquisition: *seeding*, *evolutionary growth*, and *reseeding*. The seeding step is the acquisition of domain specific knowledge from interactions between environment developers and domain experts at design time.

Evolutionary growth refers to domain designers' use of the environment to collect novel or unexpected situations generally consisting of partial designs, annotations, and discussions. Finally, at the reseeding stage, environment developers and domain experts interact again to structure, generalize, and formalize the newly gathered knowledge (Fischer et al., 1994; Fischer, 1997).

A similar approach considering continuous expert participation to enhance the knowledge-base quality of an intelligent system is presented by Ruiz-Martínez, Valencia-García, Fernández-Breis, García-Sánchez, and Martínez-Béjar (2011). The study describes ontology construction within the biomedical domain. Researchers emphasize the significance of using ontologies (a set of primitives modeling domain knowledge) for representing human knowledge, yet the construction significantly consumes both time and resources. Ontology learning is a new research area aiming to automate the ontology building process. However, there are still drawbacks to current ontology building approaches regarding the completeness and quality of the resulting ontologies. The proposed ontology learning method consists of a set of automated steps that produces the ontology elements (e.g., primary concepts, temporary concepts, relations) from an input sentence. Then, the generated ontology is reviewed, complemented, and improved by an expert. The human expert supervises the extracted knowledge every time the ontology-building framework processes a piece of text. Ruiz-Martínez et al., (2011, pp. 12366) emphasize that, "*the more knowledge the knowledge-base contains the less work the expert has to do.*"

Both cases mentioned above consider the importance and take advantage of the continuous participation of experts within the knowledge-base generation. However, while the knowledge-base generation approach described for the computer-based design environment is based on asking for domain expert participation after an undetermined time of system use, the ontology learning approach is asking the expert to review the intelligent system outcome every time it processes an input. We propose the implementation of a metacognitive skill to allow the intelligent system to determine and inform the expert when support is needed.

F. Machine Learning in Support of Knowledge Acquisition

For ill-defined domains, constructing a comprehensive ITS domain-model using the aforementioned approaches is a difficult and time consuming activity (Fournier-Viger et al., 2010). Machine Learning is a subfield of Artificial Intelligence that is concerned with the study of methods for programming computers to autonomously acquire and integrate knowledge. ML is a beneficial technique that can help to identify the most relevant correct and incorrect problem solving situations and build a partial, but potentially effective, domain-model. Algorithms and techniques already available within the machine learning field can be adopted and adapted to make an ITS learn and infer learning behaviors from empirical data at design and run time. Using this approach could significantly reduce the time and cost it takes for building an ITS's domain-model. In addition, by using ML techniques, an ITS has the capability to continuously learn,

adding a more advanced level of intellect and thus can be more applicable for ill-defined domains.

Therefore, some ITS researchers are using ML techniques to obtain the cognitive skills used in problem solving in ill-defined domains in order to populate an ITS's knowledge-base. These techniques are being used for both conventional and novel modeling approaches (Woolf, 2009), as well as combinations of the two known as hybrid approaches (Fournier-Viger et al., 2010). Primarily, these ML techniques are used to learn domain knowledge through observation of students' activities.

With regards to the ITS under investigation here, we employ an ML approach based on supervised learning. By using supervised learning, the ITS generates new knowledge from demonstrations. Thus, in contrast to those methods based only on learning from student performance, our approach considers building the domain-model from demonstrations generated from instructor-student interactions. This will continually expand the ITS's knowledge-base consisting of unexpected situations faced by students and corresponding feedback from the instructor, overcoming the limitation of a finite domain-model and improving the quality of the ITS's knowledge.

1. Learning from Demonstration

Computer learning techniques based on demonstrations are variously called *Learning by Demonstration* (LbD), *Programming by Demonstration* (PbD), or *Learning from Demonstration* (LfD) to name a few (Argall, Chernova, Veloso, & Browning, 2009). Programming by Demonstration is the most commonly used term within the ITS

community. PbD specifically refers to nonprogrammers developing computer applications from demonstrations of what actions are appropriate for the system. Most efforts using PbD within the ITS field are intended to simplify the process of authoring a tutoring system.

SimStudent (Simulated Student) is a Machine Learning-based intelligent tutor that uses PbD to help novice authors to build ITSs within the mathematics domain (Matsuda, Cohen, & Koedinger, 2005). This ITS allows an instructor to construct a GUI for a specific problem, and then use the interface to demonstrate how a student should solve problems. The ITS induces *production rules* (a set of conditions) from demonstrations that replicate the instructor's performance. More recently, *SimStudent* developers evaluated two different ways of training *SimStudent*: by instructor *demonstrating solutions* or by instructor *tutoring problem-solving* to the simulated student (Matsuda, Cohen, Sewall, Lacerda, & Koedinger, 2008). The second approach is a new training method in which the author gives *SimStudent* problems to solve and provides feedback on its solution steps taken. In this new training method, *SimStudent* applies existing knowledge to solve the assigned problem, and then the instructor provides feedback for each performed state. If *SimStudent* encounters knowledge gaps when continuing to solve the problem, it asks for help from the instructor. The instructor then teaches the ITS by demonstrating a correct step. Investigating the effects of the ITS shows that building expert knowledge by using the tutoring problem-solving approach requires less time and tends to be more precise than authoring by demonstrating solutions. Even though this approach is similar to our proposal suggesting the use of

human instructor-student interactions to populate the knowledge-base, we propose real instead of virtual students will help to identify more authentic student misconceptions. In addition, for ill-defined domains, virtual students will rarely be faced with novel states.

SimStudent is part of a set of intelligent tutor authoring tools called CTAT (Cognitive Tutor Authoring Tools). Aleven, McLaren, and Sewall (2009) present CTAT as an ITS authoring application that can use the PbD approach. Aleven et al., (2009) also describe a set of similar systems such as *Assistments Builder*, *ASPIRE*, and the *Task Tutor Toolkit*. Their goal is to allow instructors or ITS developers to build instructional and tutorial material by problem-solving demonstrations through a friendly user interface, without the necessity of learning a specific programming language. However, these demonstrations usually come only from the ITS's authors. Furthermore, problem solving demonstrations are mainly implemented at design time. Our approach proposes the consideration of learning from demonstrations from both students using the system and instructors giving feedback at run time.

CHAPTER III

MIXED-RESPONSE INTELLIGENT TUTORING SYSTEM

This chapter provides a description of the knowledge representation methods and algorithms used in this study for the implementation of the proposed mixed-response ITS framework based on Learning from Demonstration.

A. Mixed-response Framework

The mixed-response interaction we introduce is motivated from the mixed-initiative interaction approach. Allen (1999) describes mixed-initiative interaction as a "flexible interaction strategy in which each agent (human or computer) contributes what it is best suited at the most appropriate time". Mixed-initiative interaction aims to provide an effective multi-agent collaboration to solve a problem or perform a task. Allen (1999) introduced a four level mixed-initiative taxonomy: (1) agent (in our case the ITS) notifies others (instructors) of problems (i.e., student misconceptions) as they arise; (2) agent initiates interaction to clarify a situation; (3) agent takes initiative to solve a situation; and (4), agent coordinates and negotiates with other agents to determine initiative. Within the ITS field, mixed-initiative is commonly implemented to support the interaction between student and intelligent tutor. This interaction aims to provide feedback or help students carry out the problem solving process (Woolf, 2009).

Research work regarding mixed-initiative exists for the design of user interfaces. For the implementation of this mixed-initiative environment, is of relevance the level of automation a system should have to initiate collaboration with users in order to

accomplish the users' intended goals (Horvitz, 1999). Horvitz (1999) presented a set of principles to be considered in order to implement a mixed-initiative user interface that often negotiates uncertainty with regards user's goals. The recommended principles by the researcher important to our work are: (a) implement automated services that provide genuine value over direct user manipulation, (b) provide means for allowing users to analyze and refine agents work, and (c) provide the ability to continue learning by observing.

In our case, the mixed-response interaction is based on who contributes or takes control of the tutoring process. The interaction is between the ITS and the instructor. The intelligent tutor will monitor the learning environment and responding to students' tutoring requests by providing feedback or asking the human instructor for help. Specifically, whenever the ITS's knowledge confidence level is not sufficient to respond to a particular request from a student, the ITS will respond in a way that satisfies the mixed-initiative taxonomy proposed by Allen (1999) in the following way: (1) ITS notifies instructors as problems arise (ITS incapable of assisting with a novel problem solving situation); (2) ITS interacts with the instructor to clarify (contextualize the problem found) the student's problem solving situation; (3) instructor takes initiative to provide feedback; and (4), when the instructor is not available for providing immediate help to students, the ITS will take initiative to provide temporal feedback, warning students of its low confidence level. Specifically, from the principles proposed by Horvitz (1999) we are automating the tutoring of routine pedagogical situations. This is of high relevance for the instructor, so he or she can focus on interesting or novel

situations. Furthermore, instructors will always be able to access the intelligent tutor's activity and refine its work. Finally, the continuous learning approach is a key functionality in our tutoring automation proposal.

Tutoring demonstrations generated by the instructor-student interaction are used by the ITS as the data source for building the domain-model. The construction of the domain-model, and the functionality previously described, is realized by using the Learning from Demonstration (LfD) technique described next.

B. Learning from Demonstration

Learning from Demonstration is an example of a Supervised Learning approach to machine learning. Just as in Supervised Learning, the machine learning algorithm is trained on a set of labeled data. Argall et al. (2009) formally describe LfD as a world consisting of states S (i.e., student solution paths) and actions A (i.e., tutorial actions). The learning algorithm develops a policy $p : S \rightarrow A$ that selects the corresponding action in response to the observed world state. The sequence of policy derivation and its use is depicted in Figure 4.

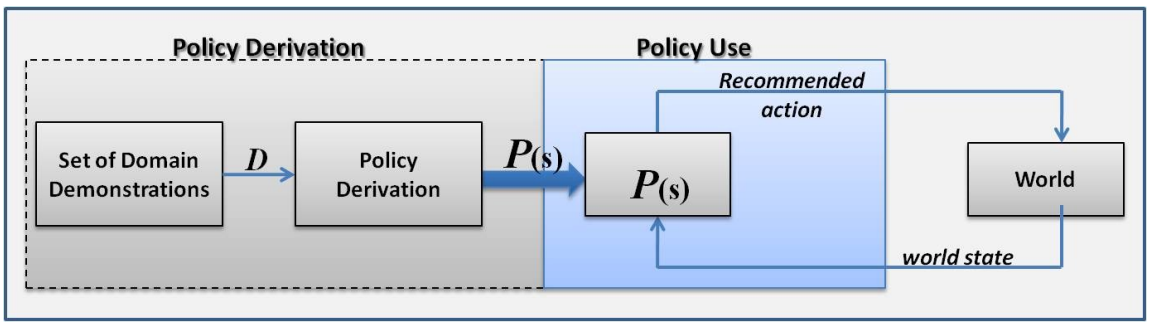


Figure 4. Policy derivation and use cycle.

The *demonstration process* and *policy derivation* techniques are two fundamental design elements that should be considered in order to obtain a policy. These elements are described next.

1. The Demonstration Process

Policy learning through demonstrations can happen at two points in time. If demonstration data is available off-line, batch learning can be applied (*batch fashion*), where the system learns the policy in a more controlled way. On the other hand, when demonstrations are available during run time, on-line learning can be applied (*on-line fashion*), where the system incrementally updates the policy as training data becomes available (Argall et al., 2009). In many cases, batch learning and on-line learning can be naturally combined, where the system learns an initial policy during the batch learning process and the policy is refined while the system is in use by real users. The on-line learning approach is recommended when the training data set is highly redundant (in our case many similar solution paths) and when the domain knowledge to learn gradually changes over time. As we describe later, we allow for a combination of batch learning (based on existing student input) and on-line learning (during system operation). The decision of how much batch learning is needed is based on the number of demonstrations needed to allow a system to start responding to common situations within the intended domain. In this study, we took advantage of both approaches. We had previous data demonstrating the most frequent correct and incorrect students' solution paths, thus, we used the batch fashion approach to train our ITS with this data in

order to generate an initial policy. Then, by using an on-line approach, we addressed unknown situations by asking for instructor support, thereby refining and improving the initial policy, see Figure 5. In Chapter IV we describe results from our experiments testing the framework's performance and speed in building a suitable classification policy starting from scratch using multiple instructors.

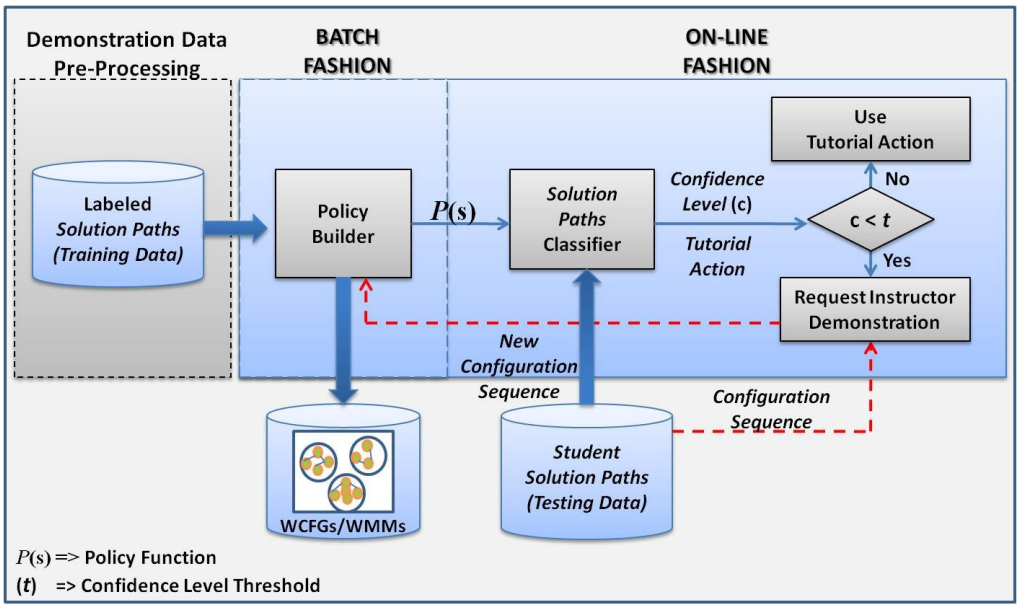


Figure 5. Implemented Learning from Demonstration process.

2. Policy Derivation

Policy derivation within LfD refers to the process of building or refining a classification policy using demonstration data that affords the reproduction of the demonstrated behavior. Policy learning algorithms are defined and use such demonstrations to derive a policy. As Argall et al. (2009) state, the goal of an LfD system is to derive a policy that improves *beyond the teacher's demonstrations*. In addition, minimal policy refining effort, fast learning time, and few training examples are desirable. Three techniques for

policy derivation are primarily used within LfD: (1) learning a simple approximation for a *state – action* mapping (*mapping function*), (2) learning a model of the world dynamics represented by *current-state – action – resulting-state* associations (*system model*), and (3) learning a model of relationships between *pre-conditions – actions – post-conditions* with which an entire solution plan can be built (*planning*) (Argall et al., 2009).

- *Mapping function*. The policy is directly approximated from the demonstration data through a function mapping from any state to the corresponding action ($f():S \rightarrow A$) (For example, a particular history of student activity triggers a particular tutorial action to guide the learner to a goal state.)

- *System model*. The demonstration data is used to estimate a world dynamics model, that is, a state transition function $T(s'|s,a)$. The policy is derived from the world model, often in combination with a state-dependent reward function to select the best action to recommend, as used within Reinforcement Learning. (For example, a particular history of student activity triggers the best tutorial action to take the student to a desired next solution step.)

- *Planning*. The demonstration data and additional user intention information is used to learn associations of *pre-* and *post-conditions* with actions ($L\{(preC,postC\}|a$.) (For example, a particular tutorial action is triggered to lead a learner to a desired next solution step (*post-condition*) only when the learner is currently in a specific solution step (*pre-condition*).)

Each of the policy learning techniques supports the policy inference process differently. The primary goal of the *mapping function* is to reproduce the expert policy to determine which actions correspond to specific states. The generated mapping function approximates the corresponding action to a specific state based on the set of available demonstrations. Therefore, the function generalizes from the demonstrated behavior to infer valid actions for unknown, yet similar states. This approach is particularly attractive to use for learning knowledge for which acquisition of a comprehensive set of demonstrations may not be feasible. The *system model* technique learns the policy by complementing the set of demonstrations with additional system explorations to determine a comprehensive state transition model of the world. The policy is intended to learn all possible associations between states and the corresponding reward when specific actions are used. The main drawback of this approach is that generalization is not often considered. Typically, demonstrations should be provided for every state; this can be extremely time consuming. Finally, the *planning* approach is designed to learn entire solution plans to accomplish a specific task. Learned plans consist of a sequence of actions that represent traversing a path from a predetermined initial state to an anticipated final target state. The sequence of actions is linked by state associations, which are considered as *pre-* and *post-conditions*. Demonstrations used to build a policy via this technique are feasible to acquire for domains where systematic solution paths exist. However, for domains in which knowledge is dynamic and context dependent, gathering a comprehensive set of solution path demonstrations is impractical.

In our context, we are interested in tutoring students based on their current problem solving state; the state is determined by the set of misconceptions present in the student's solution. Our goal was to generalize over the available demonstrations such that actions (tutorial actions) would be used for similar states (*configuration sequences*) encountered in the demonstration dataset. Based on the feedback required for the targeted ill-defined domain, we are not expecting to lead students to a specific solution state or to follow a predefined solution path. We want to provide feedback to aid students in reasoning about their current situation, and guide them in determining the best way to reach a final workable solution. Therefore, we focus on implementing and testing the first policy derivation approach: the *mapping function*. This technique will derive a function ($f():S \rightarrow A$) from the available demonstrations and generalize for similar situations. When the set of available demonstrations is not sufficient to generate appropriate actions for unknown students' solutions, more demonstrations are going to be requested from experts.

3. Intelligent Tutor Metacognitive Skill

Our policy derivation relies greatly on classification of student activities and the selection of corresponding tutorial actions. Classification errors cannot be entirely avoided. When such an error occurs, and no control mechanisms are in place to handle it, the intelligent tutor might select an inappropriate tutorial action, and so further confuse a student. The effectiveness of the ITS therefore relies on its ability to assess its confidence in individual classification results. Chernova and Veloso (2009) present the

Confidence-Based Autonomy (CBA) algorithm for policy learning from demonstration based on the *active learning* approach. Active learning requires experts to continuously participate and label new data from which agents learn (Settles, 2009). In order to most efficiently leverage the domain knowledge, the main task in active learning is to direct the expert to label the most useful examples, aiming to minimize the number of interventions (Chernova & Veloso, 2009).

The CBA algorithm is used to make requests for expert participation based on the knowledge confidence level of the agent. It is implemented by means of two main components: *confidence execution* and *corrective demonstration*. The confidence execution component represents the metacognitive skill of the agent and enables the agent to automatically estimate its confidence level based on previously determined thresholds. If the confidence level is too low, the agent requests a demonstration from the expert and reconstructs the classification policy based on the acquired data. The corrective demonstration component enables the expert to correct any agent misclassification or to provide additional demonstrations in order to reconstruct the classification policy (Chernova & Veloso, 2009).

Similar to the CBA approach, we use confidence-driven LfD to trigger requests for classification from the instructor whenever the agent's classification results indicate low confidence (Chernova & Veloso, 2009; Chernova & Veloso, 2007). Whenever the confidence level resulting from a classification falls below a given threshold, the ITS contacts the human instructor for a demonstrative tutorial action (see Figure 5). This

tutorial action is forwarded to the student and added to the ITS's knowledge-base to be used to further refine future classifications.

4. Data Representation Methods

In many ML settings, in particular classification problems, the system is trained to map from a feature vector space to the solution space. In the type of applications we are considering, constructing a straightforward definition of a feature space is difficult or impossible. This stems from the fact that the types of solutions we find for ill-defined problems consist of open-ended sentences or sequential data differing in length and content. Some examples of ITSs addressing these types of solutions are: *intercultural-competence tutor* (forum posts) (Walker et al., 2008), *CATO* (law argumentation tutor) (Aleven, 2003), *Rashi* (generic domain tutor to generate hypothesis and argumentations) (Dragon, Woolf, Marshall, & Murray, 2006), *CandarmTutor* (sequence of robot arm movements) (Fournier-Viger et al., 2008b), *SQL-tutor* (SQL sentences) (Mitrovic, Martin, & Mayo, 2002), and *LISP-Tutor* (teaching introductory programming) (Anderson, Conrad, & Corbet, 1989). In particular, for our intended domain of cybersecurity, we encountered varying solution paths, in both content and length, from each student. The conversion of this sequential data to vector structures would lead to loss of information (Cadez, Gaffney, & Smyth, 2000). Rather than using feature vectors directly, we use representations that are more appropriate for the type of sequential data encountered in problem solutions within ill-defined domains. In order to model non-vector data, we used two modeling approaches (Weighted Markov Models and Weighted

Context Free Grammars) as the primary clustering and classification policy generation methods. In Chapter IV we validate the capability of these data representation approaches to support the six functional features envisioned for a modern ITS intended to support learning within ill-defined domains. Our experiments will show that for the type of problems addressed in our study (i.e. sequences of devices' configuration commands), Weighted Context Free Grammars perform better than using Weighted Markov Models. In the following section we describe characteristics of the used knowledge representation approaches and implementation algorithms.

C. Weighted Markov Models

Various forms of Markov Models have proven effective in machine learning settings with sequential data. For example, Hidden Markov Models are popular in natural language recognition and speech processing (Rabiner & Juang, 1986). Huang, Yong, Li, and Gao (2008) use Weighted Markov Models for prediction of students' actions and behaviors. Similarly, we use a Weighted Markov Model (WMM) approach for clustering and classification of sequential data. A WMM represents a weighted collection of Markov Models (Huang et al., 2008). We describe our WMM classification method as a set of MMs that, in addition to computing the probability matrix and the initial state probability vector, computes weight values based on the state transitions' frequencies within the entire dataset in order to provide better classification predictions. In a regular Markov Model, we encounter n distinct *states* $S=\{s_1, s_2, \ldots, s_n\}$, a vector of *initial state probabilities* $B=\{b_1, b_2, \ldots, b_n\}$, where b_i denotes the probability of the model being in

state s_i. Finally, the model's dynamics is represented by the *transition probability matrix* $A=\{a_{ij}\}$, where a_{ij} denotes the probability of the system in s_i transitioning to state s_j. During the training step, the ITS is presented a set of labeled student behaviors and develops a set of Markov Models, one for each required tutorial action. At run time the system matches a student behavior against each Markov Model and identifies the best match. Then, the ITS uses the tutorial action associated to the best matched Markov Model to provide feedback to the student. Similar to Huang et al. (2008) we added an initial state weights vector $WB=\{w_{b1}, w_{b2}, ..., w_{bn}\}$ and a transition weights matrix $WA=\{wa_{si}, wa_{sj}\}$ in order to be able to make the classification of new sequences more accurate.

1. Weighted Markov Models Generation

The types of students' solutions we observe in our study consist of a set of *configuration -rules (CRs)* each of them consisting of a set of commands and parameters. For instance, the following four configuration rules used together, which represent a *configuration-sequence* (*CS*), are utilized to deny and accept certain specific network traffic:

```
iptables -A FORWARD -p tcp -dport http ACCEPT
iptables -A FORWARD -p tcp -dport https ACCEPT
iptables -in-interface eth1 -j ACCEPT
iptables -A FORWARD -j DROP
```

The set of symbols (commands and parameters) within a *CR* and the sequence of *CRs* within a *CS* are defined by the type of exercise the student is working on and by the

student's solution reasoning. For instance, if an exercise requires students to only accept web traffic in our network, a correct student solution may be exactly the one showed above or switching the first two *CRs*.

For our study we built a set of Markov Models from networking *CSs* consisting of *CRs*. The *CSs* in our training data were previously categorized a priori by an expert (see Figure 6); the set of *CSs* within each category was represented by a WMM. Each *CR* within a *CS* represents a state in the model. The sequence of *CRs* within each *CS* was used to build the Markov Model's transitions and estimate transition probabilities between states. Weight values were computed for each initial state and state-transition based on the average of occurrence within the entire dataset.

ID	Configuration Sequences	Tutorial Action	ID	Configuration Sequences	Tutorial Action
1	r1, r2, **r3**, **r4**, **r5**, **r6**	T1	11	r1, r2, **r3**, **r5**, **r6**, r2, **r7**	T3
2	r1, r2, **r3**, **r7**, **r5**, **r6**	T2	12	r1, r2, r8, **r5**, **r6**, r2	T4
3	r1, r2, **r3**, **r5**, **r6**, r2, r8, **r7**, r8	T3	13	r1, r2, r8, **r6**	T4
4	r1, r2, **r3**, **r5**, **r6**	T4	14	r1, r2, **r5**, **r6**, r2	T4
5	r1, r2, **r5**, **r6**	T4	15	r1, **r9**, **r10**, r8, r8, **r5**, r8, **r6**, r2	T2
6	r1, **r3**, r2, r2	T4	16	r1, r2, **r3**, **r5**, **r6**, r2	T4
7	r2, r1, r2, r8, **r3**, **r7**, **r5**, **r6**, r2	T2	17	r1, r2, **r5**, **r3**, **r11**, **r11**, r2, **r6**, r2	T5
8	r1, r2, **r3**, **r5**, r2	T4	18	r1, r2, **r3**	T4
9	r1, r2, **r3**, **r5**, **r6**, r2	T4	19	r1, r2, **r3**, **r3**, **r5**, **r6**, r2	T4
10	r1, r2, **r3**, **r5**, r8, **r6**, r2	T4	20	r1, r2, r8, r8, **r7**, r8, r8, **r3**, r2, r8, **r5**, r2, **r6**, r2	T2

(a) Set of Configuration-Sequences

ID	Tutorial Actions
T1	The destination IP address you are using is not representative of the internal network Web server.
T2	Excellent work, all the configuration rules are right.
T3	Deny-All policy should be the last rule in your configuration.
T4	Great, your current configuration rules are right; however you are missing some rules to finish the configuration.
T5	You've entered some incorrect rules. Before you continue entering new rules you should flush and start again.

(b) Expert Feedback

Figure 6. Configuration sequences and their tutorial actions. (a) Set of twenty configuration sequences categorized by tutorial actions (configuration rules are bolded, the rest of them are informative rules). (b) Set of tutorial actions predefined by an expert.

We used training data to build the set of Markov Models as a batch learning process. The set of Markov Models represented the initial learned classification policy. Then, the ITS used this policy to classify new *CSs* and provide feedback to students. As an illustration, consider the data representation and modeling process described next.

Figure 6(a) shows a set of twenty configuration sequences. These *CSs* were reviewed by an expert and assigned a tutorial action to each of them. Only five tutorial actions were needed for the whole set of *CSs* (see Figure 6(b)). In Figure 7 we can see the set of Markov Models generated from data in Figure 6. Each model is representing a tutorial action. Figure 8 shows the transition weights of the Markov Models based on the average of occurrence within the entire dataset. These weight values were also partially influenced by an a priori probability, which were based on the percentage of *CSs* in the entire dataset. When an unknown *CS* arises, it can be categorized with a current tutorial action; in this case, transition probabilities and weights of the corresponding Markov Model will be updated. When a new tutorial action is needed a new Markov Model will be generated. We expect a small set of tutorial actions addressing a number of *CSs*.

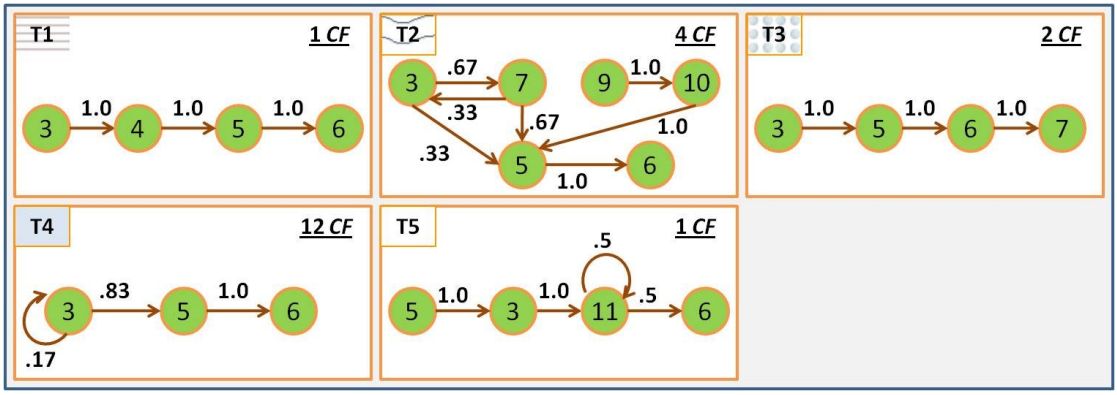


Figure 7. Set of Markov Models with transition probability distributions.

