Un modèle pour la génération d'indices par une plateforme de tuteurs par traçage de modèle

par

Luc Paquette

thèse présentée au Département d'informatique
en vue de l'obtention du grade de docteur ès sciences (Ph. D.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, juillet 2013

Your file Votre référence
ISBN: 978-0-494-95101-9

Our file Notre référence
ISBN: 978-0-494-95101-9

Your file Votre référence
ISBN: 978-0-494-95101-9

Our file Notre référence
ISBN: 978-0-494-95101-9

Le 25 juillet 2013

le jury a accepté la thèse de monsieur Luc Paquette dans sa version finale.

Membres du jury

Professeur André Mayers
Directeur de recherche
Département d'informatique

Professeur Ryan Baker
Évaluateur externe
Teachers College Columbia University

Professeur Yves Bouchard
Évaluateur interne
Faculté des lettres et sciences humaines

Professeur Marc Frappier
Président-rapporteur
Département d'informatique

Sommaire

La présente thèse décrit des travaux de recherche effectués dans le domaine des systèmes tutoriels intelligents (STI). Plus particulièrement, elle s'intéresse aux tuteurs par traçage de modèle (MTT). Les MTTs ont montré leur efficacité pour le tutorat de la résolution de tâches bien définies. Par contre, les interventions pédagogiques qu'ils produisent doivent être incluses, par l'auteur du tuteur, dans le modèle de la tâche enseignée.

La recherche effectuée répond à cette limite en proposant des méthodes et algorithmes permettant la génération automatique d'interventions pédagogiques. Une méthode a été développée afin de permettre à la plateforme Astus de générer des indices par rapport à la prochaine étape en examinant le contenu du modèle de la tâche enseignée. De plus, un algorithme a été conçu afin de diagnostiquer les erreurs des apprenants en fonction des actions hors trace qu'ils commettent. Ce diagnostic permet à Astus d'offrir une rétroaction par rapport aux erreurs sans que l'auteur du tuteur ait à explicitement modéliser les erreurs.

Cinq expérimentations ont été effectuées lors de cours enseignés au département d'informatique de l'Université de Sherbrooke afin de valider de façon empirique les interventions générées par Astus. Le résultat de ces expérimentations montre que 1) il est possible de générer des indices par rapport à la prochaine étape qui sont aussi efficaces et aussi appréciés que ceux conçus par un enseignant et que 2) la plateforme Astus est en mesure de diagnostiquer un grand nombre d'actions hors trace des apprenants afin de fournir une rétroaction par rapport aux erreurs.

Mots-clés: Systèmes tutoriels intelligents, tuteurs par traçage de modèle, interventions pédagogiques, indices par rapport à la prochaine étape, rétroactions négatives.

Remerciements

J'aimerais tout d'abord remercier mon directeur de recherche, André Mayers, pour son énorme contribution à ma formation en tant que chercheur au cours des cinq dernières années.

Merci à tous les membres d'Astus, présents et passés, avec qui j'ai eu la chance de partager de nombreuses discussions. En particulier, merci à Jean-François Lebeau, avec qui j'ai eu la chance de collaborer tout au long de mes travaux de recherche, à Mikaël Fortin, qui a grandement contribué à m'introduire à Astus, à Gabriel Beaulieu et à Amir Abdessemed.

Merci à ma famille pour leur support pendant mes longues études universitaires. Merci à mes parents, Lise et Pierre, à mon frère, David, à ma sœur, Julie, à mon beau-frère, Vincent, et à ma belle sœur, Véronique.

Merci à mes amis avec qui j'ai partagé de nombreuses années à Sherbrooke pendant mes études : Alexei Nordell Markovits, Marc Therrien, Heidi Larkin, Carine Hamel et Sébastien Richard (même s'il n'est plus à Sherbrooke). J'aimerais aussi remercier le groupe d'étudiants du RECSUS avec qui j'ai partagé de nombreuses conversations, instructives et divertissantes, sur l'heure du diner.

Table des matières

Liste des abréviations

Astus	Apprentissage par système tutoriel de l'Université de Sherbrooke
CTAT	Cognitive tutor authoring tool
MTT	Tuteur par traçage de modèle (model-tracing tutor)
SQL	Structured query language
STI	Système tutoriel intelligent
TDK	Cognitive tutor development kit

Introduction

Contexte

Les systèmes tutoriels intelligents (STI) sont des logiciels d'aide à l'apprentissage et à l'enseignement. Ils se caractérisent par leur capacité à fournir une rétroaction pédagogique adaptée au contexte de l'activité d'apprentissage et aux caractéristiques de l'apprenant. Plusieurs STI sont utilisés de façon régulière et avec grand succès dans des écoles (principalement aux États-Unis). Parmi les STI les plus connus, on retrouve : les *Cognitives Tutors* [6], une famille de tuteurs pour l'enseignement des mathématiques au secondaire ; Andes [19], un tuteur d'introduction à la physique mécanique ; et SQL-tutor [12], un tuteur pour la création de requêtes en langage SQL.

La présente thèse s'intéresse particulièrement aux tuteurs par traçage de modèle (MTT). Ce type de tuteur se distingue des autres par le fait qu'il possède un modèle exécutable de la tâche enseignée. Ce modèle lui permet de générer une ou des séquences d'actions représentant une solution valide. Les MTTs utilisent ce modèle afin de suivre l'étudiant pas à pas lors de ses activités d'apprentissages. Les MTTs sont utilisés pour l'enseignement de la résolution de tâches bien définies [13], c'est-à-dire des tâches pour lesquels il existe toujours une solution correcte ainsi qu'un algorithme permettant l'atteinte de cette solution. Les *Cogntives Tutors* [6] et Andes [19] sont des exemples de MTTs très connus.

Puisque la création d'un MTT demande un grand nombre d'heures de travail [15], plusieurs chercheurs ont élaboré des plateformes informatiques et des outils auteurs. Parmi celles-ci, on retrouve la plateforme TDK [5] et l'outil CTAT [1, 2]. L'utilisation d'une plateforme informatique ou d'un outil auteur réduit significativement les coûts reliés au développement d'un tuteur puisque ces outils fournissent des solutions logicielles réutilisables pour plusieurs composantes d'un STI. Par contre, ces plateformes sont habituellement limitées dans leurs interventions pédagogiques auprès de l'apprenant :

1. Les interventions pédagogiques qu'elles fournissent demandent l'ajout de connaissances pédagogiques au modèle de la tâche enseignée. En effet, seules les rétroactions immédiates (indiquer si une action est correcte ou non) peuvent être fournies sans l'ajout de telles connaissances. En plus de la rétroaction immédiate, les MTTs offrent habituellement des indices par rapport à la prochaine étape et une rétroaction négative (« *negative feedback* » [17]) par rapport aux erreurs. Ces deux types d'interventions demandent à l'auteur d'inclure des messages d'aide dans le modèle de la tâche. De plus, la rétroaction négative demande de modéliser chacune des erreurs pouvant faire l'objet d'une rétroaction ; une tâche qui nécessite une analyse approfondie des différentes erreurs possibles.

2. Les interventions qu'elles utilisent sont peu adaptées aux besoins de l'apprenant. Puisque le contenu des interventions pédagogiques doit être inclus dans le modèle de la tâche, il n'est pas possible de l'adapter en fonction de la situation d'apprentissage dans laquelle se trouve l'apprenant. Pourtant, plusieurs chercheurs posent l'hypothèse que l'utilisation d'interventions adaptées aux besoins de l'apprenant est bénéfique. Stellan Ohlsson propose neuf mécanismes d'apprentissages qui peuvent être sollicités par les interventions pédagogiques précises afin d'améliorer l'apprentissage [17, 18]. Dans le même ordre d'idée, Min Chi a montré que les décisions du tuteur concernant le type d'intervention à utiliser lors d'une situation précise peuvent avoir un impact significatif sur l'apprentissage [8, 9].

Objectifs

L'hypothèse du groupe de recherche Astus est qu'un système de représentation des connaissances dédié à l'enseignement facilite la conception de STI offrant des interventions pédagogiques riches. Dans cet ordre d'idée, le groupe Astus a développé une plateforme informatique pour la création de MTTs dont l'objectif est de permettre aux tuteurs d'offrir des interventions pédagogiques riches tout en conservant des coûts de conception comparables aux plateformes similaires.

La présente thèse s'appuie sur les travaux du groupe Astus afin d'exploiter l'information disponible dans le modèle de la tâche enseignée dans le but de produire des interventions pédagogiques. Plus précisément, il s'agit de déterminer comment générer automatiquement des indices par rapport à la prochaine étape et des rétroactions négatives au sein de la plateforme Astus sans que l'auteur du tuteur ait à fournir de messages d'aide et à modéliser les différentes erreurs possibles. De plus, l'atteinte de cet objectif aura comme effet de réduire les efforts nécessaires pour fournir des rétroactions négatives puisqu'il ne sera plus nécessaire de faire une analyse approfondie des erreurs possibles.

La génération automatique du contenu pédagogique permettra d'offrir des interventions qui peuvent plus facilement être adaptées à des situations d'apprentissages précises. Les messages d'aide pourront facilement être présentés dans un format approprié à la situation, ce qui n'est pas le cas des messages inclus dans le modèle de la tâche puisque ceux-ci sont écrits avant de connaître la situation d'apprentissage. Dans le cas des rétroactions négatives, une telle adaptation permettra de fournir un message ciblant la cause précise de l'erreur plutôt qu'un message décrivant l'erreur.

D'un point de vue informatique, ces objectifs se traduisent par la conception d'un ensemble d'algorithmes permettant aux tuteurs de fournir des interventions pédagogiques sans que l'auteur du tuteur ait explicitement inclus de messages d'aides dans le modèle de la tâche. Ces algorithmes forment une méthode pour :

- Générer des séquences d'indices spécifiques à la tâche afin d'expliquer, de façon de plus en plus précise, la prochaine étape que l'apprenant doit réaliser.

- Automatiquement identifier les mauvaises conceptions de l'apprenant à partir de ses actions dites hors trace[1] (« *off-path step* ») et générer une rétroaction négative par rapport aux erreurs qui en découlent.

Méthodologie

La méthode proposée afin d'atteindre l'objectif de la thèse se divise en trois phases principales :

1. Évaluer la pertinence du système de représentation des connaissances d'Astus pour l'objectif de la thèse. Au cours de cette phase, les efforts et les techniques requis pour créer un tuteur à l'aide de la plateforme Astus ont été comparés à ceux nécessaires pour créer un tuteur identique à l'aide de la plateforme CTAT. Les interventions pédagogiques fournies par les deux plateformes ont aussi été comparées.

2. Réduire les coûts de modélisation des indices expliquant la prochaine étape. Cette phase s'appuie sur les résultats de la précédente afin de concevoir une méthode pour la génération d'indices par rapport à la prochaine étape de manière indépendante de la tâche enseignée.

3. À partir de l'expertise acquise à la phase 2, étendre les interventions pédagogiques reliées aux indices en ajoutant le diagnostic des actions dites hors trace. Ce diagnostic permet une intervention pédagogique précise par rapport aux incompréhensions de l'apprenant et aux erreurs qui en découlent.

Résultats

Le principal résultat des recherches effectuées lors de la présente thèse est une méthode permettant à une plateforme pour la création de MTTs de générer automatiquement des indices. Cette méthode a été appliquée à la plateforme Astus et démontre qu'il est possible de conce-

[1] Une action est dite « hors trace » lorsqu'elle n'est pas prévue par le modèle exécutable de la tâche.

voir un algorithme qui utilise les caractéristiques de cette plateforme afin de générer des interventions pédagogiques. Plus précisément, cette démonstration s'effectue par l'implémentation de la génération de deux types d'interventions : les indices par rapport à la prochaine étape et la rétroaction négative par rapport aux erreurs. Ce résultat peut être divisé en contributions informatiques et empiriques.

D'un point de vue informatique, l'implémentation de la génération d'indice par rapport à la prochaine étape a permis de 1) identifier les caractéristiques de la représentation des connaissances d'Astus qui permettent la génération d'indices et de 2) concevoir des gabarits d'indices tirant profit de ces caractéristiques afin de générer des indices qui ont un effet bénéfique sur l'apprentissage. La conception de la méthode pour la génération de rétroactions négatives a permis de définir un modèle du diagnostic des erreurs à partir des actions hors trace des apprenants. Ce modèle inclut 1) une description de comment les concepts d'impasses et de réparations définies par Sierra [20], une théorie expliquant les erreurs procédurales des apprenants, peuvent être appliquées à la représentation des connaissances d'Astus afin d'automatiquement modéliser les erreurs des apprenants, et 2) un algorithme décrivant comment Astus fouille son modèle de la tâche en réponse à une action hors trace afin de diagnostiquer les erreurs des apprenants.

D'un point de vue empirique, des expériences réalisées à l'aide d'étudiants inscrits au cours de programmation système à l'Université de Sherbrooke ont montré que les indices par rapport à la prochaine étape générées par Astus peuvent être aussi efficaces et aussi appréciés que ceux écrits par un enseignant. Des expériences supplémentaires qui ont été réalisées à l'aide d'étudiants du cours de structure de données ont montré qu'Astus est en mesure de diagnostiquer un grand nombre des actions hors trace des apprenants. De plus, les données obtenues semblent indiquer que les rétroactions négatives fournies par Astus ont contribué à l'apprentissage des étudiants. Par contre, la faible puissance statistique des données obtenues lors de cette expérience fait en sorte que des expérimentations supplémentaires seront nécessaires afin de valider ce résultat.

Structure de la thèse

La présente thèse inclut quatre documents détaillant les trois phases de la méthodologie proposées. Le chapitre 1 est un chapitre publié dans le livre *Advances in Intelligent Tutoring Systems* [16]. Il correspond à la première phase de la méthodologie proposée et décrit une étude comparative entre l'outil auteur CTAT et la plateforme Astus. Le chapitre 2 est un article qui a été soumis pour publication à la revue *International Journal of Artificial Intelligence in Education*. Cet article présente les méthodes permettant aux tuteurs produits à l'aide d'Astus de générer des indices par rapport à la prochaine étape (phase 2) et de diagnostiquer les actions hors trace afin de générer une rétroaction négative (phase 3). Le chapitre 3 est un article publié au congrès ITS 2012. Cet article décrit la méthode utilisée pour générer des indices par rapport à la prochaine étape. Son contenu est similaire à l'article présenté au chapitre 2, mais il décrit plus en détail l'analyse statistique effectuée afin de valider l'efficacité des indices générés par Astus (phase 2). Finalement, le chapitre 4 est un article présenté au congrès UMAP 2012. Il décrit le modèle élaboré afin d'appliquer Sierra [20] à Astus. Cette étape a permis la conception de l'algorithme du diagnostic des actions hors trace (phase 3).

Chapitre 1

Comparaison d'Astus et CTAT

Ce chapitre décrit une étude préliminaire qui a été effectuée dans le but d'évaluer comment Astus se compare aux plateformes similaires. Dans cette étude, Astus a été comparée aux Cognitive Tutors créés à l'aide de l'outil CTAT. L'objectif était de comparer le processus de modélisation d'une tâche avec chacune de ces plateformes et d'évaluer leur capacité à fournir des interventions pédagogiques.

La comparaison a été effectuée par la modélisation d'une tâche de soustraction et des erreurs connues pour cette tâche. L'étude a montré qu'il est possible de modéliser la soustraction ainsi que toutes ses erreurs connues à l'aide de CTAT et Astus. Ce résultat suggère que les deux plateformes ont une puissance de modélisation comparable. Par contre, la plateforme Astus a montré un avantage en ce qui concerne les interventions pédagogiques qu'elle peut fournir. Cet avantage provient principalement du système de représentation des connaissances utilisé par Astus. Le résultat de la comparaison de CTAT et Astus a donc permis de justifier le choix d'Astus afin d'implémenter la génération d'intervention pédagogique plutôt qu'une autre plateforme telle que CTAT.

L'auteur (Luc Paquette) a contribué au contenu de ce chapitre de livre en concevant la méthodologie utilisée afin d'effectuer la comparaison. Il a modélisé l'ensemble des tuteurs CTAT et Astus en incluant la modélisation des erreurs. C'est aussi lui qui a effectué l'analyse des résultats. La plateforme Astus avait déjà été conçue et implémentée, principalement par Jean-François Lebeau. L'auteur a contribué à 75 % de la charge de travail relié à la rédaction du chapitre. Il a rédigé toutes les sections sauf celle décrivant la plateforme Astus (section 3).

Authoring Problem-Solving Tutors: a Comparison between ASTUS and CTAT

Luc Paquette, Jean-François Lebeau and André Mayers

Université de Sherbrooke, Québec, Canada

{ luc.paquette ; andre.mayers@USherbrooke.ca}

Abstract. ASTUS is an Intelligent Tutoring System (ITS) framework for problem-solving domains. In this chapter we present a study we performed to evaluate the strengths and weaknesses of ASTUS compared to the well-known Cognitive Tutor Authoring Tools (CTAT) framework. To challenge their capacity to handle a comprehensive model of a well-defined task, we built a multi-column subtraction tutor (model and interface) with each framework. We incorporated into the model various pedagogically relevant procedural errors taken from the literature, to see how each framework deals with complex situations where remedial help may be needed. We successfully encoded the model with both frameworks and found situations in which we consider ASTUS to surpass CTAT. Examples of these include: ambiguous steps, errors with multiple (possibly correct) steps, com-posite errors, and off-path steps. Selected scenarios in the multi-column subtrac-tion domain are presented to illustrate that ASTUS can show a more sophisticated behavior in these situations. ASTUS achieves this by relying on an examinable hierarchical knowledge representation system and a domain-independent MVC-based approach to build the tutors' interface.

1. Introduction

Intelligent Tutoring Systems (ITS) that support a learning-by-doing pedagogical strategy are usually developed in line with one of three established approaches: model-tracing tutors [4], constraint-based tutors [17] and example-tracing tutors [3; 23]. All of these fit VanLehn's tutoring framework [27], in which a model of a task domain is used to evaluate each of the learner's steps (which are themselves driven by mental inferences) as correct or incorrect. Model-tracing tutors such as Cognitive Tutors [6] and Andes [28] have been proven to be successful in the classroom [14], but their success is mitigated by their cost [19], which is mainly due to the effort needed to develop a generative model of the task domain. Both constraint-based and example-tracing tutors are designed to reduce this cost. The former use an evaluative model involving constraints defined over a set of pedagogically relevant solutions, and the latter, a task-specific evaluative model built from a domain expert's interactions with the learning environment. Example-tracing tutor frame-works can offer tools to generalize the resulting model [3; 15], but adding such levels of complexity is an obstacle to their democratization, and still does not make them as flexible and comprehensive as model-tracing tutors. Constraint-based tutors may be particularly effective in handling ill-defined tasks in well-defined domains, such as design-based ones; however, they cannot follow learners as closely as model-tracing tutors, a capacity which is especially interesting for well-defined tasks. To reduce the effort needed to develop model-tracing tutors, one approach is to rely on a more (Cognitive Tutors) or less (Andes) domain-independent framework. In such a context, the knowledge representation system used to build the model is a key part of the framework, and its expressivity and reasoning capacity determine which domains can be modeled and how straightforward it is to model one.

Our work is based on the hypothesis that a more sophisticated knowledge representation system not only widens the range of domains that can be modeled, but also facilitates the testing of varied domain-independent pedagogical strategies, including some that are more elaborate than the ones usually found in model-tracing tutors. In fact, our efforts can be seen as an attempt to achieve an objective similar to that of Heffernan [12], who expanded Cognitive Tutors with a domain-specific (algebra) pedagogical model in order to provide tutorial dialogs closer to those used by experienced human tutors. Thus, our frame-work ASTUS is designed with two objectives: to reduce the prohibitive effort usually associated with the development of model-tracing tutors, and to provide the ITS community with a modular framework. In terms of Wenger's classic ITS architecture [29], ASTUS's domain-independent expert and interface modules interpret domain-specific models, and pedagogical and learner model modules can be customized to try out different pedagogical avenues. To achieve this, ASTUS uses a knowledge representation system that can be seen as a middle ground between the Cognitive Tutors production systems and the Andes solution graphs, in which the set of next possible steps is updated before each of the learner's actual steps. This approach, which is online (like Cognitive Tutors) but also top-down (like Andes), was adopted under the hypothesis that the advantages of both can be combined. Designed to model domains from a pedagogical perspective rather than to model the cognitive process used to solve them, ASTUS's knowledge representation system represents procedural knowledge hierarchically, using knowledge components of different grain sizes. Of these components, the examinable ones represent the skills explicitly tutored and the black-box ones model the underlying required abilities. Finally, to ensure that the tutor has complete access to the learning environment interface, as required by Anderson et al. [6], these knowledge components act as a Model in terms of the Model-View-Controller (MVC) architectural design pattern on which the interface module is based.

In this chapter we present a study that evaluates the effectiveness of ASTUS by comparing it with Cognitive Tutor Authoring Tools (CTAT), a model-tracing tutor framework derived from the Cognitive Tutors [13]. In order to compare the two frameworks, we used each of them to author a tutor for the multi-column subtraction task domain. Each contains a generative model that can be used to trace a correct problem-solving path, but also many different incorrect ones. The objectives of this study are to evaluate whether both frameworks allow us to exhaustively model the multi-column subtraction domain; to identify the features that make each framework more or less suitable for the situations covered by a given set of scenarios; and to show which types of pedagogical behavior are made possible by each framework. The tutors were authored with the sole purpose of comparing the frameworks; we do not plan to experiment with them in a classroom.

In the next sections of this chapter, we describe the two ITS frameworks (CTAT in Section 2, ASTUS in Section 3). For each, we begin by giving an overview of its design and then present its knowledge representation system in detail, with examples from the multi-column subtraction domain. Section 4 presents the methodology of our study, detailing each step that was taken in order to achieve our objectives. For instance, we explain why we deemed the multi-column subtraction domain a good choice to evaluate the strengths and weaknesses of the two frameworks. In Section 5, we discuss the features of the resulting tutors and present scenarios which illustrate how each framework deals with different situations encountered in modeling the multi-column subtraction domain. In particular, we discuss the difference between the frameworks' respective tracing algorithms, their difficulties in modeling specific situations and the types of pedagogical behavior they support. Finally (Section 6), we conclude that authoring a tutor in either framework is a similar task, but that with ASTUS, more attention must be paid in building the task domain model so that it can be fully exploited by the domain-independent pedagogical module. However, without any extra domain-specific effort, the resulting tutor offers elaborate pedagogical behaviors that are usually not supported in problem-solving tutors. The chapter ends with a brief presentation of future work towards a new version of the ASTUS framework.

2. The CTAT Framework

CTAT[2] is a freely distributed domain-independent ITS framework that can be used to create tutors for various well-defined task domains (examples include stoichiometry and genetics). The main objective of CTAT is to reduce the amount of work required in authoring these tutors [1]. The CTAT framework allows the creation of two different types of tutor: Cognitive Tutors and Example-Tracing Tutors) [2]. Cognitive Tutors are based on the ACT-R theory of cognition [5, 7] and use a cognitive model of the skills being tutored. To create such a model,

[2] http://ctat.pact.cs.cmu.edu

expertise in AI programming is required, as the model may be implemented using the Cognitive Tutor Development Kit (TDK) [4] or the Jess rule engine[3] (only the latter is distributed along with CTAT). In our study, the CTAT tutor was created as a Cognitive Tutor implemented via the Jess rule engine. In the following sections, when we speak of the CTAT framework, we are referring to Jess-based Cognitive Tutors authored using CTAT.

2.1 Knowledge representation

CTAT tutors, being derived from the TDK-based Cognitive Tutors, use Jess production rules to represent procedural knowledge and Jess facts to represent declarative knowledge. These facts, referred to as Working Memory Elements (WMEs), can be used to represent the task, model the learner's perception of the interface and store temporary results in working memory. The content of a WME is defined by slots that can be filled with primitive data (boolean, integer, string, etc.) or references to other WMEs. For example, in our multi-column subtraction tutor, the WME for a column contains the following slots (the type of each slot is indicated here for clarity's sake; it is not specified in the actual model):

```
WME Column {
  name [string]
  nextColumn [Column]
  previousColumn [Column]
  minuend [Textfield]
  subtrahend [Textfield]
  difference [Textfield]
  hasBeenBorrowedFrom [boolean]
}

WME Textfield {
  name [string]
  value [string]
```

[3] http://www.jessrules.com

```
}

WME DecrementColGoal {
  column [Column]
}
```

Production rules provide a cognitively plausible path to explain a learner's actions, whether they are steps in the interface or mental inferences. Thus, they can exhibit two kinds of behavior: recognizing a step (no more than one per rule) from an event sent by the interface, or updating the content of the working memory (adding, modifying or removing WMEs) to reflect a learner's mental inferences.

```
(defrule AddDecrementColumnGoal
  ?problem<-(problem (subgoals $?first ?evaluateGoal $?rest)
  ?evalGoal<-(evalColumnDecrementationGoal (column ?col))
  ?col<-(column (minuend ?minued &: (neq ?minuend 0))
=>
  (bind ?decColGoal (assert (decrementColGoal (column
    ?col))))
  (modify ?problem (subgoals (create$ $?first $?rest ?decCol-
    Goal))))
)
```

The above rule is fired when there is a goal of evaluating the decrementation of a column's minuend for a column with a minuend different from zero. The WME it creates will allow other rules to match, in a chain, including the final one that will result in the action of decrementing the minuend. The following rule is triggered if there is a goal of subtracting the problem's current column, and the step is recognized if the correct value is entered in the column's difference slot, as specified by the *predict-observable-action* statement.

```
(defrule Subtract
    ?problem<-(problem (currentColumn ?column) (subgoals
      $?first ?goal $?rest))
    ?goal<-(subtractColumnGoal (column ?column))
    ?column<-(column (difference ?diff) (minuend ?minuend)
      (subtrahend ?subtrahend))
  =>
    (bind ?difference (- ?minuend ?subtrahend))
    (predict-observable-action ?diff WRITE-VALUE ?difference)
  )
```

When a step is executed in the interface, CTAT tries to find a chain of rules that leads to it. A loop is initiated in which the content of the currently available WMEs is compared with the rules' firing conditions in order to find matches. As rules can alter the content of the working memory when they are fired, additional production rules can be fired and the matching process is started over until the rule producing the learner's step is found. If no such rule is found, the step is considered an error. In CTAT, pedagogically relevant errors are modeled using production rules marked as "buggy". Buggy rules, like normal ones, can either match a step or modify the content of the working memory. In both cases, errors are detected after exactly one incorrect step, when the chain of production rules that leads to this step contains a buggy rule.