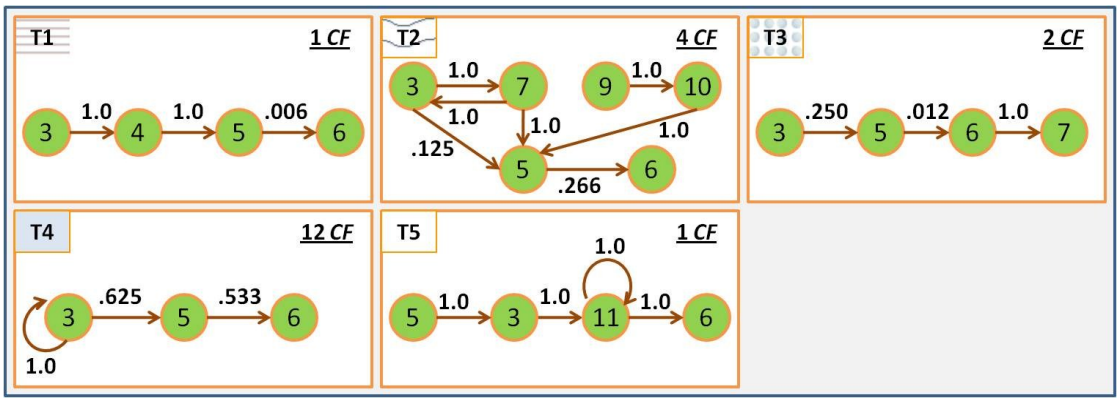


Figure 8. Set of Markov Models with transition weights based on state-transition frequency (e.g. State-Transition 3=>5 has 8 occurrences in the dataset. 1 in T2 (w=.125), 2 in T3 (w=.250), and 5 in T4 (w=.625).

2. Classification Policy Inference

The vector of initial state probabilities B and the transition probabilities matrix A for each of the WMMs are computed using the following equations:

$$b_i = \frac{\beta(s_i)}{\sum_{m=1}^{k} \beta(s_m)} \quad \textbf{(1)} \qquad a_{i,j} = \frac{\sigma(s_i,s_j)}{\sum_{m=1}^{k} \sigma(s_i,s_m)} \quad \textbf{(2)}$$

The term $\beta(s_i)$ represents the number of initial states within a model matching State s_i, and k is the number of states within the model. The term $\sigma(s_i,s_j)$ represents the number of transitions from State i to State j within a model. To compute the initial state weights vector WB and the transition weights matrix WA for each of the models we use the following equations (l represents the number of WMMs within the entire policy):

$$wb_i = \frac{\beta(s_i)}{\sum_{m=1}^{l} WMM^{(m)}\{\beta(s_i)\}} \quad \textbf{(3)} \quad \text{and} \quad wa_{i,j} = \frac{\sigma(s_i,s_j)}{\sum_{m=1}^{l} WMM^{(m)}\{\sigma(s_i,s_j)\}} \; . \quad \textbf{(4)}$$

3. Use of the Classification Policy

A new *CS* from a student is classified by determining the greatest representation and highest similarity between the new *CS* and the *CSs* represented by any of the existing WMMs. We say that a WMM *completely represents* a new *CS* when the model is able to generate the entire *CS*. Similarly we say that the WMM *partially represents* a new *CS* if it only contains part of the *CR* transitions within the entered *CS*. The representation level is determined by the number of *CR* transitions within each WMM. For complete representations, this value should be equal to the entire number of *CRs* of the input sequence minus one. Representativeness is determined by, first identifying those WCFGs that are able to completely represent the new *CS* or those with the higher partial representation, and then by selecting the one with the highest similarity (highest WMM weight value).

The highest similarity is determined by first, identifying those Markov Models that are able to best generate the new *CS* and then, selecting the one with the highest weight value. We compute the weight value by multiplying the transition probability and weight values from each state transition within the selected WMMs dictated by the sequence of *CRs* within the new *CS*. Weight values are computed for all Markov Models by using Equation (5).

$$\mathrm{WMM}_{n=1..l}^{(n)} = (b_{SI} * wb_{SI}) \prod_{i=1}^{k-1} (a_{\mathrm{Si,\,Si+1}} * wa_{\mathrm{Si,\,Si+1}}) \qquad \textbf{(5)}$$

In Equation (5), the term l denotes the number of WMMs within the classification policy, s_i represents the observed states (*CRs*) within the new *CS*, b_{S1} and wb_{S1} are the initial state probability and weight values respectively for the first state in the new *CS*, and k is the number of observed states within the new *CS*.

The classifier uses the largest weight value to select among those WMMs best representing the input *CS*. The output of the classifier is the tutorial action assigned to the selected WMM and the computed representation level. This value represents the confidence level returned by the classification policy. If the confidence level is high, the ITS will initiate interaction with students and provide feedback to them (tutorial actions) based on learned knowledge. When the confidence level is not sufficiently high, the ITS will trigger a help request asking human tutors to lead the tutoring process. When none of the current models is able to completely represent the new *CS*, those models that best represent the new *CS* are considered (*partial representation*). This partial classification approach is only considered when the representation level in the new *CS* is greater than a predefined value (e.g., 65%); otherwise the ITS´s tutoring confidence level will be 0.

In general, the WMMs approach considers the number of associations between two symbols in a new *CS* that are present within the set of previously learned *CSs*. The number of identified associations is used to determine the similarity of new *CSs* with *CSs* within available WMM clusters. Therefore, the classification of new *CSs* is based on independent pairing of symbol associations regardless of the *CR* in which they occur. Using this classification approach we found that some of the new *CSs* were identified as completely represented even though they were not identical to previously learned *CSs*. In

our experiments we realized that a considerable amount of new *CSs* in this situation resulted as incorrectly classified. Based on this WMM's drawback, we implemented an approach using context free grammars, which considers syntactic structures that associate larger amounts or the entire set of symbols within the analyzed *CS*.

D. Weighted Context Free Grammars

Context free grammars (CFGs) are well suited for modeling languages or other structures consisting of sequential data. CFGs describe the atomic units of a language and how these units can be combined to generate legal *expressions* (Tenenberg, 2001). A CFG is formally defined by a quadruple $G = (N,T,P,S)$, where:

- N represents a set of *non-terminal symbols*,
- T is a set of *terminal symbols* (grammar alphabet),
- P is a finite set of *production rules*[1] written in the form of $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)$, and
- $S \in N$ is the (non-terminal) starting symbol.

A *parsing tree* is the syntactic structure of a string generated from a formal grammar. Figure 9 shows an example of straightforward CFG consisting of three *production rules* and two *parsing trees* corresponding to the input sequence "*a b b*". This CFG is able to parse sequences represented for this regular expression: "*ab*+"[2].

---

[1] This term is different from the *if-then* statements used in the *Model-tracing* approach mentioned in Chapter II.

[2] The plus metacharacter implies repetition of the symbol *b* one or more times.

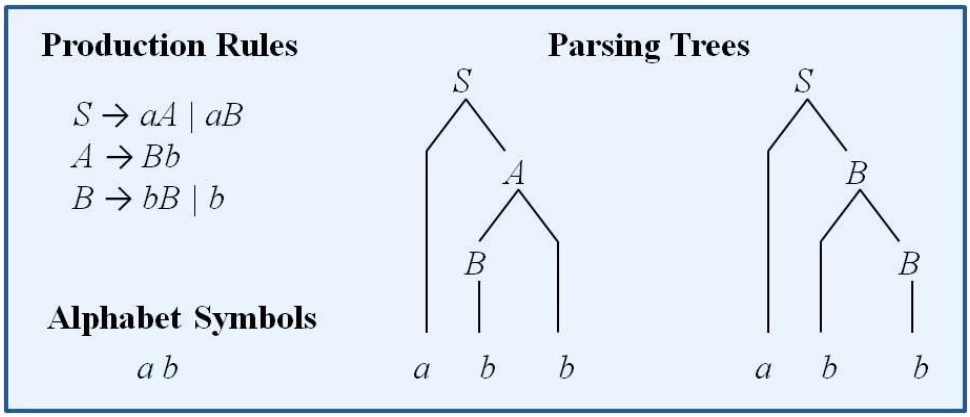


Figure 9. Context free grammar and parsing trees.

A *weighted CFG* (WCFG) is a CFG where each production rule is associated with a positive real value, which we call a *weight* (Chi, 1999). Weighted context free grammars are typically used to associate a particular parsing tree or sub-tree generated in a given grammar with a weight value. The weight of an entire parsing tree is computed by summing the weights of the production rules used to produce a given input string; including the top production rule and its children. Hence, the computed weight for different parsing trees tends to differ.

There is a variant of WCFGs called probabilistic context free grammar (PCFG). PCFGs are characterized by constraining the total sum to be one for the weights of the production rules associated with a non-terminal (Smith & Johnson, 2007). PCFGs are considered a sub-class of WCFGs. However, there is empirical evidence indicating that both approaches are equivalent and equally expressive (Chi, 1999). The weights computed from the generated parsing trees were used to classify new *CSs*. We implement the knowledge representation approach based on WCFGs using the following three steps:

- Generate the WCFGs without using initial *production rule primitives*,
- Infer the classification policy, and
- Implement the classification process using the inferred policy.

Next we will elaborate on each individual step.

1. Weighted Context Free Grammar Generation

There exist several approaches for the inference of CFGs. These approaches are principally based on algorithms that automatically generate production rules from a set of *production rule primitives* (i.e. Operator → + | - | / |*) or directly from a set of target sequence of symbols (i.e. "a + b / c"). Three of the most common approaches are: *trial-and-error* (Duda, Hart, & Stork, 2001), *genetic algorithms* (Mernik, Gerlič, Žumer, & Bryant, 2003; Tsoulos & Lagaris, 2005), and *greedy algorithms* (Nevill-Manning & Witten, 1997; Koncilia & Pozewaunig, 2002). Grammatical inference using a trial-and-error approach refers to guessing new production rules and testing them against correct and incorrect sentences (Duda et al., 2001). Genetic algorithms are a more frequently used method to generate CFGs. Production rules are generated using an evolutionary approach based on the genetic operations *crossover* and *mutation* (Mernik et al., 2003; Tsoulos & Lagaris, 2005). However, examples of production rule primitives are needed to start an evolving loop. The inference of the production rules ends when a grammar is suitably parsing a predefined set of sentences from the target language.

Greedy algorithms do not necessarily require initial production rules. Production rules are generated from scratch based on inferred structures (sequence patterns) found

within target sequences. Specifically, the inference algorithm scans a target sequence and generates production rules based on detected symbols' sequential patterns (Nevill-Manning & Witten, 1997; Koncilia & Pozewaunig, 2002). This is a necessary approach for domains where production rule primitives do not exist such as the network security domain. Since one of the target characteristics of our ITS is its ability to acquire domain knowledge without a priori information, we have to assume that nothing is known about the sequences as well. In particular, there is no prior knowledge that helps us to define production rule primitives. Hence, structure should be inferred from observed sequences, and production rules should be generated based on the inferred structure.

Several algorithms exist addressing this type of grammar inference such as the SEQUITUR algorithm (Nevill-Manning & Witten, 1997). SEQUITUR, which provides the basis for our implementation, is a greedy algorithm that infers a hierarchical structure from a sequence of discrete symbols and replaces repeated sets of symbols with grammatical production rules. Based on a hash-table indexing approach, the algorithm runs in linear-time. Table 2 shows the pseudo-code for the SEQUITUR algorithm (Nevill-Manning & Witten, 1997). The algorithm performs the following operations to infer a grammar from a given sequence: Lines 1 and 2 append new symbols from the input sequence to the start production rule *S* and build new *digrams* respectively. A digram consists of exactly two terminal or non-terminal consecutive symbols, and is used to represent sequences between two symbols. Line 3 is used to identify whether a new digram from *S* already exists within the digram index list. In Line 11, if a new digram does not exist, it is inserted into the index list. Lines 4 through 7 are

implemented to use existing production rules. When symbols on the right-hand side of an existing production rule are the same as those in the new digram, the new digram is replaced with the production rule header (on the left-hand side of the non-terminal symbol). Otherwise, Line 8 and Line 9 initiate the construction of a new production rule replacing the new digram and the detected occurrence with a new non-terminal symbol.

Table 2
Pseudo-code for the SEQUITUR algorithm

1	Read a new input symbol, and append it to start rule *S*
2	Generate a *digram* linking the last two symbols in *S*
3	***if*** the new *digram* is repeated within *S* or the right-hand side of a predicted rule
4	***if*** the other occurrence is a complete right-hand side rule
5	replace the new *digram* with the left-hand side symbol of the rule
6	***if*** there is a non-terminal symbol within the *digram* that only occurs once elsewhere
7	remove the non-terminal rule, and substitute its content in place of the other non-terminal symbol
8	***Otherwise***
9	form a new rule and replace both occurrences with a new non-terminal symbol
10	***Otherwise***
11	insert the *digram* into a *digram* index list

There is empirical research identifying suitable CFGs using SEQUITUR from different domains (e.g., natural language, biology, and music) (Nevill-Manning & Witten, 1997; Koncilia & Pozewaunig, 2002). The algorithm is able to generate production rules that describe the identified structure and efficiently regenerate the input sequence. However, the implementation of the SEQUITUR algorithm was based on the

analysis of only one stream of symbols (one sequence). Inference of comprehensive CFGs is more often than not based on the analysis of multiple sequences in order to identify dispersed characteristics within a target language. Hence, an essential improvement to the algorithm is the generation of CFGs based on inputs consisting of multiple sequences.

Koncilia and Pozewaunig (2002) introduce the *Multiple Sequence Analysis* (MSEQ) algorithm, an extension of the SEQUITUR algorithm. MSEQ addresses some of the downsides present in SEQUITUR, such as its limitation to analyze single sequences. The MSEQ algorithm produces a generalized grammar from multiple sequences based on merging of individual grammars obtained by SEQUITUR. The merging approach implemented in MSEQ allows for analyzing multiple sequences. However, this benefit comes at the cost of additional computational time. By using this approach, the SEQUITUR algorithm must be executed *n* times, where *n* is the number of sequences to analyze, plus the computational time required by the merging algorithm.

In order to handle multiple sequences, we implemented a straightforward modification to the original SEQUITUR algorithm, allowing us to maintain the run time performance of the original algorithm; a vital feature for algorithms intended to be used in real time systems because it maintains the efficiency criteria. We identified that SEQUITUR is able to handle long sequences of symbols such as texts including multiple paragraphs. SEQUITUR handles the control character *end-of-line* (EOL) as any other symbol. This functionality implies that EOL is included as a terminal symbol within generated production rules. By merely considering the EOL character as a sequence

separator and omitting it as part of the alphabet symbols, the modified SEQUITUR algorithm was able to generate production rules based on multiple sequences. We implemented this functionality by modifying Line 1 within the original SEQUITUR algorithm. The rest of the algorithm has basically the same functionality. Figure 10 shows an example of different grammars generated by using the same input sequence for the original and the modified SEQUITUR algorithms.

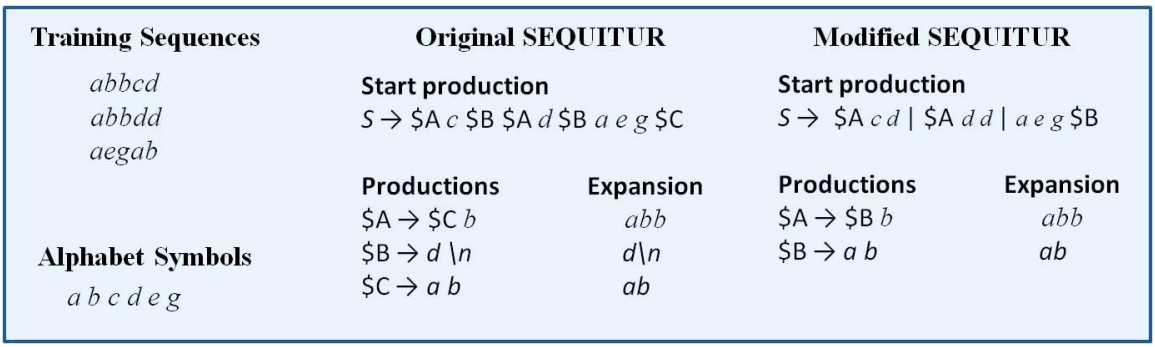


Figure 10. Grammars generated by the SEQUITUR algorithm.

The process to obtain the outputs displayed in Figure 10 is as follows. Both algorithms start scanning the training sequences and appending each symbol within the start production rule *S*. The algorithms aim to identify repeated sequences of symbols. Both algorithms identify the first repetition *ab* as a result of scanning the first eight symbols of the training sequence (including the EOL of the first line). This identification generates a temporary production rule that is not displayed in Figure 10 ($A→ *ab*). Then, using the temporary production rule consisting of *ab*, the sequence *abb* is identified in both algorithms, generating the production rule $A→$C*b* and $A→$B*b* for the original and modified SEQUITUR algorithms respectively. Both algorithms keep scanning the symbols. The original SEQUITUR algorithm identifies a new sequence

repetition consisting of the symbols *d**n* generating the production rule $B→*d**n*. On the other hand, the modified version of SEQUITUR does not consider the "\n" EOL as part of the sequence of symbols, this implies not generating production rules including this symbol. Instead of using the EOL as a sequence's symbol, the modified version of SEQUITUR uses it as generator of a new start production rule. In our example we can see the start production rules S→$A*cd* | $A*dd* generated from the first two sequences from the training sequences.

From this simple example we can observe three main characteristics of the results when using the modified SEQUITUR algorithm. First, the start production rule *S* is represented as a set of production rules (in this case three) separated by a bar character ("|"), rather than just one production rule generated by the original SEQUITUR algorithm. This implies that the resulting grammar is able to regenerate any of the input sentences independently. Second, the "\n" (EOL) character is not considered as part of the alphabet of symbols. This attribute enforces the elimination of noisy digrams generated between alphabet and EOL symbols (original SEQUITUR's production rule $B in our example) or linking the last and the first symbol of two different sequences. Finally, there is consistency with regards to the identification of the most meaningful production rules generated by the original SEQUITUR algorithm. From our example we can observe, under the expansion column, how the patterns "*abb*" and "*ab*" were equally identified by both inferring grammar approaches. Furthermore, because SEQUITUR computes the production rule frequency, we used this information to compute the weight

values per generated production rule. Hence, our final outcome of using the modified version of SEQUITUR is a WCFG.

The data representation was generated using the training dataset as input from the modified version of the SEQUITUR algorithm. Similar to the WMMs approach, the training dataset consisted of sets of *CSs* assigned to each tutorial action. Given a set of *CSs*, say $CS_1 \ldots CS_n$ and also given a set of tutorial actions, say$TA_1 \ldots TA_m$. For each *CS*; we manually assign the appropriate tutorial action TA_j. We reserve one tutorial action as the "null" tutorial action in order to confirm to the student that his answer is on the right path. A WCFG was generated per tutorial action based on its set of *CSs*. Figure 11 shows two WCFGs generated in one of our experiments. Within our experiments, configuration rules are represented by positive integer values. We also conducted experiments using different granularity levels for data representation. For instance, a finer granularity level consists of considering commands and parameters as symbols instead of entire configuration rules. A comprehensive analysis of generated WCFGs and their classification performance is described in Chapter IV.

Weights per production rule were computed locally using frequency of production rules within each tutorial action and global weight values were computed based on the percentage of *CSs* in the entire dataset. Specifically, weights associated with the start production rules *S* within each tutorial action sum to one. Differences in weight values are based on the frequency of each start production rule (see Figure 11). Weight values for terminal production rules other than *S* were computed based on their frequency within each tutorial action. These weight values were also partially influenced

by an a priori probability based on the percentage of *CSs* per cluster derived from the entire dataset. The use of the generated WCFGs and the implementation of the classification algorithms are described in the next section.

Tutorial Action 1

Production Rules:

0.2500000, S, $A 7
0.2500000, S, 54 $A 41 52 10 $B
0.2500000, S, 7 6 58 59 $B
0.2500000, S, 8 $A 6 7 42 $B

0.0116279, $A, 2 3
0.0116279, $B, 4 5

Tutorial Action 2

Production Rules:

0.0666666, S, $A $G
0.0666666, S, $A 101 102 $G
0.0666666, S, $A 12 11 $B
0.0666666, S, $A 97 $E
0.2666666, S, $C
0.0666666, S, $D $B
0.0666666, S, $D 45 $E
0.0666666, S, $A 11 84 $B
0.2000000, S, $I
0.0666666, S, 8 114 $H 37 38 $E

0.1739130, $A, 8 $H
0.1304347, $B, 13 $E
0.1739130, $C, $A 60 $E
0.0869565, $D, $A 11 12
0.1304347, $E, 4 5
0.0869565, $G, 42 $E
0.0869565, $H, 2 3
0.1304347, $I, $A 116 $E

Figure 11. Weighted context free grammars.

2. Classification Policy Inference

The classification policy is built using the set of inferred WCFGs (one per tutorial action). The intended goal of *policy inference* is, by using the generated grammars, to infer a policy able to properly classify *previously known CSs* (from the training dataset) and estimate its confidence level to determine whether the learned knowledge is sufficiently high to classify *unknown CSs* (metacognitive skill).

Based on the WCFG-based structure on which our data is represented, the types of operations performed to classify new *CSs* are:

- generate complete or partial parsing trees using existing grammars;
- compute the parsing tree weight value; and
- select the parsing tree with higher representation and probability value.

Generating parsing trees is a problem that has been extensively studied. There exist several parsing algorithms often used to determine if a given grammar is able to generate an input sequence. The *Cocke-Younger-Kasami* (CYK) algorithm is one of the most frequently used and cited parsing methods for context free grammars. CYK is an efficient algorithm based on dynamic programming (Sipser, 1997). The CYK algorithm makes use of a structure called *chart* to sort partial results using a *bottom-up* approach.

Figure 12(a) illustrates the functionality of Algorithm CYK using a basic arithmetic grammar. The standard version of CYK constrains the use of grammars in the *Chomsky Normal Form* (CNF). CNF restricts the use of grammars consisting of production rules written in the form of $A \rightarrow \alpha$, and $A \rightarrow BC$, where A, B, and C are non-terminal symbols, and α is a terminal symbol. An example of a CNF grammar can be seen in Figure 12(a). Even though any CFG can be transformed into CNF form, the size of the resulting grammar may be undesirably large (Lange & Leiß, 2009).

Extensions of the CYK algorithm attempting to efficiently work with grammars other than those in CNF form have been proposed. Ciressan, Sanchez, Rajman, and Chappelier (2001) developed an enhanced version of the CYK algorithm for unrestricted grammars. The proposed algorithm is a combination of CYK and the *Earley* algorithm.

The *Earley* algorithm is a dynamic-programming based chart parser that is not limited to CNF grammars (Earley, 1970). The additional generality of the *Earley* algorithm comes at the cost of efficiency, given that it has to rely on *look-ahead* parsing.

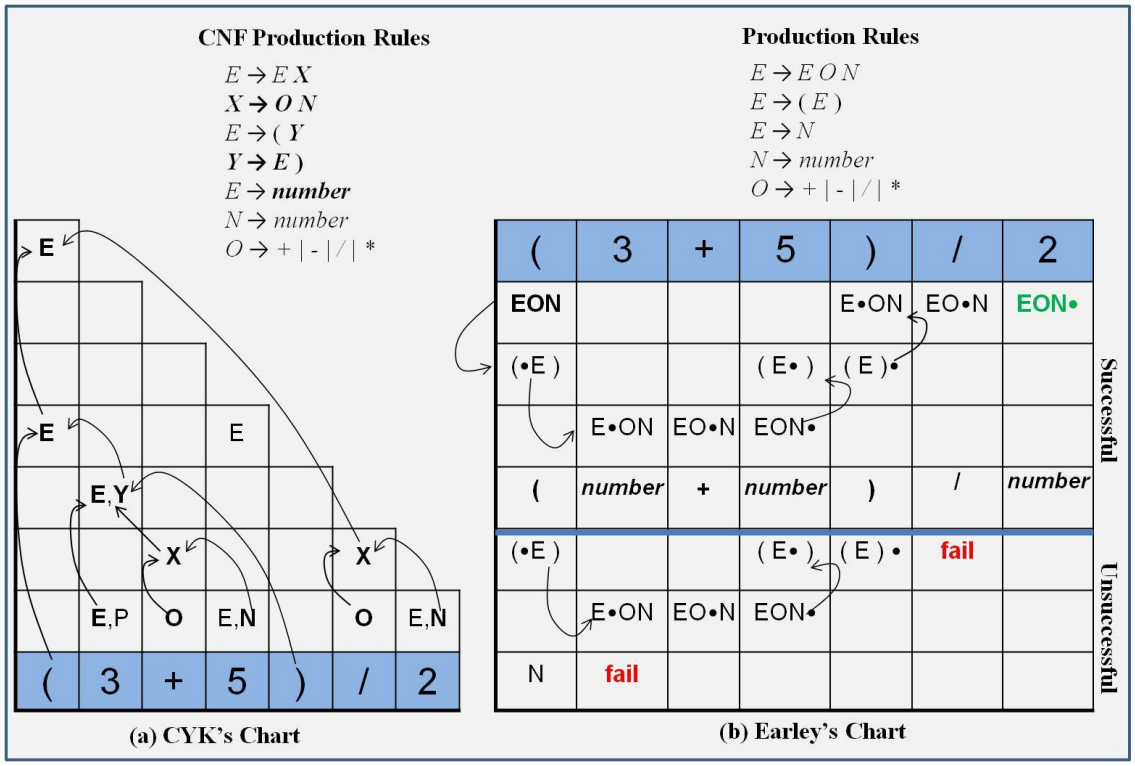


Figure 12. Parsing algorithms. (a) CYK algorithm. (b) Earley algorithm. The grammar includes the non-terminals E (*expression*), N (*number*), and O (*operators*). An e*xpression* is a list of *numbers* connected by a mathematical operator (+, -, /, *). The grammar for the CYK algorithm is in CNF version (changes are denoted in bold).

The *Earley* algorithm is based on a *top-down left-to-right* scanning approach. The algorithm iteratively performs three steps: (1) *prediction*; starting from the start rule(s), it parses all the derived production rules using a *first-depth search* method. The author of the *Earley* algorithm introduced a *dot* notation to indicate the parsed part of a right-hand side of a production rule (e.g. $A \rightarrow B \bullet D a D$ represents that the non-terminal

symbol *B* has been parsed and the rest of the production rule elements are being expected). (2) *scanning*; the *Earley* algorithm stores a list of partially completed production rules and compares the next element after the *dot* symbol with the next symbol in the input stream. If they match, the *dot* is moved one space forward and (3) *completion*; the process continues until each of the analyzed rules are completely inspected. If the algorithm fails to find a match for the inspected symbol, the production rule is then discarded (Earley, 1970). Figure 12(b) displays a portion of the parsing process using the same arithmetic grammar and input sequence used in the CYK example. The figure shows one successful and two unsuccessful paths analyzed by the algorithm. Notice that the grammar used for this algorithm is not in CNF form.

The *Enhanced-CYK* algorithm is another parsing algorithm that relaxes the restriction of CYK to CNF grammars (Ciressan et al., 2001). This algorithm is a combination of CYK and the *scanning* functionality of the *Earley* algorithm. The proposed version of this algorithm works with a type of CFG without *unitary production rules* (e.g. $A \rightarrow B$, where B is non-terminal) and terminal symbols only occur within *lexical rules*. Lexical rules are of the form $A \rightarrow w$, where w is a subset of only terminal symbols. However, these restrictions were imposed just for the researchers' domain characteristics and can be easily removed (Ciressan et al., 2001).

The *Enhanced-CYK* algorithm is a chart-based algorithm that stores two types of items within each chart cell. *Type-1* represents the set of non-terminals for which the right-hand side of their production rule is completely produced by the input sequence. *Type-2* consists of a set of partially parsed strings from the right-hand side of the

production rules. Partial parsing is represented using the *dot* notation proposed in the *Earley* algorithm. The *Enhanced-CYK* algorithm consists of two main stages, the *initialization* and chart *filling* stages. Within the chart *initialization* stage the two set types of items are initialized for each cell in the chart following the next two steps:

- For the *type-1* set, the algorithm searches for a symbol or set of symbols in the input string matching the right-hand side of a lexical rule; if a match occurs the non-terminal in the left-hand side of the lexical rule is placed in the corresponding cell.
- Regarding the *type-2* set, the algorithm assesses whether the non-terminal symbols placed in the *type-1* set are the first symbol of the right-hand side of a production rule containing more than one symbol. If this occurs, the non-terminal symbol followed by the *dot* character is placed as part of the *type-2* set within the corresponding cell.