3. The ASTUS Framework

ASTUS, like CTAT, is designed to be a domain-independent ITS framework available to the ITS community[4]. As the main objective of ASTUS is to allow experimentation with varied pedagogical strategies in the context of problem-solving tutors, much work has been focused on its foundations, the domain-independent expert and interface modules. Although we have implemented a basic knowledge-tracing [9] algorithm that fits ASTUS's hierarchical knowledge representation and a prototypal pedagogical module as a customizable expert system,

[4] An alpha version, limited to internal usage, has been completed and a beta version designed to be shared is under active development (http://astus.usherbrooke.ca).

we first made sure to support a complete inner loop [27] that can show ASTUS's potential. The next steps include developing these pedagogical and learner model modules to offer basic services and fully show the benefits of modeling a task domain with ASTUS, as well as developing authoring tools that will make modeling easier.

3.1 Knowledge representation

The knowledge representation system in ASTUS is derived from preliminary work on MIACE [16], a cognitive architecture inspired by ACT-R that proposes original twists useful in the ITS context [11; 20]. Using ASTUS's knowledge representation system, a task domain's declarative knowledge is divided into semantic (factual) and episodic (autobiographical) components, whereas procedural knowledge is modeled at three different grain sizes. First, *complex procedures* are dynamic plans generating a set of goals (intentions satisfied by procedures), according to an algorithm (e.g., sequence, condition, iteration). For example, in the subtraction tutor, the complex procedure *SubtractWithBorrow* is a partially ordered sequence:

```
SubtractWithBorrow(TopSmallerColumn c) {
  subgoal[1] := BorrowFrom(query{nextColumn(c)})
  subgoal[2] := BorrowInto(c)
  subgoal[3] := GetDifference(c)
  order-constraints {(2,3)}
}
```

Second, *primitive procedures* represent mastered abilities that correspond to steps in the learning environment. Here is one of the two primitive procedures in the subtraction tutor (the other is *ReplaceTerm*):

```
EnterDifference(Column c, Number diff) {
  c.difference := diff
}
```

Third, *queries* and *rules* represent basic or mastered mental skills, such as pattern-matching and elementary arithmetic. Along with complex procedures, they represent mental inferences. As queries and rules define how procedural knowledge components can access semantic ones, they are described in more detail in the discussion of this below.

A class of problem is associated with a goal (in the subtraction tutor, the *SubtractInColumns* goal) that can be satisfied by different procedures, complex or primitive, some of which may be marked as incorrect to represent pedagogically relevant errors (for instance, in the subtraction tutor, the incorrect procedure *AddInsteadOfSubtract*). As complex procedures specify sub-goals, the resulting graph has a goal as root and a set of primitive procedures as leaves.

Aside from goals, which define a semantic abstraction over procedures, semantic components include *concepts*, *relations*, *functions* and *contexts*, and their corresponding instances, *objects*, *facts*, *mappings* and *environments*. Concepts represent pedagogically relevant abstractions and are defined using both *is-a* relationships and essential features. Functions and relations, respectively, represent single- and multi-valued non-essential associations between objects. For example, the subtraction tutor includes these semantic knowledge components:

```
Concept Column {
  position [integer]
  minuends *[Minuend]
  subtrahends *[Subtrahend]
  difference [integer] (the value is initially unknown)
}  *a tuple where the first is the current one

Function nextColumn {
  column [Column] (argument)
  next [Column] (image)
}

Concept Term {
  units [integer] (0-9)
```

```
    tens [integer] (0-1)
  }

  Concept Minuend isA Term

  Concept Subtrahend isA Term
```

Contexts reify the subdivisions of the learning environment, which generally correspond to windows in the interface. Examples of multi-context learning environments include "wizards" and pop-up dialog boxes (the subtraction tutor has only one context). Thus an environment contains all the instances (objects, facts, mappings) related to a distinct subtask of the task domain contained in a context. Domain-level (vs. task-level) objects that represent constants (e.g., the integers 0-9 in the subtraction tutor) are part of a global context that is automatically handled by the framework.

The episodic knowledge components are instances of the semantic and procedural knowledge components that represent the learner's solution path. The current episode is a graph that contains procedures in progress, done or undone, next possible primitive procedures and planned goals that have not yet been developed.

Goals and procedures are specified with parameters and queries. The values of the former comes from the parent component (either a goal or a procedure instance) and the values of the latter comes from the current environment, according to domain-independent requests such as "get the unique object representing a given concept" or "get the image of a given function". For example, in the procedure *SubtractWithBorrow* (detailed above) a query fetches the image of the function *nextColumn* for a column specified as a parameter. Queries can also inspect the current episode. For example, in the subtraction tutor, a procedure inspects it to see whether a *BorrowFrom* goal has been satisfied or not.

Rules are used to make relations and functions operational and to classify objects (adding extra is-a relationships). Implemented using Jess, rules help to abstract many of the domain-specific computations that are not relevant in the tutoring process. For example, in the subtraction tutor, the *nextColumn* function is made operational with this rule:

```
(defrule nextColumn
  (Column ?column (position ?p))
  (Column ?next (position ?next :& (= ?next (+ ?p 1))))
=>
  (Instantiate Function "nextColumn" ("next", ?next) ("col-
    umn", ?column))
)
```

and a column is classified as a TopSmallerColumn with this rule:

```
(defrule TopSmallerColumn
  (Column ?col)
  (CurrentSubtrahend (column ?col) (subtrahend ?subtrahend))
  (CurrentMinuend (column ?col) (minuend ?minuend))
  (test (< (getValue ?minuend) (getValue ?subtrahend)))
=>
  (Classify ?col "TopSmallerColumn")
)
```

Before each of the learner's steps, the episodic graph is developed following each applicable complex procedure's plan, further specified by its arguments, to find the set of next possible primitive procedure. If a step committed by the learner, is included in this set, the graph is updated accordingly, in the other case, the step is added to the off-path steps stack. The graph is not updated while there is a least one step on the stack. Either the tutor or the learner can undo off-path steps, allowing resuming a problem-solving path that can be traced (i.e., not necessarily a correct one). Thus, the episodic components form an interpreted high-level log of the learner's steps, which are stored in a low-level log to allow an exact replay. The latter is required because even if the arguments collected from the learner's interaction in the learning environment match the arguments of a next possible primitive procedure, they may not be exactly the same. For example, the match can be limited to check for a common concept.

4. Methodology

In this study, we use incorrect procedural knowledge of the multi-column subtraction domain to add knowledge components to the model and thus yield more data to enrich our comparison. It is uncertain whether being able to give a detailed diagnosis of errors is useful in tutoring [25], but if it is not supported, it is not possible to conduct studies to evaluate the gains or lack thereof.

The methodology we followed comprises six steps. This section describes each step and its relevance to our comparison of the ASTUS and CTAT frameworks.

4.1 Choice of the task domain

We chose to model multi-column subtraction because it is representative of well-defined tasks in well-defined domains. Indeed, the arithmetic procedure of subtraction is well documented in the literature and is a clear and precise algorithm. Even though it is well-defined, the subtraction domain contains many inferences that must be deduced by the tutor from steps that may occur in a non-strict order. These characteristics give enough complexity to the solving algorithm to produce an interesting model that can be used to compare the features of the CTAT and ASTUS knowledge representation systems.

VanLehn [26] assembled a list of 121 procedural errors that can occur when a learner subtracts numbers. Incorporating these errors in the tutors adds knowledge components to implement in CTAT and ASTUS, thus introducing new and possibly complex situations for our models. These situations give us insights about the strengths and weaknesses of the two knowledge representation systems.

4.2 Framework-independent procedural model

Once we had chosen multi-column subtraction as the task domain for our comparison, we created a procedural model independent of any tutoring framework. This model is based on a procedural model presented by Resnick [24] and was used as a reference when we implemented the CTAT and ASTUS tutors. We chose this model as our reference because it can resolve any multi-column subtraction problem, and it is explicit regarding the inferences and steps that must be made and taken by the learner. To make this model more suitable for a tutoring context, we modified it slightly by adding a new way to subtract a column and including more detail in the borrowing section of the algorithm. The borrowing part of the algorithm was not deemed to be a natural extension of the semantic knowledge we assume the learner know (the base-ten numeral system's basis).

More specifically, the first modification made to Resnick's model is a new way to subtract the current column when it does not contain a number in its subtrahend (i.e., equivalent to a zero). In this case, instead of doing a subtraction, the algorithm simply copies the minuend in the difference section below the line. This modification was taken from a set of subtraction rules given by Brown and VanLehn [8] and is important in our model, since subtraction errors sometimes depend on the presence or absence of a subtrahend.

The second modification is applied to the original model's borrowing algorithm, to more closely capture the semantic knowledge of the domain in a procedure. When borrowing across zeros, our model does not change zeros to nines from right to left. Instead, it finds the first term which is not zero, decrements it and then iterates from left to right, changing zeros to nines. This procedure more closely represents the semantics of the base-ten numeral system: when borrowing one unit of the next column (to the left) is subtracted to add ten units to the current column. In the case of borrowing across a zero, ten units are borrowed into and one is borrowed from the column, thus changing the zero to a nine.

4.3 Error modeling

When solving problems using arithmetic procedures, systematic errors can be explained by the application of a faulty method [26]. Hence, for each of the 121 errors, we defined the modifications that are needed to our framework-independent model in order for it to be able to produce this error. Each error was modeled the same way: based on the error's definition, we established the modifications that had to be made to the correct model in order to recreate it. Once a model had been generated for all of the errors, we had to apply modifications to the implementation of both tutors. Also, since all the errors were successfully incorporated in our framework-independent model, we can conclude that a failure to implement an error in the CTAT or ASTUS framework is due to limitations of the framework and not because the error cannot be modeled in our problem-solving procedure.

4.4 Subtraction interface

Like the procedural model, the graphical user interface of the tutors was designed independently of the tutoring framework. It was inspired by the interface of POSIT [21], a subtraction ITS that diagnoses learners' errors while they are solving problems [22] and the traditional pen and paper approach. Our interface imposes as few limitations as possible on how learners interact with the learning environment, in order to allow them to express each of the 121 errors.

The interface we designed (shown in figure 1) allows two types of steps: 1) entering the difference for a column; and 2) replacing terms (minuends or subtrahends) in order to borrow from or borrow into a column. The UI actions used to trigger these steps are similar: the learner clicks on a column's difference input box or on the term he or she wishes to replace, and then enters a value using the numerical pad on the right. Once the "OK" button is pressed on the numerical pad, the problem display on the left is updated accordingly.

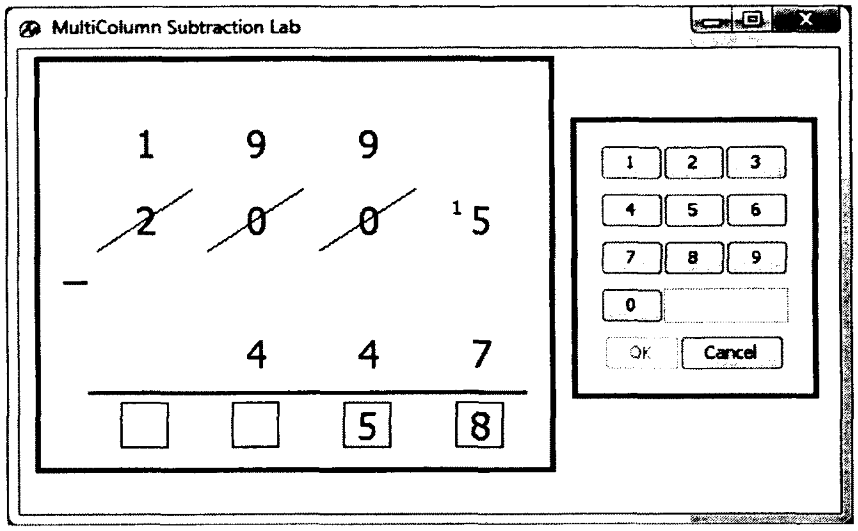


Figure 1 - The graphical user interface for the subtraction tutor

4.5 Implementation of the tutors

To implement the CTAT and ASTUS tutors, we began by covering only the knowledge required for a completely correct problem-solving path. As with the framework-independent procedural model, errors should be incorporated without modifying the model already implemented in the tutors.

The model we took from Resnick [24] accurately describes one way of executing the subtraction procedure, but we had to loosen some of its restrictions to make it applicable in the ITS context, where the model is primarily used to trace the learner's solving path. In particular, the steps that must be executed to subtract two numbers in column can be applied in many different orders while still obtaining correct results, so it is important to consider the multiple ways in which the learner can solve a problem. To be able to correctly trace the learner's solving path, we loosened the borrowing process to remove the constraint on the order of the required steps. For instance, the steps of changing a zero to a nine, decrementing a column and adding ten to a column can be executed in any order. Also, once the current column has been incremented by ten, it can be subtracted even if the borrowing process is not yet finished. Even though we gave learners more freedom, we kept the restrictions on the currently subtracted column so that they must complete all the steps relative to its subtraction before moving to the next one. This restriction has no impact on error modeling and although it is possible to subtract any column as long as subsequent borrows do not affect it, the ordered column sequence from right to left is a key part of the tutored subtraction algorithm.

The model created for our CTAT tutor is organized so that its production rules implicitly replicate a "while loop" in which each of the columns of the problem is subtracted individually, starting from the rightmost and ending with the leftmost. To trace the subtraction of individual columns, our model implements a sub-goal system. The use of such a system is common in examples of tutors given with the CTAT framework and it has the advantage of being flexible with regard to the order of the learner's steps. Sub-goals are normal WMEs that are added to the working memory with the purpose of indicating the steps that are currently possible. In our CTAT model, sub-goals are added to the working memory in the order defined by the framework-independent procedural model. An executed step can thus be accepted if it matches one of the current sub-goals, even if it does not follow the usual ordered sequence of the algorithm. The sub-goal associated with this step is then removed from the working memory to prevent it from being executed again. Once all of the sub-goals for a column have been successfully achieved, the model finds the next column and restarts the process until there is no column left to be subtracted, in which case a sub-goal is added calling for the learner to click on the "Done" button to indicate that he or she considers the problem solved. The "Done" button can also be clicked on when all the columns left to be subtracted have a difference of zero, since these extra zeros are not significant.

ASTUS's model consists of an ordered "for-each" iteration procedure on the columns of the problem. Thus, as in the CTAT model, each column is subtracted individually, from right to left. As each column is subtracted, the episodic graph is developed to obtain the set of primitive procedures that can be executed; the restriction on the order of execution of these primitive procedures is already contained in the complex procedures used to model the column's subtraction. To allow the learner to solve the problem with reasonable freedom, we ensured that the complex procedures contained only those order constraints that are required (borrowing into a column before subtracting it) or pedagogically relevant (subtracting only one column at a time). The problem is considered solved when the "for-each" iteration ends (when the last column of the iteration has been subtracted), or when the "Done" button is pressed (the idea was borrowed from CTAT, but the actual implementation is different because it is handled automatically by the framework), to let the learner finish the problem as soon as all the remaining columns have a difference of zero.

4.6 Incorporation of errors

Once we had completed the basic implementation of the two tutors, we started improving them by incorporating the errors we had modeled. Of the 121 documented errors [26], we implemented 46. The first 26 errors were modeled by systematically incorporating each error, in the same order we followed in our framework-independent model. After those errors were completed, we had enough experience to evaluate the challenges incorporation of specific errors would pose. Of the 95 remaining errors, we implemented 20 that were more challenging and required modeling behaviors that had not been previously encountered. The remaining 75 errors were not incorporated in the tutors because they were similar to the ones previously implemented and thus did not pose new challenges.

Both frameworks allowed the successful incorporation of the 46 selected errors, but the effort required showed the strengths and weaknesses of the CTAT and ASTUS knowledge representation systems. The "Results" section explains these strengths and weaknesses and the features from which they arise.

5 Results

As both frameworks allowed us to produce a comprehensive model of the subtraction domain, we need to show: how each framework's features have influenced the modeling process of the tutors and which type of pedagogical interactions are offered by each of the frameworks.

5.1 Modeling process

The results concerning the modeling process are presented in five sections. The first gives details on how ambiguous steps are managed by the two frameworks. Then we compare how specific error types are modeled and handled; in particular, we discuss errors containing multiple steps and errors containing correct steps. Next we present how each framework allows the reuse of knowledge components. Finally, we examine the coupling between the interface and the model and how this influences the implementation of the tutor.

5.1.1 Ambiguities

When tracing a learner's solving path, situations may arise in which different procedures yield the same steps. These steps are qualified as being ambiguous because they can be interpreted in more than one way. Ambiguities may be present in an error-free problem-solving path, but in the subtraction tutors, they result from the inclusion of errors. When we added errors to our tutors, we found three kinds of situations that led to ambiguities. First, two unrelated errors can lead to the same step for specific problems. For instance, the *add-instead-of-sub* (adding the term of a column instead of subtracting) and *smaller-from-larger* (subtracting the smaller number from the larger regardless of which one is the minuend) errors both result in the learner entering 2 as the difference when they are applied to a "5 – 7" column. Second, errors can involve the same behavior but in different application conditions, thus resulting in ambiguities when many sets of conditions are satisfied at the same time. For an example of this situation, consider the errors *diff-0 – N = 0* (writing 0 as the difference when subtracting a number N from 0) and *0 – N = 0-after-borrow* (the same behavior but occurring only when the column has already been borrowed from). Third, ambiguities occur when some or even all of the steps of an incorrect procedure can be considered correct. In this case, the tutor must be able to determine which procedure was executed when an unambiguous step is taken, or give priority to the correct procedure when it is not possible to decide whether the error was committed or not. An example of this situation is the *borrow-skip-equal* error (skipping columns where the minuend and subtrahend are equal during the borrowing process), in which correct steps are omitted rather than incorrect ones being performed. Details of this error are presented in the "Errors containing correct steps" subsection.

Using the model tracing algorithm of CTAT, ambiguities are resolved according to rule priority. When a step is executed, a search is performed to determine which rule path can explain it. Since the execution order of this search is based on the priority of each rule, the rules of the matching path with the highest priorities will be discovered first and will be fired, preventing the discovery of any ambiguities that might have occurred. Additionally, buggy rules always have lower priorities than valid ones, so a step that can be evaluated as correct will be so evaluated even if an incorrect path exists.

ASTUS's hierarchical procedural model allows a tracing algorithm in which an episodic graph containing all the applicable procedures is updated before each of the learner's steps. This feature allows ASTUS to detect and handle ambiguous steps. The graph is used to select the appropriate explanation for previous steps where evidence resolving the ambiguities was found. Evidence includes the execution of a non ambiguous step or the signal sent by the "Done" button. In the former case, the framework uses the episodic graph to find the nearest common ancestor of the procedure actually executed and the ambiguous one. In the latter case, the graph is searched for a procedure that is completed and that can satisfy the root goal.

In summary, CTAT prevents ambiguous states by always firing the production rules with the highest priority. This method is easy to implement and handles most of the ambiguity in a way that has no negative effects on the tracing process. It is possible to introduce special treatment of ambiguities by adding extra production rules to the model. An example is given in the "Errors containing correct steps" subsection below. CTAT is thus capable of handling any ambiguity, but the draw-back is a more complex model containing superfluous knowledge components. With ASTUS, all ambiguities are handled directly by the framework's tracing algorithm, thanks to its top-down hierarchical procedural model and the resulting episodic graph. This approach requires more effort when developing the framework, but authors do not have to worry about ambiguities. Both ASTUS and CTAT must deal with permanent ambiguities that no evidence can resolve, either by using priorities or by relying on the learner model.

5.1.2 Errors with multiple steps

Some errors require multiple steps from the learner for complete diagnosis. This is the case for errors where the whole problem is solved incorrectly, such as *add-no-carry-instead-of-sub* (the learner executes an addition without carrying instead of a subtraction). As well as involving multiple steps, these errors can also produce ambiguities. For instance, in the *add-no-carry-instead-of-sub* error described previously, columns where the sum of the minuend and the subtrahend is the same as their difference can exist. An example of this would be the 15 – 15 problem where the sum and the difference of the unit column are both 0 and, because the carry is not performed, there is no way of knowing whether the learner intended to add or subtract the columns (i.e., the ambiguity is permanent).

$$\begin{array}{r} 45 \\ -25 \\ \hline 60 \\ \text{(a)} \end{array} \qquad \begin{array}{r} 44 \\ -22 \\ \hline 62 \\ \text{(b)} \end{array}$$

Figure 2 - Examples of the add-no-carry-instead-of-sub error. (a) The first column is ambiguous. (b) The error should not be diagnosed

When it comes to reacting to the learner's errors, CTAT implements a policy of immediate feedback in which there is no delay between the execution of the error and the tutor's feedback. Thus, CTAT's error detection only allows one step to be executed before feedback is given; this poses particular challenges when dealing with errors containing more than one step. For instance, when dealing with errors that are composed of multiple steps that can be executed in different orders, since there is no way of knowing which one the learner will execute first, all of them need to trigger the error diagnosis. Also, because the learner does not have the opportunity to complete all of the steps in the error, CTAT has less evidence that the error it diagnosed is really the one being made. Other challenges come from errors that require an ordered sequence of steps. In those cases, since only one step can be executed before feedback is given, the tutor should diagnose the error when the first step of the sequence is executed. This is difficult when the first steps are ambiguous. Figure 2 illustrates such a situation in the case of the *add-no-carry-instead-of-sub* error. If the error diagnosis is made on the basis of the first column alone, the error is not detected in figure 2 (a) since the first step is ambiguous. On the other hand, if we allow the error to be diagnosed on any column, it will be detected in figure 2 (b) even if the first column was correctly subtracted[5]. To achieve accurate diagnosis in every situation, this ambiguity must be handled in the procedural knowledge, in a way that is specific to this error. Thus, when an addition is detected, the model needs to iterate on all the columns that were previously processed to see whether the entered results can be interpreted as either the subtraction or the addition of each column. If an ambiguity is found for each of the preceding columns, then the *add-no-carry-instead-of-sub* diagnosis can accurately be given.

Even if we overlook CTAT's immediate feedback policy, it would still be challenging to handle errors with multiple steps because production rules are independent of each other. The lack of an explicit hierarchy in the procedural model has two important consequences: it is 1) difficult to identify that multiple steps are caused by the same occurrence of an error and 2) difficult to evaluate how many steps are caused by an occurrence of an error. These two characteristics prevent the system from delaying feedback until the learner completes all of the steps in an error.

In ASTUS, an error with multiple steps can easily be modeled by creating an incorrect complex procedure. The model's hierarchy allows the creation of an episodic graph in which the primitive procedures caused by a specific occurrence of an error are easily identified. It is thus possible for the tutor to associate multiple steps with a specific error occurrence.

[5] Since the learner correctly subtracted the first column we can assume that he/she can subtract (without borrowing) and that the second error is not due to the *add-no-carry-instead-of-sub* error.

5.1.3 Errors containing correct steps

There are situations where an error is not described by the incorrect steps it causes but rather by the correct ones that are skipped when it is made. Being able to handle these errors is a specific case of ambiguity management, and thus the features of each framework that make it possible or challenging are the same ones de-scribed previously in the ambiguity subsection. An example of such a situation is the *borrow-skip-equal* error (skipping columns where the minuend and subtrahend are equal during the borrowing process). An example of its execution is shown in Figure 3, where all of the executed steps for the first column are correct and the error can only be diagnosed when the learner subtracts the second column and enters zero as its difference. Furthermore, in the situation shown in Figure 3 (a), there are four correct steps that must be performed before the next column can be used to diagnose the error. It is essential that the exact sequence of steps is executed before diagnosing the *borrow-skip-equal* error since, as shown in figures 3 (b) and 3 (c), the exclusion or addition of one could change the diagnosis. In 3 (b), five correct steps have been performed and one of them was changing a zero to a nine in a column where the minuend and the subtrahend are equal. Because of this step, we know that the error made is not *borrow-skip-equal*, but probably a slip where the learner forgot one of the columns. On the other hand, Figure 3 (c) shows a situation where only three of the four required steps have been executed. This shows why it is mandatory to check for the presence of the exact sequence of steps defined by the error; in this case, removing one of them can completely change the diagnosis. The solving path shown in this figure could be associated with either the *borrow-across-zero* (not borrowing on zeros and skipping to the next column) or the *always-borrow-left* (always borrowing from the leftmost column) errors.

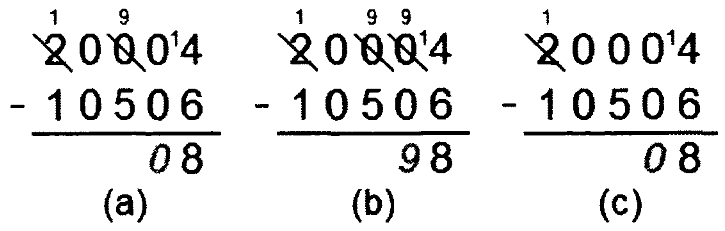


Figure 3 - An example of the borrow-skip-equal-error. (a) The error can only be diagnosed when the difference is entered in the second column. (b) When an additional step is present, we cannot evaluate the error as borrow-skip-equal. (c) One of the step is missing and the error could either be borrow-across-zero or always-borrow-left

In CTAT, we need to be able to determine when entering the difference of the next column will cause the error to be diagnosed. Since all of the correct steps are already covered by existing production rules, we have to add rules in order to 1) mark all steps that would be performed if the learner was executing the error and 2) iterate through the columns to check that each of these steps was executed. Handling errors containing correct steps is similar to handling errors with multiple steps: the mechanism used to manage the ambiguities must be implemented in the model. Implementing such a mechanism can be complex and tedious: for the *borrow-skip-equal* error, 11 production rules are required to give a correct diagnosis, while only 14 are required to trace an error-free subtraction solving path. Adding ambiguity management in the model for this error requires almost as much effort as implementing the complete model for the basic subtraction tutor. Thus, one advantage of a framework with a knowledge representation system similar to ASTUS's is its built-in approach to manage ambiguities, significantly reducing the effort needed to model complex errors.

Modeling this kind of error in ASTUS does not cause any difficulty: an incorrect complex procedure containing an ordered sequence of sub-goals is created, with a last one implicitly indicating that a part of the correct procedure was skipped. In the case of the *borrow-skip-equal* error, the last sub-goal is to subtract the next column. The main difference is that in ASTUS the error is part of the domain, whereas in CTAT it is recognized using additional rules but not explicitly modeled (e.g., the tutor cannot generate the results of this error to demonstrate it).

5.1.4 Reuse of knowledge units

Both CTAT and ASTUS allow the reuse of knowledge components in different situations. In this section we give examples of how each of the frameworks reuses knowledge components in the subtraction domain. Both knowledge representation systems allow an author to reuse previously defined procedural and semantic knowledge components to model errors. Reusing existing components reduces the complexity of the resulting model and decreases the effort needed to implement it. An example of how the modeling process can be simplified is taken from the *always-borrow* error, in which the learner systematically borrows before subtracting a column even if it is not necessary. To implement this error, we simply reuse the borrowing procedure that has been previously modeled, by forcing its use on a column that would not require it.

In CTAT, procedural knowledge can be reused with the help of buggy production rules that do not produce a step. These rules can be used to alter the content of the working memory in order to create a rule path containing valid rules ultimately flagged as incorrect. In our model, we use previously defined sub-goals to reproduce existing behaviors in an erroneous context. For instance, with the *always-borrow* error, a buggy rule adds existing sub-goals relative to the borrowing process (i.e., decrementing the next column, adding ten to the current column) in a situation where they are not required. The equivalent can be achieved in ASTUS by defining incorrect complex procedures that use existing goals. Hence, procedures that had been previously defined are used to describe erroneous behaviors. For example, the *always-borrow* error is modeled by an incorrect procedure using the goals of borrowing from and borrowing into a column.

Error composition happens when multiple erroneous behaviors are present at the same time in a solving path. In both CTAT and ASTUS, the reuse of correct procedural knowledge in modeling pedagogically relevant errors allows the recognition of composite errors because the correct procedures that are reused can themselves have erroneous alternatives. The "Pedagogical interactions" section in 5.2 shows an example of a learner displaying both the *always-borrow* (borrowing even if not required) and the *borrow-from-bottom* (borrowing from the subtrahend) behaviors.

In CTAT, production rules test conditions on existing WMEs and perform computations to modify or create new ones. In ASTUS, the complex procedures follow a fixed, domain-independent behavior so that domain-specific conditions and computations are not directly included in them, but added indirectly via queries and rules. For example, the mapping between a column and the number of zeros to its left can be used by both the *borrow-decrementing-to-by-extras* (when borrowing into, increment by 10 minus the number of zeros borrowed across) and *decrement-by-one-plus-zeros* (when borrowing from, decrement by 1 plus the number of zeros borrowed across) errors. In CTAT, this behavior can be emulated with highly prioritized production rules that perform computations and store the result in WMEs. In many cases, including the errors cited above, high priority of the rules is necessary because the computations must be performed before the steps of the borrowing process have been executed, even though the buggy rule can be executed after them. For instance, in the *decrement-by-one-plus-zeros* error, decrementing the minuend of the column that is borrowed from can be executed after the zeros have been changed to nine. It is then crucial that the number of zeros to the left of the current column be counted before the terms are changed. Another example of the reuse of knowledge components in ASTUS is the classification of a column object as having "been borrowed from". Some errors occur only for these columns and the procedures modeling them require the column to be an instance of the "column borrowed from" concept. In CTAT, WMEs can contain slots with a boolean value to emulate classification. In our subtraction model, this is indeed the case. Such a slot must be manually updated in every production rule that could change its value, whereas in ASTUS, classification is automatically re-tested when an object is modified.

5.1.5 The coupling between the model and the interface

In this section we detail how each framework manages its user interface, based on examples taken from our subtraction tutors. The two frameworks have opposite approaches in terms of how they link the user interface and the model. In CTAT, the interface and the model are almost entirely separate: the interface has no access to the content of working memory and the only information concerning the interface that the model receives is through Selection Action Input (SAI) events. A SAI event is sent when a step is performed on the interface, and it contains the element of the interface that triggered the event (selection), the action that has been performed on this element (action) and the value that was entered (input).

Since there is no direct link between the interface and the model, information concerning the problem's current state that is useful for tracing must be stored in WMEs that are updated using the SAI events. For instance, in our subtraction tutor, the working memory contains WMEs representing each of the interface's text fields, to store their value. We must ensure that the values contained in working memory are synchronized with the content shown on the interface by using the data contained in the SAI events.

Another effect of having a weak link between the interface and the model is that the interface's adaptability to the problem data is limited. Since the interface has no access to the content of working memory, it cannot use the problem data to generate its interface elements. Thus, it is difficult to dynamically add interface elements in order to match the problem state. In our subtraction tutor, this means that the interface has a fixed number of columns and a problem cannot contain more or less than that number, as this would require creating (or hiding) the text field dynamically[6]. The severity of this limitation varies according to the particular design of a tutor's interface: for example, it would have been possible to dynamically add columns if we used a "table component"[7] instead of individual text fields for each term.

The ASTUS framework enforces a stronger link between the model of the task domain and the interface [10]. Unlike CTAT, ASTUS gives the interface elements access to the instances of the current environment. This is achieved by the use of *views,* scripts that describe the representation of concepts in the interface. This MVC-based approach allows the interface to reflect the content of the current environment at all times. When the effects of a primitive

[6] It is possible for a production rule to call static Java methods to produce side-effects on the tu-tor's interface; a sample "Truth tables" tutor uses this technique.
[7] This is the solution used in sample addition and subtraction tutors.

procedure are applied to the environment, the views of the modified objects are notified so that they can update their UI representation accordingly. For example, in our subtraction tutor, when borrowing from or decrementing a minuend, the primitive procedure adds a new minuend to a column object, which view is then notified so that it displays the new minuend and strikes the old one. This approach enables a tutor built with ASTUS to have a flexible interface that adapts itself dynamically. For this reason, the ASTUS subtraction tutor can have the right number of columns to fit any problem.

Each primitive procedure is associated with an *interaction template* that triggers the execution of a step when the required basic UI actions are matched [10]. These templates offer a solution to the difficulty which CTAT tutors circumvent by "tutorable" interface elements (for example, a panel containing multiple combo-boxes and a button). Instead, with our approach, steps can be triggered by multiple interactions on multiple interface elements. The most important benefit of this approach is its capacity to generate the interactions by using the template and the episodic graph to produce a step with complete visual feedback (i.e., mouse moves and clicks). It is then possible to use the demonstration of a step as pedagogical feedback. There is an obvious cost for supporting this form of feedback, but we found out that it also helped us come up with more comprehensive models, as data implicitly defined by the interface must be explicitly encoded to define the templates. In some specific cases, we may implement multiple interactions in a single component if decomposing the step is not pedagogically relevant. For example, the "numerical pad" of our subtraction tutors is a single component.

We believe an MVC approach to the model-interface coupling has many advantages over a weaker one. It allows us to adapt the interface to the content of a particular problem, it prevents synchronization issues between the interface and the environment and it allows sophisticated pedagogical interactions such as demonstration. On the other hand, a weaker link such as the one offered by CTAT is easier to implement and, more importantly, makes it easier to develop tools that can be used to create the tutors' interfaces. These authoring tools are important, since they can greatly reduce the effort required to create a new tutor, but they are always more effective when dealing with rather simple or formatted interfaces.

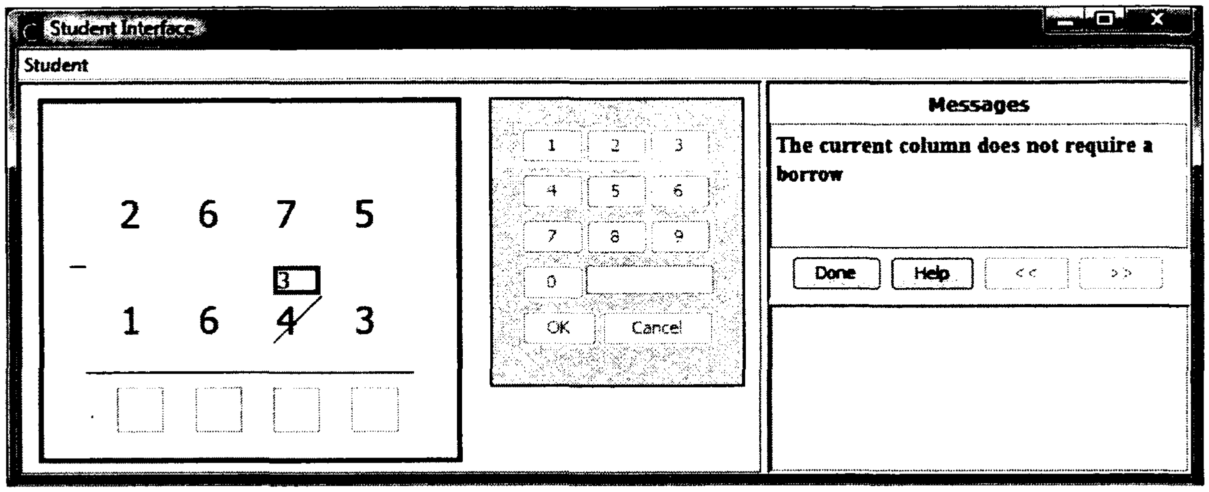
5.2 Pedagogical interactions

In this section we show how each of the frameworks gives pedagogical feedback. We start by describing the different types of interactions that are supported by at least one framework. We then use four scenarios taken from the subtraction tutors to give examples of how CTAT and ASTUS behave when reacting to learners' steps. Table 1 (presented at the end of this section) summarizes how the individual interaction types are supported by each framework.

5.2.1 Pedagogical interaction types

- Immediate feedback: minimal feedback to indicate whether a step is correct or incorrect. This includes flag feedback: changing the color of an input value to indicate whether it is correct or incorrect.
- Interface highlights: focusing the learner's attention on a specific part of the interface, for example by painting a rectangle over it.
- Next-step hint: giving a hint towards one of the next correct steps.
- Error-specific feedback: giving feedback regarding an error that is contained in the model.
- Off-path error recognition: recognizing certain off-path steps and giving feed-back accordingly.
- Demonstration: demonstrating, with full visual feedback, how to perform a step (i.e., the tutor takes control of the mouse and keyboard).

5.2.2 Scenario 1



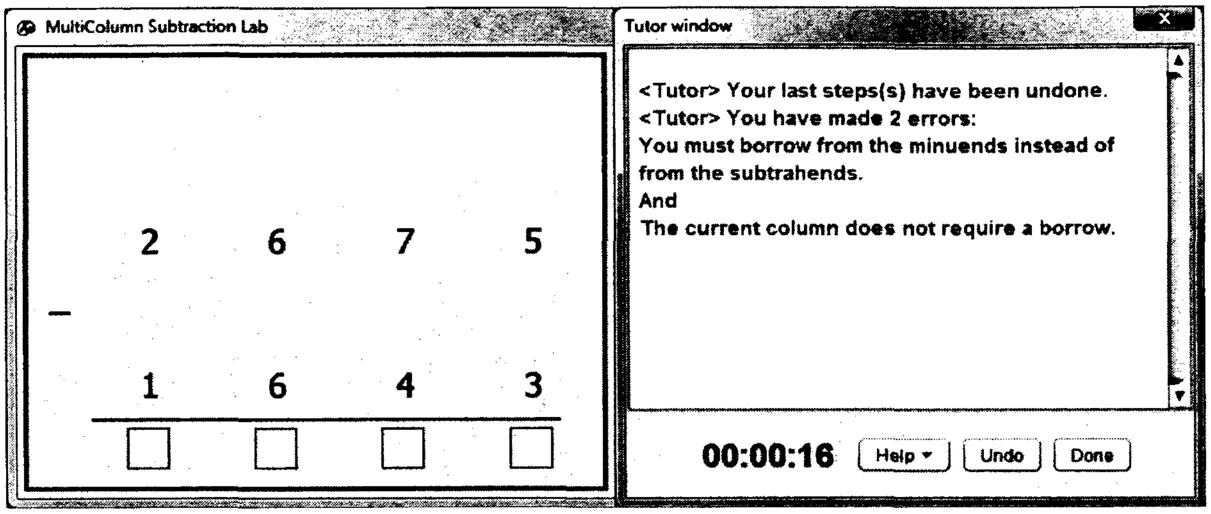
The current column does not require a borrow

Figure 4 - An example of CTAT's reaction to a composite error

The first scenario illustrated in Figure 4 shows a case of error composition: solving the problem 2675 – 1643, the learner changes the 4 (subtrahend of the second column from the right) to a 3. This action is caused by the composition of two errors: *always-borrow* (borrowing even if it is not required) and *borrow-from-bottom* (borrowing from the subtrahend). For both frameworks, the recognition of composite errors is made possible by reusing procedural knowledge, but the feedback they produce is different.

In this situation, the CTAT tutor reacts (see Figure 4) by: 1) flagging the incorrect value entered by writing it in red; 2) highlighting the replaced term by drawing a blue frame around it; and 3) giving a message related to the error. This message is produced from a template associated with the first buggy rule encountered (*always-borrow*) while the second (*borrow-from-bottom*) is ignored.

In the same situation, the ASTUS tutor reacts by first undoing the learner's step and updating the interface accordingly; and second, displaying a message that indicates which errors have been traced. As shown in Figure 5, in cases of error composition, ASTUS's written feedback includes all of the errors found.



<Tutor> Your last steps(s) have been undone.
<Tutor> You have made 2 errors:
You must borrow from the minuends instead of from the subtrahends.
And
The current column does not require a borrow.

Figure 5 - An example of ASTUS's reaction to a composite error

5.2.3 Scenario 2

The second scenario shows how the tutors interact with the learner when a next-step hint is requested. We show how the learner's solving path affects the chosen hints by illustrating how the tutors react to two hint requests. These requests lead to the same step but at two different points in the solving path of the 2005 – 1017 problem. The first hint request is executed when no step have been yet performed by the learner (Figure 6) while the second one is made when the only remaining step is to decrement the minuend (2) from the leftmost column (Figure 7).

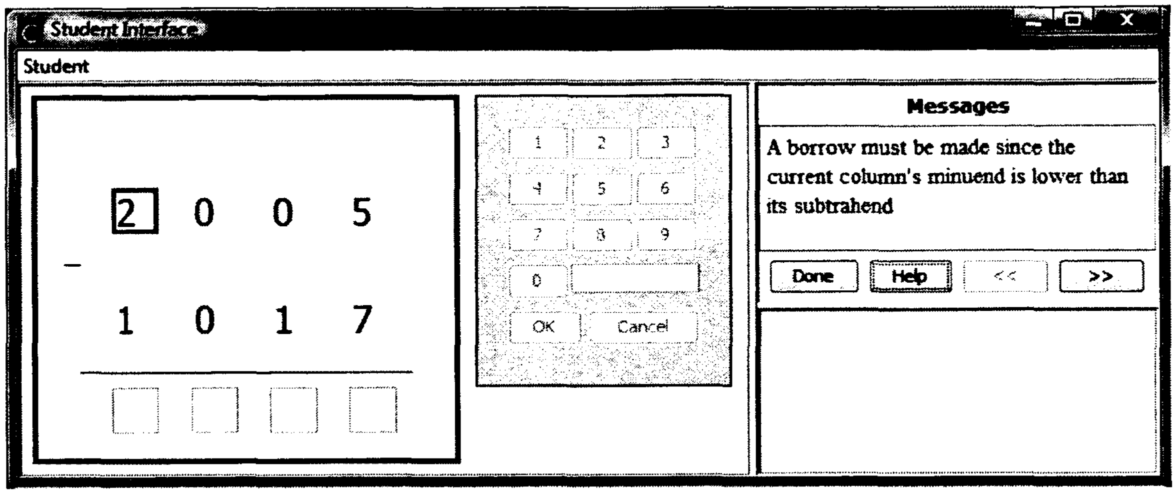

A borrow must be made since the current column's minuend is lower than its subtrahend

Figure 6 - CTAT's feedback when a hint is requested at the beginning of the problem

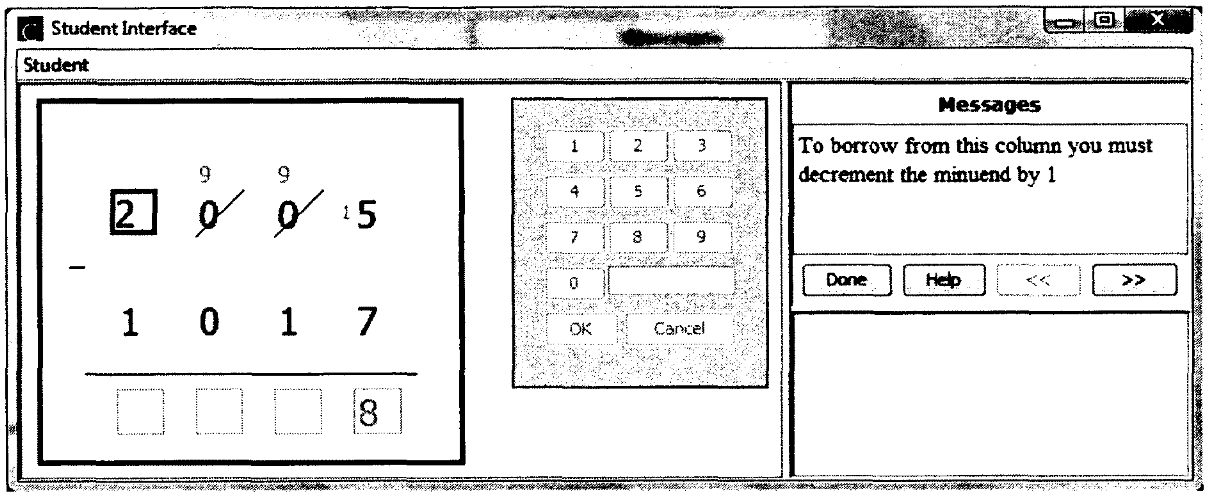
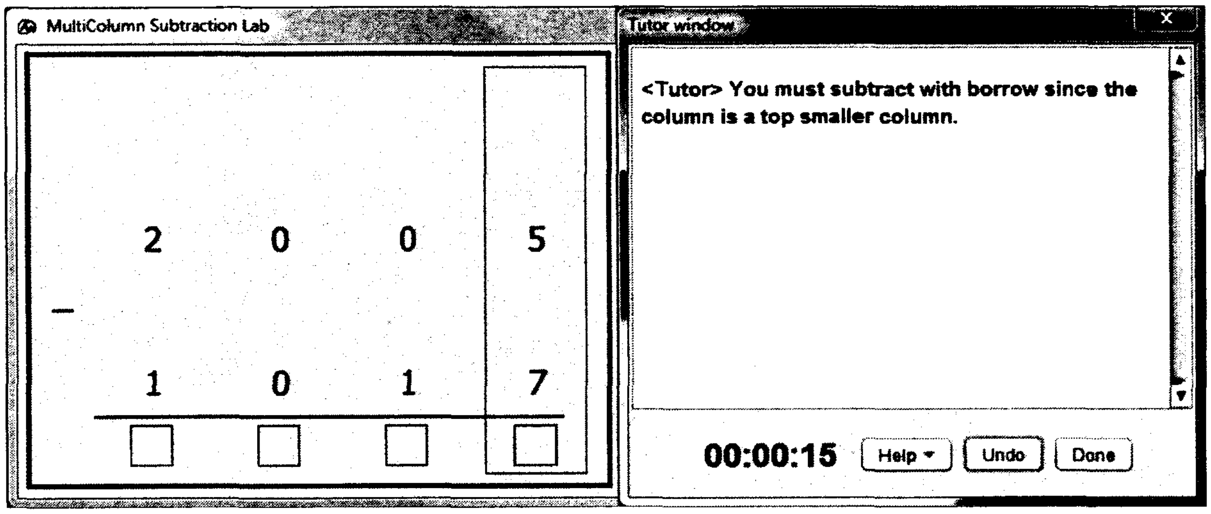

To borrow from this column you must decrement the minuend by 1

Figure 7 - CTAT's feedback when a hint is requested and only the decrementing remains

The method used by CTAT in providing hints is to 1) find the rule chain that produces the next step; 2) generate messages using the templates associated with each of the rules and display them in the same order the rules were fired in (the arrow buttons can be used to navigate the hint list); and 3) highlight the interface component on which the action must be executed. In this scenario, the two hint requests generate different production rule chains and thus the first hint message displayed differs depending on the solving path. For the first request, the hint given concerns the condition required to borrow (Figure 6) and, for the second, the hint concerns decrementation of the minuend (Figure 7). In both cases, even if the messages displayed are different and refer to different parts of the problem, the highlight is applied to the minuend of the leftmost column. In the case illustrated by Figure 6, the highlight should focus the user's attention on the current (rightmost) column but, since the highlight is determined by the SAI event contained in the production rules, the displayed highlight does not correspond to the hint message. Even though highlighting is only supported for rules containing SAI events, we see nothing in the knowledge representation system that would prevent its implementation in rules that modify the working memory.



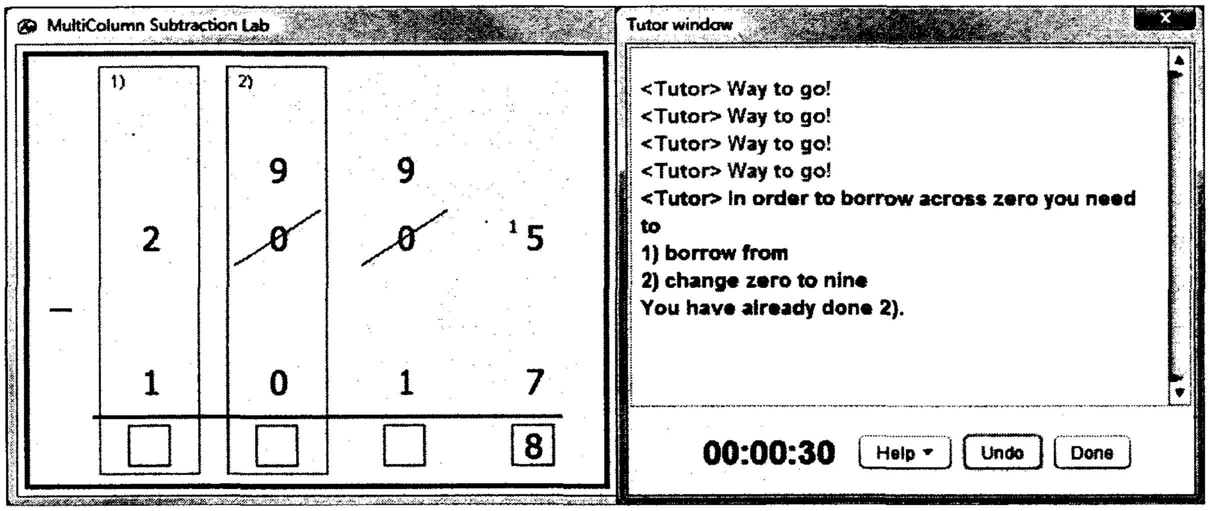
<Tutor> You must subtract with borrow since the column is a top smaller column.