Figure 13(a) shows the outcome of the *initialization* stage for the arithmetic grammar and the input sequence used in Figure 12. For this example, because of the lexical rules and non unitary rules restriction, a new production rule was added for the parenthetical terminal symbols (i.e., $P \rightarrow (\mid)$) and a start production rule was directly linked to the numerical terminal symbols.

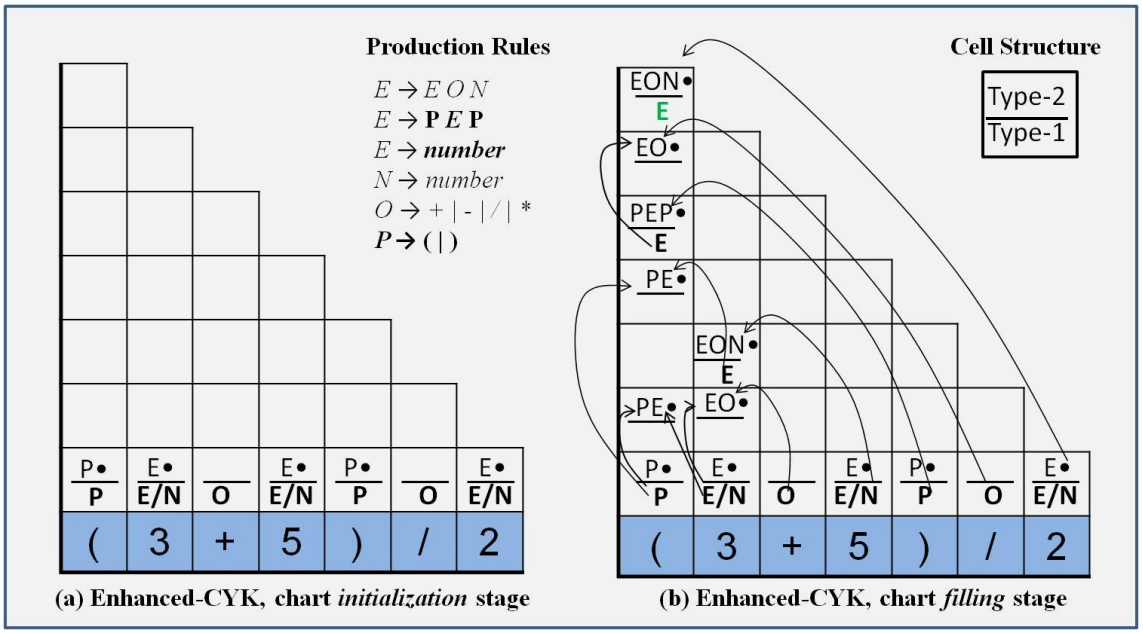


Figure 13. Enhanced-CYK algorithm. (a) chart *initialization* stage. (b) chart *filling* stage. The grammar for the enhanced-CYK algorithm was slightly modified from the original in Figure 12 (changes are bolded).

The chart *filling* stage consists of three main steps to determine the *type-1* and *type-2* sets of items for the empty cells (see Figure 13(b)). In a *bottom-up left-to-right* way the algorithm performs the following steps:

- For each empty cell, the algorithm inspects whether cells in the corresponding diagonal have a non-terminal symbol within the *type-1* set, which matches the next symbol in a partially parsed substring within the *type-2* set for any cell below the one being inspected.
- Depending on whether we have a successful match from the previous step, the matched symbol is added to the partially parsed string and is placed within the *type-2* set of the inspected cell.

- Depending on whether the new parsed string placed in *type-2* is equal to the right-hand side of any existing production rule, the left-hand side non-terminal symbol producing the parsed string is placed within the *type-1* set of the inspected cell.

The chart *filling* process for the analyzed arithmetic grammar is depicted in Figure 13(b). The *Enhanced-CYK* algorithm reduces the number of analyses performed by the *Earley* algorithm and maintains the *efficiency* of the CYK algorithm without the CNF restriction. We use an extended version of the *Enhanced-CYK* algorithm, which we call the *Enhanced*-CYK* algorithm, that is not restricted to lexical and non-unitary rules. Furthermore, we use the WCFGs generated in our data representation phase for our classification process. Hence, the outcome of our algorithm consists of parsing trees and their corresponding weight values. Figure 14 shows an example using a grammar and input sequence from our domain data.

For natural language grammars, terminal symbols are previously categorized as verbs, nouns, pronouns, etc. For arithmetic grammars, terminal symbols are categorized as numbers or operators. Symbols previously categorized are assigned to non-terminal symbols to represent them (known as *lexical rules*). In our domain's grammar, there is only one category of terminal symbol; each symbol represents a configuration rule. Therefore, using a lexical rule written in the form of $R \rightarrow \alpha$, where α represents any terminal symbol in the alphabet, would lead to situations where we could not distinguish between types of symbols. In order to address this situation, we slightly modified the *initialization* step of the *enhanced-CYK* algorithm. We omitted the use of lexical rules.

Instead of placing a non-terminal symbol within the *type-1* set, which represents a specific set of symbols in the alphabet, we used the actual symbols from our input string. Then, those input symbols that initiate the left-hand side of a production rule were placed within the *type-2* set using the *dot* notation, see Figure 14(a). Other than the use of terminal symbols as part of the parsing process, the chart *filling* process in the *Enhanced*-CYK* algorithm is essentially the same as the one originally proposed by the *Enhanced-CYK* algorithm (see Figure 14(b)). Next we explain how this algorithm is used to perform the classification process.

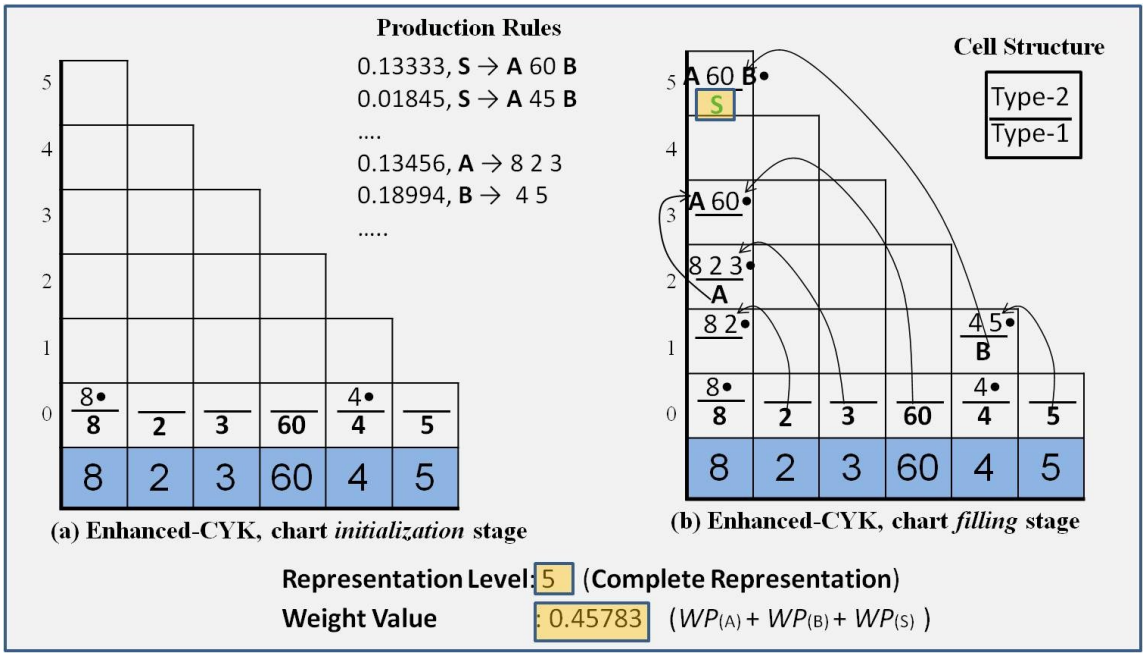


Figure 14. Enhanced*-CYK algorithm using WCFG. (a) chart *initialization* stage, (b) chart *filling* stage. Weight values are assigned to each production rule within the grammar. Final weight value and level of representation of the input sequence is depicted.

3. Use of the Classification Policy

The classification process begins when a student enters a new *CS,* which is then classified by determining the most likely representation and the highest WCFG weight value of the new *CS* among the existing WCFGs. We say that a WCFG *completely represents* a *CS* when the model is able to generate the entire *CS*. Similarly, we say that the WCFG *partially represents* a *CS* if it can only generate sub-sequences of the entered *CS*. The representation level is determined by the row number in the chart in which the parsing process ends (for complete representations, this value should be equal to the length of the input sequence minus one), see Figure 14. Representativeness is determined by first identifying those WCFGs that are able to completely represent the new *CS* or those with the higher partial representation, and then by selecting the one with the highest likelihood (highest WCFG weight value).

The WCFG weight value is computed by summing the weight values of each satisfied production rule by using Equation (1). A satisfied production rule implies that its right-hand side string is entirely contained within the entered *CS*. For *completely represented CSs*, the satisfied production rules are those included within the resulting parsing tree. In Equation (1), *l* represents the number of WCFGs within the classification policy; *m* represents the number of satisfied production rules within each WCFG for the new *CS*; and $wp_{(k)}$ is the weight value of production rule *k*.

$$\mathrm{WCFG}_{n=1..l}^{(n)} = \sum_{k=0}^{m} wp_{(k)} \qquad \textbf{(1)}$$

The classifier uses the largest weight value to select among those WCFGs best representing the input *CS*. The output of the classifier is the tutorial action assigned to the selected WCFG and the computed representation level. Similar to WMMs, we use the representation level value of the WCFG to measure the confidence level returned by the classification policy for the selected tutorial action. When none of the current models is able to completely represent the new *CS*, those models that best represent the new *CS* are considered (*partial representation*).

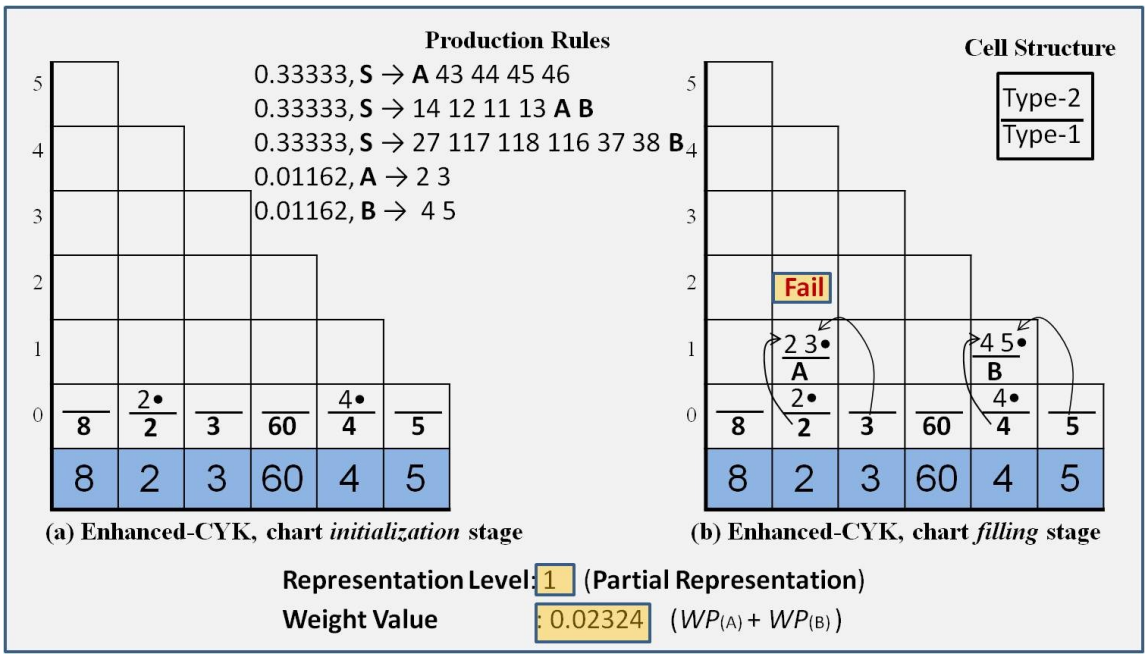


Figure 15. Enhanced-CYK algorithm using WCFG and a partially represented input sequence. (a) chart *initialization* stage. (b) chart *filling* stage. Weight values are assigned to each production rule within the grammar. Final weight value and level of representation of the input sequence is depicted.

Figure 14 shows the computed weight value for a completely represented *CS*. In Figure 15 we show an example of partial representation where the *CS* is the same as in

Figure 14, but a different set of production rules is used. Here we can see that the representation level is equal to one, and the weight value is the sum of the two production rules contained within the entered *CS*. More comprehensive experimental results from our study are presented in the Chapter IV.

E. Summary

This chapter provided details of the knowledge representation methods and algorithms used for the implementation of the proposed mixed-response ITS framework. In this chapter we described the phases of the Learning from Demonstration technique used as a machine learning approach. We included details of Weighted Markov Models and Weighted Context Free Grammars, the two implemented knowledge modeling approaches for non-vector data. Finally, in this chapter we described and illustrated the generation of the models, inference of the classification policy and how this policy is utilized.

In the following chapter we present the series of evaluation studies we conducted using the implemented knowledge modeling approaches. We evaluate the capability of these methods to address the six functional characteristics envisioned for an ITS: *effectiveness*, *scalability*, *efficiency*, *portability*, *metacognition* and *robustness*.

CHAPTER IV

ITS FRAMEWORK EVALUATION

This chapter describes a set of experiments that were performed to evaluate the proposed mixed-response ITS framework. We implemented two knowledge representation and policy inference approaches described in Chapter III, WMMs and WCFGs, for the proposed framework and evaluated them using sequential data from community college students using the Web Access Exercise System (WAES). The WAES learning environment is described in section B within this chapter. The initial evaluation process was focused on testing the framework performance regarding the first five envisioned characteristics for a modern ITS supporting students learning within ill-defined domains, as laid out in Chapter I:

- *effectiveness* in learning domain knowledge from instructor-student demonstrations and using that knowledge whenever it is required,
- *scalability* by supporting the expansion of the ITS's knowledge-base influenced by a larger number of students,
- *efficiency* in terms of reducing the amount of time required by an expert to build the domain model,
- *metacognitive* skills for understanding its existing knowledge level in order to know when to provide help to students or when to refer to an instructor and learn from instructor-student interaction, and

- *robustness* in terms of enabling the ITS to handle varying levels of student background knowledge and instructors' pedagogical perspectives.

As part of this evaluation we compared the performance of the two implemented knowledge representation and policy inference approaches (i.e., WMMs and WCFGs). Additionally, in order to assess the sixth envisioned characteristic for a modern ITS framework, i.e. its capability to be used within different domains (*portability*), we conducted evaluations using data from a different domain. This will be discussed later in the chapter.

A. The Cybersecurity Domain

The proposed mixed-reponse ITS framework was tested using data from the ill-defined *cybersecurity* domain. The cybersecurity domain encompasses a variety of subjects (e.g., Firewalls, Intrusion Detection, and Operating Systems Security), each of them addressing different security concerns.

In a typical real-world cybersecurity scenario, practitioners are given, or participate in the creation of, an organizational *security policy*, which includes a set of *security requirements*. Afterward, based on the security policy and several contextual pieces of information (e.g., network topology, network users' profile, and organizational budget), network security professionals select, configure, and monitor the set of *security systems* or *services* (e.g., firewall, VPN server, and antivirus server) that will protect the organization's information.

Even when a possible correct solution tends to use some of, all of, or more of the security systems mentioned above, a security system's configuration should always be customized to the specific organizational requirements. Furthermore, as the number of security requirements increases, the possibility of neglecting or overlapping *configuration rules* increases. A configuration rule is often represented as a command and a set of parameters intended to configure a security system (hardware or software devices) based on a security requirement. While configuring a security system, network security professionals usually build scripts composed of a number of configuration rules ranging anywhere from 10 to more than 1000, leading to a number of configuration paths and the possibility of multiple correct, partially correct, or incorrect configurations. Abbes, Bouhoula, and Rusinowitch (2008) summarized a set of studies that report anomalies on firewall configurations managed by experts, common errors leading to a poor configuration that includes security holes, and the success of certain worm and virus attacks that could be blocked with better firewall configurations. Hence, the types of problems faced in this domain are sometimes well-defined, but mostly ill-defined.

B. The WAES Learning Environment

The proposed mixed-response ITS framework was applied using data from the Web Access Exercise System (WAES), a case-based instructional system that provides training in well- and ill-defined cybersecurity problems (Cifuentes, Bettati, Marti, Alvarez, & Mercer, 2009). Specifically, we used data from problems currently included in the WAES' *Firewalls* unit. The Firewalls unit in WAES consists of a set of exercises

addressing specific topics on cybersecurity. Each exercise consists of a set of security requirements and an organization's network topology, which includes a firewall security device. Students are asked to protect the organization's network by configuring the firewall device based on the given security requirements and network IP addresses. Students configure the security device by entering a set of configuration rules. WAES also includes cases that, in order to be solved, require the application and integration of the knowledge gained throughout the solution of the unit's exercises. Upon successful completion of the set of exercises in a WAES unit, students are expected to have the domain knowledge necessary to address more complex real-world cases (Cifuentes et al., 2009).

Throughout the exercise and case solution process, students are asked to perform three major activities: (1) internal representation of the network security problem and solution planning; (2) implementation of their solution by configuring the network security devices; and (3) evaluation of the implemented solution by using the system's traffic generation functionality. First, the students are asked to analyze the system at hand, identify the security problems and draft a plan to address the problems. Next, the students implement their solution's plan by configuring the network elements (in this case firewalls devices) to address the problems identified in the first step. In the third and final step, the students are asked to validate their approach and implementation by testing the system. The WAES provides support for this by having tailored testing sequences available that can be triggered by the student in order to simulate networking attacks. Based on the results of this last evaluation step, the WAES system indicates

whether the student configured a successful solution (see Appendix A for more details). In this research study, we were focused on demonstrations from solution paths generated during configuration of security devices (use of configuration rules), activity (2).

The students' data collection was conducted automatically by the WAES system. The WAES system logs all the commands typed by the students within the command line interface while students configure a security device (i.e., firewall). Then, WAES generates log-files that allow for the review of students' activities while they were solving each exercise. Instructors can review the log-files in order to grade the students' configurations, as well as, provide feedback to students when needed. The configuration log-files were used as our data source. The collected log-files provided a set of correct, partially correct, and incorrect security device configurations.

The learning task we are using is ill-defined because of the undetermined number of solutions and solution paths. Each security requirement to be addressed within an exercise implies the use of one or more configuration rules, each of which consists of an open combination of mandatory and optional parameters. Furthermore, there is a direct influence among security requirements. The set of parameters within a configuration rule, used to address a security requirement, should consider parameters within other configuration rules. The sequence of the configuration rules is also affected by which rules were used in the final configuration. Hence, even for small sets of security requirements, the number and content of required configuration rules and the set of correct sequences of these rules is difficult to predefine at design time.

C. Participants

For the evaluation of our mixed-response ITS framework we used data collected from several cybersecurity classes that used WAES from approximately 10 different community colleges around the country. Specifically, data was obtained from 130 students from 10 different classes over a time period of three semesters. Student demographics indicated that these community colleges were predominantly Hispanic or African-American. Approximately 74% of participants were male and 26% were female. Most of the students had medium to advanced computer and networking skills and were asked to configure network devices.

In addition, four experts in cybersecurity participated in the development of the tutorial actions and evaluation of the ITS framework performance. One of the four experts that participated in this study has more than 20 years of experience in working with network security and reliability issues. He currently serves as the Director of a Network Security Office and teaches upper level courses in Networking at a major university. Two experts are university professors with more than seven years of experience in teaching network security courses. The fourth expert is a graduate student in computer science. He has experience in installing and configuring networks, servers, and network security devices.

Demonstration data consisted of students' *CSs* labeled with experts' tutoring actions. For our experiments, a total of 515 *CSs* consisting of more than 3600 *CRs* were gathered from the collected dataset. From these *CSs* we obtained more than 1150 different *CRs*, and 100 different configuration parameters.

D. Evaluation Studies

In order to evaluate the six functionalities envisioned for an ITS supporting students learning how to solve ill-defined problems, we conducted five experiments using two testing approaches: *batch learning* and *interactive learning*. When using the batch learning approach we evaluated the data representation and tutoring performance of the ITS framework by using the two proposed methods, WMMs and WCFGs. In the interactive learning environment we only tested the ITS performance using the WCFGs method because it out performed the WMMs method within the *batch learning* experiments. The main goal of the interactive learning approach evaluations was to measure the *efficiency* in learning of the ITS framework, as well as, its capability to be *portable*.

Within the batch learning approach, we evaluated the performance of the proposed ITS framework regarding four different functionalities by conducting the three following experiments:

Experiment 1: *effectiveness* and *metacognitive* skills evaluation,

Experiment 2: *robustness* evaluation, and

Experiment 3: *scalability* evaluation.

In all three experiments, we used the data collected from students using WAES. Data was provided to the intelligent tutor in different amounts and sequences per experiment. The categorization of solutions as correct or incorrect with similar misconceptions and the corresponding feedback was performed by the most experienced expert. Within this

evaluation environment we replicated experiments several times in order to determine initial threshold values for the confidence level of the ITS.

Within the interactive learning environment we conducted two experiments evaluating the ITS's performance regarding four functionalities:

Experiment 4: *effectiveness*, *efficiency* and *robustness* evaluation, and

Experiment 5: *portability* evaluation.

In Experiment 4 to 5, we used partial data previously used within the batch learning approach. The intelligent tutor started learning from scratch by interacting directly with multiple experts. Data was randomly sorted and selected from the entire dataset. Experts interacted directly with the ITS every time the intelligent tutor performed a student's solution classification. Although we were able to evaluate four functionalities within these experiments, measuring *effectiveness* and *portability* was our main goal.

E. Data Preprocessing

Data preprocessing is an important, and often neglected, activity within the machine learning field. Kotsiantis, Kanellopoulos, and Pintelas (2006) say that preprocessing tasks such as cleaning, transformation, and selection should be considered in order to generate suitable training and testing datasets. In addition, for supervised learning, the success of knowledge acquisition during the learning phase is highly influenced by the representation and quality of the datasets. Thus, in this study, *cleaning* and *selection* preprocessing tasks were primarily considered for dataset preparation; then, transformation of students' solution paths into sequential datasets was implemented.

1. Data Preparation

We performed a data cleaning (fixed incorrect data and eliminated of confidential data) and selection (elimination of noisy data) analysis before performing any data transformations; we were mainly looking for potential incorrect or invalid data and outliers that could have a significantly negative impact on the ITS learning process. However, most of the collected observations were considered acceptable. Interestingly, the entire dataset was considered representative of our learning domain by our expert with more experience in network security. And even though we encountered some configuration sequences significantly different in length (number of configuration rules) and content (use of parameters), those outliers were considered necessary for our classification process. Therefore, all *CSs* were selected and no significant cleaning, other than eliminating students' identification data, was necessary.

An additional data preparation step for a portion of our experiments was the *categorization* of each *CS*. An expert labeled each *CS* as *correct*, *partially correct*, or *incorrect* based on whether the sequence addressed the intended security requirements. Then, *CSs* labeled as incorrect were sub-categorized based on the major type of configuration error. *CSs* identified with the same configuration error were grouped and assigned a tutorial action that would help students to identify and correct the configuration misconception. The result of this data preparation step was a set of *clusters* consisting of similar *CSs* as defined by the domain expert. By two *CSs* being similar we mean that they require the same tutorial action from the ITS. Hence, for each cluster the expert formulates a tutorial action that is appropriate for all *CSs* in the cluster.

Interestingly, we observed that the sizes of clusters (in terms of number of *CSs*) generated during preprocessing were generally highly unbalanced. While some clusters contained many *CSs* (i.e., tutorial action 2 and 11 in Figure 16), which were typically insignificantly different from each other, several other clusters have only one or a small number of *CSs* (i.e., tutorial action 1 and 10 in Figure 16). It was expected that the small clusters, as a result of students' misconceptions, sometimes might be quite common. This type of behavior was expected because of the degree to which our domain was ill-defined. It is an advantage of our mixed-response approach over design time approaches that such misconception-driven clusters can be identified and (as we will illustrate later) handled by the ITS.

ID	Tutorial Actions
1	Your *iptables* configuration is empty, are there any packet filtering rules needed to initiate the configuration?
2	You are using some incorrect source or destination IP address.
3	Great! Correct configuration.
4	The Deny-All policy and allowance of traffic from the internal network should be at the end of the configuration rules.
5	You are missing the allowance or rejection of certain traffic before the Deny-All policy. You should flush and start over.
6	You entered some incorrect and repetitive rules. Before you continue entering rules you should flush your configuration and start again.
7	The spoofed Packets rule should be at the very beginning of the configuration rules.
8	You are using REJECT or DROP parameters incorrectly.
9	You are rejecting or accepting some traffic incorrectly.
10	NAT rules are not needed to address this exercise's security requirements.
11	Your current configuration rules are correct. However, you are still missing the allowance or rejection of certain traffic.

Figure 16. Set of tutorial actions predefined by an expert.

2. Data Transformation

As described earlier, solution paths selected by students consist of *configuration sequences* (*CSs*), which in turn consist of *configuration rules* (*CRs*). An example illustrating *CSs* from three students is shown in Table 3: Student 1 uses two *CSs* with three *CRs* respectively, while Student 2 and Student 3 each only use one *CS*.

In order to better represent *CSs*, we further parse each *CR* as a sequence of parameters with two different levels of granularity: *token level* and *paired-symbols level*. For instance, the rule "iptables -A FORWARD -j DROP" has five tokens and three paired symbols respectively. At token level the rule consists of individual symbols ("iptables", "-A", "FORWARD", "-j", and "DROP") while at paired-symbols level it consists of individual symbols and parameter-value pairs ("iptables", "-A FORWARD", and "-j DROP"). For data representation, a *configuration-rule level* was used as well. This coarse-grained data representation level consists of entire *CRs* ("iptables -A FORWARD -j DROP").

Table 3
Frequent behaviors and misconceptions in configuration rules

Student ID	Configuration Rules	Command Type
1	iptables –**flush**	reset configuration command
	iptables **–L**	informative command
	iptables -A FORWARD -p tcp --dport http -j ACCEPT	
1	iptables **–F**	different/correct command
	iptables -A FORWARD -j DROP	different/incorrect rule order
	iptables -A FORWARD **--dport http –p tcp** -j ACCEPT	different parameter order
2	iptables -A FORWARD -p tcp **--dport httt** -j ACCEPT	incorrect parameter name
	iptables -A FORWARD -p tcp **--dport ftp** -j ACCEPT	incorrect parameter value
3	iptables **–F**	different/correct command
	iptables -A FORWARD -p tcp **--dport 80** -j ACCEPT	different/correct parameter value

We note that the more coarser-grained the representation is, the more domain specific the parser must be. While any tokenizer is able to process input at token level, it must understand the concept of command-line input to generate paired symbols from student input. In order to parse input at configuration rule level, the parser must understand what configuration rules are, and thus have a detailed understanding of the underlying domain.

Table 3 also displays examples of different configuration types within each *CS* (e.g., the third *CR* within both *CSs* of Student 1 are semantically the same but different regarding the order of parameters, the first CR within both CSs of Student 1 are semantically the same but differ using complete command name and alias respectively), semantic and syntactic misconceptions, and parameter functionality type (e.g. the first *CS* of Student 1 consists of a resetting command, an informative command, and finally of a configuration command).

F. Effect of Symbol Granularity Level

We performed the evaluation of our approaches' knowledge representation using the three different data representation granularity levels: *token*, *paired-symbols*, and *configuration-rule*. By configuration rule level we mean that the ITS operates with student input at configuration rule level. At the other extreme the ITS is presented with an unprocessed token stream from the student. We expected and obtained varying ITS framework performance results in classifying new *CSs* using these granularity levels. The way in which data representation levels were built is described next:

- *Token level*. At this level the ITS is given an unprocessed stream of tokens that have been generated by a generic tokenizer from the student input. As a result, a vocabulary catalog of terms is generated from the students' sets of *CRs*. These terms include the entire set of individual configuration symbols (configuration commands, parameters, and values) used by students for each specific exercise. Term frequency is computed and each term within the catalog is paired with a unique identification number. Figure 17(a) shows an example of a vocabulary catalog using the token granularity level from students' *CRs* in Table 3. By using this vocabulary we encode the parameters within a *CR* using the identification numbers assigned to each term. Term frequency is computed, and a unique identification number is assigned, see Figure 17(b). Then, *CSs* are represented by linking the encoded *CRs*, see Figure 17(c).