Figure 8 - ASTUS's feedback when a hint is requested at the beginning of the problem

In ASTUS, the tutor uses the examinable knowledge components of the model to automatically generate next-step hints[8]. In contrast with CTAT, no message templates are required, although such templates are also supported for situations where customized messages are helpful. Using the hierarchical procedural model, the episodic graph and the learner model, the tutor can evaluate which procedures the learner is most likely to be executing and choose the one on which help should be given. For example, in figure 8, the next step is to decrement the minuend of the leftmost column. Since the borrowing process has not been initiated yet,

[8] Hint generation requires that knowledge components receive meaningful names in all supported languages; internationalization issues may arise.

the tutor evaluates that the learner needs help on the conditional procedure which determines whether borrowing is required. A hint is then generated using this procedure's definition. In figure 9, the next step is also to decrement the minuend, but the tutor recognizes that the borrowing process has been started and now gives feedback for the sequence procedure used when borrowing across a zero. With ASTUS, hints can be given on partially completed procedures (borrow across zero) while in CTAT, once a rule has been fired, it will never produce a hint again. ASTUS also behaves differently from CTAT in highlighting the interface. The use of views allows the tutor to highlight any object, even if its view is composed of multiple interface components, as is the case for columns. Additionally, ASTUS uses the procedure parameters to determine which components to highlight; high-lights are thus supported for every procedure, not only the primitive ones. All objects showing up as arguments that have a view in the current environment may be highlighted, but complex procedures may give special purposes to specific parameters and may use this information to select which ones will be highlighted (e.g., a for-each procedure has a parameter that designates the set to iterate on).



<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> Way to go!
<Tutor> In order to borrow across zero you need to
1) borrow from
2) change zero to nine
You have already done 2).

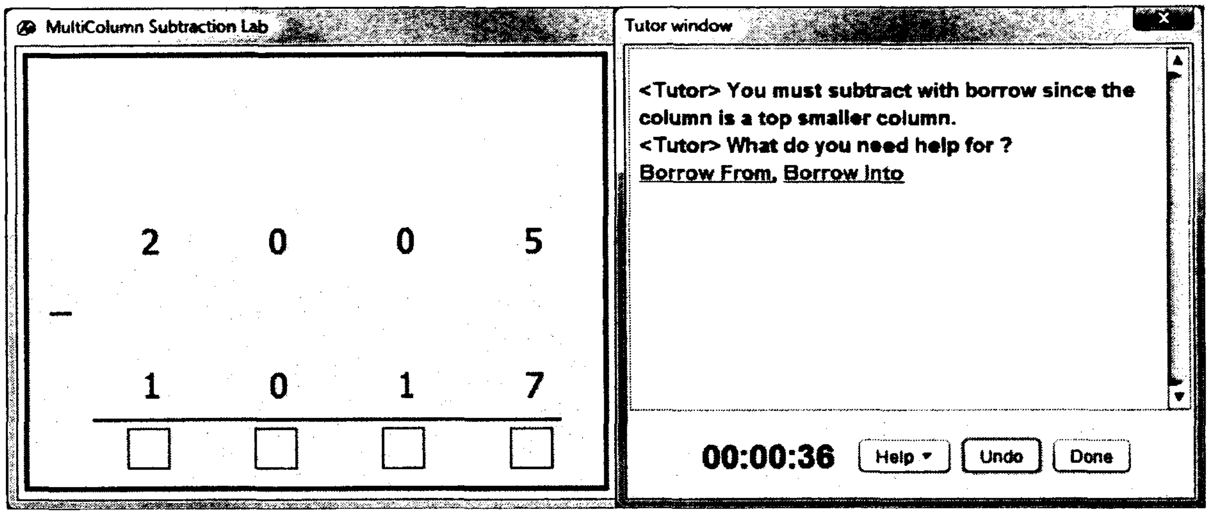
Figure 9 - ASTUS's feedback when a hint is requested and only the decrementing remains

5.2.4 Scenario 3

As shown in the second scenario, both CTAT and ASTUS produce multiple messages when a hint is requested. These messages are linked to the different procedural components used to produce the next step from the chain of matched rules in CTAT or the episodic graph in ASTUS. The tutor can provide hints at multiple levels, from more abstract (such as subtracting a column) to more specific (such as entering the difference).

Besides being able to vary the hint level between top-down and bottom-up, ASTUS can also offer help on any of the paths leading to a step. When the learner requests help and multiple different steps can be executed, ASTUS can ask for additional information to determine the best hint to provide. For this purpose, ASTUS uses links similar to the "Explain more" feature employed by Andes [28]. Each link represents a goal the learner can choose to request help on. For example, in the "2005 – 1017" problem, if the learner asks for help twice before performing any step, the tutor asks which step he/she intends to do. In this situation, the tutor would propose the "Borrow From" and the "Borrow Into" goals (Fig. 10).

We did not find an equivalent mechanism in CTAT's Jess-based cognitive tutors, although example-tracing tutors can give hints for different next steps depending on which interface component currently has the focus. A similar behavior could be implemented, as it would consist of searching for a production rule which results in an action that uses the interface component currently being focused on.



<Tutor> You must subtract with borrow since the column is a top smaller column.
<Tutor> What do you need help for ?
Borrow From, Borrow Into

Figure 10 - In ASTUS, the tutor asks the learner which goal he/she wishes to be helped with

5.2.5 Scenario 4

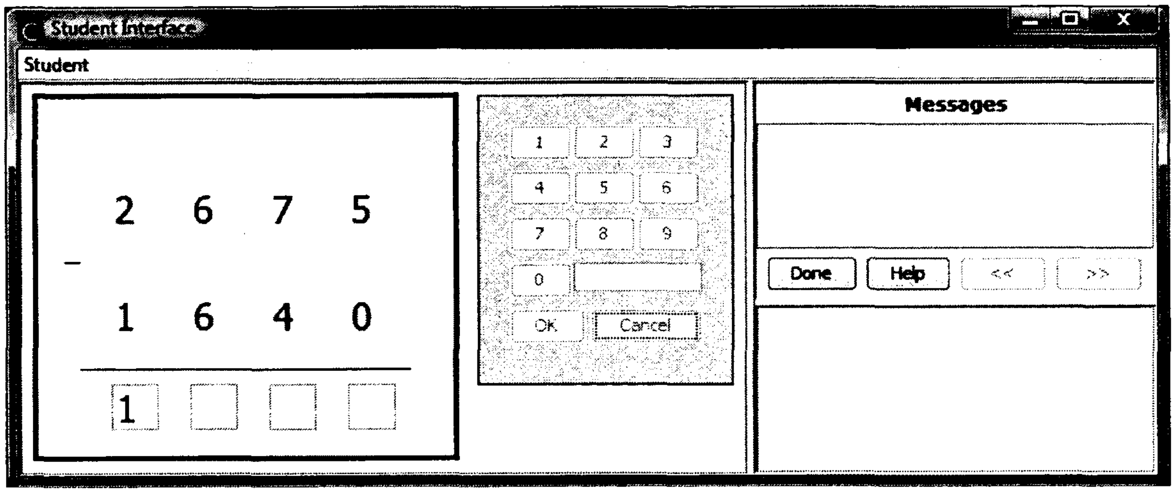
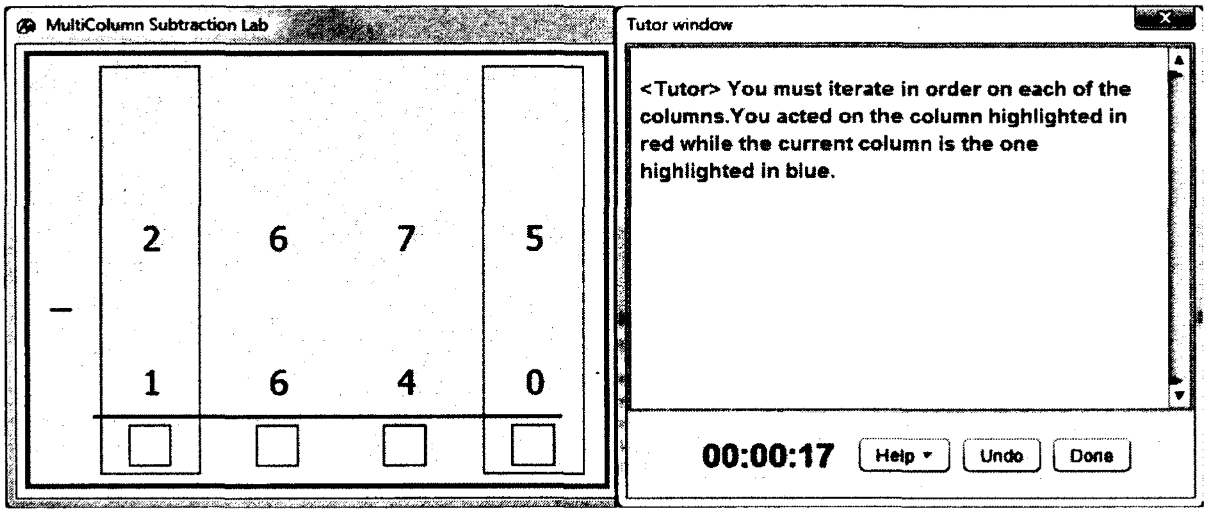


Figure 11 - CTAT's feedback for a step executed in a column other than the current one

The last scenario we present illustrates feedback automatically generated when off-path steps are recognized by the tutor. In this example, the learner tries to solve the "2675 – 1640" problem by starting with the leftmost column and entering 1 as the difference.

The feedback given by CTAT in this situation is shown in figure 11. Since there are no buggy rules that lead to the executed step, CTAT treats it as a non-specific error and only uses flag feedback to indicate that the step is incorrect.

With ASTUS's features, it is possible to implement recognition of errors that go against the scripted behavior of a complex procedure when an off-path step is executed, and to generate error messages accordingly. Figure 12 illustrates ASTUS recognizing an error in the iteration on the problem's columns. To make this diagnosis, ASTUS examines the currently available procedures to identify one describing an ordered iteration on a set of columns. It then checks the arguments of the primitive procedure executed by the learner to link one of them to a column on which the iteration is applied. If the identified column is not the one for which steps are currently available, the tutor generates feedback according to a domain-independent predefined template.



<Tutor> You must iterate in order on each of the columns.You acted on the column highlighted in red while the current column is the one highlighted in blue.

Figure 12 - ASTUS's feedback for a step executed in a column other than the current one

Type of feedback	CTAT	ASTUS
Immediate feedback	Flag feedback Red is used for errors, green for correct steps	Messages in the tutor window ("Way to go!" and "step has been undone")
Interface highlights	Can highlight interface component such as text fields	Can highlight any semantic knowledge viewable on the interface
Next-step hints	Abstract to specific	Abstract to specific Hints on multiple next-steps
Error-specific feedback	Supported for modeled errors	Supported for modeled and composite errors
Off-path step recognition	Not supported	Supported for planning errors such as forgetting an iteration in a "for-each procedure
Demonstration	Not supported	Supported

Table 1 - Summary of the supported feedback types

6. Conclusion

The goal of the study presented in this chapter was to compare the knowledge representation system of the ASTUS framework with a well known production rule-based system such as the CTAT framework. We especially wanted to see whether the two frameworks allow us to model the well-defined task domain of multi-column subtraction in its entirety, what kind of pedagogical interactions they can offer and which situations are more difficult to model in each of them. To achieve our objectives, we compared the modeling process of two subtraction tutors (one in each of the frameworks) into which we incorporated 46 pedagogically relevant errors.

Both CTAT and ASTUS were flexible enough to correctly model the subtraction task domain while still allowing freedom in the order of the learner's steps and being able to model all the errors found in the literature [26]. Even though all errors can be diagnosed by both frameworks, limitations were encountered in incorporating some of them into the models.

Regarding the pedagogical aspect of the tutors, both frameworks can provide immediate feedback, produce hint or error-specific message from templates associated to a procedural knowledge component and highlight elements in the interface. ASTUS has the advantages of being able to: recognize error composition, generate next-step hints or error-specific feedback on off-path steps, offer more sophisticated highlights, demonstrate how a step must be performed on the interface and giving the learners more control over the kind of help they need.

As we incorporated errors into our tutors, we encountered many situations that posed challenges to the modeling process. One challenge that appeared frequently was the management of ambiguous steps. CTAT's solution is simple to implement and ensures that ambiguities are prevented by always selecting the procedures with the highest priority. The author can still implement more complex management by adding production rules to the model. On the other hand, ASTUS handles complex ambiguity cases in its tracing algorithm. Another difficulty that the frameworks must be able to handle is the reuse of knowledge components. In our study, we encountered many situations where previously defined procedural or semantic knowledge components could be reused in order to keep the model as simple as possible. Both CTAT and ASTUS implement mechanisms that allow the author to easily reuse previously defined components. The nature of the problem can also bring difficulties in authoring a tutor In the case of subtraction task domain, the tutor must adapt the interface in function of the number of columns specified for each different problem. With CTAT, it is more difficult because of the lack of a direct link between the model and the interface.

The two frameworks follow different approaches regarding the authoring process of a tutor. CTAT's tracing algorithm and knowledge components are kept simple, as the primary focus has been on decreasing the effort needed to create tutors [1], in order to reduce the time needed by experienced modelers and make the modeling process accessible to non-programmers (Example-Tracing tutors [13] goes further in this way). On the other hand, ASTUS is designed to be used by programmers able to manipulate more complex knowledge components. These components allow us to incorporate a number of domain-independent behaviors such as ambiguity management and the generation of next-step hints or error-specific feedback on off-path steps.

With this study, we have shown how ASTUS's knowledge representation system allows us to model complex well-defined task, and that it can be compared to other existing frameworks such as CTAT. There is still much work that can be done to improve our framework. Our efforts have been largely focused on the knowledge representation and our next steps will be to develop other domain-independent modules such as the learner model, the outer loop's task selection and sophisticated domain-independent pedagogical strategies. It would also be interesting for us to do a similar comparative study with a constraint-based framework such as ASPIRE [18]. This would allow us to evaluate whether our system has advantages over the constraint-based approach and which elements of this approach can be adapted to improve the system we are developing.

Acknowledgments We would like to thank Vincent Aleven, Ryan S.J.D. Baker and Jean-Pierre Mbungira for their comments and suggestions on our study and its results.

References

[1] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R, The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains, *Proceedings of ITS 2006*, Ikeda, M., Ashlay, K., Chan, T.-W. (Eds), Lecture Notes in Computer Science 4053, Springer-Verlag, 2006, pp 61-70.

[2] Aleven, V., Sewall, J., McLaren, B.M., Koedinger, K.R, Rapid Authoring of Intelligent Tutors for Real-World and Experimental Use, *Proceedings of ICALT 2006*, 2006.

[3] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R., A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors, *International Journal of Artificial Intelligence in Education*, in press.

[4] Anderson, J.R., Pelletier, R., A Development System for Model-Tracing Tutors, *Proceedings of the International Conference of the Learning Sciences*, 1991.

[5] Anderson, J.R., *Rules of the Mind*, Lawrence Erlbaum Associates Inc, 1993.

[6] Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R., Cognitive Tutors: Lessons Learned, *The Journal of the Learning Sciences*, 4(2), 1995, pp 167-207.

[7] Anderson, J.R., Lebiere, C., *The Atomic Components of Thought*, Lawrence Erlbaum Associates, Inc, 1998.

[8] Brown, J.S., VanLehn, K., Towards a Generative Theory of "Bugs", in *Addition and Subtraction: A Cognitive Perspective*, Lawrence Erlbaum Associates Inc, 1982.

[9] Corbett, A.T., Anderson, J.R., Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge, *User Modeling and User-Adapted Interaction*, 4, 1995, pp 253-278.

[10] Fortin, M., Lebeau, J.-F., Abdessemed, A., Courtemanche, F., Mayers, A., A Standard Method of Developing User Interfaces for a Generic ITS Framework, *Proceedings of ITS 2008*, Woolf, B.P., aïmeur, E., Nkambou, R., Lajoie, S. (Eds), Lecture Notes in Computer Science 5091, Springer-Verlag, 2008, pp 312-322.

[11] Fournier-Viger, P., Najjar, M., Mayers, A., Nkambou, R., A Cognitive and Logic Based Model for Building Glass-Box Learning Objects, *Interdisciplinary Journal of Knowledge and Learning Objects*, 2, 2006, pp 77-94.

[12] Heffernan, N.T., Koedinger, K.R., Razzaq, L., Expanding the Model-Tracing Architecture: A 3rd Generation Intelligent Tutor for Algebra Symbolization, *International Journal of Artificial Intelligence in Education*, 18(2), 2008, pp 153-178.

[13] Koedinger, K.R., Aleven, V., Heffernan, N.T., Toward a Rapid Development Environment for Cognitive Tutors, *Proceedings of AIED 2003*, IOS Press, 2003, pp 455-457.

[14] Koedinger, K.R., Anderson, J.R., Hadley, W.J., Mark, M.A., Intelligent Tutoring Goes to School in the Big City, International Journal of Artificial Intelligence in Education, 8, 1997, pp 30-43.

[15] Matsuda, N., Cohen, W.W., Koedinger, K.R., Applying Programming by Demonstration in an Intelligent Authoring Tool for Cognitive Tutors, AAAI Workshop on Human Comprehensible Machine Learning, 2005, pp 1-8.

[16] Mayers, A., Lefebvre, B., Frasson, C., Miace: a Human Cognitive Architecture, *SIGCUE Outlook*, 27(2), 2001, pp 61-77.

[17] Mitrovic, A., Koedinger, K.R., Martin, B., A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling, *Proceedings of the 9th International Conference on User Modeling*, Springer-Verlag, 2003, pp 313-322.

[18] Mitrovic, A., Suraweera, P., Martin, B., Zakharov, K., Milik, N., Holland, J., Authoring Constraint-Based Tutors in ASPIRE, Proceedings of ITS 2006, Ikeda, M., Ashlay, K., Chan, T.-W. (Eds), Lecture Notes in Computer Science 4053, Springer-Verlag, 2006, pp 41-50.

[19] Murray, T., An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art, in *Authoring Tools for Advances Learning Environment*, Murray, T., Blessing, S., Ainsworth, S. (Eds), Kluwer Academic Publishers, 2003, pp 491-544.

[20] Najjar, M., Mayers, A., Fournier-Viger, P., A Novel Cognitive Computational Knowledge Environment, *The WSEAS Transactions on Advances in Engineering Education*, 3(4), 2006, pp 246-255.

[21] Orey, M.A., Interface Design for High Bandwidth Diagnosis: The Case of POSIT, *Educational Technology*, 30(9), 1990, pp 43-48.

[22] Orey, M.A., Burton, J.K., POSIT: Process Oriented Subtraction-Interface for Tutoring, Journal of Artificial Intelligence in Education, 1(2), 1990, pp 77-105.

[23] Razzaq, L., Patvarczki, J., Almeida, S.F., Vartak, M., Feng, M., Heffernan, N.T., Koedinger, K.R., The ASSISTment Builder: Supporting the Life Cycle of ITS Content Creation, *IEEE Transactions on Learning Technologies*, 2(2), 2009, pp 157-166.

[24] Resnick, L.B., Syntax and Semantics in Learning to Subtract, in *Addition and Subtraction: A Cognitive Perspective*, Moser, J. (Ed), Lawrence Erlbaum Associates Inc, 1982.

[25] Sleeman, D., Kelly, E.A., Martinak, R., Ward, R.D., Moore, J.L., Studies of Diagnosis and Remediation with High School Algebra Students, *Cognitive Science*, 13, 1989, pp 551-568.

[26] VanLehn, K., *Mind Bugs: The Origin of Procedural Misconceptions*, MIT Press, 1990.

[27] VanLehn, K., The Behavior of Tutoring Systems, International Journal of Artificial Intelligence in Education, 16(3), 2006, pp 227-265.

[28] VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M., The Andes Physics Tutoring System: Lessons Learned, International Journal of Artificial Intelligence in Education, 15(3), 2005, pp 1-47.

[29] Wenger, E., *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan Kaufmann Publishers, 1987.

Chapitre 2

Générer des interventions pédagogiques

L'article présenté dans ce chapitre est une vue d'ensemble de la recherche effectuée dans le but de permettre à Astus de générer des interventions pédagogiques. Il met en évidence les différentes caractéristiques du système de représentation des connaissances d'Astus qui permettent de rendre disponible l'information contenue dans le modèle de la tâche afin qu'elle puisse être utilisée lors de la génération d'interventions. L'article illustre, à l'aide d'exemples, l'impact de chacune de ces caractéristiques sur la génération d'indices par rapport à la prochaine étape. Il présente aussi la méthode qui a été développée afin de permettre à Astus de diagnostiquer les erreurs des apprenants à partir de leurs actions hors trace et montre comment Astus peut utiliser ce diagnostic afin de générer des rétroactions négatives. Finalement, l'article présente le résultat de cinq expérimentations qui ont été conçues dans le but de valider de façon empirique les interventions générées par Astus.

L'auteur (Luc Paquette) a contribué à 90 % de la charge de travail pour la rédaction de l'article. C'est lui qui a conçu et implémenté les méthodes et algorithmes permettant la génération des interventions pédagogiques. Il a aussi implémenté les deux tuteurs qui ont été utilisés lors des expérimentations. Il a conçu, exécuté et analysé les résultats pour les cinq expérimentations. Le système de représentation des connaissances utilisé par Astus a été conçu et implémenté par Jean-François Lebeau, mais l'auteur a participé à un grand nombre de discussions à ce sujet avec Jean-François.

A Method for the Generation of Pedagogical Interventions by a MTT Authoring Framework

Luc Paquette, Jean-François Lebeau, Gabriel Beaulieu, André Mayers,

Université de Sherbrooke, Sherbrooke, Québec, Canada,

luc.paquette@usherbrooke.ca

Abstract. Model-tracing tutor (MTT) authoring frameworks are great tools for producing MTTs with less effort, but the pedagogical interventions they produce are limited and usually require the inclusion of pedagogical knowledge, such as text message templates, in the model of the task. Enhanced ability to generate pedagogical interventions would be beneficial to MTT frameworks, as they would retain their advantage of reduced authoring effort while producing interventions closer to those of task-specific tutors. In this paper, we show how MTT frameworks can generate pedagogical interventions. This is achieved by defining features of a knowledge representation system that can be implemented in a framework to make the content of its task model explicit in the MTT it produces. We explain how the Astus framework implements these features and show how it allows the creation of MTTs that can generate pedagogical interventions such as next-step hints and negative feedback on error.

Keywords. Model-tracing, knowledge representation system, pedagogical intervention, next-step hints, negative feedback

INTRODUCTION

Model-tracing tutors (MTTs) have proven successful for the tutoring of well-defined tasks such as LISP programming [6], middle-school mathematics [13] and physics [32]. Their main distinguishing feature is their capability to follow learners on a step-by-step basis by tracing their actions against an executable model of the tutored task. This allows MTTs to provide pedagogical interventions such as flag feedback and next-step hints.

Research on MTTs has slowed in recent years as researchers have focused their efforts on defining or improving other paradigms for the creation of intelligent tutoring systems that address some of the limitations of MTTs. Constraint-based tutors [17] have obtained good results when modeling ill-defined tasks [19] such as writing SQL queries [16, 18]. The example-tracing approach [4] is an efficient way to produce tutors with similar behaviors to MTTs with less authoring effort [3]. Machine learning has been used to model ill-defined tasks such as logic proofs [7].

We believe there is still room to improve MTTs by giving them the capability to provide more sophisticated pedagogical interventions. Neil Heffernan [10] proposed to including pedagogical knowledge in MTTs in order to improve their interventions. Unfortunately, this approach increases the effort required to author an MTT, a factor that already limits the use of current MTT authoring frameworks.

The aim of our research is to determine the types of pedagogical interventions that can be provided by MTTs without increasing the effort required to author them. To achieve this objective, we first studied how an MTT can generate interventions using the content of its model of the tutored task; then we developed a knowledge representation system that would facilitate the generation of interventions by MTTs; and finally we implemented the system in Astus, an MTT authoring framework [21].

There have been multiple attempts in the past to enable intelligent tutoring systems to generate pedagogical interventions, but this approach has not been widely adopted by modern authoring frameworks. Rickel [27] proposed a framework (TOTS) that would allow intelligent tutoring systems for well-defined tasks to generate interventions by using a knowledge representation based on procedural networks [29]. Unfortunately, we found no examples of tutors and interventions generated by TOTS. Steve [28] uses a similar approach to generate explanations in an intelligent tutoring system that helps students learn to perform physical tasks such as operating complex machinery. Likewise, REACT [11], a trainer for operators of deep-space communication stations, uses a similar knowledge representation system to implement impasse-driven tutoring [12] that includes generated pedagogical interventions on errors. Finally, the GIL tutoring system [26] is able to generate interventions for LISP programming tasks, using a production system specially extended for that purpose.

In this paper, we first present a brief summary of production systems, a knowledge representation system that is usually associated with MTT authoring frameworks. Then, we examine how knowledge representation systems can be adapted for the generation of pedagogical interventions, and apply the resulting system to the generation of next-step hints in Astus. We go on to show how Astus can generate a second type of pedagogical intervention: negative feedback on errors. Finally, we present the results of experiments conducted to provide an initial evaluation of the interventions generated by Astus and reveal possible improvements.

PRODUCTION SYSTEMS

Production systems are classically used by MTT authoring frameworks to model the tutored task [1], for two main reasons. First, production rules can be considered as the base unit of procedural memory and can thus be used to model the procedural knowledge of an ideal learner. Classical MTTs make the hypothesis that such a model is efficient for tutoring [5]. Second, production rules are modular and expressive (in the practical sense). The author of a classical MTT can design a set of production rules, for which the execution produces a list of the steps[9] required to perform a task, without being constrained by the structure of the production rules themselves.

In production systems, the tutored task is modeled using two main structures: working memory elements (WMEs) and production rules. WMEs are declarative knowledge units describing the objects in working memory (WM) by specifying a set of attributes whose values are references to other WMEs or primitive data (numbers, strings, etc). Production rules are IF-THEN structures, where the IF part specifies conditions describing the WM state required to execute the rule and the THEN part specifies the rule's actions (modifications to WM or steps in the learning environment). Instances of the WME are used to describe the tutor's interpretation of the learner's mental representation of the task. This allows the tutor to interpret the learner's action by matching production rules against the content of WM to find a chain of rules that explains the executed step.

The practical expressivity of WMEs and production rules make them a powerful tool for modeling the tutored task, but it can be difficult for MTTs that use them to generate pedagogical interventions. Indeed, their expressivity makes it difficult for the tutor to analyze their content. This greatly limits how the tutor can use the content of the task model to produce pedagogical interventions. Although some interventions, such as flag feedback, can be achieved without requiring additional pedagogical knowledge, this is not always the case. For example, to provide next-step hints, classical MTTs require that specific hint templates be associated with each of the model's production rules. Figure 1 shows how a rule taken from CTAT's [3] fraction addition tutor[10] uses the "construct-message" keyword to include next-step hints in the model of the task.

[9] A step is an atomic action in the learning environment that modifies the state of the problem (VanLehn 2006).

[10] http://ctat.pact.cs.cmu.edu

```
(defrule same-denominators
  ?problem <- (problem (given-fractions ?f1 ?f2) (subgoals))
  ?f1 <- (fraction (denominator ?denom1))
  ?f2 <- (fraction (denominator ?denom2))
  ?denom1 <- (textField (value ?d &:(neq ?d nil)))
  ?denom2 <- (textField (value ?d))
=>
  (bind ?sub (assert (add-fractions-goal (fractions ?f1 ?f2))))
```

Figure 1 - Example of a production rule taken from CTAT's fraction addition tutor

KNOWLEDGE REPRESENTATION SYSTEM

To facilitate the generation of interventions in MTT authoring frameworks, we designed a knowledge representation system that models the teacher's instructions rather than the procedural knowledge of an ideal learner. This system was designed to extend the capabilities of classical MTTs by providing them with a model of the task that can be used to generate a wide range of pedagogical interventions.

In this section we present three features of a knowledge representation system designed to generate pedagogical interventions: a hierarchical procedural knowledge structure, explicit procedural knowledge units and semantically rich declarative knowledge units. For each feature, we describe the relevant aspects of production systems and explain how it is handled by Astus's knowledge representation system. Finally, using next-step hints as an example, we show how these features allow us to generate increasingly complete pedagogical interventions. The examples presented in this section are taken from MTTs for the insertion of elements into an AVL tree [23] and subtraction [21].

Hierarchical procedural knowledge structure

The hierarchical structure of procedural knowledge is used to locate specific knowledge during the global process of performing a task. When included in an MTT, this allows the MTT to intervene by providing the learner with information regarding his/her progress toward accomplishing the task.

When modeling a task for a classical MTT, the production rules are designed to be chained in a specific order that models the learner's cognitive processes. The result of the execution of these chains forms an implicit hierarchy that is determined by the content of the rules: the rules' application conditions are designed to match specific actions resulting from other rules. Since the chaining is not explicitly described in the rule syntax and the MTT cannot interpret the task-specific content of production rules, it is difficult for the MTT to infer the hierarchical structure of the model's procedural knowledge.

One way to make the hierarchical structure of the procedural knowledge explicit in production systems is to introduce the concept of goals. In such a system, the application condition of every rule must specify a goal that will be achieved by its execution, and the rule's actions can add goals to WM. Goals have been used in production systems such as CTAT's Cognitive Tutors to control rule chaining (Figure 1), but they are not required by the rule syntax (goals are standard WMEs). By making goals mandatory and including them in the rule syntax, they become an explicit part of the MTT, allowing it to analyze the hierarchical structure of the rules to produce a procedural graph of the interactions between the rules and the goals (Figure 2).

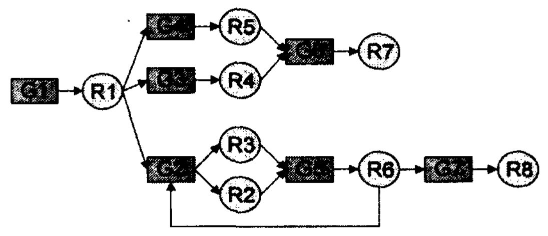


Figure 2 - Example of a procedural graph. Rules are marked as R1 to R8 and goals are marked as G1 to G7

Astus

Rather than using production rules as its base procedural knowledge unit, Astus's knowledge representation system uses goals and procedures organized in a graph similar to a procedural network [29]. A goal represents an intention behind the performance of a task, whereas a procedure represents a particular way to satisfy a goal. Because procedures are associated with a unique parent goal and because their syntax is designed to make explicit the sub-goals they instantiate, Astus can produce a procedural graph very similar to the one shown in Figure 2, but instead of rules (R1 to R8), Astus uses procedures (goals are depicted as rectangles and procedures as ovals).

Next-step hints

Using the information contained in its procedural graph, Astus can create hints identifying the sub-goals the learner must complete in order to achieve his/her current goal. To do this, we can define a hint template, such as "`In order to [parent goal], you need to [sub-goals]`", utilizing the information provided by the procedural graph. This template can then be instantiated for any procedure, using natural-language names assigned by the author to the relevant knowledge units.

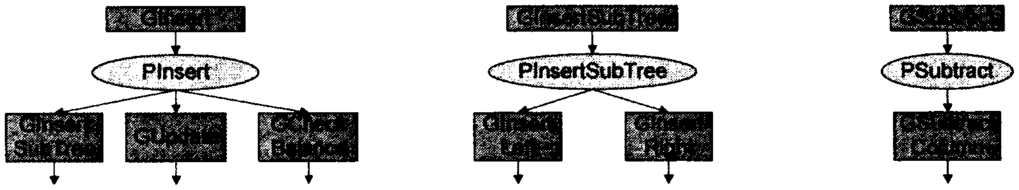


Figure 3 - The procedural graph of three procedures for which Astus can generate next-step hints

For example, the template can be instantiated for any of the procedures illustrated in Figure 3. The first procedure (*PInsert*) is taken from our tutor for the insertion of elements into an AVL tree. Three sub-goals are available for the execution of this procedure; thus the hint template would be instantiated as "`In order to` *`insert an element,`* `you need to` *`insert in a sub-tree,`* *`update`* `and` *`check for imbalances`*". The second procedure (*PInsertSubTree*) is executed to achieve one of the *PInsert* sub-goals and allows us to generate a similar hint using the same template: "`In order to` *`insert in a sub-tree,`* `you need to` *`insert to the left`* `and` *`insert to the right`*".

Finally, the third procedure is taken from our subtraction tutor and produces the hint "`In order to` *`subtract,`* `you need to` *`subtract a column`*".

The hints generated using only the information contained in the procedural graph are incomplete. Although they tell the learner which goals should be achieved, they offer no information about how to organize them. Is it necessary to achieve all of the goals? Is achieving one of them enough? Is the order of the goals relevant? Should the goal be achieved more than once? To answer these questions, the procedural knowledge units need to provide additional information regarding the organization of their subgoals.

Explicit procedural knowledge units

In production systems, the rule execution order is implicit in the content of the model's production rules.[11] In order for a rule to be fired, its application condition must match the content of WM. Either the condition matches the initial state of WM and the rule can be executed at the beginning of the task, or the execution of an available rule modifies the content of WM in a way that allows additional rules to match.[12] Thus, the rules are executed following a specific organization that is implicit in the content of their application conditions and actions.

To make the organization of the rules' execution explicit in the MTT, the knowledge representation system needs to allow the MTT to interpret the rules' application conditions and the effects of their execution on WM. This would allow the MTT to infer the rule organization by associating the outcomes of a rule's action and the application conditions of other rules. This information could then be used to generate pedagogical instructions.

Astus

In Astus, each of the procedures contained in the task model has a specific type that explicitly describes the organization of its sub-goals. When creating a model, the author specifies the

[11] The author can also specify static priorities that are used when two or more rules can be executed simultaneously.

[12] The WM effects of a rule can also prevent rules that were previously available from firing.

desired type for each procedure. This determines the template that will be used to define the procedure. The author is then required to fill in the template to describe the procedure's behavior and the organization of its sub-goals.

Procedures fall into two main categories: primitive and complex. Primitive procedures are perceptual motor skills and model steps. They are reified as interactions in the learning environment. As these procedures correspond to steps, they do not specify sub-goals. Complex procedures are scripts specifying sets of sub-goals whose execution is organized according to specific templates. The sub-goals of a complex procedure can be organized according to different types of scripts: a sequence of sub-goals, the selection of a sub-goal or the repetition of a sub-goal.

Formalizing the procedural knowledge by the use of complex procedures that describe the organization of their sub-goals is analogous to the way procedural knowledge is treated in VanLehn's Sierra theory [30]. According to this theory, learners use three main control structures when performing tasks: AND, OR and FOR-EACH goals. Astus's complex procedures extend these basic structures to offer unordered, partially ordered and totally ordered sequences, for-each repetitions (over a sequence of objects), conditional selections and repetitions (loops) and nondeterministic selections (over a set of objects).

Next-step hints

Whereas the hints generated using only the information contained in the procedural graph all employed the same template, it is now possible to provide hints that refer to the organization of a procedure's sub-goals. Since each procedure has a specific type whose structure is known to Astus, it can interpret the procedure's content to produce its pedagogical interventions. Hence, the next-step hints generated by Astus can be improved by using an appropriate template, based on the type of procedure the hint refers to.

Using the additional information available from the procedure type, we can improve the hints generated for the procedures presented in Figure 3. For example, when provided with the additional knowledge that the procedure *PInsert* is a totally ordered sequence, the hint can indicate that all of the procedure's sub-goals must be completed in the correct order:

```
In order to insert an element, you need to do the fol-
lowing in the correct order:
1) insert in a sub-tree
2) update
3) check for imbalances
```

Similarly, the hint for *PInsertSubTree* can be improved using the knowledge that the procedure's behavior is a selection of a sub-goal:

```
In order to insert in a sub-tree, you need to either
insert to the left or insert to the right.
```

Finally, *PSubtract* is the repetition of a single sub-goal:

```
In order to subtract, you need to repeatedly subtract a
column.
```

Although these hints provide instructions regarding the sub-goals' organization, there is still room for improvement. For example, when providing a hint for a selection, the MTT should be able to explain when each sub-goal should be selected; when generating a hint regarding a repetition, the MTT should be able to indicate how many times the sub-goal should be repeated. In order to include this information in the hints it generates, the MTT must be able to interpret the content of the procedural knowledge units in more detail. This requires that the MTT be able to interpret how procedural knowledge units access and manipulate the content of WM.

Semantically rich declarative knowledge units

In order to provide instructions about how to execute specific procedural knowledge units, the MTT needs to be able to interpret their full content, not just their hierarchical structure and the organization of their execution. For production rules, this implies that the MTT needs to be able to analyze the application condition of a rule to determine which objects from WM are useful to the rule, and how the rule manipulates those objects.

To manipulate objects from WM, production systems offer one type of declarative knowledge unit: working memory elements (WMEs). WMEs are practically expressive, as a WME is composed of a set of untyped attributes, but their structure is not a rich source of information regarding their usage. Since the attributes are not typed, the MTT cannot analyze the links between the different types of WMEs. Thus, it is not possible to distinguish the role of a particular WME to determine whether it models the task itself, a mental calculation, a mental representation of the learning environment or a goal.

Astus

Three main features have been added to Astus's knowledge representation system in order to extend the classical representation of declarative knowledge in MTTs. First, declarative knowledge units can be of three types: concepts, relations and functions. Second, the manipulation of declarative knowledge units is restricted to a small number of task-independent operators whose semantics is known to the MTT. Third, task-dependent manipulations of declarative knowledge units are achieved through a fixed interface that the MTT can interpret. Since the semantics of these features is known to the MTT, it has the capability to include them in the pedagogical interventions it generates.

The first type of declarative knowledge unit defined in Astus is a concept: a pedagogically relevant abstraction used to model the task's objects. The concept is defined by a set of features that are essential to the description of the object. Each feature has a type that is either an atomic value (numbers, symbols, Booleans) or a reference to another concept. For example, in our MTT for the insertion of elements into an AVL tree, the concept "Node" has only one feature: the node's content (a number).

Astus allows inheritance between concepts in order to further specify the type of an object. For example, the "Node" concept can be specialized as a "BinaryNode" which adds two new features to the object: a left and a right pointer to the node's sub-trees. Likewise, the "AVLNode" concept is a specialization of a "BinaryNode" containing a balance factor. Concept inheritance can be either asserted directly or dynamically inferred from the values of the object's features. For example, a "BinaryNode" can be classified as a "Leaf" when both of its pointers are null.

In addition to concepts, Astus's knowledge representation system allows authors to define relations and functions to model links between objects. A function is defined by a list of arguments and an image (these variables can refer to atomic values or concepts). For example, in our AVL MTT, the function "parentOf" has one argument, a node, and returns its parent node as an image. Similarly, relations are defined by lists of places. For example, the relation "childOf" identifies whether a specific node is a child of another one. This relation has two places, two nodes, and is instantiated if the value of the first place is a child of the value of the second one. As for concept inheritance, functions and relations can be either asserted or inferred.

In order for procedures to interact with the content of WM, Astus provides task-independent operators and an interface for task-dependent manipulations. The task-independent operators are Boolean operators to express conditions and navigation operators to access the features of an object. For example, the navigation operator "→" can be used to obtain the value of a feature called "content" from an instance of the concept "Node" ("node → content"). Boolean operators can be used to check whether an object is an instance of a specific concept (*isA*), whether two variables refer to the same object (*same*), whether two or more objects are linked by a given function or relation (*exists*), and also to check the order of two objects (*greater*, *lesser*). These operators can be combined using logical operators (*and*, *or*, *not*).

Task-dependent manipulations of WM are achieved through a query interface specified by Astus's knowledge representation system. This interface allows procedures to specify the information they want to retrieve from WM. In particular, it can specify the type of declarative knowledge to retrieve by using the name of the concept, relation or function. Additionally, the query interface specifies whether all instances of the desired knowledge units are to be retrieved or whether a unique instance is sought.

For example, the query interface allows a procedure to retrieve all instances of the concept "Leaf" by using the instruction "`all(Leaf)`". Likewise, the query "`unique(Leaf)`" retrieves an instance of the concept "Leaf" and ensures that it is the only instance of this concept in the WM. Queries can also be used to retrieve instances via relations and functions. For functions, the query will retrieve the image of a function's instance. For example, "`unique(parentOf, [node])`" will retrieve the parent (image of the function) for the argument node. A relation query will return the objects associated with the free place (the one that is not constrained by the query). For example, "`all(childOf, [_, parentNode]`" will return all the children of a node, whereas "`unique(childOf, [childNode, _])`" will return the unique parent of a node. It is also possible to filter the results of a query according to a logical condition, using the keyword "`where`". For example, the query "`all(Node, where{ not(same($e → leftPointer, nullPointer)) })`" will retrieve a set containing all of the instances of the concept "Node" and will then filter the result to find the ones for which the feature "leftPointer" is not the same as the variable "nullPointer".[13] The result of the query will thus be all of the nodes that have a child to the left.

Astus's query interface allows it to interpret how procedures access and manipulate the content of WM, even though it does not have the capability to interpret how the requested information is produced. For example, Astus can access the content of a query to explain to the learner that he/she needs to retrieve all the nodes that are leaves (task-independent access), but it cannot explain why a specific node is a leaf, since this classification is achieved through task-specific processes.

[13] The "`where`" clause is evaluated for each node retrieved by the query, with the variable "$e" taking the value of specific nodes.

Next-step hints

Astus can improve the interventions it generates by providing information about how procedures access WM and how they manipulate the retrieved objects. In particular, the hints generated can specify the objects that should be passed as arguments for each sub-goal of a procedure and detail the conditions defining the procedure's behavior.

With this information, we can provide additional instructions when generating next-step hints. Until now, we have only used the procedural graph and the procedure type. By using all the information contained in a procedure's script, Astus is able to generate more complete hints. First, we can look at the script for the *PInsert* procedure and its parent goal *GInsert*:[14]

```
Goal 'GInsert' eng-name 'insert an element' {
    param 'tree' type 'Tree' eng-name 'tree'
    param 'element' type 'int' eng-name 'element to
      insert'
}

Fully ordered sequence 'PInsert' achieves 'GInsert' {
    goal 'GInsertSubTree' using 'tree → root',
      'element'
    goal 'GUpdate' with 'tree'
    goal 'GCheckBalance' using 'tree → root'
}
```

This procedure makes very limited use of Astus's interface for the manipulation of the WM objects. The only manipulation is to access the root of the tree in which the insertion is taking place. For this procedure, Astus can improve the generated hint by indicating which objects should be used as arguments for the sub-goals:

[14] The goal's script shows an example of how the textual names used by the MTT to generate its messages are encoded by the author as part of the goals, concepts, functions and relations, using the keyword "eng-name".

In order to *insert an element*, you need to do the following in the correct order:
1) *insert in a sub-tree* using the *root* of the *tree* and the *element to insert*
2) *update* using the tree
3) *check for imbalances* using the *root* of the *tree*

For a conditional procedure such as *PInsertSubTree*, being able to interpret the access to WM allows Astus to detail the conditions used to decide which sub-goals should be executed. These conditions are defined in the procedure's script:

```
Conditional 'PInsertSubTree' achieves 'GInsertSubTree'
{
    if lesser('element', 'node → content')
        goal 'GInsertLeft' using 'node', 'element'
    if greater('element', 'node → content')
        goal 'GinsertRight' using 'node', 'element'
}
```

By analyzing the content of these conditions, the MTT can include them in its hints:

In order to *insert in a sub-tree*, you need to either *insert to the left*, if the *element to insert* is less than the *content* of the *node*, or *insert to the right*, if the *element to insert* is greater than the *content* of the *node*.

Finally, the hint generated for *PSubtract* can also be improved using the information contained in the procedure's script:

```
Ordered For-Each 'PSubtract' achieves 'GSubtract' {
    iterator 'column' from 'task → columns'
```

```
        goal 'GSubtractColumn' with 'column'
    }
```

In order to *subtract*, you need to *subtract a column* for each of the *columns* of the *task*.

The previous examples make use of all the information available in the procedure's scripts to generate next-step hints. In order to continue improving the interventions generated by Astus, it needs to have access to information regarding the execution context for specific instances of the procedures. We must thus define how Astus traces the learners' steps and ensure that the tracing process can be easily interpreted.

TRACING IN ASTUS

When an MTT created with Astus is used to perform a task, the procedural knowledge contained in the procedural graph is instantiated to produce an episodic tree (Figure 4), a task tree dynamically generated according to the state of WM. To achieve this, starting from a task's main goal, the procedure associated with this goal is executed. Its queries and conditions are evaluated to determine the behavior of its script and its sub-goals are instantiated. According to the script, the sub-goals can be instantiated as currently executing (E) or waiting (W). The same process is applied recursively for each of the new executing goals.

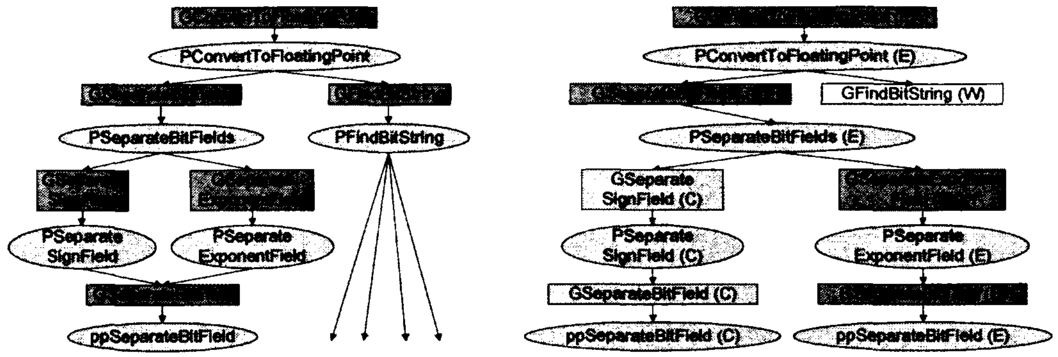


Figure 4 - Procedural graph from a floating-point number conversion MTT (left) and its instantiation as an episodic tree (right). Units marked (E) are currently executing, (W) are waiting and (C) are completed

For example, when executing a sequence procedure, depending on its order constraints, all its sub-goals might be instantiated in the executing state or some might be instantiated as waiting (*PConvertToFloatingPoint* in Figure 4). In the case of conditional procedures, the execution of the procedure will instantiate only one sub-goal, depending on the evaluation of the conditions contained in the procedure's script (Figure 5a). As a final example, a for-each procedure will evaluate the sequence of objects on which its sub-goal should be repeated and will instantiate one sub-goal for each of the objects from the sequence. Depending on whether or not the order is important for the iteration's execution, the episodic tree might contain only one executing goal, with the other marked as waiting (Figure 5b).

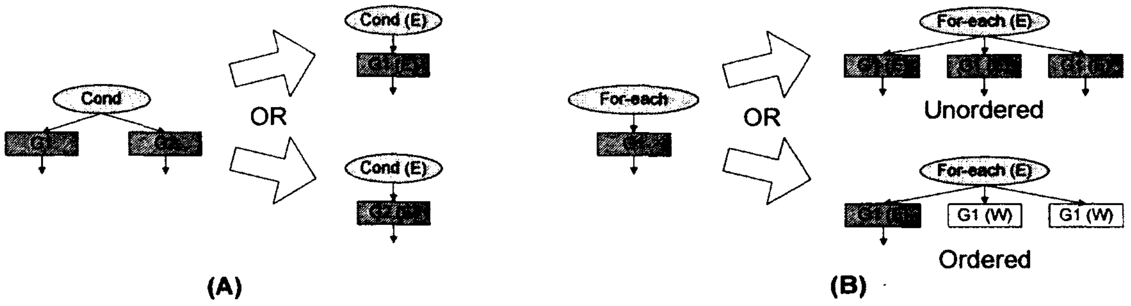


Figure 5 - Instantiation in an episodic tree of a conditional procedure (A) and a for-each procedure (B)

Astus uses the episodic tree to trace the learner's steps. In this tree, the leaves are either primitive procedures or goals waiting to be expanded in the future. When the learner performs a step, it is compared to each of the tree's primitive procedures to find a match.[15] If no match is found, the step is marked as off-path and is considered erroneous. If a match is found, the step will either be attributed to a known error, if one of its parent procedures is marked as erroneous, or be considered a correct step. When the step is considered correct, the state of the primitive procedure associated with it is changed to completed (C) and the tree is updated accordingly.

Starting from the newly completed primitive procedure (one of the tree's leaves), its parent goal is also marked as completed. Then, the individual goals and procedures are updated, going upward through the tree towards the root. A goal is marked as completed if its child procedure is completed. Procedures are updated according to their scripts. A conditional procedure is marked as completed once the selected sub-goal is completed. For sequences, sub-goals marked as waiting are executed if their order constraints are satisfied and the procedure is completed when all of the sub-goals are completed. For-each procedures have a similar behavior to sequences. A loop procedure is marked as completed when its iteration condition is met, otherwise the script creates a new instance of the sub-goal. A task is considered completed once the tree's root goal is completed.

Next-step hints

Using the content of the episodic tree allows Astus to contextualize the hints it generates according to the current state of the task. This includes taking into account the results of conditional expressions when selecting a sub-goal and adapting hints for procedures that are partially completed.

[15] If more than one match is available for a step, this will cause an ambiguity that will be resolved in the following steps.

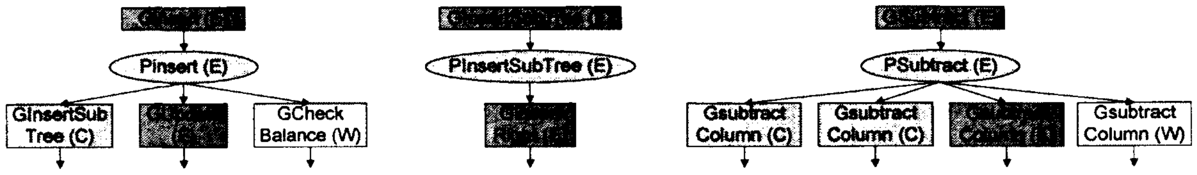


Figure 6 - Episodic trees associated with the three generated hints

For a partially completed sequence procedure such as *PInsert*, the generated hint can be modified to focus on the next goal the learner should accomplish. If the first sub-goal (*GInsertSubTree*) has already been successfully achieved (Figure 6), the generated hint can notify the learner and explain what his/her next objective should be:

> You are currently doing *insert an element*, you already did *insert in a sub tree*, your next objective should be to *update* using the *tree*.