- *Paired-symbols level*. At this level, the ITS is handed pre-pruned input, where an intermediate step parses the input from the tokenizer first before passing partly parsed commands to the ITS. Specifically, the parser determines pairs of commands and their arguments and passes these paired symbols to the ITS. We notice that at this level a parser must be implemented that has a general understanding of the syntax of the input to be pruned by the ITS. It therefore needs to have some understanding of the application domain at design time. At run time a vocabulary catalog of terms is generated from the students' sets of *CRs*. For this representation, terms consist of individual symbols (commands) and paired-symbols (parameter-value pairs) used by students for each specific

exercise. Term frequency is computed and a unique identification number is assigned as well. Figure 18(a) is an example of a vocabulary catalog using the paired-symbol granularity level from students' *CRs* in Table 3. *CRs* are encoded using the identification numbers assigned to each term, term frequency and unique identification numbers are assigned as well (see Figure 18(b)). Then *CSs* are represented as string sequences by linking the encoded *CRs*, see Figure 3(c).

(a) Vocabulary Catalog

ID	Term	Frequency
1	iptables	14
2	--flush	1
3	-L	2
4	-A	8
5	FORWARD	8
6	-p	2
7	tcp	5
8	--dport	5
9	http	2
10	-j	8
11	ACCEPT	5
12	DROP	3
13	-F	3
14	httt	1
15	ftp	1
16	80	1

(b) Configuration Rules

ID	Conf. Rules	Frequency
1	1,2	1
2	1,3	1
3	1,4,5,6,7,8,9,10,11	1
4	1,13	2
5	1,4,5,10,12	1
6	1,4,5,8,9,6,7,10,11	1
7	1,4,5,6,7,8,14,10,11	1
8	1,4,5,6,7,8,15,10,11	1
9	1,4,5,6,7,8,16,10,11	1

(c) Configuration Sequences

Student-ID	Token level
1	1,2 _ 1,3 _ 1,4,5,6,7,8,9,10,11
1	1,13 _ 1,4,5,10,12_1,4,5,8,9,6,7,10,11
2	1,4,5,6,7,8,14,10,11_1,4,5,6,7,8,15,10,11
3	1,13_1,4,5,6,7,8,16,10,11

Figure 17. Dataset transformation based on token granularity level. (a) Sample of configuration terms based on token granularity level with unique identification number and frequency displayed. (b) Encoded *CRs* with unique identification number and frequency displayed. (c) *CSs* from three students represented as sub-strings of symbols (the former is an example of an ITS's training data file).

- *Configuration-rule level*. At this level, the ITS is presented with fully parsed configuration rules. The parser must be specifically designed to understand configuration rules. At run time, using the unique identification number assigned to the encoded *CRs* generated within the paired-symbols granularity level (see right side of Figure 18(b)), a coarse-grained *CSs* representation was generated. This data representation consisted of a string of encoded *CRs*, see left side of Figure 18(c).

(a) Vocabulary Catalog

ID	Term	Frequency
1	iptables	14
2	--flush	1
3	-L	2
4	-A FORWARD	8
5	-p tcp	5
6	--dport http	2
7	-j ACCEPT	5
8	-F	3
9	-j DROP	3
10	--dport httt	1
11	--dport ftp	1
12	--dport 80	1

(b) Configuration Rules

ID	Conf. Rules	Frequency
1	1,2	1
2	1,3	1
3	1,4,5,6,7	1
4	1,8	2
5	1,4,9	1
6	1,4,6,5,7	1
7	1,4,5,10,7	1
8	1,4,5,11,7	1
9	1,4,5,12,7	1

(c) Configuration Sequences

Student-ID	Paired symbols level	Configuration Rule level
1	1,2 _ 1,3 _ 1,4,5,6,7	r1, r2, r3
1	1,8 _ 1,4,9 _ 1,4,6,5,7	r4, r5, r6
2	1,4,5,10,7 _ 1,4,5,11,7	r7, r8
3	1,8 _ 1,4,5,12,7	r4, r9

Figure 18. Dataset transformation based on paired-symbols and *CRs* granularity levels. (a) Sample of configuration terms based on paired-symbols granularity level with unique identification number and frequency displayed. (b) *CRs* with unique identification number and frequency displayed. (c) *CSs* from three students represented as sub-strings of paired-symbols and as strings of *CRs* (the former is an example of an ITS's training data file).

G. Batch Learning Experiments

We implemented a *batch learning* approach for the evaluation of the mixed-response ITS framework based on Learning from Demonstration. Specifically, within this evaluation we conducted three experiments in order to measure the ITS framework's *effectiveness* and *metacognitive skill*, *robustness*, and *scalability*. The main goal of this evaluation was to determine initial threshold values in which the confidence level of the ITS is effective in providing feedback, robust in addressing diverse pedagogical perspectives and scales consistently when demonstration data increases.

For this evaluation we carried out three main phases, 1) *data preprocessing*, 2) *policy learning*, and 3) a *classification process*. Within the *data preprocessing* phase we performed data transformation steps that converted the raw input data (sets of configuration sequences) into sequences of symbols (as described in the previous section). This process was necessary in order to construct an easy to use data structure for the two implemented knowledge representation and policy inference approaches: WMMs and WCFGs. This initial phase produced the training and testing datasets used in Phase 2 and Phase 3 respectively.

Within the *policy learning* phase we used the 70/30 system training approach commonly used for the evaluation of learning systems. About two-thirds (70%) of the available data was used for building the sets of WMMs and WCFGs; representing the initial learned classification policy; from now on referred to as classifiers. Since the observations were randomly distributed between the datasets, training data for our learning algorithms were randomly selected. The remaining one-third was used for

testing the ITS framework performance regarding the classification of new *CSs* and generation of the appropriate tutorial action.

During the *classification process* phase, an interactive mode was simulated for testing purposes. By using the testing data produced from Phase 1, the classification policy established in Phase 2 was evaluated by an expert. We implemented a non-intrusive evaluation of the ITS framework into the WAES architecture. Since the ITS framework was not tightly integrated into the WAES, these analyses were performed off-line. We were able to "replay" student activities and associated ITS feedback and thus fine tune ITS framework elements to optimize effectiveness. Details and illustrations regarding each phase are described in the following subsections. Specifically, we describe how the classifiers, built with the proposed knowledge representation methods, address four of the six proposed functional characteristics: *effectiveness*, *metacognition*, *robustness*, and *scalability.*

1. Effectiveness and Metacognitive Skill Evaluation Results

To assess the *effectiveness* and the *metacognitive skill* of the ITS framework in learning and using the domain knowledge, we measured the *precision* and *accuracy* of the classification policy derived from the testing data. Precision and accuracy are two measures that are commonly used in the evaluation of classifiers. The precision of a classifier measures how many items identified as belonging to class C_i actually belong to C_i. However, precision does not say anything about items that belong to C_i that have been identified as not belonging to C_i. In our context, precision refers to the capability of

the ITS to classify students' configurations with similar misconceptions within the same cluster. Precision of the classification methods therefore measures the effectiveness of the ITS framework. The accuracy of a classifier on the other hand is the percentage of correctly identified items (i.e., *true positives* and *true negatives*). In our context, accuracy is the percentage of correctly proposed tutorial responses of the ITS. It represents the capability of the intelligent tutor to determine whether or not a particular tutorial action resulting from the classification of a new *CS* should be used. Specifically, accuracy of the classification methods was used to measure the metacognitive skill of the ITS framework. Hence, the metacognitive skill is based on the capability of the ITS to determine whether or not to use a tutorial action based on the estimated confidence level. We measured the precision and accuracy of the ITS by having a domain expert grade each ITS tutorial action selection based on correctness.

Precision of the ITS Framework

In our experiment an expert analyzed approximately 350 *CSs* from students. Based on these *CSs*, the expert recommended use of 16 different tutorial actions initially. A unique tutorial action was assigned to each *CS*. Based on this categorization we built 16 WMMs and WCFGs using the three different granularity levels used for data representation. Then, *CSs* within the testing dataset were classified using both the WMMs- and WCFGs-based classifiers by using the largest *representation level* and highest weight value. The representation level is the number of sub-sequences of symbols identified as previously known from the total number of symbols within a new

CS. In this experiment, the representation level was used as the ITS's confidence level. The classifier's ability to represent new *CSs* according to granularity level is presented in Table 4, and the results of the classification process are depicted in Table 5.

Table 4
Representation percentages according to granularity level and approach

	WMMs		**WCFGs**	
	Representation Level		**Representation Level**	
Granularity Level	**Complete**	**Partial**	**Complete**	**Partial**
Token	79%	21%	59%	41%
Paired Symbols	68%	32%	59%	41%
Config. Rules	59%	41%	59%	41%

During the classification process, the experiment's results indicated that 59% of the new *CSs* were completely represented by the WCFG approach using all three implemented granularity levels. This 59% was exactly the percentage of *CSs* in the testing dataset that were also learned from the training dataset. These results indicate that the WCFG approach was able to effectively and consistently regenerate learned *CSs* (see Table 4). On the other hand, 59% of the new *CSs* were completely represented by the WMM approach using the configuration-rules granularity level. However, the use of paired-symbols and token granularity levels increased the portion of completely represented new *CSs* by up to 68% and 79% respectively. We found that some of the new *CSs* within the testing dataset were identified as completely represented even though they were not present in the training dataset. This situation occurred because the WMM approach is considering the number of times two symbols appear together within the analyzed *CS*, while the WCFG approach considers syntactic structures that allow for

the identification of the occurrence of the entire sequence of symbols within the analyzed *CS*. Hence, the limitation in using WMMs is that the classifier is considering independent symbol associations regardless of the *CR* in which they occur.

Regarding the precision of classifying new *CSs* within clusters with similar misconceptions, 100% of the completely represented *CSs* from the WCFG approach were classified correctly for all three levels of granularity (see Table 5). These outcomes emphasize the effectiveness of this approach in representing knowledge based on sequential data, and its capability in using that representation to regenerate and identify similar sequences. However, by using the WMM approach, we obtained a reduction in the percentage of correct classifications for completely represented *CSs*. This reduction in effectiveness was because of the number of unknown *CSs* within the testing dataset that were classified as completely represented. For the additional *CSs* that were not completely represented, the outcomes indicate that, when using the WCFG approach, a finer granularity level of data representation helps to improve the classification process (see Table 5).

Table 5

Effectiveness results. Percentage of correct classifications of *CSs* according to representation level for different granularity levels of two classifiers

	WMMs		**WCFGs**	
	Representation Level		**Representation Level**	
Granularity Level	**Complete**	**Partial**	**Complete**	**Partial**
Token	77%	74%	100%	85%
Paired Symbols	80%	76%	100%	80%
Config. Rules	100%	75%	100%	75%

In Table 5 we can see similar results regarding correct classification for the WMMs approach from coarser to finer granularity levels. We obtained around 75% of correct classifications. Using the WCFG-based approach led to an increase in correct classifications by 10%. We obtained 75% of correct classifications when using configuration-rules as opposed to 85% of correctness when using a token granularity level. Results of this first analysis indicated that the use of a finer granularity level for data representation combined with the WCFG approach causes a better classification performance. However, the use of a finer granularity level was not satisfactory for the WMM approach. The remaining experiments were conducted using the three granularity levels for the WCFG approach and only the configuration-rule granularity level for the WMM approach.

Accuracy of the ITS Framework

During the *classification process*, new *CSs* were classified and placed into a specific cluster using both approaches (WMMs and WCFGs). Classifying completely represented *CSs* resulted in 100% accuracy for the WCFG approach using all three implemented granularity levels and for the configuration-rule granularity level using the WMM approach. However, the framework generated some incorrect classifications for partially represented *CSs*. Hence, we focused on identifying those *CSs* with a partial representation that were incorrectly classified. The ITS will determine its confidence level for a previously performed classification; aiming to reach a 100% accuracy in

recommending only those partially represented *CSs* that were correctly classified. This is the point at which the intelligent tutor demonstrates its metacognitive feature.

The ITS determined its confidence level by comparing the representation level obtained from the selected cluster and a previously specified threshold. In this study we found that the final estimation of the confidence level is a function of the representation level and the length of the *CSs*, based on the number of symbols within a new *CS*. Based on our experiment results, to consider a confident classification, we set a representation threshold of 65% of the intended to classify *CS*. The level of representation "≥ 65%" means that, from the total number of symbols within a new *CS*, at least 65% were identified as previously known sub-sequences by the classifier. The value of 65% was determined as the most appropriate threshold value to consider a classification of a new *CS* as relevant. This threshold value is a result of several experiments in which we determined the best performance of the two classifiers using the three granularity levels used for data representation.

The threshold allowed the ITS framework to confidently classify partially represented *CSs* that were sufficiently supported (at least 65%) by one or more of the data representation clusters. *CSs* classified with a high confidence level were provided with a tutorial action by the ITS. The *CSs* insufficiently represented (less than 65%) were added to the set of *CSs* where the ITS had a low confidence level; therefore, no tutorial action was recommended. The defined threshold approach gave us an increase of correct classifications for all three granularity levels. Specifically for the finer granularity level, we obtained an increase from 85% to 98% for partially represented

CSs using the WCFG-based classifier, and 75% to 89% for the WMM-based classifier using the coarser granularity level. This increase is a result of not considering some of the *CSs* that were incorrectly classified. The results of this final classification step for *CSs* partially represented are presented in Table 6.

Table 6
Effectiveness and metacognition results. Percentage of correct classifications of partially represented *CSs* according to granularity level

	WMMs		**WCFGs**	
Granularity Level	**Previous Classification**	**Using ITS Confidence**	**Previous Classification**	**Using ITS Confidence**
Token	N/A	N/A	85%	98%
Paired Symbols	N/A	N/A	80%	92%
Config. Rules	75%	89%	75%	89%

These results indicate that threshold methods must be considered for classifying new *CSs* that are only partially represented. Again, the combination of fine granularity and WCFG-based classification gave the best results in providing correct feedback to students. The collected evidence illustrates that the metacognitive skill we implemented ensured that most of the feedback provided by the intelligent tutor was going to be correct.

2. Robustness Evaluation Results

Given the positive results in terms of effectiveness, we needed to evaluate the robustness of the policy-building mechanisms. By this we mean the ability of the ITS to effectively

build the knowledge-base independently of the type of student input received, and the order in which the student input was received and processed. For this we performed the same evaluations we described in the previous section by using two more, randomly selected, sets of training and testing datasets. Then we performed the classification process again. Finally, we compared the classification results of the three datasets, the initial and the two new datasets, in order to determine whether or not the initial results we obtained were influenced by the way the training and testing datasets were formed. We conducted the comparison of results by analyzing outcomes obtained using the three granularity levels for the WCFG-based classifier and only the coarser granularity level for the WMM-based classifier. The results of this comparison considered partially represented *CSs* as well and are presented in Table 7. For easier reference, we are going to refer to the three datasets as Set-1, Set-2, and Set-3.

The outcomes of repeating the batch learning process for policy learning by using new training and testing datasets indicated consistency in the classification outcomes. The collected evidence indicates that both of the classifiers used behaved consistently for each of the tested granularity levels of data representation we implemented. Specifically, from these multiple comparisons, we confirmed that the use of the WCFG-based classifier based on a token granularity level can be considered as a dependable classification method. The experiment results indicate between 96% and 98% of correct classification when the WCFG based classifier was used in combination with a token granularity level for data representation.

Table 7
Robustness results. Comparison of percentage of correct classifications of partially represented *CSs* according to granularity level

	WMMs			**WCFGs**		
Granularity Level	**Set-1**	**Set-2**	**Set-3**	**Set-1**	**Set-2**	**Set-3**
Token	N/A	N/A	N/A	98%	97%	96%
Paired Symbols	N/A	N/A	N/A	92%	93%	93%
Config. Rules	89%	79%	86%	89%	91%	90%

3. Scalability Evaluation Results

In general, the scalability of a system is a characteristic measured by its capability to maintain or increase its performance proportionally as it is applied to large situations. The scalability of an ITS is determined by its flexibility to support the addition of new resources or tools (Viccari et al., 2008). Within an ITS framework in which the knowledge is obtained or improved at run time, the ITS's knowledge-base represents a resource that continuously grows. Therefore, we assessed the scalability of the proposed ITS framework by measuring its capability to scale up and maintain its tutoring effectiveness when applied to large numbers of student input, in our case configuration sequences.

The scalability characteristic was evaluated by testing the ITS performance with three training and testing datasets with varying size. Each set consisted of approximately 33%, 66%, and 100% of the available data respectively. Since the number of students in each course is comparable, between 10 and 15 students per course, we randomly assigned the entire collection of data from each of the 10 courses to one of the three

generated datasets. In Table 8 we present the portion of the collected data used to build each of the three datasets used for testing the scalability performance of our proposed ITS framework. Table 8 displays the number of configuration sequences and configuration rules assigned to each dataset. Information presented in this table illustrates that the increment in the proportion of different configuration rules and different commands used by students, decreased as the dataset grew. However, there were always new *CRs* or sequence of *CRs* the ITS should learn.

Table 8
Distribution of collected data for scalability testing

	Courses	**Students**	***CSs***	***CRs***	**Diff. *CRs***	**Commands**
Dataset 1	3	42	170	750	411	85
Dataset 2	6	81	320	2110	812	103
Dataset 3	10	130	515	3612	1150	127

To assess whether the ITS framework performance scaled up properly in building and using the knowledge-base when applied to large situations, we carried out the same evaluations we described previously within each of the produced datasets. We randomly selected the training and testing datasets. Then we performed the classification process. Finally, we compared the results of the classification to determine whether the performance of the two data representation and policy learning approaches (WMMs and WCFGs) increased proportionally with regards to the ITS' growing knowledge-base.

To compare the classification results using different datasets we analyzed the final classification results obtained from the tested granularity levels. The comparison

was conducted for both WMM-based and WCFG-based classifiers. The results of this comparison considering partially represented *CSs* are presented in Table 9.

Table 9
Scalability results. Comparison of percentage of correct classifications of *CSs* according to granularity level

	WMMs			**WCFGs**		
Granularity Level	**Dataset-1**	**Dataset-2**	**Dataset-3**	**Dataset-1**	**Dataset-2**	**Dataset-3**
Token	N/A	N/A	N/A	94%	96%	98%
Paired Symbols	N/A	N/A	N/A	87%	87%	92%
Config. Rules	79%	82%	89%	80%	85%	89%

The outcomes of using a growing dataset approach for policy learning indicated a consistent increase in the classifiers' performance. Both of the classifiers used behaved consistently for the tested granularity levels of data representation we implemented. In particular, the results displayed in Table 9 show how for all granularity levels the percentage of correctly classified *CSs* increases monotonically with the available amount of data. For example, at finer granularity, the WCFG´s performance grows from 94% to 98% as the dataset increases. We also observed consistency among outcomes of previous analysis regarding the superior performance of the WCFG-based classifier based on a token granularity level.

H. Interactive Learning Experiments

In order to simulate a more realistic domain knowledge learning environment for our mixed-response ITS framework, we implemented an interactive learning module within the WAES system where instructors interacted with the ITS. Within this learning environment, the intelligent tutor starts learning from scratch by receiving students' *CSs* from the available dataset and then recommending tutorial actions based on its current knowledge confidence level. The instructor evaluates the ITS's tutoring performance and recommends new tutorial actions when needed. In this evaluation method we integrated the *policy learning* and *classification process* phases conducted within the batch learning approach presented in Section B. We used a portion of the data generated during the batch learning approach and then used the WCFG-based classifier at the token granularity level. We chose to use this particular classifier because of the positive results associated with it in the experiments conducted in Section B.

Within the *classification and policy learning* phases, around half of the entire dataset was used (227 *CSs*) for building the set of WCFGs. This dataset was obtained from five courses randomly selected from the 10 available. Then, the classification policy was generated at run time. Every time that a domain expert evaluated the correctness of tutorial actions provided by the ITS, the classification policy was rebuilt and prepared to classify a new *CS*. Specifically, within this evaluation we conducted three experiments in order to measure the ITS framework's *effectiveness* and *metacognitive skill*, *robustness*, and *scalability*. The main goal of this evaluation was to determine initial threshold values in which the confidence level of the ITS is effective in

providing feedback, robust in addressing diverse pedagogical perspectives and scales consistently when demonstration data increases.

Within the interactive learning approach, in order to assess efficiency, robustness and portability of the ITS, three evaluations were conducted: (1) an evaluation was conducted using four different instructors or experts in cybersecurity. The sequence of the *CSs* within the used dataset was identical for all four instructors. (2) A second experiment consisted of three evaluations conducted for one of the instructors. This instructor used the same dataset but with three different sequences of *CSs* that had been randomly reordered. The primary goal of these first two evaluations was testing the efficiency and robustness of the ITS framework performance without decreasing its effectiveness. The ITS efficiency was measured based on how quickly the intelligent tutor began working independently of the human instructor to provide adequate feedback. The ITS's robustness refers to its consistency regarding the classification of *CSs* and generation of the appropriate tutorial action under different pedagogical perspectives and occurrences of *CSs*.

Finally, (3) a third experiment was conducted to evaluate the portability of the proposed ITS framework when applied in a domain different from cybersecurity. Sequential data was obtained from an undergraduate *database* course. The evaluation was carried out with the help of an instructor who used the interactive learning environment for the cybersecurity dataset. Details and illustrations regarding the implemented ITS learning environment and the performed evaluations are described in the following subsections.

1. ITS Interactive Learning Environment

The *interactive learning* environment for ITS training, as shown in Figure 19, is a workspace for evaluating the ITS tutoring performance and providing additional tutoring demonstrations for policy learning. The implemented environment allows instructors to interact with the intelligent tutor and provide tutoring demonstrations.

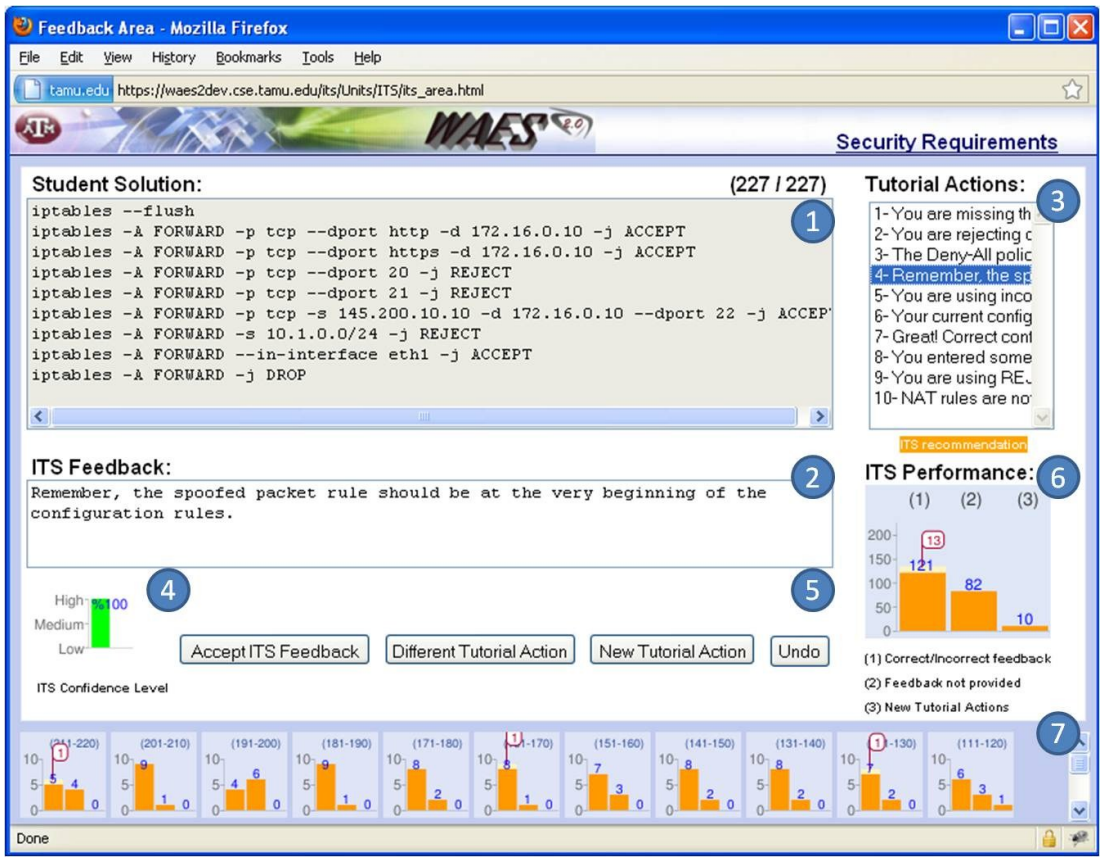


Figure 19. Workspace for training and evaluating the ITS tutoring performance.

The environment consists of seven main sections: (1) the "Student Solution" section provides the student answer to a given exercise; (2) the "ITS Feedback" is an editable area where the ITS displays its proposed feedback or the human expert enters new tutorial actions; (3) the "Tutorial Actions" list includes the set of learned tutorial

actions from the expert; (4) the "ITS Confidence Level" meter shows the level of confidence computed by the ITS regarding the tutorial action recommended for the currently analyzed student solution; (5) the set of buttons used by the instructor to "Accept ITS Feedback", select a "Different Tutorial Action" from the available, recommend a "New Tutorial Action", and "Undo" the previous instructor action; (6) the "ITS Performance" chart which displays statistics about the ITS's performance regarding correct and incorrect feedback given (the number of incorrect feedback is shown within the flag), number of times when the ITS determined not provided feedback, and number of learned tutorial actions; and (7), a set of charts describing in detail the ITS tutoring performance for every ten analyzed solutions. The charts are ordered sequentially from left to right according to the most recent activity.

In Figure 19 we present an example of a student's *CS* for which the intelligent tutor performed the classification of the *CS* with a 100 percent confidence level. This percentage of confidence is the result of an identical *CS* learned by the ITS from a previous student. Therefore, the ITS determined to use the corresponding tutorial action as a result of the classification's high confidence level. The classification confidence level might vary based on the similarity of the *CS* in need of classification and the current knowledge available in the classification policy (clusters of *CSs*). In Figure 20 we can see an example of a classified *CS* in which the intelligent tutor estimated its confidence level as 32 percent. Even though the ITS classified the new *CS* within the cluster corresponding to tutorial action number 2, the ITS determined not to use this tutorial action because of its low confidence level. This low confidence level is a result

of partial similarity of the new *CS* with those included within the selected cluster of previously learned *CSs*. Specifically, for this given *CS*, the ITS detected that the student is rejecting or accepting traffic incorrectly. However, there is a combination of incorrect configurations (e.g., accepting traffic incorrectly, rejecting traffic correctly but using the DROP parameter instead of REJECT, and incorrect order of configuration rules) that made the ITS estimate a low confidence level. In this case, the instructor should intervene in order to indicate to the ITS which of the current tutorial actions should be used or to provide a new tutorial action in case the new *CS* includes unknown misconceptions (see Appendix B for specific student and instructor scenarios).

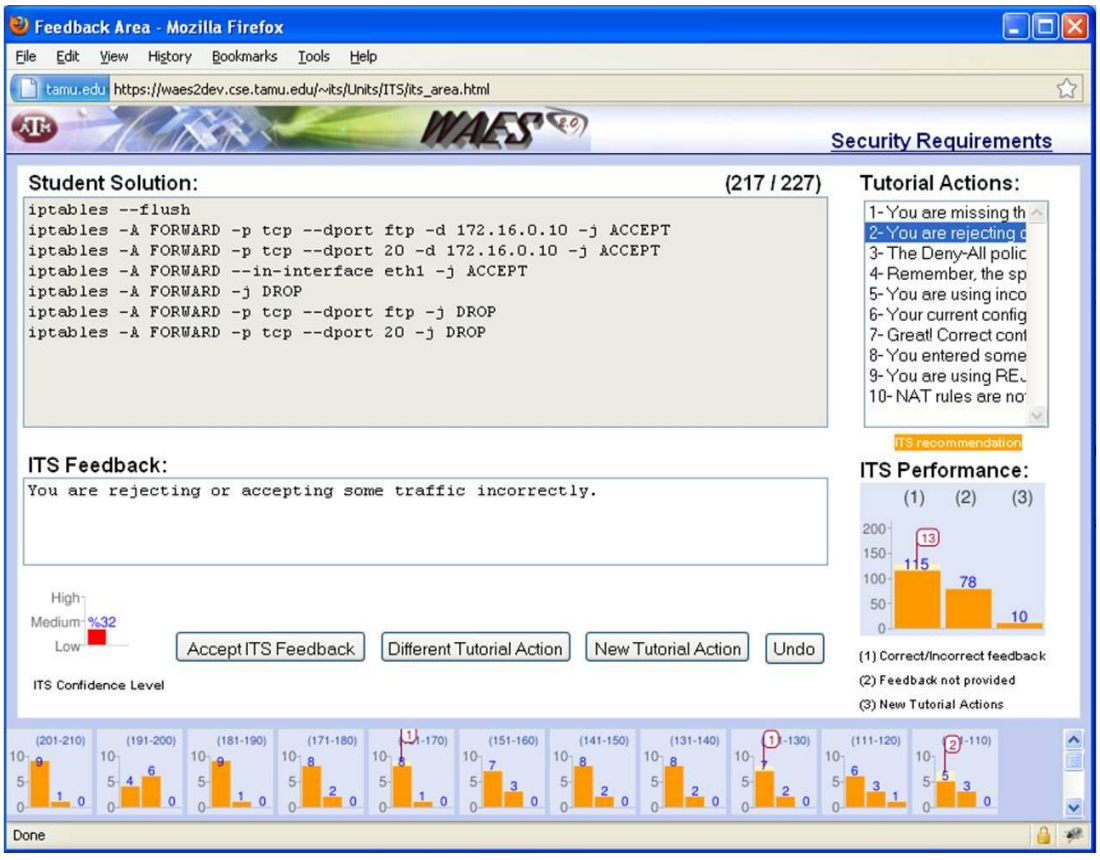


Figure 20. Low confidence level classification.