For a conditional procedure (*GInsertSubTree*), only one of its sub-goals is instantiated in the episodic tree (Figure 6). Thus, it is not necessary for the generated hint to list all the possible conditions. The hint can focus the learner's attention on the currently executing goal:

> In order to *insert in a sub-tree*, you need to *insert to the right* since the *element to insert* is greater than the *content* of the *node*.

Finally, for the procedure *PSubtract*, a for-each procedure, the MTT can offer additional information concerning the number of sub-goals left to complete:

> In order to *subtract*, you need to *subtract a column* for each of the *columns* of the *task*. You have already completed this for the first 2 columns. You must repeat the process for the 2 remaining ones.[16]

[16] Interface highlights can be used as a supplement to the help message to visually indicate the remaining columns.

The examples we provided to illustrate the generation of next-step hints are a small subset of the templates that can be used to generate hints. The instructions given for any of the procedures can be adjusted to specific learning situations. One hint might provide information regarding the procedure's queries whereas another does not, or a hint may or may not specify the sub-goals' arguments. Hints may be specific to the current state of the task or more abstract. The choice of the template's content depends on the pedagogical strategy applied by the MTT.

The hints provided by Astus are currently generated using task-independent templates filled with task-dependent content extracted from the model of the task. This method limits the readability of the generated hints. One step towards improving them would be to use natural language generation techniques.

NEGATIVE FEEDBACK

The same features that allowed the generation of next-step hints can also be applied to additional types of interventions. The second type of intervention we implemented in Astus is negative feedback on error. Drawing inspiration from the Sierra theory [30], we designed a method by which Astus can diagnose errors from the learner's off-path steps in order to provide negative feedback.

Sierra

Sierra is a theory to explain the origin of learners' procedural errors [30]. It proposes plausible cognitive processes that could lead learners to execute incorrect steps. According to this theory, errors can be observed when learners face impasses – situations in which their current knowledge of the task is insufficient to perform it – and try to repair them. The combination of an impasse and a repair strategy determines the learner's erroneous behavior.

Three types of repair strategy are proposed: no-op, back-up and barge-on. When the no-op strategy is applied, the learner ignores the goal he/she does not know how to accomplish. The back-up strategy is very similar to no-op, but, instead of simply ignoring his/her current goal, the learner returns to a previously unfinished goal and resumes from that point. Finally, when the barge-on strategy is applied, the learner modifies his/her procedural knowledge in order to resolve the impasse.

The Sierra theory has been validated by creating a computational model of learning that includes the impasse and repair processes. This model was applied to the subtraction task and has successfully modeled the acquisition of multiple errors observed in learners' behavior. Despite this success, little effort has been made to incorporate elements of the Sierra theory into tutoring. Applying this theory in Astus would improve the resulting MTTs by allowing them to diagnose many of the learner's errors without the need for erroneous knowledge units, thus reducing the effort required to provide negative feedback on error similar to that offered by REACT's impasse-driven tutoring [12].

Error diagnosis

Taking our inspiration from Sierra, we designed a method that allows Astus to diagnose many of the learner's errors. We have shown in a previous paper [22] that Astus's knowledge representation system is compatible with the assumptions formulated in Sierra and that Astus can automatically disrupt procedural knowledge units to model erroneous behaviors analogous to those resulting from Sierra's impasse and repair process. In this section, we show how our Sierra-inspired method is used to diagnose learners' errors.

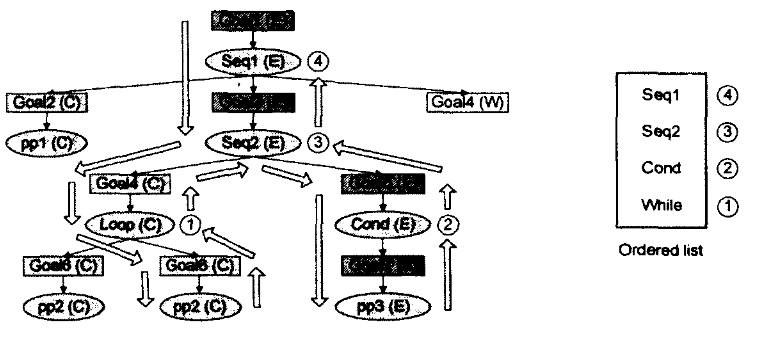


Figure 7 - Construction of an ordered list of all the procedures that might have been the source of the learner's impasse. The next correct step is *pp3*; *pp1* and both instances of *pp2* have already been completed.

In Astus, the episodic tree contains all the steps that are predicted by the MTT's executable model of the task. When an off-path step is executed, Astus attempts to diagnose the learner's error by manipulating the content of the episodic tree to try to instantiate a step that matches the learner's off-path step. To generate a diagnosis, the MTT starts by searching the tree to identify all the complex procedures that might be the source of the learner's impasse. The result is an ordered list of procedures, with the first ones being those closest to the steps contained in the tree. Figure 7 illustrates the process of constructing this list. A depth-first search, exploring the branch that explains the learner's last step and the currently executing branches, is carried out, starting from the tree's root (*Goal1*). The branch that explains the last step (the second instance of *pp2*) is searched first, as its incorrect execution might explain errors in which the learner considers a procedure, such as *Loop*, as not completed even though in fact it is. Steps completed prior to that one are ignored. The complex procedures encountered during the search are added to the ordered list after their sub-goals have been completely searched.

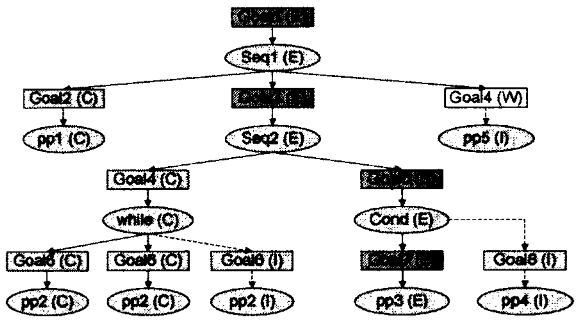
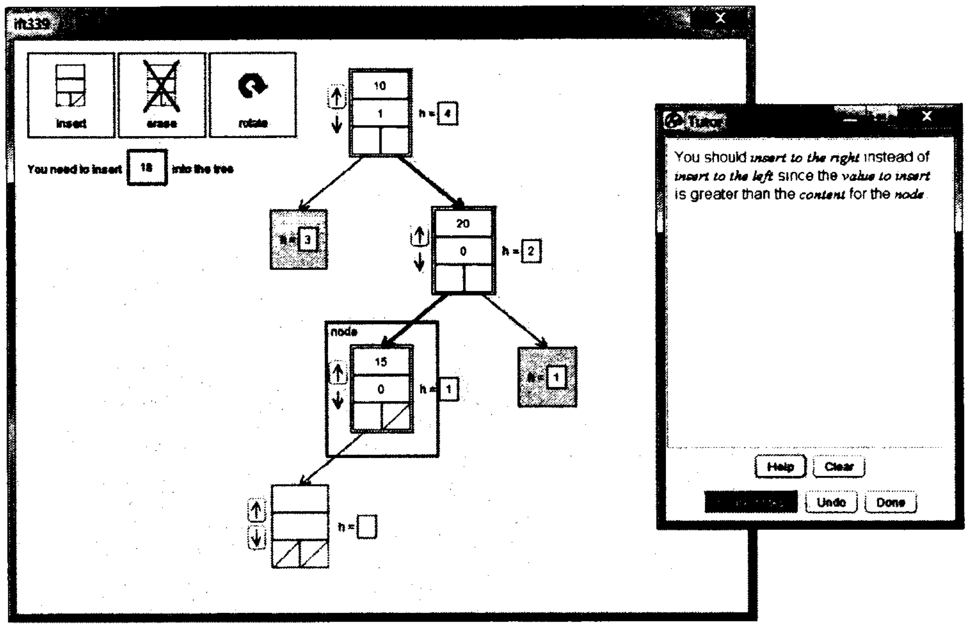


Figure 8 - Result of the interpolation. The interpolated goals are marked with an (I).

For each procedure identified by this search, Astus interpolates the steps resulting from its incorrect execution by examining how it can be incorrectly executed. Figure 8 shows examples of interpolation applied to the episodic tree from Figure 7. First, the learner might use a *barge-on* repair to modify the condition of the *Loop* procedure, thus repeating its sub-goal

(*Goal6*). Likewise, he/she might modify the conditional procedure's (*Cond*) conditions (*barge-on*) and achieve the wrong sub-goal (*Goal8*). Finally, he/she might use the *back-up* repair to avoid having to complete a current goal (*Goal3*) and instead try to achieve a sub-goal (*Goal4*) that is still waiting for the completion of a previous one.

When interpolating procedures, the MTT can also apply a process similar to the *barge-on* repair strategy to modify how the procedures access WM. When querying WM to access an instance of a function or a relation, the interpolation process can replace the accessed relation or function by a similar one (one that is defined over the same concepts). In our subtraction MTT, this can cause the function *differenceOf*, which takes two integers as its arguments and has one integer as its image, to be replaced by the similar function *sumOf*. Likewise, the interpolation can swap two arguments that are of the same type. For example, in our AVL MTT, the function *heightDifference* computes the difference in height for two sub-trees. When applying the argument swap interpolation, Astus will inverse the two sub-trees to produce behaviors such as $5 - 6 = -1$ instead of $6 - 5 = 1$ (with 5 and 6 being the value of the heights for the two sub-trees passed as arguments). Astus can also apply a *barge-on* repair when using the access operator ($\rightarrow$) and access the wrong feature of a concept. This can only be applied if both the original feature and the incorrect one are references to objects of the same type. In our AVL MTT, this repair can have the effect of accessing a node's left pointer (node $\rightarrow$ leftPointer) instead of its right pointer (node $\rightarrow$ rightPointer).



You need to insert 18 into the tree

You should *insert to the right* instead of *insert to the left* since the *value to insert* is greater than the *content* for the *node*.

Figure 9 - Our MTT for the insertion of elements into an AVL tree. The learner has performed an off-path step that was diagnosed by the MTT.

Astus uses the set of all interpolated steps to try to find a match for the learner's off-path step. If such a match is found, the branch of the episodic tree containing the interpolated step is used as a diagnosis of the learner's error. It is possible that multiple interpolated steps match the learner's off-path step. In our current implementation, such an ambiguity is resolved by using the simplest interpolation (the one with the least repairs) as the diagnosis, but it would be possible to use additional information, such as the learner model, to resolve the ambiguity. Once a diagnosis has been produced, the MTT can use it to react to the learner's step.

Feedback generation

When Astus diagnoses an off-path step, it can generate negative feedback. To achieve this behavior, we use Astus's capability to generate interventions by examining the content of the task model. In this section, we give examples of the negative feedback generation taken from our MTT for the insertion of elements in an AVL tree (Figure 9).

Figure 9 illustrates a task where the value 18 needs to be inserted into an existing AVL tree. The learner has reached the node containing the value 15 and must decide on which side of this node to continue the insertion process. Figure 10 shows part of the episodic tree for this specific state of the task. The procedure *PInsertSubTree* is a conditional procedure that determines whether the new value should be inserted to the left or to the right of the current node. As the value 18 is greater than 15, the goal *GInsertRight* is instantiated by *PInsertSubTree* and the learner has to insert to the right of the current node.

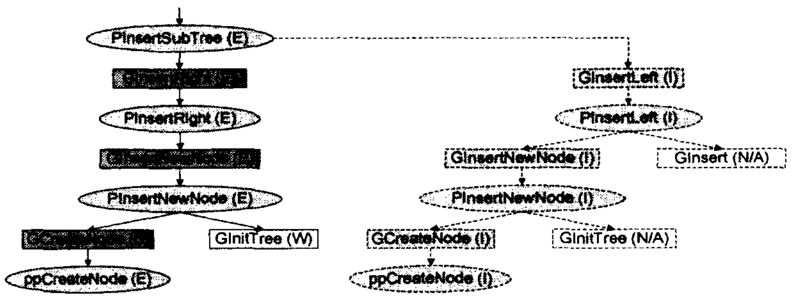


Figure 10 - Part of the episodic tree for the insertion of 18 and the interpolated branch for the off-path diagnosis.

The next step for the learner is to create a new node (*ppCreateNode*) to the right of the current one. If he/she executes the similar step of creating a new node to the left, Astus will try to diagnose this off-path step by interpolating new branches in the episodic tree. Starting from the procedure *PInsertSubTree* (Figure 10), Astus will instantiate the goal *GInsertLeft* to interpolate the behavior of incorrectly executing *PInsertSubTree*. This interpolation will lead to an instance of the primitive procedure *ppCreateNode* that corresponds to the learner's step. Having found a match for the off-path step, Astus will use this interpolation as the diagnosis for the learner's error.

To produce feedback on errors, Astus associates a feedback template with every type of error that it can diagnose. In the above example, the source of the errors is a *barge-on* repair executed on the conditional procedure *PInsertSubTree*. As the learner did not fully understand when to insert to the left or to the right, he/she incorrectly chose to insert 18 to the left of the current node (*GInsertLeft*). Using this diagnosis, Astus can provide feedback by instantiating the corresponding template:

```
You should [correct sub-goal name] instead of [used
sub-goal name] since [condition for the correct goal].
```

Using the content taken from the *PInsertSubTree*'s script (page 65), the negative feedback can be instantiated (Figure 9 shows how this feedback is communicated to the learner):

> `You should` *`insert to the right`* `rather than` *`insert to the left`* `since the` *`element to insert`* `is greater than the` *`content`* `for` *`node`*`.`

A second example of an error can be observed in our AVL MTT when the learner calculates a node's balance factor. This is achieved by subtracting the height of the right sub-tree from the height of the left sub-tree (left height - right height). A common error occurs when the learner does not remember which height to subtract from the other and performs the opposite subtraction (right height - left height).

When interpolating the episodic tree to diagnose this error, Astus will encounter the procedure *PUpdateBalanceFactor*. This procedure queries WM for an instance of the function *heightDifference* that takes two pointers to sub-trees and returns the difference of their heights. While interpolating, Astus will try to incorrectly execute this query to identify the erroneous behaviors it would cause. One way to incorrectly execute the query is to invert the function's two arguments. This will result in the error of subtracting right height - left height. Using this diagnosis, Astus can instantiate a feedback template:

```
While trying to [goal name], you have inverted the
[pair of arguments] for the [function name]. You
should have used [correct arguments].
```

The instantiation of this template will result in the following feedback:

```
While trying to *update the balance factor*, you have
inverted the *first term* and the *second term* for the
*height difference*. You should have used the *left
pointer* as the *first term* and the *right pointer* as the
*second term*.
```

Astus can adapt the templates used to generate its negative feedback according to the desired pedagogical strategy. An experiment by McKendree [14] evaluated the effectiveness of goal-oriented feedback on error and feedback explaining the reason for an error. According to this experiment, goal-oriented feedback is the most useful for the learner as it helps him/her correct his/her mistake and it also seems to have beneficial effects on subsequent encounters with the problematic knowledge. It can be combined with feedback explaining the reason for the error, a type of feedback that can improve later performance, but does not indicate how to correct the error. The feedback we provide uses a combination of pointing to the correct goal and explaining the cause of the error.

The format we chose is also advantageous in situations where Astus's diagnosis does not accurately identify the source of the learner's error. In those situations, we expect learners will still benefit from the MTT's interventions, since Astus provides goal-oriented feedback that not only explains the error, but also indicates what should be done next.

EMPIRICAL RESULTS

We have shown how Astus can use its knowledge representation system to generate pedagogical interventions and given examples of two types of such interventions: next-step hints and negative feedback on error. In this section, we present the results of five experiments conducted with students from the computer science department at the Université de Sherbrooke. The purpose of the experiments was to obtain an initial assessment of the effectiveness of the generated interventions.

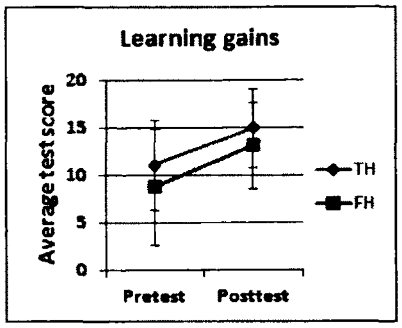
Next-step hints

We evaluated the next-step hints generated by Astus in the context of a floating-point number conversion MTT that was used in a system programming course. This section presents a summary of three experiments that are described in more detail elsewhere [25, 24].

The objective of our first experiment [25] was to compare learning gains and student assessment for next-step hints generated by Astus to those for hints authored by a teacher. In this experiment, 34 students were divided into two groups: 19 students received teacher-authored hints[17] (TH) and 15 received framework-generated hints (FH). The statistical analyses of the results (illustrated in Figure 11 and summarized in Table 1) did not show a significant difference in learning gains between the two conditions.

Table 1 contains the results of four statistical tests. First, we used a two-sample t-test to compare the pretest scores for the two conditions. Then, we used paired t-tests to evaluate the learning gains between the pretest and the posttest for both conditions. Finally, we used an ANCOVA to evaluate the difference in the posttest scores between the two conditions, using the pretest scores as a covariate. Figure 11 (left) shows the similar learning gains (slope) between pretest and posttest for the TH and FH conditions. Figure 11 (right) shows the students' subjective assessment of hints from four different types of procedures: loop, conditional, sequence with more than one sub-goal (sequence N) and sequence with only one sub-goal (sequence 1). A score of 0 indicates a strong preference for the generated hints and a score of 4 indicates a strong preference for those authored by a teacher.

[17] The teacher-authored hints were encoded as text templates similar to those used by Cognitive Tutors.



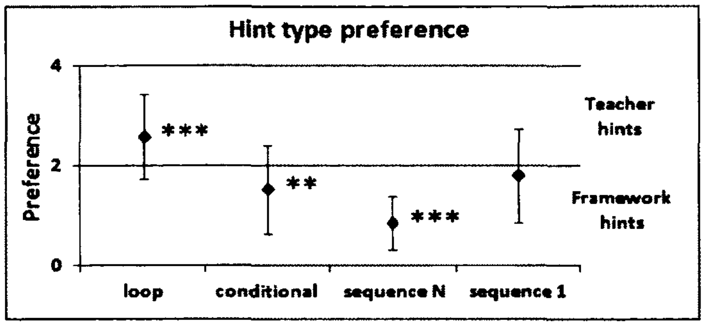


Figure 11 - The results of our first next-step hint experiment. The '*' character indicates statistical significance (** for $p < 0.01$ and *** for $p < 0.001$).

Table 1 - Summary of the statistical analysis for our first next-step hint experiment.

	Stat	p	Effect size	Power
Pretest scores	$t(32) = -1.258$	0.218	$d = 0.43$	22.67%
Learning gain (NH)	$t(14) = -3.485$	0.004**	$d = 0.79$	89.65%
Learning gain (WH)	$t(18) = -4.926$	< 0.001***	$d = 0.86$	97.49%
ANCOVA	$F(1, 31) = 0.234$	0.632	$\eta^2_p = 0.0075$	7.56%

Our first experiment did not find any significant difference between framework-generated and teacher-authored hints, but does not validate that the students' learning gains can be attributed to the hints they received. The observed gains could simply be caused by the activity of performing the task using an MTT. To determine whether next-step hints were helpful, we conducted a second experiment [24] comparing the learning gains of an MTT without next-step hints (flag feedback only) to those of an MTT that also provided framework-generated hints. A group of 32 students was divided into two sub-groups: 16 students used an MTT that provided no hints (NH) and 16 students used an MTT with framework-generated hints (WH). The results of this experiment are summarized in Table 2 and shown in Figure 12 (left).

Table 2 - Summary of the statistical analysis for our second next-step hint experiment.

	Stat	p	Effect size	Power
Pretest scores	$t(23.629) = 0.576$	0.570	$d = 0.20$	8.50%
Learning gain (NH)	$t(15) = 6.213$	< 0.001***	$d = 1.35$	99.89%
Learning gain (WH)	$t(15) = 5.550$	< 0.001***	$d = 1.37$	99.91%
ANCOVA	$F(1, 29) = 3.057$	0.046*	$\eta^2_p = 0.091$	39.40%

To further validate the results of our second experiment, we conducted a third one using the same two conditions with 33 learners: 16 for the NH condition and 17 for the WH one. The results of our third experiment are summarized in Table 3 and shown in Figure 12 (right).

Table 3 - Summary of the statistical analysis for our third next-step hint experiment.

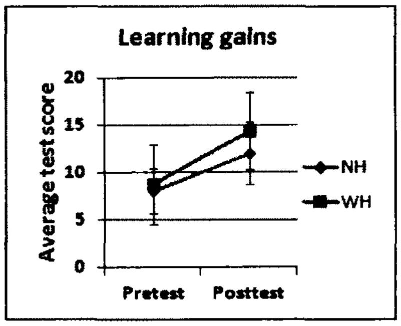
	Stat	p	Effect size	Power
Pretest scores	$t(31) = 0.318$	0.753	$d = 0.11$	61.00%
Learning gain (NH)	$t(15) = 2.970$	0.010**	$d = 0.52$	49.46%
Learning gain (WH)	$t(16) = 4.401$	< 0.001***	$d = 0.77$	86.64%
ANCOVA	$F(1, 30) = 2.818$	0.052	$\eta^2_p = 0.086$	36.40%

Discussion

The results from our first experiment suggest that, for our floating-point conversion MTT, the nature of the next-step hints (framework-generated or teacher-authored) did not have a significant impact on learning gains. This is supported by the fact that both conditions had significantly higher posttest scores, with similar effect sizes ($d = 0.79$ and $d = 0.86$). Additionally, the ANCOVA comparing the posttest scores of the two conditions, using the pretest scores as a covariate, did not show a significant difference and had a very low effect size (η^2_p =

0.0075). The similar slopes for the two conditions in the graphical visualization of the students' scores also suggest similar learning gains (left of Figure 11).

The second experiment was conducted to ensure that the use of next-step hints had a significant effect on learning gains and that the learning gains observed in our first experiment were not merely a consequence of using an MTT to perform the task. Both conditions had significant learning gains, with very similar effect size ($d = 1.35$ and $d = 1.37$). Although the mean gain between pretest and posttest was higher for the WH condition ($M = 5.63$) than for the NH condition ($M = 3.93$), the similar effect size might be due to the higher standard deviation for WH's pretest scores ($SD = 3.30$) than for NH's pretest scores ($SD = 2.34$). The effect size for WH would have been higher if its standard deviation had been closer to that of the NH condition. The result of the ANCOVA was significant, with an effect size of $\eta^2_p = 0.091$, which seems to indicate that the framework-generated hints provided to the students did improve learning gains. Finally, the steeper slope for the WH condition in Figure 12 (left) also suggests that the next-step hints were beneficial.



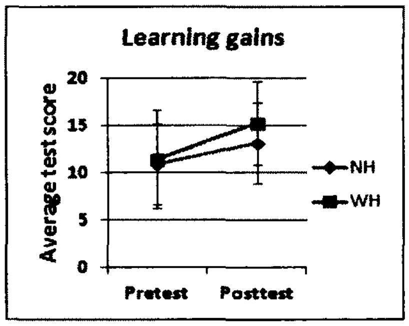


Figure 12 - The results of our second (left) and third (right) next-step hint experiments

The results of our second experiment suggest that the framework-generated hints had a positive impact on learning gains, but its statistical power was low. We therefore conducted a third experiment to try to reproduce similar results. Both the NH and the WH conditions showed significant learning gains, although the higher effect size for WH ($d = 0.77$) seems to indicate that the hints had a positive impact on the students' learning gains. The ANCOVA was not statistically significant, but its effect size ($\eta^2_p = 0.086$) was very similar to that obtained in our second experiment ($\eta^2_p = 0.091$) and the results of the test were close to a statistically significant difference ($p = 0.052$). These two facts suggest that a more powerful test could have found a significant difference. Finally, the steeper slope for the WH condition in Figure 12 (right) suggests greater learning gains when receiving next-step hints.

Overall, the results of our experiments suggest that our framework-generated hints have a positive impact on learning gains that is similar to that of teacher-authored hints. Additional experiments could improve our empirical validation by reproducing similar results for different tasks. Such results would suggest that the efficiency of our generated hints can be generalized to multiple types of task of different complexity. It would also be very helpful to use learners with no background in computer science. In our experiments, the learners were computer science students with a facility for understanding computer-generated hints. Experiments conducted with learners of more varied backgrounds would be interesting, as they would allow us to better evaluate the impact of hint readability on learning gains.

Negative feedback

We conducted two experiments in a data structure course, using an MTT for inserting elements into an AVL tree (Figure 9). The objectives of the first experiment were to evaluate whether the diagnoses produced by our MTT were accurate and whether the negative feedback produced by the MTT helped the learner. The second one was designed to analyze logs of the students' interactions with the MTT.

In the first experiment, our MTT was used by 45 students randomly divided into two groups. The first 23 students used an MTT that provided both negative feedback on error and flag feedback (WF condition), whereas the remaining 22 students received only flag feedback (NF condition). The students were first asked to complete a pretest (20 minutes), then to use the MTT (30 minutes) and finally to complete a posttest (20 minutes). Two versions of the test (graded on a total of 40) were used. Half the students received the first version as pretest and the second as posttest, whereas the order was reversed for the other half.