In order to support the ITS's training and evaluation, the instructor has access to the exercise's requirements and to an example of a correct solution, as shown in Figure 21. Also, the instructor can revisit the set of solutions already classified within each tutorial action's cluster (see Figure 22). By allowing instructors to see the set of students' solutions assigned to one specific tutorial action, they are able to verify consistency among the provided solution-tutorial action demonstrations.

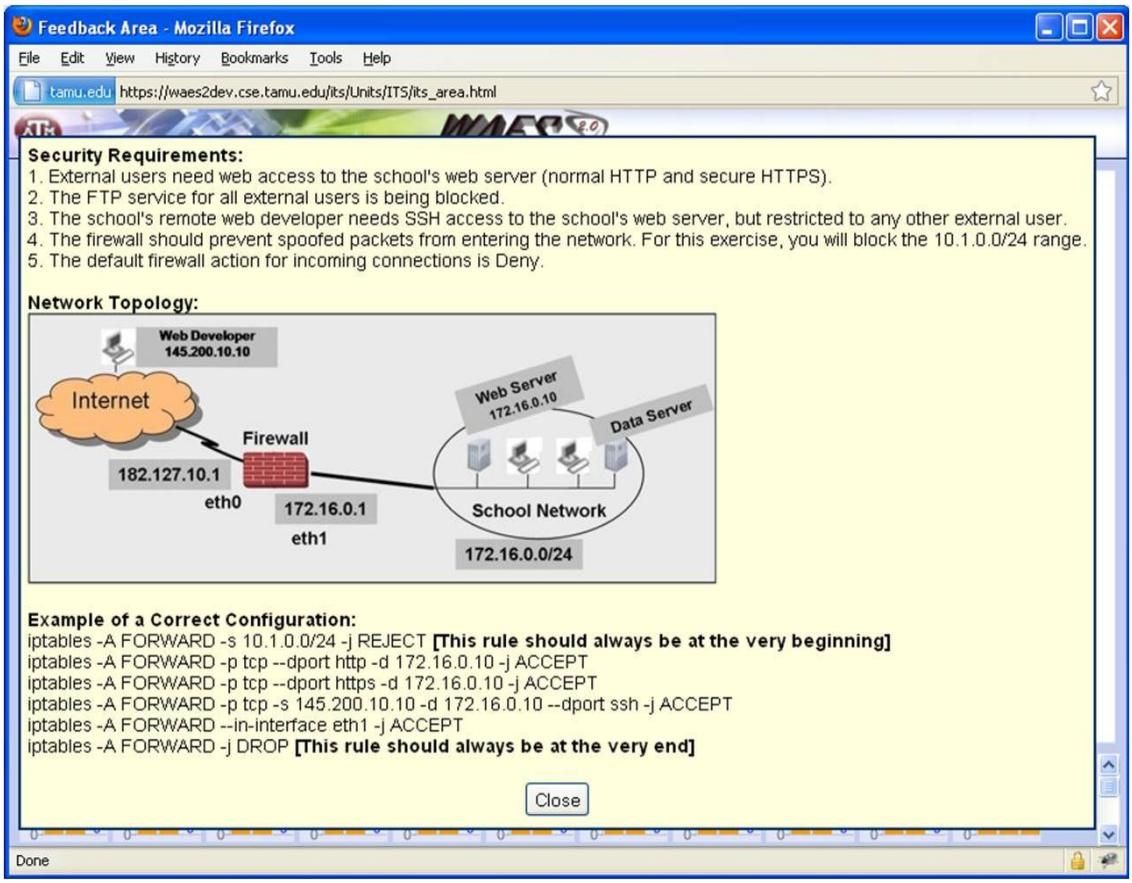


Figure 21. Description of the exercise's requirements and example of correct solution.

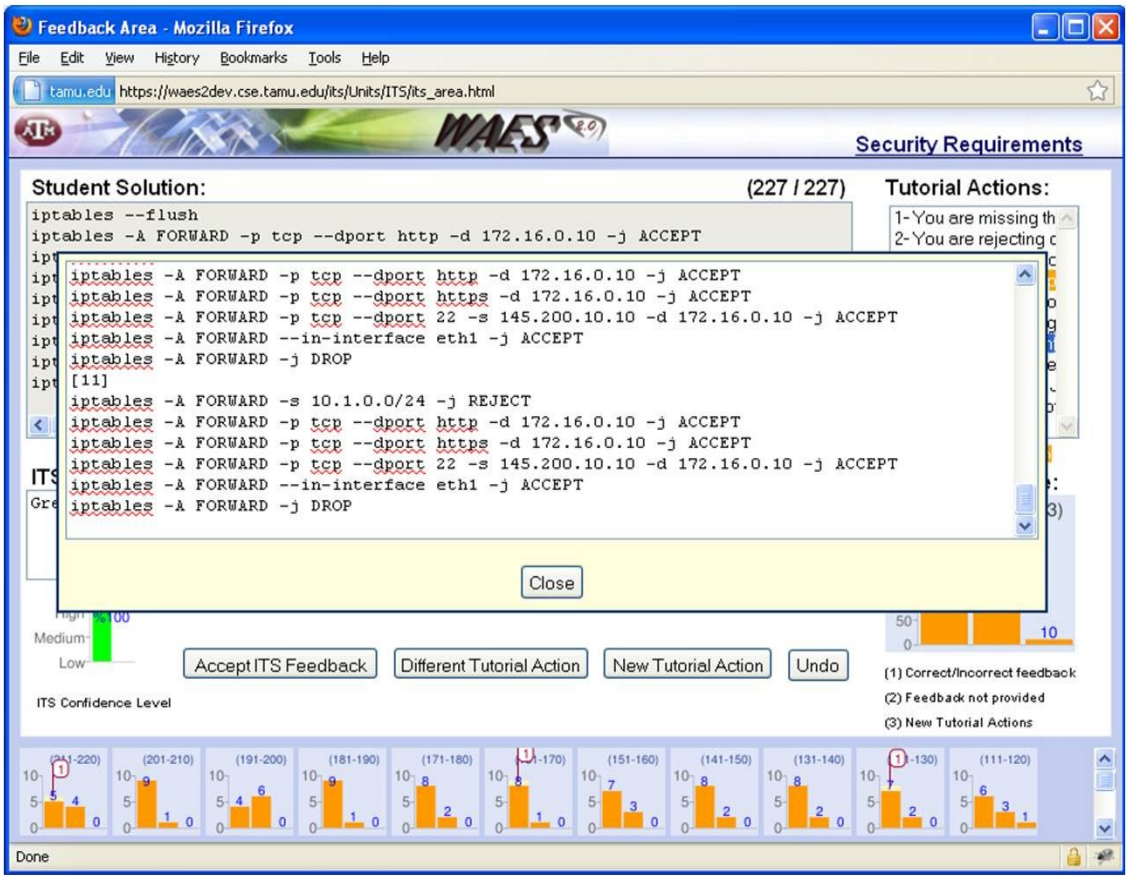


Figure 22. Set of student's solutions assigned to a tutorial action's cluster.

Once the instructor and the ITS have finished the classification of the entire dataset including the students' solutions, a report summarizing the ITS performance is given to the expert. This report includes the number of analyzed students' solutions, the total number of learned tutorial actions, the number of students' solutions in which the ITS provided feedback, and the percentage of correct and incorrect feedback provided (see Figure 23). In addition, a graphic depicting the evolution of the ITS performance throughout the entire dataset is included. This graphic includes information regarding correct and incorrect feedback, as well as the solutions not addressed by the ITS.

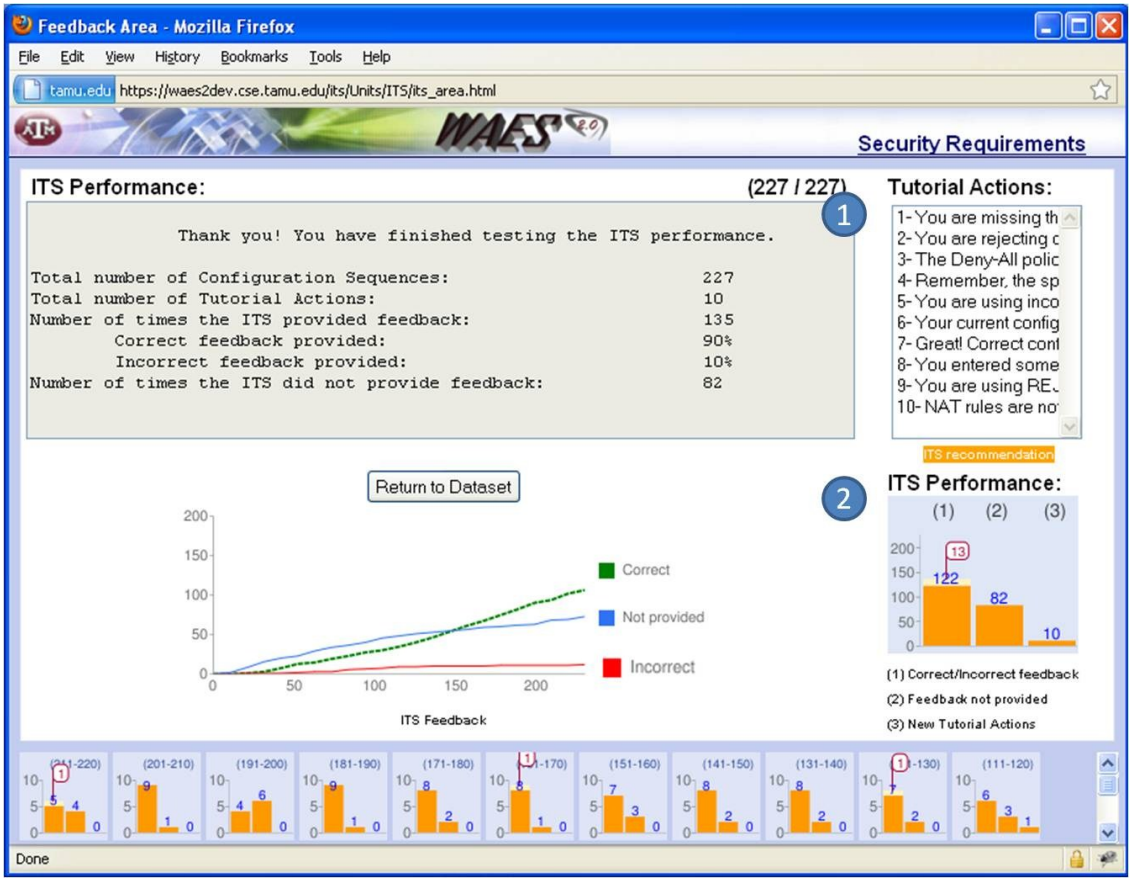


Figure 23. Statistical report on ITS performance.

2. Efficiency and Robustness Evaluation Results

Most of the research aiming to improve the ITS efficiency is based on reducing the time required for students to learn specific knowledge (Cen, Koedinger, & Junker, 2007). However, some research measuring the efficiency of an intelligent tutor authoring system considers efficiency as the ability to reduce the amount of time required by an expert to build an ITS (Lynch et al., 2006). The efficiency we measured within our study focuses on this latter approach. Given that our research work focuses on enhancing the ITS performance while building its knowledge-base, the efficiency analysis we

conducted was based on the effort it took for the instructors to obtain reliable tutoring responses from our proposed ITS framework.

The *efficiency* and *robustness* evaluation of the proposed ITS framework in building its knowledge-base was twofold. By conducting these experiments we evaluated the similarity of the ITS's performance when: (1) the same content and sequence of *CSs* is used to train the ITS by a set of different experts, as opposed to a single expert; and (2), similar to the evaluation conducted within the *batch learning* approach, the same expert interacted with the intelligent tutor by using the same dataset sorted in three different ways. Therefore, we used *Multiple Expert Evaluation* and *Multiple Dataset Evaluation* settings to measure the ITS's efficiency and robustness.

Multiple Expert Evaluation Results

To compare the classification results using different instructors we analyzed the final outcomes obtained from the WCFG-based classifier and token granularity level for data representation. In Figure 24 we graphically present the outcomes of the ITS learning and tutoring performance from four experts using 227 *CSs*. Table 10 shows the summary of the evaluation results. Table 11 shows the percentage of *CSs* that received a tutorial action from the ITS and their evaluation result.

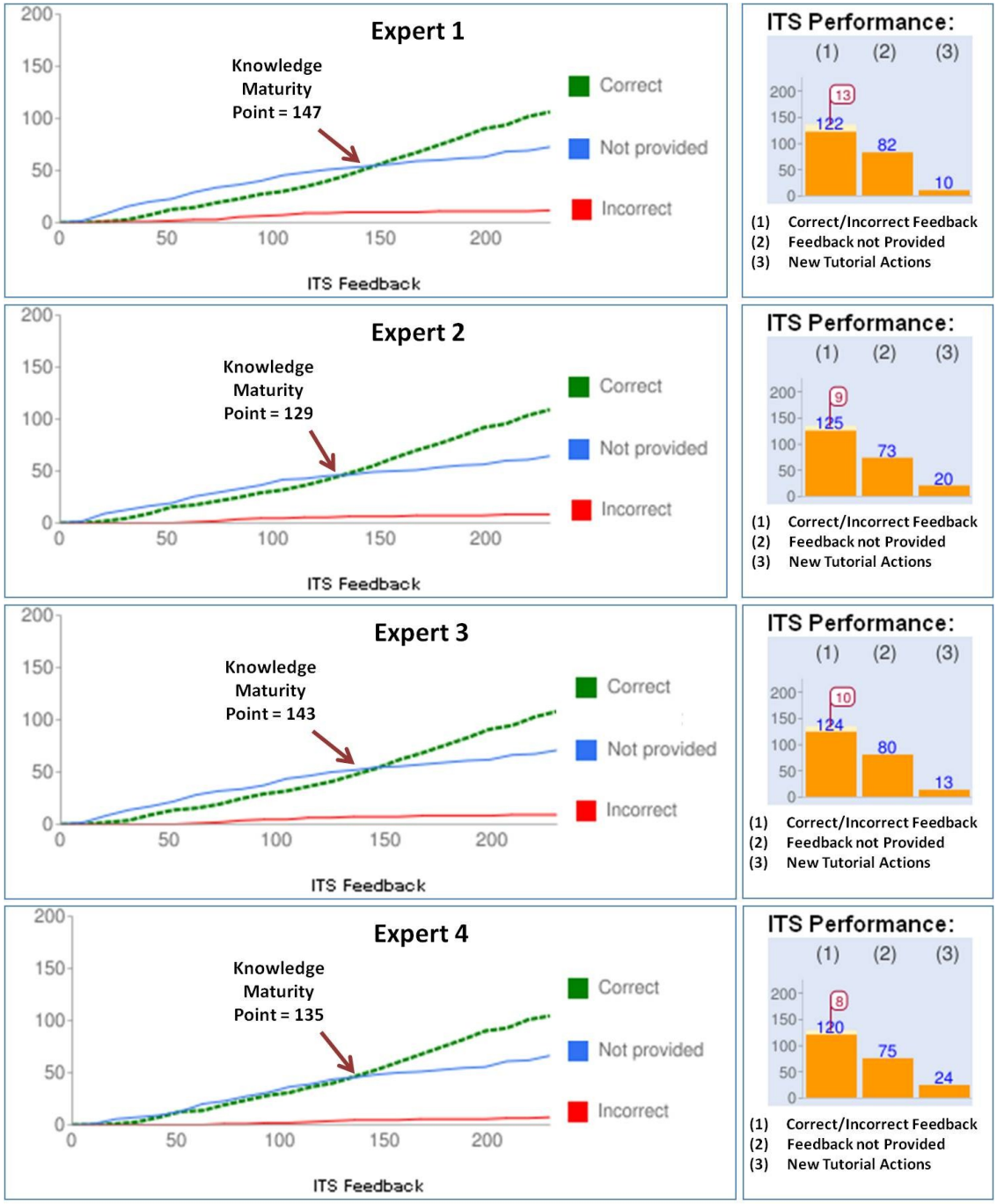


Figure 24. ITS's Learning and tutoring performance from four experts.

Table 10
ITS's learning and tutoring performance from four experts

	Feedback Provided		Feedback Not Provided	Tutorial Actions	Knowledge Maturity	Training Time
	Correct	Incorrect				
Expert 1	122	13	82	10	147	1 hr
Expert 2	125	9	73	20	129	1.25 hrs
Expert 3	124	10	80	13	143	1.42 hrs
Expert 4	120	8	75	24	135	3 hrs

Table 11
Percentage of feedback provided by the ITS from four experts

	Feedback Provided	Feedback Not Provided	Feedback Provided		Knowledge Maturity	
			Correct	Incorrect	Correct	Incorrect
Expert 1	59%	41%	90%	10%	97%	3%
Expert 2	59%	41%	93%	7%	96%	4%
Expert 3	59%	41%	93%	7%	96%	4%
Expert 4	56%	44%	94%	6%	95%	5%

From the obtained outcomes we observed very little variance in the classification performance of the ITS: for most of the instructors, around 59% of the time the ITS recommended a tutorial action (see feedback provided in Table 11). From this 59%, around 93% of the time the provided feedback was correct (see correct feedback provided in Table 11). Around 7% of the feedback provided by the ITS turned out to be incorrect. However, in three out of four cases, the majority of the incorrect feedback occurred earlier, i.e. in the first half, in the ITS learning process. The occurrence and amount of incorrect feedback is illustrated in Figure 25 by the number within the little flag in each chart. We also observed little variability in the number of *CSs* in which the ITS did not provide feedback because of its low confidence level, around 41% (see "Feedback Not Provided" in Table 11). Furthermore, there was consistency in the point

where the ITS started having a better confidence level to provide feedback. We refer to this point as the ITS's *knowledge maturity point* (see Figure 24). We say that the ITS reaches knowledge maturity when the amount of *CSs* tutored by the ITS starts to be larger than those not tutored. We also observed that the percentage of correct feedback provided after the knowledge maturity point increased to 97% (see "Correct Feedback under Knowledge Maturity" in Table 11).

We highlight here that the time spent by the instructor ranges from 1.0 to 3.0 hours to help the ITS in building the set of tutorial actions (see "Training Time" in Table 10). These numbers are significantly lower than the ten hours of work to produce production rules and constraints at design time within the mathematics domain reported by Koedinger et al. (2004) and Anderson et al. (1995). The time spent by instructors was slightly larger than that reported by Mitrovic (1998) while building the ITS knowledge-base using the constraint based approach within the databases domain. However, in our experiments, the spent time was used to build an entire database exercise as opposed to 1.1 hours used to implement a constraint. We assume this time can be further reduced as the instructors gain experience using the training environment. Furthermore, the advantage of the ITS learning at run time is that the required time to train the ITS is distributed and continuously decreases during the use of the ITS.

Expert 1

Expert 2

Expert 3

Expert 4

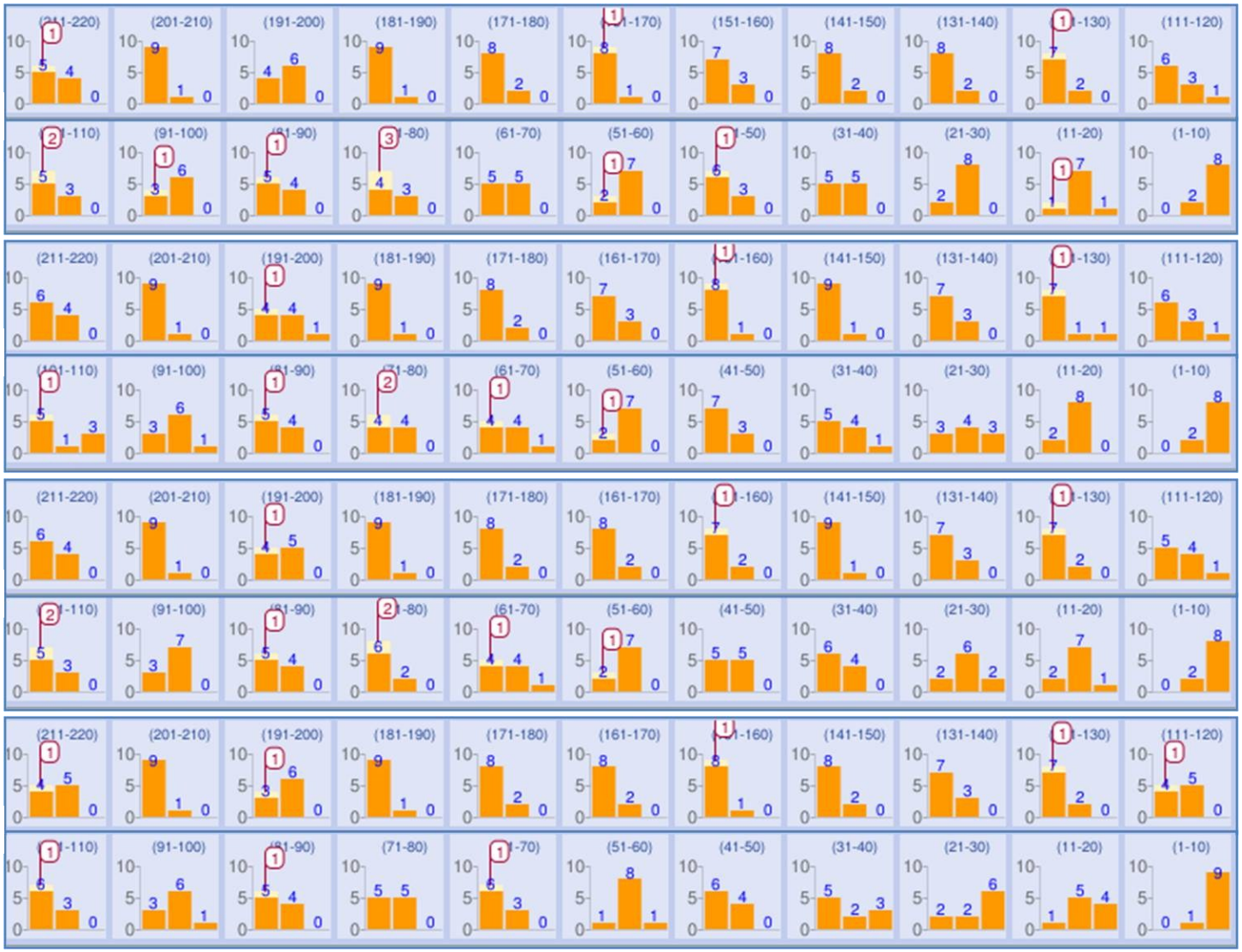


Figure 25. ITS's learning and tutoring detailed performance from four experts.

Figure 25 describes how the ITS tutoring performance develops during the training period. The four sets of charts in Figure 25 illustrate the tutoring behavior per every ten *CSs*; a set consists of 22 charts obtained from each expert's evaluation. The chart including the first ten *CSs* is located in the bottom-right hand side of each set of charts, and the chart located in the top-left hand side represents the last ten *CSs* analyzed by the expert. Table 12 gives a detailed overview of the ITS's tutoring performance for each of the four experts.

Table 12
ITS's learning and tutoring detailed performance from four experts

	Tutorial Actions	**Provide Less Feedback**	**Provide More Feedback**	**Incorrect Feedback**	**Correct Feedback**
Expert 1	First 20 *CSs*	1-110	110- +	50-130	130- +
Expert 2	First 40 *CSs*	1-110	110- +	50-130	130- +
Expert 3	First 30 *CSs*	1-110	110- +	50-130	130- +
Expert 4	First 40 *CSs*	1-100	100- +	60-130	120- +

The first column in Table 12 indicates that most of the tutorial actions were provided by instructors within the first 20 to 40 analyzed *CSs*. However, the intelligent tutor did not start using those tutorial actions until more demonstrations per tutorial action were obtained; this happened after analyzing around 40 to 50 *CSs*. We observed that after having between 5 to 10 *CSs* per tutorial action the ITS started computing high confident levels for partially represented *CSs*. The pattern of the ITS not providing feedback to most of the analyzed *CSs* continues until around 110 *CSs*. In addition, during this time, the ITS had a poor effectiveness providing feedback. Around 75% of

the incorrect feedback occurred within *CSs* 50 and 130. After analyzing around 120 or 130 *CSs* the ITS started providing more feedback, and the percentage of correct feedback was around 96%.

Important information was obtained from this data examination, such as when a stricter threshold for confidence level estimation is needed and the amount of demonstrations needed to run batch learning training. For instance, we observed that we needed a stricter threshold value for clusters where the number of *CSs* is lower than 10. Furthermore, the number of demonstrations to train an ITS using our proposed ITS framework should consist at least of 150 demonstrations in order to reach the knowledge maturity level required by the ITS. Based on results from the batch learning evaluations, we observed little variability regarding the effectiveness of the ITS framework performance. Furthermore, evaluation outcomes supported the ITS framework's robustness when applied to various learning situations (different instructors' pedagogical perspectives). Finally, efficiency regarding the number of demonstrations and time required by instructors to complete the evaluation and training process of the ITS was significantly lower than the time reported by previous research work.

Multiple Datasets Evaluation Results

The interactive learning environment we implemented allowed us to conduct further tests regarding the robustness of the ITS performance, similar to the one conducted within the batch learning approach. Within this second study we evaluated consistency in effectiveness of the ITS interacting with the same instructor when applied to a dataset

sorted in different ways. We analyzed the final outcomes obtained after using the confidence level step for the WCFG-based classifier and token granularity level for data representation. In Figure 26 we graphically present the outcomes of the ITS's learning and tutoring performance from the three datasets through 227 *CSs*. Table 13 presents the summary of the evaluation results. Table 14 shows the percentage of *CSs* that received a tutorial action from the ITS and their correctness evaluation.

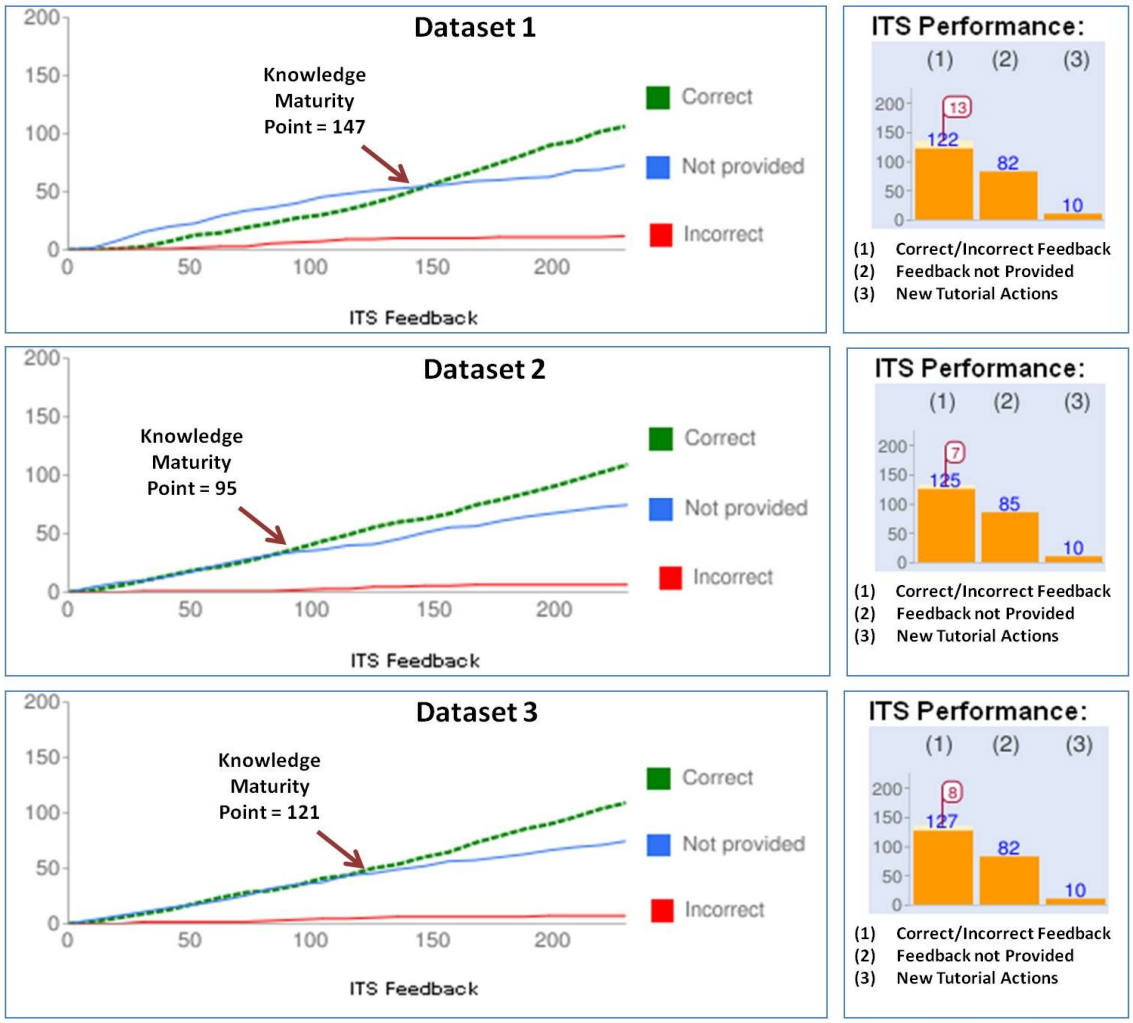


Figure 26: ITS's learning and tutoring performance from three datasets.

Table 13
ITS's learning and tutoring performance from three datasets

	Feedback Provided		Feedback Not Provided	Tutorial Actions	Knowledge Maturity	Training Time
	Correct	Incorrect				
Dataset 1	122	13	82	10	147	1 hr
Dataset 2	125	7	85	10	95	.83 hrs
Dataset 3	127	8	82	10	121	.75 hrs

Table 14
Percentage of feedback provided by the ITS from three datasets

	Feedback Provided	Feedback Not Provided	Feedback Provided		Knowledge Maturity	
			Correct	Incorrect	Correct	Incorrect
Dataset 1	59%	41%	90%	10%	97%	3%
Dataset 2	58%	42%	95%	5%	95%	5%
Dataset 3	60%	40%	94%	6%	96%	4%

We observed that for all of the datasets, around 59% of the time the ITS recommended a tutorial action. Of these 59%, the ITS provided correct feedback up to 95% of the times, while around 6% of feedback was incorrect (see "Correct and Incorrect Feedback" provided in Table 14). We observed that for around 41% of the *CSs*, the ITS did not provide feedback because of lack of sufficient confidence. All these results are consistent with the outcomes of the experiment using multiple experts. We observed a small reduction in the point of knowledge maturity. However, similar to initial studies, an average of 120 *CSs* were required as a minimum to reach the maturity of knowledge.

This time, the expert reported a significant reduction in the time required to review the entire set of *CSs* from 1 to .75 hours. He mentioned that this was because he

already knew how to use the training environment and was familiar with the potential errors within the *CSs* dataset.

In Figure 27 we present the sets of charts detailing the ITS tutoring performance while training using the three datasets. In Table 15 we present relevant information obtained from these sets of charts. From information in Table 15 we observe that tutorial actions were provided by the instructor within the first 20 to 50 analyzed *CSs*. This difference was because of the ill-definedness of our domain. Hence, we expected new tutorial actions even at later time during the ITS learning process, as is illustrated by the new tutorial action within the last set of ten analyzed *CSs* in Dataset 2 (see top-left hand side chart from Dataset 2 in Figure 27).

Again, during the analysis of the first 50 to 130 *CSs* the ITS provided poor effectiveness providing feedback. After analyzing around 120 *CSs* the ITS performance improved regarding the feedback provided. This time, the ITS consistently reported correct feedback for around 96% of the cases. This experiment's outcomes support the previous conclusions regarding when to apply a stricter threshold for confidence level estimation, as well as the amount of demonstrations needed for batch learning training.

Table 15
ITS's learning and tutoring detailed performance from three datasets

	Tutorial Actions	**Provide Less Feedback**	**Provide More Feedback**	**Incorrect Feedback**	**Correct Feedback**
Dataset 1	First 20 *CSs*	1-110	110- +	50-130	130- +
Dataset 2	First 40 *CSs*	1-70	70- +	80-120	120- +
Dataset 3	First 50 *CSs*	1-90	120- +	80-130	130- +

Dataset 1

Dataset 2

Dataset 3

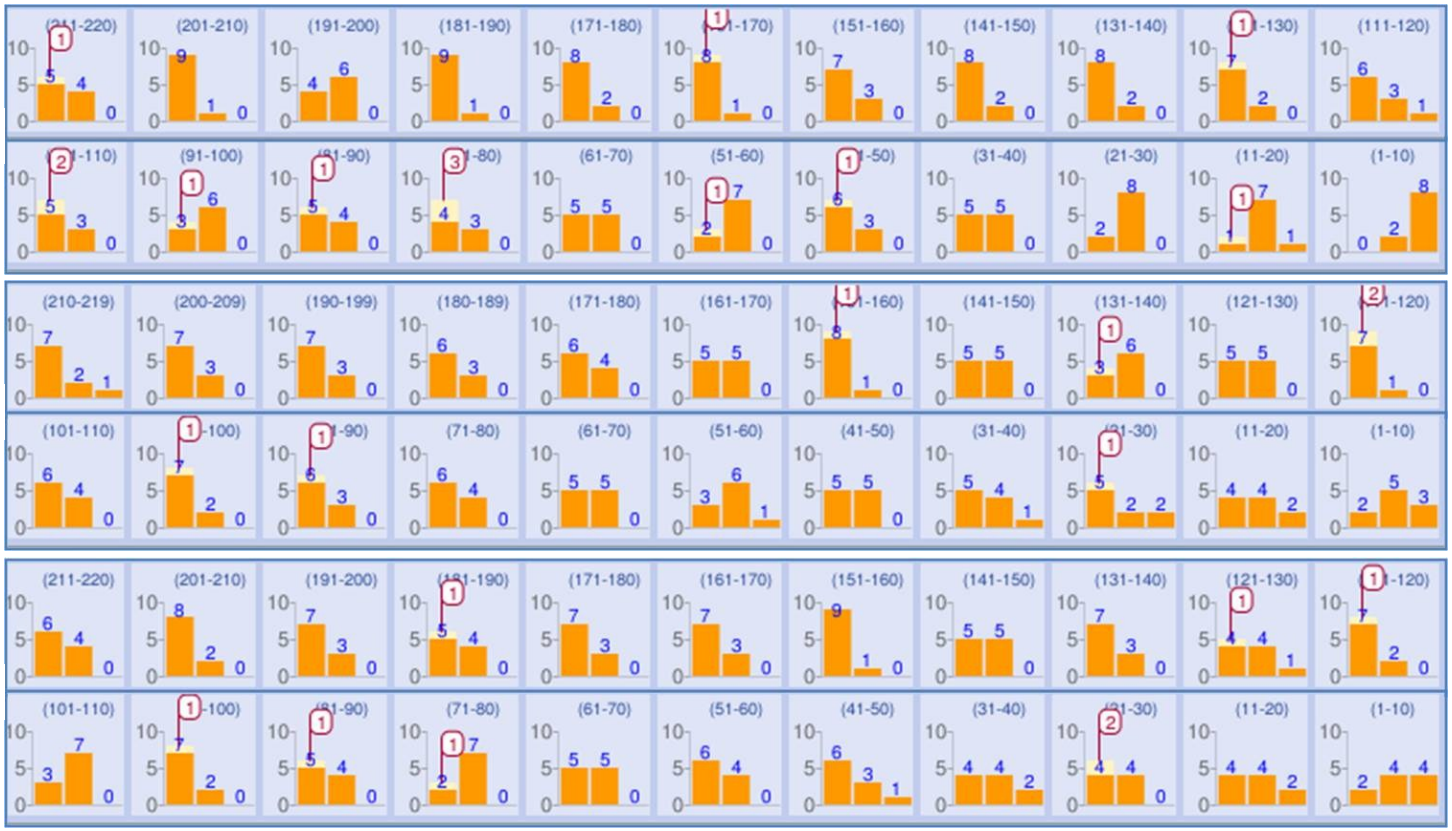


Figure 27: ITS's learning and tutoring detailed performance from three datasets.

3. Portability Evaluation Results

ITS frameworks supporting *portability* across domains is an open research topic within the intelligent tutoring field. Development of portable ITS's frameworks is an important goal primarily within the intelligent tutors authoring systems area (Lynch et al., 2006). Domain independent frameworks aim to avoid the redesign of internal functionality or modules within an ITS. Hence, they can be utilized by end users focused on the delivery of instructional content and problem solving expertise instead of the ITS internal functionality and knowledge representation.

Next we describe some research about ITS authoring systems supporting the development of domain-independent ITSs for well- and ill-defined problem solving. Within the well-defined area, Cognitive Tutor Authoring Tools (CTAT) is a freely available and well known ITS authoring system adopted by the ITS community (Lynch et al., 2006). Examples of implementations using CTAT include domains such as genetics, politeness rules, and arithmetic. ASTUS is a more recent domain-independent ITS framework supporting the development of intelligent tutors for helping students in problem-solving (Paquette, Lebeau, & Mayers, 2010). There are some empirical studies regarding advantages and disadvantages between these two frameworks (Nkambou, et al., 2010). However, both of them have been successfully adopted by the ITS community because their domain-independent effectiveness.

Researchers within ill-defined domains have developed some domain-independent ITS frameworks as well (Fournier-Viger et al., 2010). Rashi is a domain-independent ITS authoring system based on an inquiry learning teaching mode (Dragon

et al., 2006). This authoring system is intended to develop ITSs based on requesting students to generate hypotheses regarding a situation and argumentations supporting or refuting the posed hypotheses. There exist implemented ITSs using Rashi in domains such as geology, art history, and human biology (Dragon et al., 2006). Specifically, for the implementation of ITSs supporting free-text discussions using Rashi, the domain-knowledge is represented by a *direct-acyclic* graph linking related concepts with evidence supporting or refuting those concepts. The domain knowledge is generated at design time by domain experts (Dragon, Floryan, Woolf, & Murray, 2010).

Different from the free-text data handled by Rashi authoring system, our approach is focused on representing knowledge consisting of sequence of configuration rules and commands (sequential data). In order to evaluate the potential portability of using our proposed ITS framework, we conducted an additional evaluation using data from a domain different than cybersecurity using a similar type of data. The data was gathered from the *database* domain. Specifically, we experimented with the use of our system to teach Structured Query Language (SQL) to computer science students.

Data from SQL-query problems and the *CRs* used in the previous setting have some similarities and differences. Two examples of straightforward solutions from the analyzed domains are shown in Figure 28. In Table 16 we present a comparison of characteristics between solutions from the two domains. Most of the characteristics described in Table 16 indicate similarity between the solutions from the two domains. Principally, we emphasize similarities in: (a) the existence of optional and non-optional string elements (*CRs'* parameters/Queries' clauses) and use of aliases; (b) the sequence

of the string elements is affected by other used elements (context sensitivity); (c) syntactic mistakes are addressed by the learning environment; and (d), the use of discrete data.

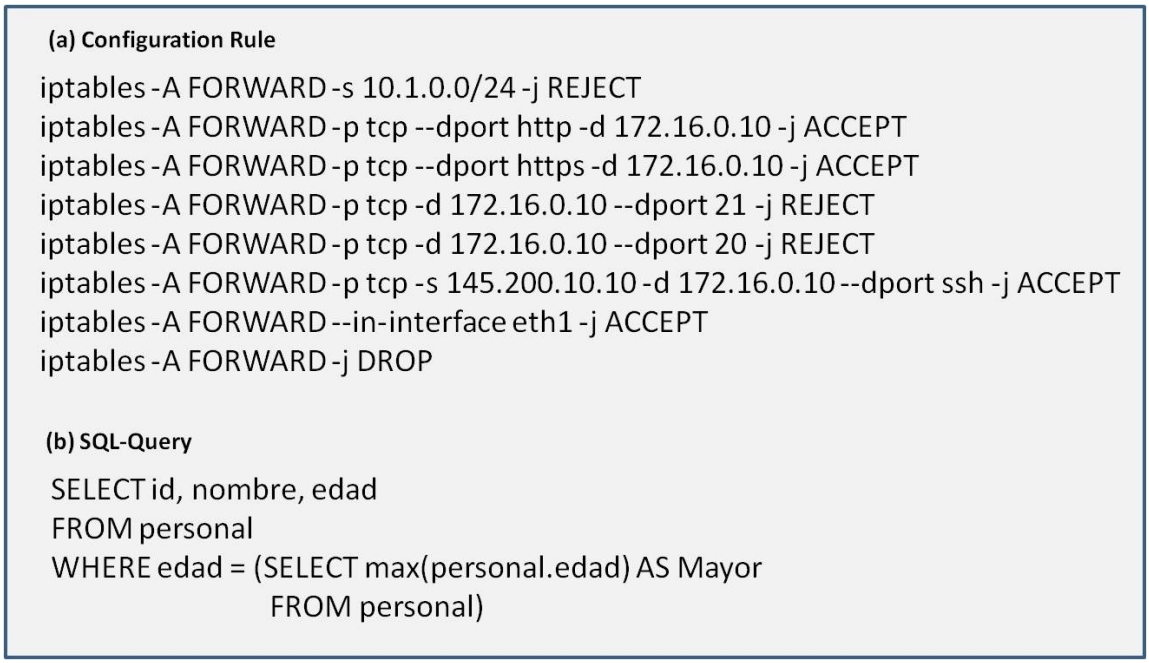


Figure 28. Solution examples from cybersecurity and database domains. (a) Solution from the cybersecurity domain including multiple *CRs*. (b) Solution from the database domain including nested queries and aliases.

There also exist partial similarities such as: (a) the sequence of clauses within queries is completely constrained, while for *CRs* sequences there exist constraints only for some parameters; (b) there exist multiple "correct" solutions for both domains, however, the number of "correct" solutions in the cybersecurity domain tend to be larger than those in SQL-query domain; and (c), the level of difficulty to determine whether a given solution is completely correct is larger for the cybersecurity domian. However, for

highly complex problems, practitioners from both domains often detect misconceptions quickly by analyzing erroneous solutions.

Table 16
Comparison of characteristics between solutions from SQL-query and *CRs*

Characteristic	***CRs***	**SQL-Query**
Multiple SQL-queries or *CRs* are required in one solution?	Yes	No
Is constrained the order of *CRs'* parameters/Queries' clauses?	Regularly	Yes
Use of optional and required *CRs'* parameters/ Queries' clauses?	Yes	Yes
String elements (*CRs'* parameters/Queries' clauses) dependency?	Yes	Yes
Use of nested string elements (*CRs'* parameters/Queries' clauses)?	No	Yes
Use of discrete values for *CRs'* parameters/Queries' clauses?	Yes	Yes
Use of aliases or abbreviations?	Yes	Yes
The working environment provides syntactic errors feedback?	Yes	Yes
Multiple "correct" solutions?	Yes+	Yes-
Easy to identify "correct" solutions?	No	Regularly

Finally, the main differences between these domains are: (a) multiple *CRs* are required in one solution (implying larger solutions) while often a single SQL-query is sufficient to solve a database querying problem; and (b), nested SQL-queries (sub-queries) can be used to combine multiple outcomes in order to address complex database information retrieval needs. The nested functionality is a characteristic not similarly implemented within the use of *CRs*.

Portability Evaluation Results

In order to perform our evaluation, data for testing the portability criteria was obtained from a *database* course taught in a university in México. Data was obtained from 30 students over a time period of three semesters. Data consisted of students' SQL-queries from an exercise consisting of seven different SQL-query requests. From this dataset, we selected three of the most complex queries for the evaluation of the ITS framework. Around 150 students' SQL-queries were used for each of the tested exercises. We used the implemented token level WCFG-based classifier for data representation. The evaluation of the ITS framework using the SQL-queries data was conducted by the author of this research and the help of another instructor.

In Figure 29 we graphically present the outcomes of the ITS learning and tutoring performance from three datasets consisting of SQL-queries. In Table 17 we present the summary of the evaluation results. Table 18 shows the percentage of SQL-queries that received a tutorial action from the ITS and their correctness evaluation. We obtained similar outcomes from this evaluation. Specifically, the third experiment reported low variability in the classification performance across the datasets; while the other two were not an exact match, their differences were minor. The ITS provided feedback for around 65% of the SQL-queries. Around 95% of the provided feedback was correct and 5% incorrect (see Table 18). Hence, the percentage of correct feedback was only marginally smaller than the one obtained for the cybersecurity datasets, and the number of SQL-queries receiving feedback was 5 percentage points larger. We obtained slight differences in the knowledge maturity points among the three SQL-query datasets.

We considered that this difference was primarily influenced by the smaller size of solutions required to solve exercises within the SQL-query exercises, and by differences in the number of tutorial actions provided per each exercise. The time required to analyze each of the datasets was around .94 hrs. The improvement in the time spent was primarily influenced by the size of the students' solutions and the amount of analyzed demonstrations.

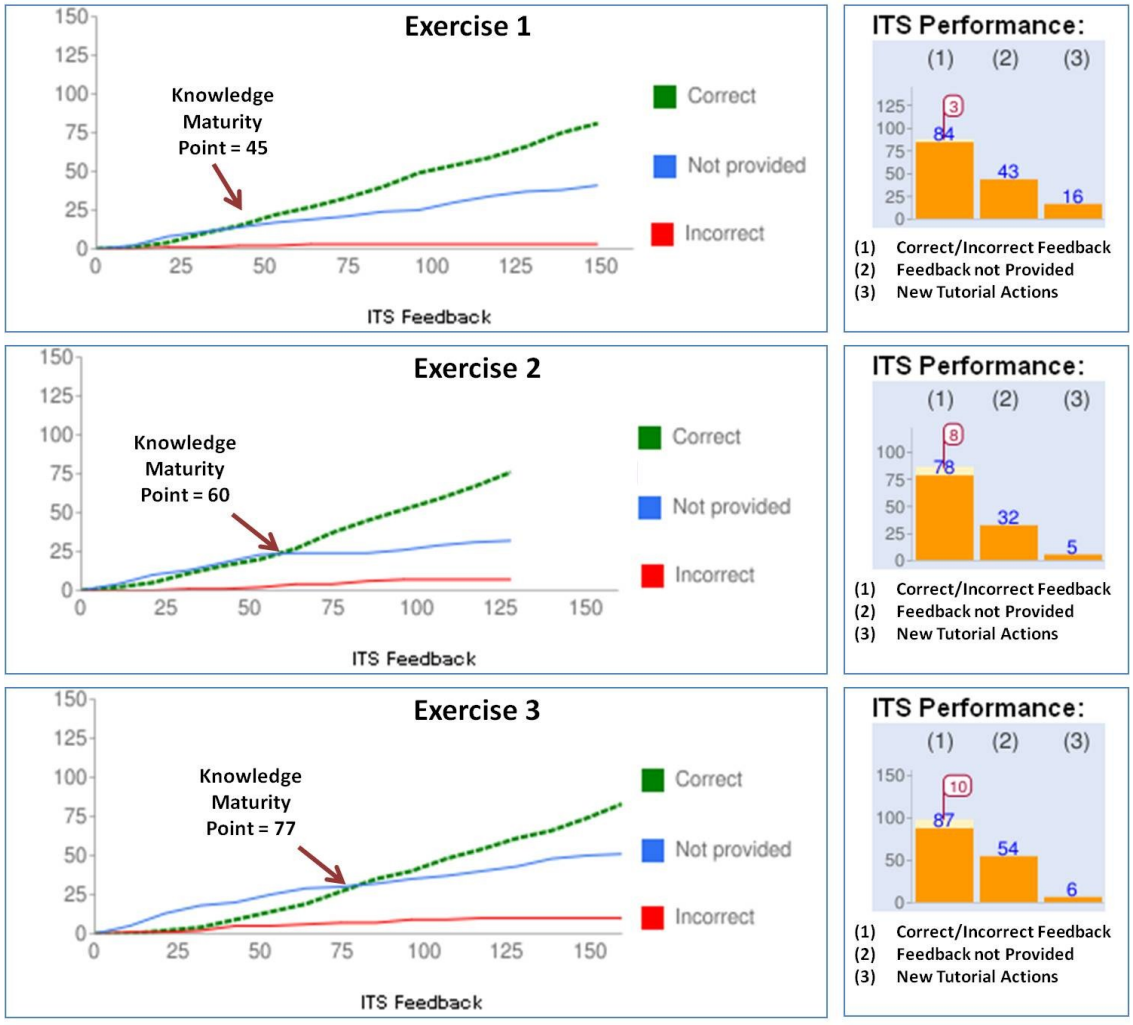


Figure 29. ITS's learning and tutoring performance from three SQL-query exercises.

Table 17
ITS's learning and tutoring performance from three SQL-query exercises

	Feedback Provided		Feedback Not Provided	Tutorial Actions	Knowledge Maturity	Training Time
	Correct	Incorrect				
Exercise 1	84	3	43	16	45	1.10 hrs
Exercise 2	78	8	32	5	60	.92 hrs
Exercise 3	87	10	54	6	77	.81 hrs

Table 18
Percentage of feedback provided by the ITS from three SQL-query exercises

	Feedback Provided	Feedback Not Provided	Feedback Provided		Knowledge Maturity	
			Correct	Incorrect	Correct	Incorrect
Exercise 1	60%	40%	96%	4%	98%	2%
Exercise 2	69%	31%	91%	9%	93%	7%
Exercise 3	62%	38%	90%	10%	95%	5%

These outcomes support the results obtained in the cybersecurity domain tests. Within the database domain, the proposed ITS framework demonstrated: (a) *effectiveness* by providing correct feedback; (b) *robustness* by being consistent using different datasets; (c) *efficiency* by reducing the time required to build the knowledge-base; (d) *metacognitive* skill by determining when to provide feedback and a point of *knowledge maturity*; and (e) *portability* by its ability to be used within domains using sequences of symbols.

I. ITS Framework Implementation

Determining the adequate amount of knowledge needed to release a reliable intelligent tutor is not always an evident task. The complexity of this task increases when the knowledge to be represented corresponds to tutoring students how to solve ill-defined

problems. As we discussed earlier, for ill-defined problems, continuous experts' participation is recommended in order to address unexpected or novel situations and to provide the adequate feedback. We propose to leave outliers and pedagogically interesting situations to the human instructor to handle and routine situations to the ITS. However, an important question should be considered for a successful implementation of the proposed ITS framework: how does one determine the amount of knowledge needed by the ITS before releasing it to be used by students? Results from our experiments, mainly those using the interactive learning environment, gave us the following insights regarding how to determine when the ITS has enough knowledge to identify the most common acceptable solutions and students misconceptions for solving ill-structured problems:

1. Even though students can use the intelligent tutor starting with an empty knowledge-base, there are two relevant points within the ITS learning process: (1) the proposed ITS framework tends to not provide feedback until a certain number of demonstrations have been provided, and (2) an approximation of the number of required demonstrations to accomplish the higher percentage of correct feedback (knowledge maturity) can be identified. Hence, we recommend using a batch learning approach to train the ITS prior to system release.
2. The number of demonstrations required for the batch learning approach in order to accomplish the ITS's knowledge maturity level is influenced by the domain and complexity of the exercises. We observed consistency among exercises from the same domain and similar complexity regarding the number of demonstrations

needed to reach the ITS knowledge maturity point. We suggest, for each specific domain, the computation of the knowledge maturity point for exercises with varying levels of complexity in order to estimate the range of demonstrations needed per complexity level. This estimation will help to avoid under and over training before system release.

3. Data used to train the intelligent tutor should represent the target audience's context. Within ill-defined domains, solutions are influenced by the context or students' backgrounds. Accordingly, the feedback provided by experts should be based on that context. We recommend gathering demonstrations of exercises' solutions from real and contextualized situations. Furthermore, experts used to categorize and assign feedback to the student's solutions should be familiar and well-informed of the target audience's context and background.

The main goal of these recommendations is to prevent the release of an intelligent tutor untrained in routine pedagogical situations that can be easily learned during a batch learning process. On the other hand, we want to avoid over training and delaying the release of an ITS that is capable of addressing routine situations and only inexperienced with outliers.

J. ITS Framework Evaluation Summary

We evaluated the framework's capability to support the six criteria envisioned for a modern ITS framework supporting learning within ill-defined domains. The ITS

framework evaluation was conducted in two different learning environments: (1) the ITS knowledge acquisition based on a batch learning approach; and (2) the ITS learning in an interactive learning environment. We conducted five studies to evaluate the proposed mixed-response ITS framework regarding its performance building the domain model and learning tutorial actions by using an LfD technique. Both of these approaches were tested using multiple datasets and domain experts. The study using the batch learning approach was conducted by running an off-line evaluation, while an interactive training environment was implemented for the second approach. Three experiments were conducted using the batch learning approach in order to evaluate: (1) the *effectiveness* and *metacognitive skill*, (2) *robustness*, and (3) *scalability* of the proposed ITS framework. Within the interactive learning evaluation we conducted two additional experiments evaluating the: (4) *effectiveness*, *efficiency* and *robustness*, and (5) *portability* of the framework.

We tested two implemented methods (WMMs and WCFGs) for knowledge representation and policy learning based on learning from instructor-student demonstrations. For knowledge representation we used three different levels of data granularity to evaluate the ITS framework performance. The used levels of granularity gave us insight about the impact of using finer granularity levels to better represent sequential data.

Results of the study support our four hypotheses. Outcomes of experiments one and four were used to evaluate the first and second hypotheses. First, both evaluation approaches (batch and interactive learning), demonstrated that "*the ITS's domain*

knowledge and tutorial actions can gradually be built at run time from instructor-student interactions". Both data representation and policy learning methods (WMMs and WCFGs) implemented were able to represent and use the acquired knowledge from the available demonstration datasets. The second hypothesis, "*the proposed ITS framework was able to estimate its knowledge confidence level by using a confidence-based approach to LfD*", was supported by the two conducted evaluation approaches as well. The intelligent tutor was able to decide whether to recommend feedback to the student or request for the instructor's response. This study demonstrated that the mixed-response ITS framework was extremely effective in producing tutorial actions for students' solutions. We identified a point of ITS knowledge maturity where the intelligent tutor becomes more knowledgeable regarding its intended learning goal; results providing correct feedback after the ITS reached this knowledge maturity point indicated up to 97% effectiveness.

The second and third experiment results supported our third hypothesis regarding whether "*the implemented knowledge representation and policy learning methods support efficiency, scalability, and robustness functional criteria envisioned for modern ITS frameworks*." First, from experiment 4 within the interactive learning environment we obtained superior ITS framework effectiveness using the WCFG-classifier approach and a granularity level based on tokens. Efficiency of the framework was primarily tested within the interactive learning evaluation; results indicate that the time required to train the ITS was around 1.4 hours per exercise. We provided a metacognitive skill for the ITS where the intelligent tutor was able to reject incorrectly classified tutorial actions

based on a computed confidence level. This functionality increased the ITS performance in providing feedback up to ten percentage points. Second, from experiment two and fourth (batch and interactive learning approaches respectively), the robustness of the ITS was tested applying the framework to different pedagogical perspectives and multiple datasets different in size and sequence. Using the batch learning approach in experiment 3 we tested the scalability criteria by using datasets of different lengths. Results testing robustness and scalability indicated consistency in the ITS's performance when applied to different learning situations.

Finally, the fifth experiment's results supported our fourth hypothesis regarding whether "*the implemented knowledge representation and policy learning methods support transferability to different content domains*." The portability capability was tested using the interactive learning environment by measuring the ITS framework performance using data from a different domain other than cybersecurity. Results regarding portability indicated consistency in the ITS's effectiveness providing feedback; using data from a second domain we obtained up to 98% effectiveness.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

In this chapter, we recapitulate the challenges of developing ITSs for ill-defined domains that motivated this research work. We then summarize the major results of our proposed solution. Finally, we discuss directions for future work regarding limitations of this work and proposed research avenues.

A. Conclusions

The success of intelligent tutors within the educational field is evident. Examples of these types of systems enhancing students' learning gains can be found in use within different domains. However, the time and resources required to design, implement, and release those ITSs as finished and stable systems are extremely high. As we discussed in this dissertation, developers of successfully implemented ITSs report that hundreds of hours are required for ITS designers and domain experts to gather the required domain and pedagogical knowledge. Furthermore, considerable time must be invested by ITS developers to convert the collected domain-knowledge into the intended knowledge representation structures (e.g., *production rules* and *constraints statements*).