At the end of the experiment, students from the WF condition were asked to answer a series of questions regarding their perception of the quality of the negative feedback they received. Out of 23 students, 22 said they had received feedback while using the MTT. Of those 22, 2 answered that they received very few feedback interventions (1-3), 11 received few (4-9), 8 often received feedback (9-15) and 1 very often (16+). Students were also asked to indicate whether they agreed or disagreed with 6 statements regarding the feedback (Table 4).

Table 4 - Students' responses to the feedback assessment questionnaire.

	Strongly disagree	Disagree	Agree	Strongly agree
1. Their content corresponded to my error	0	2	13	7
2. They helped me perform the task	0	3	8	11
3. They helped me learn how to perform the task	1	6	12	3
4. They were easy to understand	1	8	8	5
5. They hindered my understanding of the task	15	4	3	0

A two-sample t-test showed no statistically significant differences ($t(43) = -1.503$) between the students' pretest scores for the WF ($M = 20.02$; $SD = 6.32$) and NF ($M = 23.41$; $SD = 8.67$) conditions. The low statistical power (31.06%) and the medium effect size ($d = 0.45$) for this test seem to indicate that a significant difference could have been found with a more powerful test.

Paired t-tests were used to assess the learning gains between pretest and posttest. Both the WF ($t(22) = -7.400, p < 0.001$) and NF ($t(21) = -4.252, p < 0.001$) conditions showed significant gains. The condition-WF pretest ($M = 20.02$; $SD = 6.32$) and posttest ($M = 27.61$; $SD = 7.44$) scores indicate a large effect size ($d = 1.09$) and the condition-NF pretest ($M = 23.41$; $SD = 8.67$) and posttest ($M = 28.27$; $SD = 9.22$) scores indicate a medium effect size ($d = 0.54$). The results are illustrated in Figure 13.

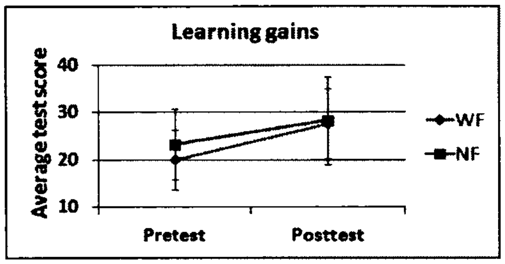


Figure 13 - The results of pretest and posttest results for our negative feedback experiment.

A one-tailed analysis of covariance (ANCOVA), with the pretest scores as the covariate, did not show a significant difference between the posttest scores ($F(1, 43) = 2.187, p = 0.074$) for conditions WF ($M_{aj} = 29.066$) and NF ($M_{aj} = 26.749$). The power of this test is low (44.25%), with a medium effect size ($\eta^2_p = 0.049$). Table 5 presents a summary of the analysis of the learning gains.

Table 5 - Summary of the statistical analysis for our negative feedback experiment.

	Stat	p	Effect size	Power
Pretest scores	$t(43) = -1.503$	0.140	$d = 0.45$	31.06%
Learning gain (NH)	$t(22) = -7.400$	< 0.001***	$d = 1.09$	99.88%
Learning gain (WH)	$t(21) = -4.252$	< 0.001***	$d = 0.54$	67.98%
ANCOVA	$F(1, 43) = 2.187$	0.074	$\eta^2_p = 0.049$	44.25%

We conducted a second experiment designed to retrieve logs of the students' interactions with the MTT and compile statistics relevant to our diagnosis of errors and the negative feedback provided by the MTT. We divided a class of 34 students into two groups: 18 students used an MTT providing negative feedback on error (WF) and 16 used an MTT that provided only flag feedback (NF). Although the students did complete a pretest and a posttest, we did not use this data, as the pretest scores were too strong (more than 25% of the students had perfect or almost perfect scores) and did not leave any room for improvement. This might be due to the fact that the students had to implement an AVL tree for an assignment due the week after the experiment.

We analyzed logs from 30 students: 18 from the WF condition and 12 from the NF condition (the logs from 4 students were lost due to issues uploading their data to our servers). In total, the students performed 192 tasks (113 for WF and 79 for NF) and executed 1120 off-path steps (576 for WF and 544 for NF) on 585 different instances of errors.[18] Our MTT was able to diagnose as errors, and provide negative feedback for, 381 out of the 576 off-path steps executed by students from the WF condition (66.15%).

[18] We consider consecutive off-path steps as being caused by the same error instance.

A two-sample t-test ($t(28) = -1{,}719$, $p = 0.097$) showed a marginal difference in the number of off-path steps per student for the WF ($M = 32.00$; $SD = 17{,}38$) and the NF ($M = 45.33$; $SD = 25.21$) conditions. The statistical power of this test was low (34.21%), and effect size was medium ($d = 0.62$).

We examined the number of correct steps executed by a student after an off-path step. For the WF condition, the students corrected their errors on the next attempt 65.62% of the times they received negative feedback on their error and 38.97% of the times they did not receive negative feedback. Overall, the chances of correcting an error, whether or not the MTT provided negative feedback, was 56.60%. For the NF condition, the students received no negative feedback and corrected their error on the next attempt 47.61% of the time.

Discussion

The two experiments we conducted had for objectives to assess the quality of the negative feedback generated by Astus and its effect on learning. In our first experiment, we asked the students to evaluate the quality of the negative feedback they received (Table 4). Almost all of the students who received feedback (20 out of 22) agreed that the feedback they received accurately identified their errors, but this result is attenuated by the fact that students' subjective report of the accuracy of feedback can be unreliable. Additional experiments will be required to formally evaluate the accuracy of Astus's diagnoses.

In addition, the students' evaluation of the negative feedback they received suggested many improvements that could be made to increase the efficiency of our feedback. Although most students (19 out of 22) answered that the feedback helped them perform the tasks, about a third (7 out of 22) also answered that the feedback did not help them learn (Table 4). This might be due to the fact that they had difficulty understanding the feedback (9 out of 22). With this in mind, it would be necessary for us to improve the readability of the feedback generated by our framework. We also intend to use this opportunity to revise the pedagogical content of the feedback to maximize its efficiency.

The pretest and posttest scores of our first experiment were used to evaluate whether the learners' performance was improved by negative feedback on error. Although the ANCOVA did not show a statistically significant difference between the two conditions, the t-tests evaluating the learning gains yielded a much higher effect size ($d = 1.09$) for the WF condition than for the NF condition ($d = 0.54$) (Table 5). This is illustrated by the steeper slope for condition WF on the graphical representation of the learners' scores (Figure 13).

The higher learning gains for the WF condition might be explained by its lower average score on the pretest, but the effect size for the ANCOVA, which controls for pretest scores, is medium. This suggests that, although the ANCOVA did not reveal a statistically significant difference, the difference between the two conditions is meaningful. The observed effect size ($\eta^2_p = 0.049$) is about half that obtained for similar experiments measuring the learning gains of next-step hints ($\eta^2_p = 0.091$ and $\eta^2_p = 0.086$). This is consistent with what we expected, as next-step hints are on-demand help, whereas negative feedback is targeted at specific learning situations and is only available on errors for which a diagnosis is possible. Hence, our negative feedback can only help learners when there is a plausible explanation for their errors, and is mainly targeted at learners who have a minimal understanding of how to perform the task. It will not help learners who have greater difficulty, as many of their errors will be the result of trial and error.

Although the ANCOVA's effect size is consistent with the expected value, a power analysis indicated that 122 students would have been required for the test to have a statistical power of 80%. This is much higher than the number of students per course in the computer science department at Université de Sherbrooke, and it considerably reduced our chances of finding a statistically significant difference.

We examined logs from the students' usage of our MTT in our second experiment. Our MTT was able to diagnose about two-thirds (66.15%) of the students' off-path steps. Thus, our tutor was able to provide negative feedback for many of the learners' off-path steps, but this number might include diagnosis on minor slips or errors related to using the learning environment. In those situations, the feedback will not be helpful to the learner.

We examined how often students were able to correct an off-path step on their next attempt. For the WF condition, we observed a higher percentage of correct steps following an off-path step for which the MTT provided negative feedback (65.62%) than for one with flag feedback alone (38.97%). This difference could be due to the fact that slips, errors related to the learning environment and errors that are easy to correct are more often diagnosed by Astus. To verify whether this is the case, we compared the total number of correct attempts following an off-path step for both the WF and NF conditions. If the negative feedback did not contribute to learners being able to correct their errors, the percentage of correct steps following an off-path step should be similar for both conditions. We observed a higher percentage of correct steps for the WF condition (56.60%) than for the NF condition (47.61%). This suggests that negative feedback did help learners correct their error more quickly.

Overall, the use of negative feedback seems to have had a positive impact on the students' use of our MTT. The number of off-path steps per student was significantly reduced in the WF condition and the students in the WF condition seem to have been able to correct their errors more quickly.

CONCLUSION

In this paper, we have shown how an MTT authoring framework can be designed to produce MTTs capable of using the content of the task model to generate pedagogical interventions. To achieve this objective, we proposed, as an extension to classical MTTs, three features of a knowledge representation system that facilitates the generation of interventions by modeling the teacher's instructions.

We introduced the individual features (hierarchical procedural knowledge structures, explicit procedural knowledge units, semantically rich declarative knowledge units) by comparing them to production systems and describing how they are implemented in Astus's knowledge representation system. We showed how each feature provides information that can be used to generate interventions such as next-step hints and how the combination of these features allows the generation of increasingly complete hints. Likewise, we explained how the information obtained by Astus's tracing of the learners' steps can be used to further specify the instructions provided by the hints.

We showed that the features which allow Astus to generate next-step hints can also be used to generate other types of interventions. By manipulating the episodic tree used to trace the learner's steps, Astus can provide negative feedback on many of his/her errors. To do so, Astus can interpolate incorrect executions of the task's procedural knowledge to determine whether they provide a plausible explanation of the learner's off-path step.

We presented the results of five experiments conducted in order to obtain an initial assessment of the effectiveness of the interventions generated by Astus. Results from these experiments suggest that the next-step hints generated by Astus can be as effective and as well rated as those written by a teacher. Although the results of our two experiments concerning negative feedback did not show strong statistical results, they gave us great insights on how to improve the generation of negative feedback.

One of the main advantages of generated interventions is the capability to customize them. The interventions generated by Astus can be customized for different groups of learners, specific learners within a group or specific learning situations. For example, our hints can be generated in different languages (English and French), made culturally aware [8] and consider the learner's emotional state [33]. The available content for the interventions would remain the same, but their presentation would vary according to these parameters. Although it would be possible for a teacher to author multiple versions of every intervention to take these parameters into account, having the framework generate them would require much less effort.

In addition to customizing the interventions we generate, we are interested in studying how Astus's knowledge representation system can be used to generate additional types of interventions such as worked examples [15], self-explanation prompts [2, 9] and analogies with other instances of the task [20]. The generation of such interventions by Astus would aid in investigating their effectiveness for the tutoring of various tasks and their interaction when combined in sophisticated pedagogical strategies.

REFERENCES

[1] Aleven, V. (2010). Rule-Based Cognitive Modeling for Intelligent Systems. In Nkambou R., Bourdeau J. & Mizoguchi R. (Eds.) *Advances in Intelligent Tutoring Systems*, 33-62.

[2] Aleven, V., Koedinger, K.R. (2008). An Effective Metacognitive Strategy: Learning by Doing and Explaining with a Computer-Based Cognitive Tutor. *Cognitive Science*, 26, 147-179.

[3] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. In *Proceedings of ITS 2006*, 61-70.

[4] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R. (2009). Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems. *International Journal of Artificial Intelligence in Education*, Special Issue on "Authoring Systems for Intelligent Tutoring Systems", 105-154.

[5] Anderson, J.R., Boyle, C.F., Corbett, A., Lewis, M.W. (1990). Cognitive Modeling and Intelligent Tutoring. *Artificial Intelligence*, 42, 7-49.

[6] Anderson, J.R., Conrad, F.G., Corbett, A.T. (1989). Skill Acquisition and the LISP Tutor. *Cognitive Science*, 13, 467-505.

[7] Barnes, T., Stamper, J., Croy, M., Lehman, L. (2008). A Pilot Study on Logic Proof Tutoring Using Hints Generated from Historical Student Data. In *Proceedings of EDM 2008*, 197-201.

[8] Blanchard, E.G., Mizoguchi, R. (2008). Designing Culturally-Aware Tutoring Systems: Toward an Upper Ontology of Culture. In *Cutlturally Aware Tutoring Systems CATS'08*, 23-24.

[9] Conati, C., VanLehn, K. (1999). Teaching Meta-Cognitive Skills: Implementation and Evaluation of a Tutoring System to Guide Self-Explanation while Learning from Examples. *Proceedings of AIED 1999*, 297-304.

[10] Heffernan, N.T., Koedinger, K., Razzaq, L. (2008). Expanding the Model-Tracing Architecture: A 3rd Generation Intelligent Tutor for Algebra Symbolization, *International Journal of Artificial Intelligence in Education*, 18, 153-178.

[11] Hill, R.W., Johnson, L.W. (1993). Impasse-Driven Tutoring for Reactive Skill Acquisition. In *Proceedings of the Conference on Intelligent Computer-Aided Training and Virtual Environement Technology*.

[12] Hill, R.W., Johnson, L.W. (1995). Situated Plan Attribution. *International Journal of Artificial Intelligence in Education*, 6, 35-66.

[13] Koedinger, K., Anderson, J.R. (1993). Effective Use of Intelligent Software in High School Math Classrooms. *Proceedings of AIED 1993*, 241-248.

[14] McKendree, J. (1990). Effective Feedback Content for Tutoring Complex Skills. *Human-Computer Interaction*, 5, 381-413.

[15] McLaren, B.M., Isotani, S. (2011). When is it Best to Learn with All Worked Examples? *Proceedings of AIED 2011*, 222-229.

[16] Mitrovic, A. (1998). A Knowledge-Based Teaching System for SQL. In *Proceedings of ED-MEDIA 1998*, 1027-1032.

[17] Mitrovic, A. (2010). Modeling Domains and Students with Constraint-Based Modeling. In Nkambou R., Bourdeau J. & Mizoguchi R. (Eds.) *Advances in Intelligent Tutoring Systems*, 63-80.

[18] Mitrovic, A., Ohlsson S. (1999). Evaluation of a Constraint-Based Tutor for a Database Language. *Artificial Intelligent in Education*, 10, 238-256.

[19] Mitrovic, A., Weerasinghe, A. (2009). Revisiting Ill-Definedness and the Consequences for ITSs. In *Proceedings of AIED 2009*, 375-382.

[20] Ohlsson, S. (2008). Computational Models of Skill Acquisition. *The Cambridge Handbook of Computational Psychology*, Cambridge University Press, 359-395.

[21] Paquette, L., Lebeau, J.-F., Mayers, A. (2010). Authoring Problem-Solving Tutors: A Comparison Between CTAT and ASTUS. In Nkambou R., Bourdeau J. & Mizoguchi R. (Eds.) *Advances in Intelligent Tutoring Systems*, 377-405.

[22] Paquette, L., Lebeau, J.-F., Mayers, A. (2012). Automating the Modeling of Learners' Erroneous Behaviors in Model-Tracing Tutors. In *Proceedings of UMAP 2012*, 316-321.

[23] Paquette, L., Lebeau, J.-F., Mayers, A. (accepted). Diagnosing Errors from Off-Path Steps in Model-Tracing Tutors. In *Proceedings of AIED 2013*.

[24] Paquette, L., Lebeau, J.-F., Beaulieu, G., Mayers, A. (2012). Automating Next-Step Hints Generation Using ASTUS. *In Proceedings of ITS 2012*, 201-211.

[25] Paquette, L., Lebeau, J.-F., Mbungira, J.P., Mayers, A. (2011). Generating Task-Specific Next-Step Hints Using Domain-Independant Structures. In *Proceedings of AIED 2011*, 525-527.

[26] Reiser, B.J., Kimberg, D.Y., Lovett, M.C., Ranney, M. (1992). Knowledge Representation and Explanation in GIL, An Intelligent Tutor for Programming. In Computer-Assisted Instruction and Intelligent Tutoring Systems, 111-149.

[27] Rickel, J. (1988). An Intelligent Tutoring Framework for Task-Oriented Domains. In *Proceedings of ITS 1988*, 109-115.

[28] Rickel, J., Johnson, L.W. (1999). Animated Agents for Procedural Training in Virtual Reality: Perception, Cognition, and Motor Control. *Applied Artificial Intelligence*, 13, 343-382.

[29] Sacerdoti, E.D. (1975). *A Structure for Plans and Behavior*. Elsevier, New York.

[30] VanLehn, K. (1990). *Mind Bugs: The Origin of Procedural Misconceptions*. MIT Press.

[31] VanLehn, K. (2006). The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education*, 16, 227-265.

[32] VanLehn, K., Lynch, C., Schultz, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *International Journal of Artificial Intelligence in Education*, 15, 147-204.

[33] Woolf, B., Burleson, W., Arroyo, I., Dragon, T., Cooper, D., Picard, R. (2009). Affect-Aware Tutors: Recognising and Responding to Student Affect. *Journal of Learning Technology*, 4, 129-163.

Chapitre 3

Générer les indices par rapport à la prochaine étape

L'article présenté dans ce chapitre décrit la méthode utilisée par Astus afin de générer des indices par rapport à la prochaine étape. Son contenu est similaire aux parties relatives aux indices par rapport à la prochaine étape du chapitre 2, mais il est présenté dans une perspective différente avec des exemples différents. L'article présenté dans ce chapitre décrit plus en détail l'analyse statistique des expérimentations qui ont été faites dans le but de valider de façon empirique les indices produits par Astus.

Comme pour l'article présenté au chapitre 2, la contribution de l'auteur (Luc Paquette) inclut la méthode pour la génération des indices par rapport à la prochaine étape et les expérimentations utilisées pour valider l'efficacité des indices. L'auteur a contribué à 80 % de la charge de travail pour la rédaction de cet article.

Automating Next-Step Hints Generation Using ASTUS

Luc Paquette, Jean-François Lebeau, Gabriel Beaulieu and André Mayers

Université de Sherbrooke, Québec, Canada

{Luc.Paquette, Andre.Mayers}@USherbrooke.ca

Abstract. ASTUS is an authoring framework designed to create model-tracing tutors with similar efforts to those needed to create Cognitive Tutors. Its knowledge representation system was designed to model the teacher's point of view of the task and to be manipulated by task independent processes such as the automatic generation of sophisticated pedagogical feedback. The first type of feedback we automated is instructions provided as next step hints. Whereas next step hints are classically authored by teachers and integrated in the model of the task, our framework automatically generates them from task independent templates. In this paper, we explain, using examples taken from a floating-point number conversion tutor, how our knowledge representation approach facilitates the generation of next-step hints. We then present experiments, conducted to validate our approach, showing that generated hints can be as efficient and appreciated as teacher authored ones.

Keywords: Hint generation, knowledge representation, model-tracing tutors

1. Introduction

The "intelligence" of intelligent tutoring systems (ITS) results from their ability to offer relevant pedagogical feedback tailored to the learner's needs. In order to achieve this objective, most systems offer different services [1] such as:

- An expert module that analyzes the task's model to assess the learner's progression towards a solution.
- A learner model that assesses the learner's mastery of the task's knowledge.
- A pedagogical module that provides relevant feedback.

Ideally, in order to reduce the development costs, those modules would be independent from the task. In this context, the creation of a tutor would only require modeling the knowledge relevant to the task and implementing the learning environment's graphical user interface (GUI). Unfortunately, it is difficult for a tutor to provide sophisticated feedback using only the task's model. For this reason, the pedagogical module usually relies on domain specific content integrated to the model. We designed the ASTUS framework and its knowledge representation approach as a step towards solving this problem for model-tracing tutors (MTTs) [2].

Our objective is to offer a framework [3] that can take advantage of the content of the task's model in order to generate different types of sophisticated pedagogical feedback [4]. This approach is inspired by Ohlsson's learning mechanisms theory [5]. According to this theory, learning can be achieved using nine different mechanisms. Each of these mechanisms can be more or less effective according to the learning context and can be activated by different types of learning activities and feedback. Tutors, such as those created using ASTUS, would greatly benefit from being able to generate feedback targeting a maximum number of those mechanisms.

In order to apply Ohlsson's [5] theory to our framework, we first focused our efforts on a mechanism classically used by MTTs: instructions provided as next-step hints. Whereas most MTT provide next-step hints [6, 7], they are usually authored by a teacher and integrated to the knowledge units contained in the task's model. Barnes and Stamper [8] worked on associating teacher authored hints to automatically generated task models, but few efforts have been made to automate the generation of the hints themselves. The automation of this feedback would contribute to the reduction of the efforts required to author MTTs.

The work presented in this paper describe how, using the ASTUS framework's knowledge representation system, we are able to automatically generate next-step hints. We describe, and illustrate using examples, the processes of generating hints and we present the results of experiments conducted in order to validate our approach.

2. ASTUS

ASTUS is an authoring framework for the creation of MTTs similar to the Cognitive Tutors [9]. One of its main differences is the use of a novel knowledge representation system instead of the more traditional production rule based ones. This system was designed to facilitate the manipulation of the task model by task independent processes such as the automatic generation of pedagogical feedback.

Rather than modeling the cognitive processes used by learners to execute a task, ASTUS's knowledge representation models the teachers' point of view of the task. The format used to model the task is designed to make the content of each knowledge unit explicit. This property allows the manipulation of the model by the framework and is crucial for the generation of feedback such as next-step hints.

In this section, we present a summary of the main structures of ASTUS's knowledge representation system [2]. Semantic knowledge is modeled using concepts: task specific abstractions that are pedagogically relevant. Each concept defines a set of essential features that can refer to other concepts or primitive values (integer, decimal number, symbol, boolean).

Procedural knowledge is modeled using goals and procedures that together form a procedural graph. Figure 1 (left) shows part of the procedural graph in the case of our floating-point tutor. Goals are shown as rectangles and procedures as ovals.

Goals can be achieved by the execution of a procedure (primitive or complex). Primitive procedures model skills that are considered already mastered by the learners. They are reified as atomic interactions in the learning environment's GUI. Complex procedures specify sets of sub-goals the learner has to achieve. Those sub-goals are arranged according to dynamic plans specific to the procedure's type (a sequence, a selection or iteration). Both procedures and goals can specify variables (parameters) used to refine their behavior.

During the tutor's execution, goals and procedures are instantiated in order to produce an episodic tree (right of figure 1). This tree contains all of the completed (C) or currently executing (E) goals and procedures as well as goals that will be expended in the future (W). The episodic tree is used to match the learner's steps and indicate whether they are valid or not. This is achieved by using the complex procedures' scripts to expand the tree up to each of the possible next-steps.

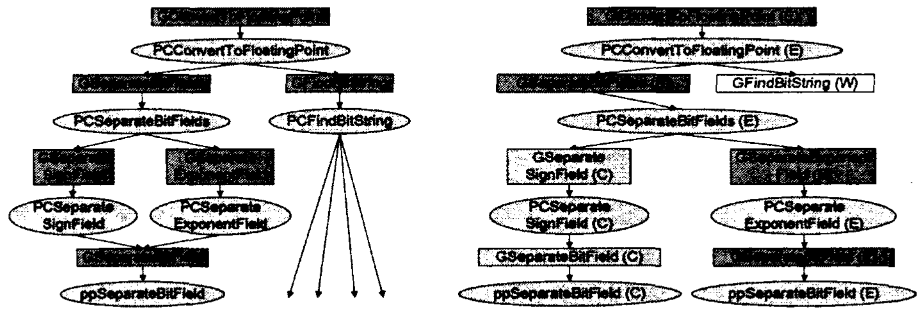


Figure 1 - Examples of part of the procedural graph (left) and its instantiation as an episodic tree (right) for our floating-point number conversion tutor

3. Hint Generation

Since the task's model is defined using structures that the framework can manipulate, it is possible to automatically generate pedagogical feedback such as next-step hints. To achieve this, the framework mainly benefits from the information contained in the procedural graph and the episodic tree.

We distinguish two main features for hints: their structure (independent from the task) and their content (specific to the task). We defined the structures of our hints as text templates to be filled using task specific content extracted from the knowledge units defined in the task's model.

We illustrate the process of next-step hint generation using a conditional procedure from our floating-point conversion tutor. More precisely, this example is taken from the sub-task of converting a decimal number to a binary format. The following procedure is used while converting the integer part of a decimal number:

```
Conditional procedure 'PCDivideInt' achieves 'GDivideInt' {
  if 'current_line' instanceOf 'FirstLine'
    goal 'GDivideInitialInt' with 'int', 'current_line'
  if not ('current_line' instanceOf 'FirstLine')
    goal 'GDividePrevQuotient' with 'current_line'
}
```

The definition of this procedure contains information that can be used by the framework in order to generate next-step hints. The header contains the procedure's identifier (*PCDivideInt*) and the identifier of the goal it achieves (*GDivideInt*). The body of the procedure specifies two sub-goals that are available to the procedure (*GDivideInitialInt* and *GDividePrevQuotient*) and the parameters that will be used to instantiate them. The body also specifies two conditions (one for each sub-goals). In addition, the procedure's type (conditional) specifies how it will be executed: the conditions will be evaluated and the sub-goal associated with the first condition evaluated as true will be instantiated.