The downside of ITS development regarding time and resources is highly influenced by the amount of knowledge the ITS should learn before starting to perform as an effective instructional tool. Furthermore, the effort to obtain adequate and comprehensive domain and pedagogical knowledge varies depending on the complexity

of the domain. Specifically, for ill-defined domains, the time and resources required to build ITSs tends to increase significantly. As we described in Chapter II, the reason for this greater effort is based on the large number of correct solutions and solution paths characterized by ill-defined problems. The characteristics underlying ill-defined domains lead to two main challenges: First, gathering comprehensive domain knowledge and a solid representation of potential students' misconceptions from domain experts is a hard task to be performed during the system's design time. Second, using the conventional knowledge representation structures, developed for ITSs used within well-defined domains, tends to be ineffective in representing knowledge for ill-defined problems.

In this research, we focused on the generation of the domain-knowledge for an ITS intended to support students' learning in the type of ill-defined domains described previously. In our study, we explored ways to address previously mentioned problems by using the Learning from Demonstration (LfD) technique. We leveraged basic LfD by a mixed-response approach in which the instructor has a permanent participation within the tutoring loop. In addition, we implemented and evaluated two knowledge representation methods, based on Weighted Markov Models and on Weighted Context Free Grammars respectively, which proved to be well suited for modeling languages and sequential data.

To minimize the time required to gather comprehensive domain knowledge during the ITS design time, the presented mixed-response ITS framework based on LfD aims to allow an intelligent tutor to continuously learn from instructor-student interactions at run time. By using an implemented *metacognitive* skill, the ITS is able to

estimate its current knowledge confidence level in order to defer to the human instructor in cases where the ITS does not have sufficient confidence in the applicability of any of its available tutorial actions. As a result of this, the time required to release an ITS can be shortened by leaving the task of identifying more sophisticated or unexpected situations to the intelligent tutor at run time. Then, by asking for instructor intervention, the domain knowledge will continuously grow from the instructor tutoring demonstrations.

We validated the proposed ITS framework based on six functionality characteristics we envisioned for a modern ITS supporting learning within ill-defined domains: *effectiveness, scalability*, *efficiency*, *portability, metacognition*, and *robustness*. We established that, in addition to always providing an *effective* tutoring performance, an ITS framework should offer: (1) *scalability* in data magnitude, in other words, an ITS should offer a means to expand its knowledge-base and preserve suitable performance; (2) *efficiency* reducing the time and human effort required to integrate new knowledge into the knowledge-base; (3) *portability* to be used across multiple domains with nil to minor alteration to the framework functionality; (4) *metacognitive* skill to infer its knowledge confidence level in order to trigger adequate tutorial responses or request for instructor help; and (5), sufficient *robustness* to deal with different pedagogical and tutoring criteria used by instructors to teach the intricate skills and knowledge required to solve ill-defined problems.

Furthermore, we introduced the implemented methods (WMMs and WCFGs) and algorithms used for knowledge representation. In our pursuit to choose the best ITS

feedback classification model, we conducted five different experiments using various granularity levels for knowledge representation. We found that by using our WCFG-based classifier in conjunction with a finer granularity level for knowledge representation, the implemented intelligent tutor was able to reach around 97% effectiveness in providing correct feedback when considering its knowledge confidence level. This percentage of effectiveness was accomplished after a training period when the ITS frequently asked for instructor demonstrations. Most of the tutorial actions were provided by instructors within the first 20 to 40 analyzed *CSs*. The ITS started using the learned tutorial actions after around 50 demonstrations. Then, after approximately 120 or 130 *CSs* the ITS started providing more feedback and the instructor´s intervention decreased.

Moreover, the performance of the ITS demonstrated its robustness and predictability when applied to different datasets and pedagogical perspectives. Furthermore, contrary to the time required to build production rules reported by previous ITS researchers, a relatively smaller amount of time was reported by instructors using the proposed framework. The time spent by the instructor ranged from 1.0 to 3.0 hours to construct the knowledge-base and required tutorial actions per exercise. Results from our experiments within the SQL-queries domain were similar to the 1.1 hours reported by Mitrovic (1998). However, even though we conducted experiments using data from the same domain (SQL-queries), the level of complexity of exercises and differences between the two implemented approaches prevented us from making a direct comparison regarding implementation time. As mentioned previously, the time spent reported by

Mitrovic (1998) building the SQL-tutor is based on the implementation of a constraint while our reported time is based on the construction of an entire exercise. However, each constraint within the SQL-tutor can be used in multiple exercises. While currently in our case, each new exercise must be constructed from scratch. Finally, the proposed ITS framework proved to perform well when it was applied to a second domain. This is significant because the ITS framework could be directly applied without modifications. However, more empirical research is suggested regarding the *portability* criteria.

B. Future Work

There have been few research proposals integrating human tutors into the ITS tutoring loop thus far. This is surprising given the challenges and cost of gathering comprehensive domain knowledge during ITS design time, principally for ill-defined domains. Therefore, the analysis of the impact of continuous participation of instructors beyond design time has been evaluated on a relatively small scale (Segedy et al., 2010; Johnson & Barnes, 2010). In order to evaluate performance of intelligent tutors when they are continuously trained by human instructors, more experiments are needed. Specifically, our subsequent experiments aim to further investigate: (1) the positive and negative impact of this tutoring approach in terms of the students' learning outcomes and experiences; and (2) issues and new functionalities regarding the instructor and intelligent tutor interaction such as mechanisms to detect and resolve inconsistencies in instructor demonstrations, support asynchronous instructor participation, and integrate multiple instructors' knowledge into a single knowledge-base. In addition, our

subsequent studies aim to compare the ITS performance when knowledge is gathered from experts exclusively at design time to the use of human instructors participating every time they are needed by the ITS.

Additional research venues resulted from this study. First, we found that the two methods (WMMs and WCFGs) we implemented for knowledge representation and tutorial action classification performed satisfactorily for students' solutions consisting of sequential data (primarily the WCFG approach). However, we believe that building domain knowledge from instructor-student interactions at run time, as proposed by our mixed-response ITS approach based on LfD, can be successfully utilized in domains in which students' solutions consist of different data types such as the open-ended discussions existing within several ill-defined domains. We believe that the proposed data representation methods (WMMs and WCFGs) or similar approaches can be used.

Second, the weight value computed by our classifiers used to initially categorize new *CSs* and the knowledge confidence level used to fine tune the ITS's final classification can be improved. As expected, within the batch and interactive learning experiments we conducted, we found that around 41% of *CSs* classified by the ITS were entirely new or only partially known by the intelligent tutor. By using a minimum recognition level as a threshold value (65% of representativeness) the intelligent tutor determined not to provide student feedback for those *CSs* that fell below the threshold value. However, around 20% of below threshold *CSs* were also initially correctly classified by the ITS, but because they didn't meet the minimum representation requirement students did not receive feedback. This implies the ITS could have used the

assigned feedback of an additional 20% of *CSs*; thereby, helping more students. Hence, we expect to improve the final classification of partially recognized *CSs* by improving or using different threshold strategies.

Finally, in order to verify the proposed ITS framework capability to be used across domains, validation studies will be conducted to evaluate the framework portability to other domains. Initially, our goal is to evaluate the framework *portability* using domains in which students' solutions can be represented as sequential data.

C. Final Remarks

We have discussed the strong evidence regarding the educational impact of Intelligent Tutoring Systems (ITS) as well as the corresponding implementation challenges. Many of the most respected researchers in the ITS field have reported effectiveness of ITSs in increasing students' learning outcomes by up to one or even two standard deviations. However, researchers also reported that, in order to produce effective intelligent tutors, months of intense interaction between ITS developers and domain-experts were needed just to build the domain knowledge. Typically, domain experts convey the target knowledge, ITS developers interpret and formalize the representation of that knowledge, and then experts are asked to verify the final knowledge representation in some way. Therefore, the time from start to finish for knowledge acquisition and representation is lengthy. This is the case particularly within domains in which students need to learn how to solve ill-defined problems.

ITS authoring systems have been helping developers overcome these design problems by allowing instructors to construct customized intelligent tutors. However, the traditional version of building domain knowledge at design time is often still taking place. Similar to evolving knowledge acquisition approaches found in other research fields, we replaced this restricted approach of the ITS knowledge learning at design time with an incremental approach that kept training the ITS during run time. In order to address this incremental knowledge learning approach, our main point is that the ITS should have the capability to identify when new knowledge is needed, inform instructors when this occurs, and learn from instructors' tutoring demonstrations.

The main contribution of this research is the implemented methods, algorithms, and techniques used to materialize this incremental knowledge learning approach. Our empirical results have shown us that the ITS domain model can be produced rapidly by including instructors with minimal intervention from the ITS framework developers. Furthermore, research outcomes indicate effectiveness of the ITS in providing feedback and determining when instructor intervention is necessary. We believe that building ITSs with metacognitive capabilities is the key to reducing necessary construction time for knowledge acquisition and representation. For the ITS to be able to autonomously determine when more knowledge is needed to address new situations is a novel approach to the ITS field that will provide a more significant, customizable and dependable teaching and learning environment for both instructors and students using Intelligent Tutoring Systems.

REFERENCES

Abbes, T., Bouhoula, A., & Rusinowitch, M. (2008). An inference system for detecting firewall filtering rules anomalies. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC '08)*. ACM , New York, NY, 2122-2128. DOI=10.1145/1363686.1364197

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE '95)*, Philip S. Yu and Arbee L. P. Chen (Eds.). IEEE Computer Society, Washington, DC, 3-14.

Aleven, V. (2003). Using background knowledge in case-based legal reasoning: A computational model and an intelligent learning environment. *Artificial Intelligence,* 150(1-2), 183–237.

Aleven, V., Ashley, K., Lynch, C., & Pinkwart, N. (2008). Intelligent tutoring systems for ill-defined domains: Assessment and feedback in ill-defined domains. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems, Ill-defined Domains Workshop (ITS '08)*, Springer-Verlag, Berlin, Heidelbrg, 1-3.

Aleven, V., McLaren, B. M., & Sewall, J. (2009). Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE Transactions on Learning Technologies,* 2(2), 64-78. DOI=10.1109/TLT.2009.22

Allen, J. F. (1999). Mixed-initiative interaction. *IEEE Intelligent Systems,* 14(5), 14-23. DOI=10.1109/5254.796083

Amershi, S., & Conati, C. (2009). Combining unsupervised and supervised machine learning to build user models for exploratory learning environments. *The Journal of Educational Data Mining,* 1(1), 18–71.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1996). ACT: A simple theory of complex cognition. *American Psychologist,* 51, 355-365.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science,* 13, 467-506.

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

Argall, B., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems,* 57(5), 469-483.

Arroyo, I., Murray, T., Woolf, B., & Beal C. (2003). Further results on gender and cognitive differences in help effectiveness. In *Proceedings of the 11th International Conference on Artificial Intelligence in Education*, ISO Press, Sydney, Amsterdam, 368-370.

Bernardini, A., & Conati, C. (2010). Discovering and recognizing student interaction patterns in exploratory learning environments. In *Proceedings of Intelligent Tutoring Systems: Lecture Notes in Computer Science,* Springer Press, Pittsburgh, PA 6094, 125-134. DOI: 10.1007/978-3-642-13388-6_17

Blessing, S. B. (1997). A programming by demonstration authoring tool for model-tracing tutors. *International Journal of Artificial Intelligence in Education,* 8(3), 233-261.

Bloom, B. (1984). The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring. *Educational Researcher,* 13(6), 4-16.

Cadez, I. V., Gaffney, S., & Smyth P. (2000). A general probabilistic framework for clustering individuals and objects. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*, ACM, New York, NY, 140-149. DOI:10.1145/347090.347119

Cen, H., Koedinger, K. R., & Junker, B. (2007). Is over practice necessary? --Improving learning efficiency with the cognitive tutor through educational data mining. In R. Luckin, K. R. Koedinger, & J. Greer (Eds.), In *Proceedings of the 2007 conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work*, IOS Press, Amsterdam, The Netherlands, The Netherlands, 511-518.

Chernova, S., & Veloso, M. (2007). Confidence-based learning from demonstration using gaussian mixture models. In *Proceedings of the 6th International joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '07*), ACM, New York, NY, 233(1), 1-8. DOI=10.1145/1329125.1329407

Chernova, S., & Veloso, M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research,* 34(1), 1-25.

Chi, Z. (1999). Statistical properties of probabilistic context-free grammars. *Computational Linguistics,* 25(1), 131–60.

Cifuentes, L., Bettati, R., Marti, W., Alvarez, O., & Mercer, R. (2009). Systematic design of a case-based learning environment. In G. Siemens & C. Fulford (Eds.), In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, AACE Press, Chesapeake, VA, 4176-4183.

Ciressan, C., Sanchez, E., Rajman, M., & Chappelier, J. C. (2001). An FPGA-based syntactic parser for real-life almost unrestricted context-free grammars. In G. J. Brebner & R. Woods (Eds.), In *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, Springer-Verlag, London, UK, 590-594.

Dragon, T., Floryan, M., Woolf, B., & Murray, T. (2010). Recognizing dialogue content in student collaborative conversation. In V. Aleven, J. Kay, & J. Mostow (Eds.), *Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelbrg, 6095(2), 113-122. DOI:10.1007/978-3-642-13437-1_12

Dragon, T., Woolf, B. P., Marshall, D., & Murray, T. (2006). Coaching within a domain independent inquiry environment. In *Proceedings of the Fifth International Conference on Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelbrg, 4053, 144-153.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification (2nd Edition)*. New York, NY: John Wiley & Sons Press.

Earley, J. (1970). An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery,* 13(2),94-102. DOI=10.1145/362007.362035

Fischer, G. (1997). Domain-oriented design environments: Knowledge-based systems for the real world. *Special Issue on Successes and Pitfalls of Knowledge-based Systems in Real-world Applications, International Journal Failure & Lessons Learned in Information Technology Management,* 1(2), 123-133. DOI: 10.1007/BF00872289

Fischer, G., McCall, R., Ostwald, J., Reeves, B., & Shipman F. (1994). Seeding, evolutionary growth and reseeding: Supporting the incremental development of design environments. In *Proceedings of Human Factors in Computing Systems (CHI '94)*, ACM, New York, NY, 292-298. DOI: 10.1145/191666.191770

Fournier-Viger, P., Nkambou, R., & Mayers, A. (2008a). Evaluating spatial representations and skills in a simulator-based tutoring system. *IEEE Transactions on Learning Technologies,* 1(1), 63–74. DOI: 10.1109/TLT.2008.13

Fournier-Viger, P., Nkambou, R., & Mephu-Nguifo, E. (2008b). A sequential pattern mining algorithm for extracting partial problem spaces from logged user interactions. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems, Ill-defined Domains Workshop (ITS '08)*, Springer-Verlag, Berlin, Heidelbrg, 46-55.

Fournier-Viger, P., Nkambou, R., & Mephu-Nguifo, E. (2009). Exploiting partial problem spaces learned from users' interactions to provide key tutoring services in procedural and ill-defined domains. *Artificial Intelligence in Education*, 200, 383-390.

Fournier-Viger, P., Nkambou, R., & Mephu-Nguifo, E. (2010). Building intelligent tutoring systems for ill-defined domains. In R. Nkambou, R. Mizoguchi, & J. Bourdeau, (Eds.), *Studies in Computational Intelligence*, Springer-Verlag, Berlin, Heidelberg, 308, 81-101. DOI:10.1007/978-3-642-14363-2_5

Horvitz, E. (1999). Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: The CHI is the limit (CHI '99)*, ACM, New York, NY, 159-166. DOI=10.1145/302979.303030.

Huang, X., Yong, J., Li, J., & Gao, J. (2008). Prediction of student actions using weighted Markov models. In *Proceedings of the IEEE International Symposium on IT in Medicine and Education*, 12-14.

Jarvis, M. P., Nuzzo-Jones, G., & Heffernan, N. T. (2004). Applying machine learning techniques to rule generation in intelligent tutoring systems. In J. C. Lester (Eds.), In *Proceedings of the International Conference on Intelligent Tutoring Systems*, Springer-Verlag, Berlin, Heidelbrg, 541-553.

Johnson, M. M., & Barnes, T. (2010). EDM Visualization Tool: Watching Students Learn. In R. S. Joazeiro de Baker, A. Merceron, P. I. Pavlik (Eds.), In *Proceedings of the 3rd International Conference on Educational Data Mining,* JEDM, Pittsburgh, PA, 297-298.

Jonassen, D. (1997). Instructional design models for well-structured and ill-structured problem-solving learning outcomes. *Educational Technology Research & Development,* 45, 65–94.

Jones, L. (2007). *The Student-Centered Classroom*. Cambridge, MA: Cambridge Press.

Kabanza, F., Nkambou, R., & Belghith, K. (2005). Path-planning for autonomous training on robot manipulators in space. In *Proceedings of the 19th International Joint Conference in Artificial Intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1729–1731.

Koedinger, K. R., Aleven, V., Heffernan, N. T., McLaren, B. M., & Hockenberry, M. (2004). Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Proceedings of the 7thIntelligent Tutoring Systems Conference (ITS '2004)*, Springer-Verlag, Berlin, Heidelbrg, 162-174.

Koncilia, C., & Pozewaunig, H. (2002). Identifying data sources for data warehouses. In H. Yin, N. M. Allinson, R. Freeman, J. A. Keane & S. J. Hubbard (Eds.), In *Proceedings of the 3rd International Conference on Intelligent Data Engineering and Automated Learning (IDEAL '02),* Springer-Verlag, London, UK, 213-218.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science,* 1(2), 111-117.

Lange, M., & Leiß, H. (2009). To CNF or not to CNF? An efficient yet presentable version of the CYK algorithm. Retrieved from http://www.informatica-didactica.de/cmsmadesimple/index.php?page=LangeLeiss2009_en, april 2011.

Lu, X. (2006). Expert tutoring and natural language feedback in intelligent tutoring systems. In R. Mizoguchi, P. Dillenbourg & Z. Zhu (Eds.), In *Proceedings of the 2006 Conference on Learning by Effective Utilization of Technologies: Facilitating Intercultural Understanding*, IOS Press, Amsterdam, The Netherlands, 657-658.

Lynch, C. F., Ashley, K. D., Aleven, V., & Pinkwart, N. (2006). Defining "ill-defined domains"; A literature survey. In *Proceedings of the 2nd International Workshop on Intelligent Tutoring Systems in Ill-defined Domain (ITS '06),* Springer-Verlag, Berlin, Heidelbrg, 1-10.

Matsuda, N., Cohen, W., & Koedinger, K. (2005). Building cognitive tutors with programming by demonstration. In S. Kramer & B. Pfahringer (Eds.), Technical report: TUM-I0510. In *Proceedings of the International Conference on Inductive Logic Programming*, Institut fur Informatik, Technische Universitat Munchen, 41-46.

Matsuda, N., Cohen, W., Sewall, J., Lacerda, G., & Koedinger, K. (2008). Why tutored problem solving may be better than example study: Theoretical implications from a simulated-student study. In B. Woolf, E. Aimeur, R. Nkambou & S. Lajoie (Eds.), In *Proceedings of the International Conference on Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelberg, 111-121.

McLaren, B., Koedinger, K. R., Schneider, M., Harrer, A., & Bollen, L. (2004). Bootstrapping novice data: Semi-automated tutor authoring using student log files. In *Proceedings of the Workshop on Analyzing Student-Tutor Logs (ITS '04),* Springer-Verlag, Berlin, Heidelbrg, 3220, 1-10.

Mernik, M., Gerlič, G., Žumer, V., & Bryant., B. R. (2003). Can a parser be generated from examples? In *Proceedings of the 2003 ACM Symposium on Applied computing (SAC '03)*, ACM, New York, NY, 1063-1067. DOI:10.1145/952532.952740

Mitrovic, A. (1998). Experiences in Implementing Constraint-based Modeling in SQL-Tutor. In B. P. Goettl, H. M. Halff, C. L. Redfield & V. J. Shute (Eds.), In *Proceedings of the 4th International Conference on Intelligent Tutoring Systems* (ITS '98), Springer-Verlag, London, UK, 414-423.

Mitrovic, A., Martin, B., & Mayo, M. (2002). Using evaluation to shape ITS design: Results and Experiences with SQL-Tutor. *International Journal User Modeling and User-Adapted Interaction*, 12(2-3), 243-279.

Mitrovic, A., Martin, B., & Suraweera, P. (2007). Intelligent tutors for all: Constraint-based modeling methodology, systems and authoring. *IEEE Intelligent Systems,* 22(4), 38-45.

Mostafavi, B., & Barnes, T. (2010). Towards the creation of a data-driven programming tutor. In *Proceedings of the 10th International Conference on Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelberg, 6095(2), 239-241. DOI: 10.1007/978-3-642-13437-1_31

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research,* 7(1), 67-82.

Nkambou, R., Bourdeau, J., & Mizoguchi, R. (2010). Introduction what are intelligent tutoring systems, and why this book? In R. Nkambou, J. Bourdeau & R. Mizoguchi (Eds.), *Advances in Intelligent Tutoring Systems, (1st ed.)*. Berlin, Heidelberg: Springer Publishing Company, pp. 1-14.

Nkambou, R., Mephu-Nguifo, E., Couturier, O., & Fournier-Viger, P. (2007). Problem-solving knowledge mining from users' actions in an intelligent tutoring system. In Z. Kobti & D. Wu (Eds.), In *Proceedings of the 20th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence (CAI '07)*, Springer-Verlag, Berlin, Heidelberg, 393-404. DOI=10.1007/978-3-540-72665-4_34

Nkambou, R., Mephu-Nguifo, E., & Fournier-Viger, P. (2008). Using knowledge discovery techniques to support tutoring in an ill-defined domain. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems for Ill-defined Domains Workshop (ITS '08),* Springer-Verlag, Berlin, Heidelberg, 395-405. DOI=10.1007/978-3-540-69132-7_43

Noronha, R. V., & Fernandes, C. T. (2005). Model to represent ill-structured problems in ITS environments. In *Proceedings of the 8th IFIP World Conference on Computers in Education (WCCE '05),* University of Stellenbosch, Cape Town, South Africa 436-446.

Noronha, R. V., Galante, D., & Fernandes, C. T. (2005). Preliminary ideas to provide intelligent tutoring systems with abilities to deal with ill-structured problems. In G. Chiazzese, M. Allegra, A. Chifari, & S. Ottaviano (Eds.), In *Proceedings of the International Conference Methods and Technology for Learning,* WIT Press, Great Britain, UK, 83-88.

Nwana, H. (1990). Intelligent tutoring systems: An overview. *Artificial Intelligence Review*, 4(4), 251-277.

Ohlsson, S. (1994). Constraint-based student modeling. In *Student Modeling: The Key to Individualized Knowledge-based Instruction*, Springer-Verlag, Berlin, Heidelberg, 167-189.

Paquette, L., Lebeau J.-F., & Mayers, A. (2010). Authoring Problem-solving Tutors: A comparison between ASTUS and CTAT. In R. Nkambou, J. Bourdeau & R. Mizoguchi (Eds.), *Advances in Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelberg, pp. 377-405.

Psotka, J., Massey, L. D., & Mutter, S. A. (1998). *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Rabiner, L. R., & Juang, B. H (1986). An introduction to hidden Markov models. *IEEE Acoustics, Speech & Signal Processing Magazine,* 3, 4–16.

Ruiz-Martínez, J. M., Valencia-García, R., Fernández-Breis, J. T., García-Sánchez, F., & Martínez-Béjar, R. (2011). Ontology learning from biomedical natural language documents using UMLS. *Expert Systems Applications,* 38(10), 12365-12378.

Segedy, J., Sulcer, B., & Biswas, G. (2010). Are ILEs ready for the classroom? Bringing teachers into the feedback loop. In *Proceedings of the 10th International Conference on Intelligent Tutoring Systems,* Springer-Verlag, Berlin, Heidelberg, 6095(2), 405-407.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin-Madison. Retrieved from http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf, may 2011.

Simon, A., Egidi, M., Viale, R., & Marris, R. L. (1992). *Economics, Bounded Rationality and the Cognitive Revolution*. Brookfield, VT: Edward Elgar.

Sipser, M. (1997), *Introduction to the Theory of Computation* (1st ed.). Boston, Massachusetts: International Thomson Publishing, pp. 99.

Sleeman, D., & Brown, J. S. (1982). Introduction: Intelligent tutoring systems. In D. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems,* Orlando, Florida: Academic Press, Inc.1-11.

Smith, N. A., & Johnson, M. (2007). Weighted and probabilistic context-free grammars are equally expressive. *Computational Linguistics*, 33(4), 477-491.

Stevens, R., & Soller, A. (2005). Machine learning models of problem space navigation: The influence of gender. *Computer Science and Information Systems,* 2(2), 83-98.

Tenenberg, J. D. (2001). On the meaning of computer programs. In M. Beynon, C. L. Nehaniv & K. Dautenhahn (Eds.), In *Proceedings of the 4th International Conference on Cognitive Technology: Instruments of Mind (CT '01)*, Springer-Verlag, London, UK, 165-174.

Tsoulos, I. G., & Lagaris, I. E. (2005). Grammar inference with grammatical evolution. Retrieved from http://www.cs.uoi.gr/~lagaris/papers/PREPRINTS/meta_grammars .pdf, june 2011.

Urban-Lurain, M. (1996). Intelligent tutoring systems: An historic review in the context of the development of artificial intelligence and educational psychology. Retrieved from http://www.cse.msu.edu/rgroups/cse101/ITS/its.htm, june 2011.

VanLehn, K. (2006). The behavior of tutoring systems. *International Journal of Artificial Intellience,* 16(3), 227-265.

Viccari, R. M., Jaques, P. A., & Verdin, R. (2008). *Agent-based Tutoring Systems by Cognitive and Affective Modeling*. Hershey, PA: IGI Global.

Walker, E., Ogan, A., Aleven, V., & Jones, C. (2008). Two approaches for providing adaptive support for discussion in an ill-defined domain. In *Proceedings of Intelligent Tutoring Systems for Ill-defined Domains Workshop (ITS '08)*, Springer-Verlag, Berlin, Heidelbrg, 4-12.

Woolf, B. (2009). *Building Intelligent Interactive Tutors: Student-Centered Strategies for Revolutionizing e-Learning*. Elsevier, Amsterdam: Morgan Kaufmann Publishers.

APPENDIX A

WEB ACCESS EXERCISE SYSTEM

The Web Access Exercise System (WAES) is a web-based environment that provides instruction in *cybersecurity* (Cifuentes et al., 2009). WAES is instructionally founded on Case-based cybersecurity problem solving. A WAES's unit consists of multiple modules which group a set of exercises (see Figure A.1 (a)).

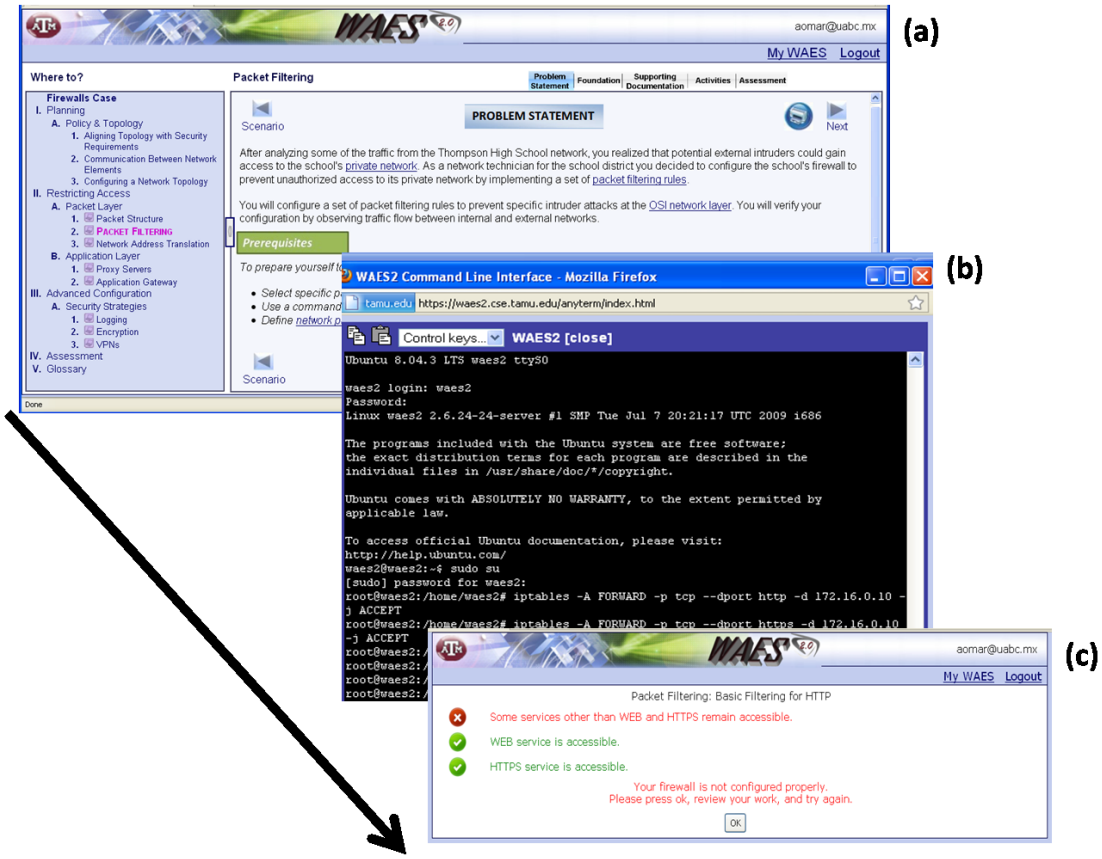
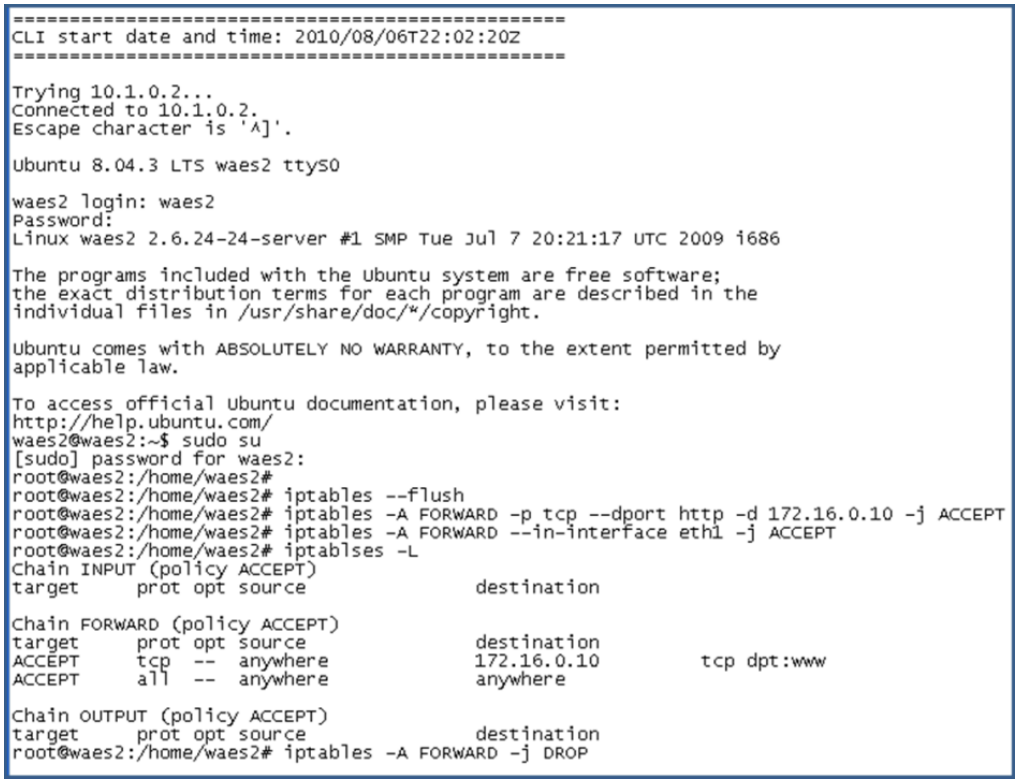


Figure A.1. Web Access Exercise System. (a) Firewalls unit and exercise content. (b) Command Line Interface. (c) Configuration outcome and feedback.

The students' goal is to solve cases within each exercise by planning configuration strategies and configuring a set of remote, but real, security devices. Hence, the WAES replaces the physical contacts between students and security devices with a web-based configuration environment. Remote device configuration is executed by entering a set of *configuration rules* through a web-based Command Line Interface (CLI) (see Figure A.1 (b)). Upon successful solution of an exercise, the WAES system indicates that the student configured a successful solution. If the security systems selected by the student and the configuration of those systems were not acceptable, then the WAES system provides a predefined feedback according to misconceptions (see Figure A.1 (c)).



```
================================================
CLI start date and time: 2010/08/06T22:02:20Z
================================================

Trying 10.1.0.2...
Connected to 10.1.0.2.
Escape character is '^]'.

Ubuntu 8.04.3 LTS waes2 ttyS0

waes2 login: waes2
Password:
Linux waes2 2.6.24-24-server #1 SMP Tue Jul 7 20:21:17 UTC 2009 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
waes2@waes2:~$ sudo su
[sudo] password for waes2:
root@waes2:/home/waes2#
root@waes2:/home/waes2# iptables --flush
root@waes2:/home/waes2# iptables -A FORWARD -p tcp --dport http -d 172.16.0.10 -j ACCEPT
root@waes2:/home/waes2# iptables -A FORWARD --in-interface eth1 -j ACCEPT
root@waes2:/home/waes2# iptablses -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             172.16.0.10         tcp dpt:www
ACCEPT     all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@waes2:/home/waes2# iptables -A FORWARD -j DROP
```

Figure A.2. Student's configuration log-file.

The WAES system tracks and saves all the commands typed by the students within the CLI while configuring a security device. Log-files are generated from students' activity and used to evaluate students' performance by instructors. In Figure A.2 we show an example of the partial content of a log-file generated by WAES.

APPENDIX B

OPERATIONAL EXAMPLES OF THE MIXED-RESPONSE ITS FRAMEWORK

B.1 Architecture of the Mixed-response ITS Framework

The architecture of the proposed mixed-initiative ITS framework is illustrated in Figure B.1. Students activity is monitored and logged by the ITS while they configure security devices through the command line interface. The proposed ITS framework is primarily reactive; it responds with tutorial actions in response to explicit student help requests.

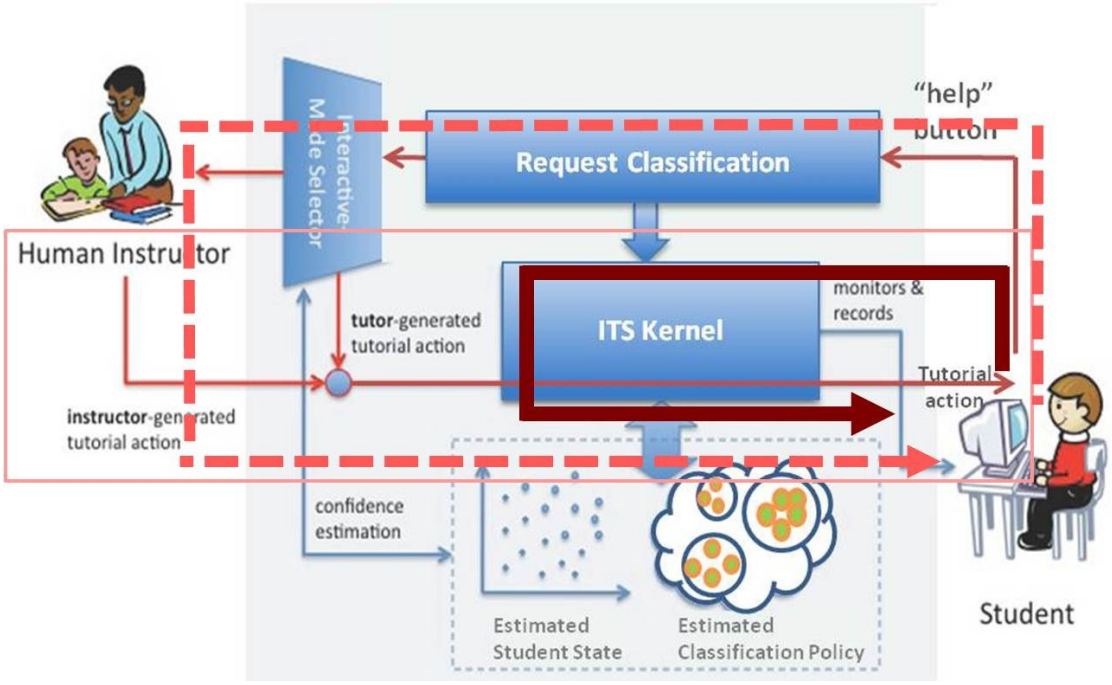


Figure B.1. Architecture of the mixed-response ITS framework.

When a student indicates he needs help in a particular situation, the intelligent tutor performs four main actions: (1) gathers the student's problem solution from the student's activity log-file, (2) classifies the student's problem solution based on

previously learned domain knowledge and select the corresponding tutorial action, (3) estimates whether its classification confidence level is above a predetermined threshold, (4) determines whether to use the selected tutorial action or asks for instructor intervention (and learn from the instructor intervention). The *ITS Kernel* receives and classifies student's problem solutions. The *Interactive Mode Selector* implements the mixed-response functionality of the proposed ITS framework by determining the mode of interaction that will be used to respond to the student's requests (see Figure B.1).

B.2 Mixed-response ITS Framework Users Scenarios

There are two main functionalities performed by the proposed ITS framework: (1) interacting with students or (2) interacting with instructors. During ITS-student interaction, the intelligent tutor receives students' help requests and responds with a tutorial action whenever the intelligent tutor's knowledge confidence level is considered high enough. ITS-instructor interaction is activated after the intelligent tutor receives a student request for which the ITS' knowledge confidence level is insufficient to provide feedback to the student. The following two scenarios exemplify the possible intelligent tutor interactivity with students and instructors.

1. Scenario 1: Intelligent Tutor-student

Susan is a college student learning how to configure a firewall security device by solving an exercise using WAES (see Figure B.2 (a)). While completing the exercise, she begins to doubt the correctness of her current configuration work. All her work is

logged by the system to which the intelligent tutor has access (see Figure B.2 (b)). Knowing she has access to an intelligent tutor, Susan decides to ask for help before continuing the configuration of the security device. She presses the help button available on the CLI interface and the intelligent tutor responds by providing customized feedback (see Figure B.2 (c)).

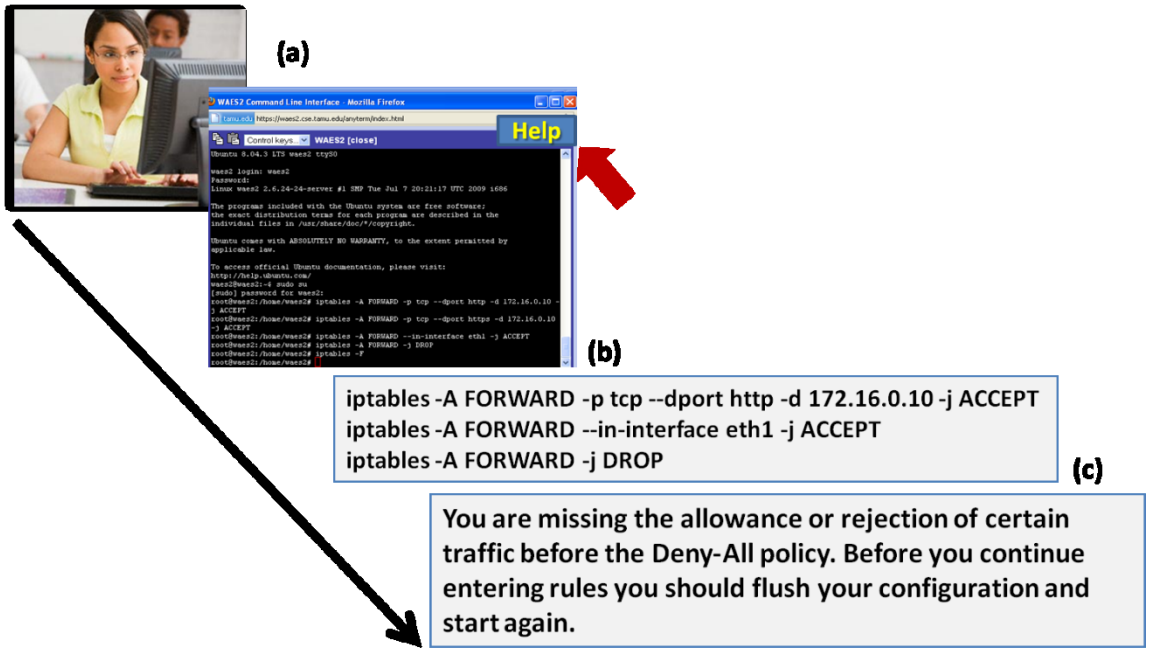


Figure B.2. Intelligent tutor-student scenario: (a) The student presses the help button while configuring a security device in WAES. (b) The intelligent tutor gathers and classifies the student's configuration sequence. (c) The intelligent tutor provides customized feedback to the student.

How does the intelligent tutor select the provided feedback? When Susan pressed the help button, she activated the ITS tutoring process in the following way: The ITS: (1) received a help request from a student and the corresponding configuration sequence (only those configuration rules influencing the device configuration are considered) was

retrieved from the student's log-file (see Figure B.3), (2) selected the corresponding tutorial action by classifying the student's configuration sequence within the most similar cluster of learned configuration sequences (see Figure B.4 (a)), (3) estimated the classification confidence level of the selected tutorial action (see Figure B.5), and (4) considering that the confidence level was high, the intelligent tutor provided the selected tutorial action to the student (see Figure B.2 (c)).

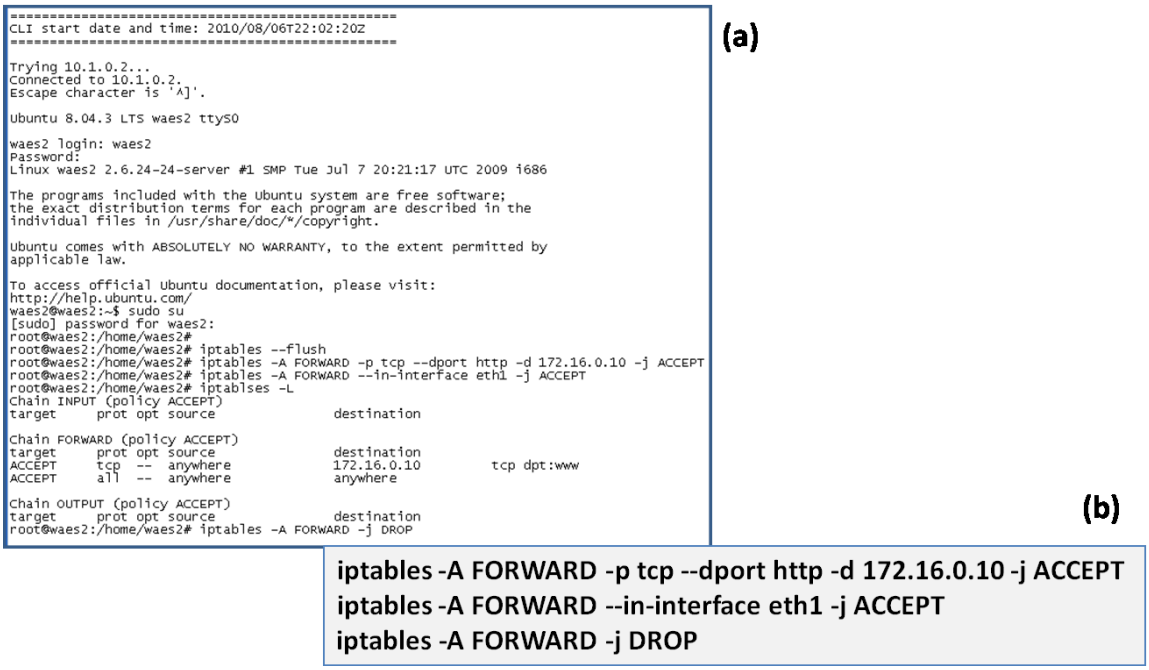


Figure B.3. Configuration sequence retrieved from a student's log-file: (a) Student's log-file. (b) Retrieved configuration sequence.

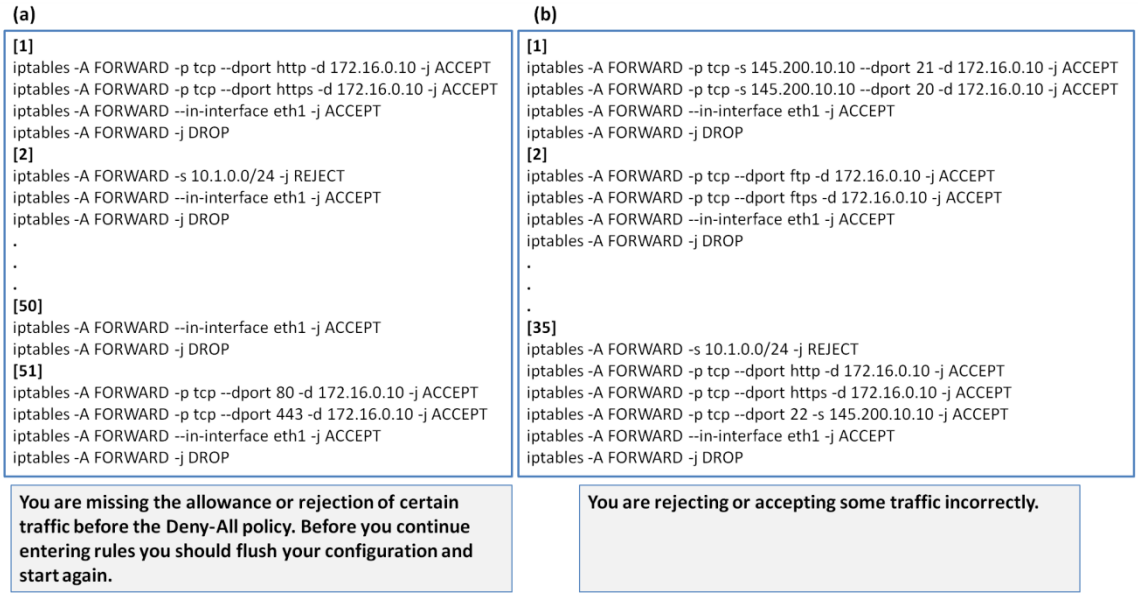


Figure B.4. Example of two clusters including configuration sequences and tutorial actions learned by the intelligent tutor.

In Figure B.4 we illustrate how the ITS classified previous configuration sequences in two different clusters. Within cluster (a), all the entered configuration rules are correct. However, the student is forgetting to reject some traffic and the required configuration rules for this rejection should come before the Deny-All rule (iptables -A FORWARD -j DROP). On the other hand, configuration sequences in cluster (b) were grouped together because they include configuration rules incorrectly rejecting or accepting traffic. Even though all the configuration rules within the student's configuration sequence in our example (Figure B.3 (b)) are present in some configuration sequences within clusters (a) and (b) (Figure B.4), the implemented classification algorithm was able to correctly classify the student's configuration sequence within cluster (a) with a high confidence level.

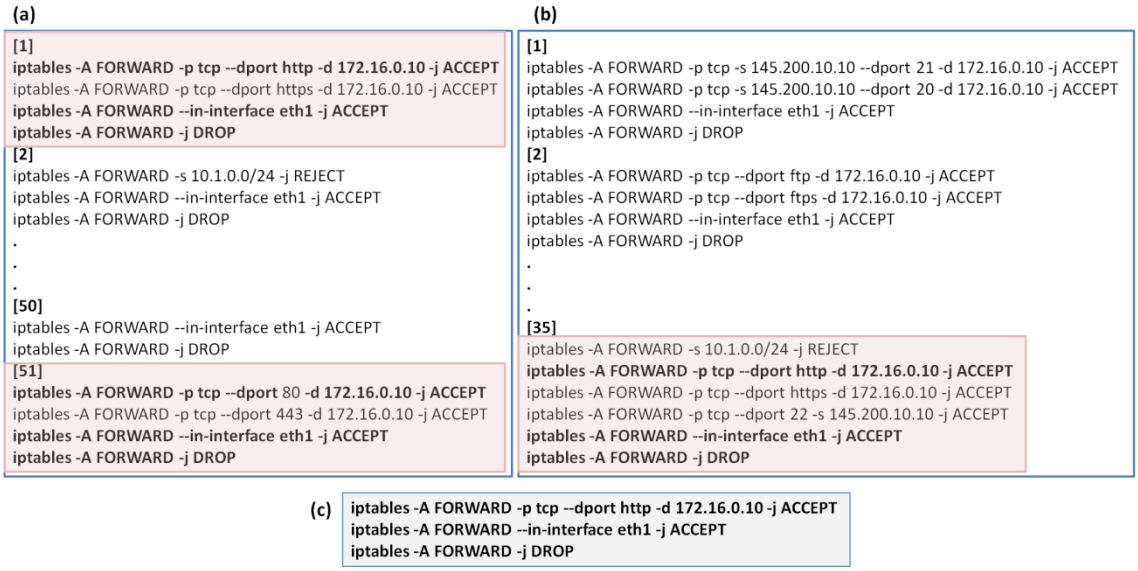


Figure B.5: Configuration sequence classification. (a) Cluster grouping of partially correct configuration sequences. (b) Cluster grouping of configuration sequences including incorrect configuration rules. (c) Student's configuration sequence.

The classification process is performed by identifying those clusters which include configuration sequences that best represent the student's configuration sequence (target configuration sequence). In Figure B.5 we illustrate how clusters (a) and (b) contains a configuration sequence ([1] and [35] respectively) containing the entire set of configuration rules within the target configuration sequence. Given that the clusters' configuration sequences include more configuration rules than the target one, the representation is considered partial. The classifier selects cluster (a) with the higher representation because the number of configuration rules within the analyzed configuration sequence ([1]) is lower than the one analyzed within cluster (b) ([35]).

The confidence level is computed based on weights assigned to token structures that are identified within a cluster. Structures consist of sequences of tokens that are repeated within each cluster. For instance, as shown in Figure B.5, within cluster (a) the

sequence of tokens that were considered to estimate the confidence level for the classification of the target configuration sequence are in bold within configuration sequence [1] and [51]. In this example the computed confidence level was not 100 percent but above a predefined threshold, indicating to the intelligent tutor that the corresponding tutorial action should be used.

2. Scenario 2: Intelligent Tutor-instructor

In this example we use a student's configuration sequence classified with a low confidence level. Mr. García, an instructor using WAES in networking security course has received a request to lead the tutoring process from the ITS as a result of a classification with low confidence level. The instructor accesses a tutoring module (see Figure B.6 (b)) that he receives from the ITS including: (1) the student's configuration sequence, (2) the selected tutorial action, (3) the classification confidence level, and (4) the list of available tutorial actions. Next, Mr. García analyzes the student's configuration sequence (see Figure B.6 (c)), the intelligent tutor's selected feedback, and determines whether to use the proposed feedback or recommend a new tutorial action (see Figure B.6 (d)). Finally, the instructor submits the feedback to be delivered to the student (see Figure B.6 (e)).

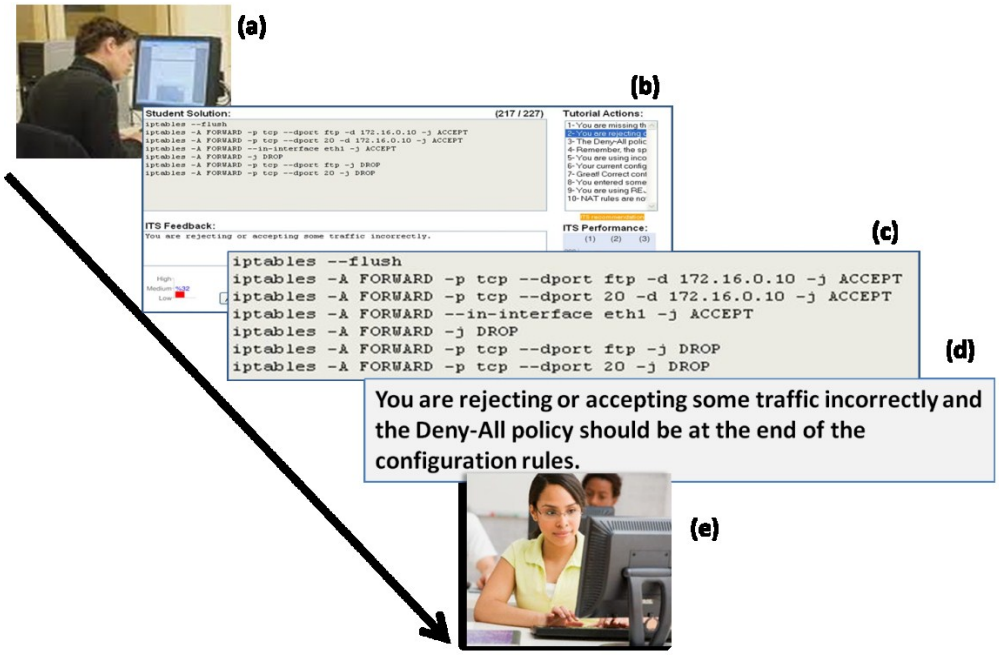


Figure B.6. Intelligent tutor-instructor scenario. (a) The instructor receives an intervention request. (b) The instructor accesses the tutoring interface. (c) The instructor analyzes the student's solution and the intelligent tutor's proposed feedback. (d) The instructor determines and submits the appropriate feedback. (e) The student receives the feedback through the system's interface.

To lead the tutoring process, the instructor has access to an interface similar to the interactive learning environment that we used to train the ITS. Throughout this interface the instructor receives from the ITS: (1) the student's configuration sequence, (2) the selected tutorial action, (3) the classification confidence level, and (4) the list of available tutorial actions (see Figure B.7). In addition, the instructor can: (a) accept the tutorial action selected by the ITS, (b) select one of the other available tutorial actions, or (c) provide a new tutorial action when needed (see Figure B.7 (5)).

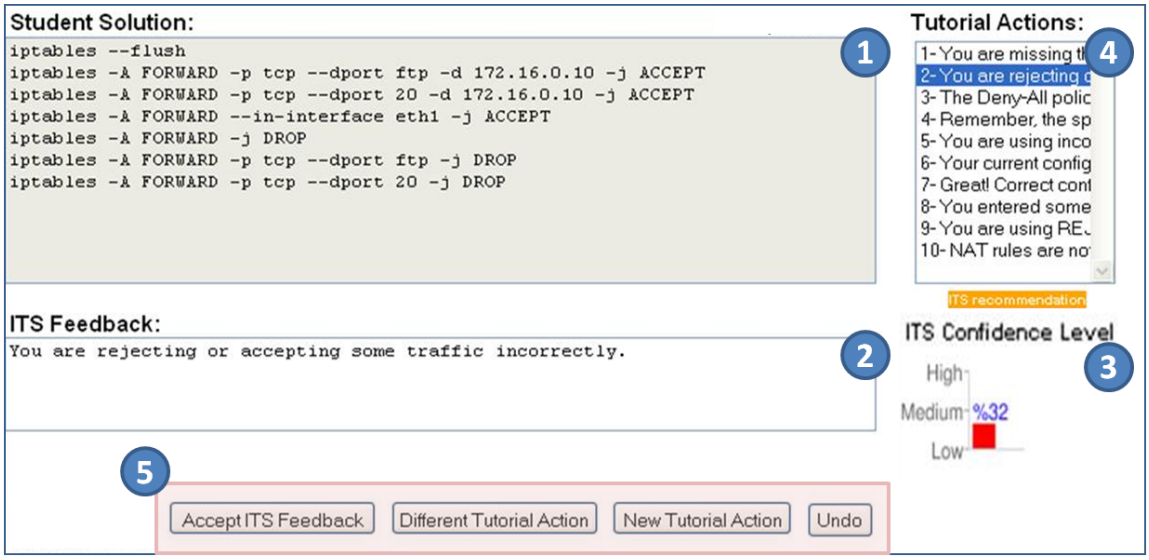


Figure B.7. Instructor interface designed for carrying out the tutoring process.

In this example the student's configuration sequence includes several misconceptions: the student's configuration is accepting traffic that should be rejected, the Deny-All policy is not at the end of the configuration sequence, and some configuration rules are missing (see Figure B.6 (c)). The intelligent tutor classified the student's configuration sequence correctly assigning a tutorial action which is addressing one of the main configuration problems (the Deny-All policy is not at the end of the configuration sequence). However, because of the student's multiple misconceptions and the sequence in which he/she entered the configuration rules (not similar to previously learned configuration sequences) the resulting confidence level computed by the ITS was low (32%).

VITA

Omar Álvarez Xochihua received his B.S. degree in computer science from the Autonomous University of Baja California, México in 1991 and his M.A.I. degree from the Institute of Technology of Monterrey (Tecnológico de Monterrey), México in 1994. He entered the Computer Science doctoral program at Texas A&M University in August 2006 and received his Ph.D. degree in May 2012. Since 1995, he has worked as an university professor on the Computer Science Faculty at Autonomous University of Baja California, México. His research interests lay primarily in intelligent tutoring systems, especially in building models for knowledge representation within ill-defined domains. He is also interested in computer science instruction through using computer-based agents and distance learning technologies. He may be reached at:

Name: Omar Álvarez Xochihua

Address: Autonomous University of Baja California,
Faculty of Science,
Ensenada B.C., México.

Email Address: axomar@gmail.com