Once we have determined the content available for the generation of hints, we can choose the structure of the desired hint. For instance, the definition of a conditional procedure could be used to generate a "pointing-hint":

```
You need to [parent goal name].
```

This message can be instantiated using the information contained in the parent goal to become:

You need to *divide the integer.*

The text used in the message comes from the name associated to the *GDivideInt* goal:

```
Goal 'GDivideInt' eng-name 'divide the integer' {
  parameter 'int' type 'Integer' eng-name 'integer'
  parameter 'current_line' type 'IntLine' eng-name 'current
    line'
}
```

In fact, all of the domain specific text used to generate hint messages comes from the name associated to the knowledge units. This approach requires less effort than asking a teacher to write each hint, especially if multiple hints are associated to the same knowledge unit or the hints have to be translated in multiple languages.

The first "pointing-hint" message is abstract and does not provide much help regarding how to execute the procedure. In fact, this template could be used for any type of procedure. Producing more helpful messages requires more specific content. We can examine how a conditional procedure is executed (select the appropriate sub-goal) and combine this information with the knowledge of the available sub-goals to produce the following hint:

```
In order to [parent goal name], you must either [sub-goal
name] or [sub-goal name].
```

Which would be instantiated as:

In order to *divide the integer,* you must either *divide the initial integer* or *divide the previous quotient.*

While this hint is more explicit regarding how to execute the procedure than the "pointing-hint", it could still be made more specific. In order to solve this problem, the tutor can refer to the conditions associated to each of the sub-goals.

The conditions are explicitly defined using a combination of logical expressions (and, or, not, exists, isInstance, equals). This information can be used to generate hints by starting from a condition's root expression and generating hints for each of its sub expressions. During this process, the "not" expression can be used as a modifier for a "positive" attribute that impacts the templates used to generate each expression's hint. Table 1 presents the different templates we used for each expression types.

Table 1 - Templates associated to the conditions' expressions. The bracketed text indicates a sub template and the parenthesis indicates whether a sub expression is positive (T) or not (F)

Positive	not	and	or	exists	isInstance	equals
True	[*expr*(F)]	[$expr_1$(T)] and [$expr_2$(T)]	[$expr_1$(T)] or [$expr_2$(T)]	a [concept] exists	[var] is a [concept]	[var1] equals [var2]
False	[*expr*(T)]	[$expr_1$(F)] or [$expr_2$(F)]	[$expr_1$(F)] and [$expr_2$(F)]	no [concept] exists	[var] is not a [concept]	[var1] does not equal [var2]

Using those templates, the previous hint can be modified to provide additional instruction regarding when to apply each of the procedure's sub-goals. The condition expressions described in the "PCDivideInt" procedure (defined previously) can be used to generate messages that are integrated to the hint:

> In order to *divide the integer*, you must either *divide the initial integer*, if *current line* is a *first line*, or *divide the previous quotient*, if *current line* is not a *first line*.

This hint takes advantage of all of the information contained in the procedure's definition, but can still be modified by using the information contained in the episodic tree regarding the current state of the problem being solved. This can be used to reduce the size of the hint and to focus the learner's attention on the correct sub-goal:

```
In order to [parent goal name], you must [active sub-goal
name] since [active condition].
```

Which would be instantiated as:

`In order to` *`divide the integer,`* `you must` *`divide the initial integer`* `since` *`current line`* `is a` *`first line.`*

This last template is the one currently used by our framework, but this decision is specific to how we decided to provide next-step hints. Any combination of one or more templates (those given as examples or new templates using the available information) can be used to provide next-step hints.

The examples given in this section show how the information contained in the definition of knowledge units can be used to generate next-step hints. Those hints can be customized according to the desired pedagogical strategy: they can provide different amounts of instruction and they can be contextualized using the current state of the learning environment. The current implementation uses text templates in order to generate the hints, but could be improved by using natural language techniques. For instance, in our previous examples, the condition expression "*current line* is a *first line*" could be rewritten as "the current line is the first line". Such small modifications would greatly improve the readability of the generated hints.

In this paper, we only described how next-step hints can be generated for conditional procedures. A similar process has been applied to every type of procedural knowledge units. Among them are inferences, expressions that model mental skills applied to fill in the parameters of goals and procedures. They can be used to further contextualize next-step hints by specifying how a parameter is deduced from known ones, recalled from memory or perceived in the learning environment's GUI.

4. Experiments

In order to validate our hint generation approach, we conducted multiple experiments during a computer science course at the University of Sherbrooke. We used a floating-point number conversion tutor designed using ASTUS. The objective of our first experiment was to evaluate the learning gains and the students' appreciation of next-step hints generated by our framework. This first experiment is detailed in [10], but we present here a summary of its methodology and the analysis of its results. In this first study, 34 students were separated in two groups: 19 students received teacher authored hints (TH) and 15 received framework generated hints (FH). Statistical analyses of the results did not show a significant difference in learning gain when comparing pretest and posttest scores (left of figure 2) for both conditions and showed that framework generated hints can be as appreciated as equivalent teacher authored hints (right of figure 2) for different types of complex procedures: while iteration, conditional, sequence with N sub-goals and sequence with 1 sub-goal.

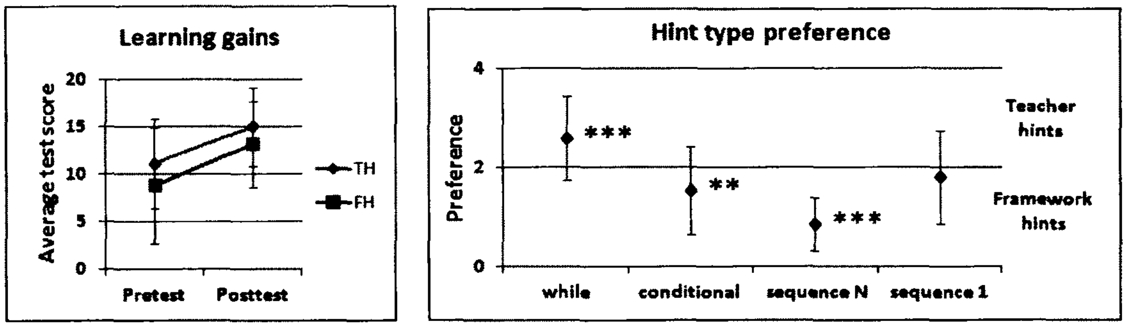


Figure 2 - Graphs illustrating the results of our first experiment. The '*' character indicates the statistical significance (** for $p < 0.01$ and *** for $p < 0.001$)

Table 2 - Summary of the statistical analysis for our second experiment

	Stat	p	Effect size	Power
Pretest scores	$t(23.629) = 0.576$	0.570	$d = 0.20$	8.50%
Learning gain (NH)	$t(15) = 6.213$	< 0.001***	$d = 1.35$	99.89%
Learning gain (WH)	$t(15) = 5.550$	< 0.001***	$d = 1.37$	99.91%
ANCOVA	$F(1, 29) = 3.057$	0.046*	$\eta^2_p = 0.091$	39.40%

While our first experiment did not find any significant difference between framework generated and teacher authored hints in the context of our floating-point tutor, it does not validate that the learning gains can be attributed to the received hints. Indeed, the observed gains could simply be caused by the activity of solving problems using a tutor. In order to determine if next-step hints were helpful while solving problems with our tutor, we conducted a second experiment comparing the learning gains of a tutor without next-step hints (only flag feedback) to those of a tutor also providing framework generated next-step hints. A group of 32 students was separated, at random, in two sub-groups: 16 students used a tutor that did not provide next-step hints (NH) and 16 students used one that provided framework generated hints (WH). The students were first asked to complete a pretest (20 minutes), then use the tutor (40 minutes) and finally complete a posttest (20 minutes). There were two versions of the test (graded on a total of 20). Half the students received the first version as pretest and the second as posttest while the order was reversed for the other half. Table 2 summarizes the results of our analysis.

A two-sample t-test showed no statistically significant differences between the participants' pretest scores for the NH ($M = 8.13$; $SD = 2.34$) and the WH ($M = 8.82$; $SD = 4.16$) conditions. Although no significant differences were found, the standard deviation of the WH condition is much higher than the one for the NH condition.

The learning gains between the pretests and posttests were validated using paired t-tests. Both conditions showed significant gains. The NH condition's pretest ($M = 8.13$; $SD = 2.34$) and posttest ($M = 12.09$; $SD = 3.30$) scores indicate a large effect size, and so do the WH condition's pretest ($M = 8.81$; $SD = 4.16$) and posttest ($M = 14.44$; $SD = 4.06$) scores. The effect sizes are very similar even though the mean learning gain is higher for the WH (5.63) condition when compared to the NH (3.96) condition. This lack of difference results from the difference in standard deviations between the two conditions. The effect size for the WH condition would have been higher if its standard deviations were closer to those of the NH condition.

A one-tailed analysis of covariance (ANCOVA), with the pretest scores as the covariate, showed a significant differences between the posttest scores for the NH ($M_{aj} = 12.31$) and WH ($M_{aj} = 14.22$) conditions. This suggests that the use of next-step hints during problem solving allows learners to achieve higher learning gains. In order to further validate this result, we conducted a third experiment using the same methodology as the second one. In this experiment there were 16 learners for the NH condition and 17 for the WH condition. Its results are summarized in table 3.

Table 3 - Summary of the statistical analysis for our third experiment

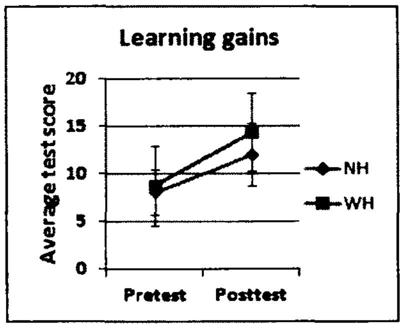
	Stat	*p*	Effect size	Power
Pretest scores	$t(31) = 0.318$	0.753	$d = 0.11$	61.00%
Learning gain (NH)	$t(15) = 2.970$	0.010**	$d = 0.52$	49.46%
Learning gain (WH)	$t(16) = 4.401$	< 0.001***	$d = 0.77$	86.64%
ANCOVA	$F(1, 30) = 2.818$	0.052	$\eta^2_p = 0.086$	36.40%

A two-sample t-test showed no statistically significant differences between the learners' pretest scores for the NH (M = 10.97; SD = 4.28) and the WH (M = 11.50; SD = 5.24) conditions.

The learning gains between the pretests and posttests were validated using paired t-tests. Both conditions showed significant gains. The NH condition's pretest (M = 10.97; SD = 4.28) and posttest (M = 13.19; SD = 4.25) scores indicate a medium effect size, and the WH condition's pretest (M = 11.50; SD = 5.24) and posttest (M = 15.26; SD = 4.43) scores indicate a large effect size. The higher effect size for the WH condition suggests it yielded higher learning gains than the NH condition.

A one-tailed analysis of covariance (ANCOVA), with the pretest scores as the covariate, showed no significant differences between the posttest scores for the NH (M_{aj} = 13.83) and WH (M_{aj} = 15.26) conditions. The results of the test were very close to a statistically significant difference (p = 0.052). This, combined with the differences in effect size for the paired t-tests and the higher adjusted mean score for the WH, suggests that the WH condition yielded higher learning gains.

While neither our second nor our third experiments yielded strong statistical results, the results of both suggest that the WH condition leads to higher learning gains. Figure 3 shows the results of those two experiments in graphic form. In both, the steeper slopes for the WH conditions illustrate how the students in the WH conditions improved their posttest results by a greater amount than those in the NH condition. Those graphs can be compared to the equivalent graph for our first experiment (left of figure 2) for which the two slopes (FH and TH) are very similar, which is consistent with the absence of a significant difference.



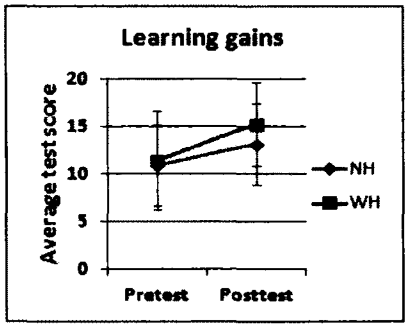


Figure 3 - Graphs illustrating the results of our second (left) and third (right) experiments

5. Discussion

The results of our most recent experiments show how framework generated next-step hints yielded higher learning gains compared to the use of a tutor offering only flag feedback. This shows that the floating-point number conversion task is complex enough for the use of next-step hints to improve learning gains, but it does not evaluate the efficiency of framework generated hints. A previous experiment [10] showed that framework generated hints can be as efficient and as appreciated as teacher authored ones in the context of our floating-point tutor.

Additional experiments could be used to improve our empirical validation by reproducing similar results for different tasks. Such results would suggest that the efficiency of framework generated hints can be generalized to multiple types of task of different complexity. These experiments would also benefit from bigger groups of learner to increase the statistical power of their result. Additionally, they would benefit from learners with no background in computer science. In our experiments, the learners were all computer science students that are well suited to understanding computer generated messages. Reproducing similar results with learners from more varied backgrounds would support our hypothesis that efficient hints can be automatically generated by a framework regardless of the task taught.

We described how our framework has access to the information required to generate efficient next-step hints. It would be interesting to research how the hints' efficiency can be improved by modifying how they present this information. The use of natural language techniques might impact the hints' efficiency by improving their readability, thus fostering better communication between the learner and the tutor. Their efficiency might also be improved by the use of learning theories optimizing the content and the format of the hints provided to the learners.

The use of generated next-step hints is useful in order to reduce the authoring efforts required to create a tutor by only having to associate readable names to knowledge units instead of complete message templates. They could also be used to customize the hints to groups of learners, specific learners within that group or even specific learning situation. For example, hints could be generated in different languages, they could be made culturally aware [11] and they could consider the learner's current emotional state [12]. The content of the hints would remain the same but their presentation would vary according to those parameters. Although it would be possible for a teacher to author multiple versions of every hint to account for those parameters, having the framework generate the hints would require much less efforts.

In addition to reducing the efforts of authoring hints, being able to generate them can be essential for situations where it is not possible to enumerate all the possible hints. An example of such a situation is negative feedback on errors. In order to provide such feedback, model-tracing frameworks usually require the tutor's author to model erroneous procedural knowledge. This process requires a lot of efforts due to the very high number of different errors. In order to reduce the required efforts, we are currently working on a model, based on Sierra's theory of procedural error [13], to allow our framework to diagnose as many of those errors as possible without modeling additional erroneous knowledge [14]. Since the errors are automatically diagnosed by the framework while a learner solves a problem, they are not explicitly defined in the task and it is not possible to enumerate all the required hints. It is thus essential for the framework to be able to generate efficient hints in order to provide feedback regarding the diagnosed errors.

The example of providing negative feedback on errors illustrates how being able to generate next-step hints is a first step toward achieving our objective of developing a framework able to provide feedback for many of Ohlsson's learning mechanisms. We started by automating the generation of next-step hints feedback for instruction, but our work will also be extended to support other mechanisms such as negative feedback on error, a type of feedback usually provided by constraint-based tutor [15].

6. Conclusion

In this paper, we explained how the ASTUS framework generates next-step hints using domain independent knowledge structures. We presented experiments showing that these hints can be as effective and as appreciated as teacher-authored hints in the context of our floating-point number conversion tutor.

Future work will focus on expanding the number of different types of feedback the framework can generate in order to take advantage of as many of Ohlsson's learning mechanisms [5] as possible. Our hypothesis is that the same characteristics that allow the generation of next-step hints will be helpful when generating other types of feedback. Our next objective is to diagnose and offer negative feedback regarding the learners' errors without requiring the modeling of knowledge marked as erroneous.

Acknowledgements. We would like to thank Richard St-Denis for allowing the use of our tutor in his class, Mikaël Fortin for authoring the "teacher authored hints" used during our experiment and François Bureau du Colombier and Jean Pierre Mbungira for their help with the implementation of our tutor.

References

[1] Wenger, E.: *Artificial Intelligence in Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan Kaufmann Publishers Inc, 1987.

[2] Paquette, L., Lebeau, J.-F., Mayers, A.: Authoring Problem-Solving Tutors: A Comparison Between CTAT and ASTUS, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 377-405.

[3] Lebeau, J.-F., Paquette, L., Fortin, M., Mayer, A.: An Authoring Language as a Key to Usability in a Problem-Solving ITS Framework, *Proceedings of ITS 2010 Part II*, Aleven, V., Kay, J, Mostow, J. (Eds), Lecture Notes in Computer Science 6095, Springer-Verlag, 2010, pp 236-238.

[4] Paquette, L., Lebeau, J.-F., Mayers, A.: Integrating Sophisticated Domain-Independent Pedagogical Behaviors in an ITS Framework, *Proceedings of ITS 2010 Part II*, Aleven, V., Kay, J, Mostow, J. (Eds), Lecture Notes in Computer Science 6095, Springer-Verlag, 2010, pp 248-250.

[5] Ohlsson, S.: *Deep Learning: How the Mind Overrides Experience*, Cambridge University Press, 2011.

[6] VanLehn, K.: The Behavior of Tutoring Systems, *International Journal of Artificial Intelligence in Education*, 16(3), 2006, pp 227-265.

[7] Aleven, V.: Rule-Based Cognitive Modeling for Intelligent Systems, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 33-62.

[8] Barnes, T., Stamper, J: Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data, *Proceedings of ITS 2008*, Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (Eds), Lecture Notes in Computer Science 5091, Springer-Verlag, 2008, pp 372-382.

[9] Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R.: Cognitive Tutors: Lessons Learned, *The Journal of the Learning Sciences*, 4(2), pp 167-207.

[10] Paquette, L., Lebeau, J.-F., Mbungira, J.P., Mayers, A.: Generating Task-Specific Next-Step Hints Using Domain-Independant Structures. *Proceedings of AIED 2011*, Biswas, G., Bull, S., Kay, J., Mitrovic, A. (Eds), Lecture Notes in Artificial Intelligence 6738, Springer-Verlag, 2011, pp 525-527.

[11] Blanchard, E.G., Mizoguchi, R.: Designing Culturally-Aware Tutoring Systems: Toward an Upper Ontology of Culture, *Culturally Aware Tutoring Systems CATS'08*, Blanchard, E.G., Allard, D. (Eds), 2008, pp 23-24.

[12] Woolf, B., Burleson, W., Arroyo, I., Dragon, T., Cooper, D., Picard, R.: Affect-Aware Tutors: Recognising and Responding to Student Affect, *Journal of Learning Technology*, 4, 2009, pp 129-163.

[13] VanLehn, K.: Mind Bugs: The Origin of Procedural Misconceptions, MIT Press, 1990.

[14] Paquette, L., Lebeau, J.-F., Mayers, A.: Modeling Learner's Erroneous Behaviors in Model Tracing Tutors, *Proceedings of UMAP 2012*, Masthoff, J., Mobasher, B., Desmarais, M.C., Nkambou, R. (Eds), Lecture Notes in Computer Science 7379, Springer-Verlag, pp 316-321.

[15] Mitrovic, A.: Modeling Domains and Students with Constraint-Based Modeling, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 63-80.

Chapitre 4

Modéliser automatiquement les erreurs des apprenants

L'article présenté dans ce chapitre décrit comment Astus est en mesure de modéliser automatiquement un grand nombre d'erreurs des apprenants. Pour y arriver, la théorie Sierra [20], une théorie décrivant l'origine des erreurs procédurales des apprenants, a été appliquée au système de représentation des connaissances d'Astus. Selon cette théorie, les erreurs procédurales surviennent lorsque les apprenants rencontrent des impasses ; des situations pour lesquelles ils ne savent pas comment résoudre la tâche correctement. Dans une telle situation, ils utilisent un mécanisme de réparation qui leur permet de continuer à résoudre la tâche. La combinaison d'une impasse et d'une réparation détermine l'erreur effectuée par un apprenant. Cette théorie a été appliquée à Astus en déterminant les différentes façons dont le modèle de la tâche peut être perturbé afin de modéliser les connaissances incorrectes qui découlent des impasses des apprenants. L'algorithme du diagnostic des erreurs, présenté dans le chapitre 2, a été élaboré à partir du modèle de perturbation des connaissances d'Astus décrit dans l'article présenté dans ce chapitre.

La contribution de l'auteur (Luc Paquette) représente 80 % de la charge de travail de rédaction de l'article. Comme il a été dit précédemment, le système de représentation des connaissances utilisé par Astus a été conçu par Jean-François Lebeau. L'application de la théorie Sierra à Astus par un modèle de perturbations des connaissances est une contribution de l'auteur.

Automating the Modeling of Learners' Erroneous Behaviors in Model-Tracing Tutors

Luc Paquette, Jean-François Lebeau and André Mayers

Université de Sherbrooke, Québec, Canada

{Luc.Paquette, Andre.Mayers}@USherbrooke.ca

Abstract. Modeling learners is a fundamental part of intelligent tutoring systems. It allows tutors to provide personalized feedback and to assess the learners' mastery over a task domain. One aspect often overlooked is the modeling of erroneous behaviors that can be used to provide error specific feedback. This is especially true for model-tracing tutors that usually require erroneous procedural knowledge associated to each of the possible error. This process can be automated thanks to a task independent model describing the learners' erroneous behaviors. The model proposed in this paper is inspired by the Sierra theory of procedural error and is developed for ASTUS, an authoring framework for model-tracing tutors.

Keywords: Erroneous behaviors, learner modeling, model-tracing tutors

1. Introduction

Intelligent tutoring systems (ITS) model different aspects of the learner's interaction with the tutor: how they solve problems [1]; how they manipulate the learning environment's user interface [2]; and how they acquire new knowledge [3]. Those models can be exploited by the tutor to provide personalized feedback.

One aspect of tutoring less frequently modeled is the erroneous behaviors exhibited by the learners. Modeling such behaviors can be benefic for ITSs as it allows them to provide negative feedback on errors, one of Ohlsson's nine learning mechanisms [4]. Constraint based tutors [5] can diagnose errors and provide feedback for them. Baffes and Mooney [6] designed a system to automatically compile bug libraries for classification tasks. Although both those systems can diagnose errors, they are less efficient than model-tracing tutors (MTTs) for modeling the sequence of steps committed by learners while they solve problems [7]. On the contrary, MTTs do not efficiently diagnose the learners' errors and would thus greatly benefit from a model of erroneous behaviors.

The modeling of erroneous behaviors is often neglected in MTTs as it usually requires adding erroneous procedural knowledge to the task's model, a process that requires much effort. In order to solve part of this problem, we took inspiration from computational theories explaining the source of procedural errors. More specifically, we were inspired by Sierra [8], a theory explaining the development of the learners' procedural misconceptions in procedural tasks such as subtraction. This theory provides cognitively plausible explanations for the learners' erroneous behaviors that could be used by MTTs to provide error specific feedback.

Whereas Sierra was designed to simulate the learner's behavior while solving a problem, MTTs are designed to teach how to solve problems. This fundamental difference influences the definition of their task's models. Sierra constructs multiple possible models of the learners' knowledge of the task at different moments during their learning process. MTTs, on the other hand, start with a complete model of the task. This model is designed by an expert to ensure that the tutor can provide efficient pedagogical feedback. Because of this difference, it is not possible to directly apply the Sierra theory in the context of MTTs.

Thus, we defined a model describing how an MTT's procedural knowledge can automatically be disrupted in order to produce erroneous behaviors analogous to those generated by Sierra.

We developed this model for ASTUS [9], an authoring framework for MTTs. ASTUS's knowledge representation system was designed so that it can be manipulated by task independent processes such as the automatic disruption of a task's knowledge model.

In this paper we present our model of the learners' erroneous behaviors. First, we show how ASTUS's knowledge representation system respects Sierra's assumptions. Then, we explain how its procedural knowledge can be disrupted using processes inspired by Sierra. Finally, we describe how the repair strategies defined by Sierra can be adapted to ASTUS's knowledge representation system.

2. Knowledge Representation

The first step towards elaborating our model was to make sure that the knowledge representation system used by ASTUS is compatible with the Sierra theory [8]. This theory enumerates assumptions regarding its knowledge representation approach. Even though ASTUS's approach is significantly different from Sierra's, we can show that it is compatible with Sierra's assumptions. In this section, we specify the relevant assumptions for our model and explain how ASTUS implements them.

Sierra is a theory for procedural errors and thus does not define a specific representation for semantic knowledge, but it asserts that the procedural knowledge contains patterns used to access semantic knowledge. The *pattern assumption* indicates that those patterns are defined using a formalism that can be interpreted by the system (Sierra uses predicate calculus). In ASTUS, the patterns are queries that can be used by the procedural knowledge in order to access elements from the knowledge base such as concepts (pedagogically relevant abstraction) and relations (n-ary predicates defined over concepts). Their structure can be manipulated by the system to disrupt a task's procedural knowledge and generate erroneous behaviors.

Sierra's *recurrence assumption* specifies that, during the execution of a task, a goal stack is minimally required to describe its control regime. In ASTUS, the control regime is described using a goal tree, a structure containing more information than a stack. In addition to the current goals and the procedures that can be applied to achieve them, this episodic tree (figure 14) also contains all the previously satisfied goals and goals already planned.

Sierra also specifies the *"goal types" assumption* which asserts that the execution of each goal is determined by its type (AND, OR and FOR-EACH). Those three types were chosen to accurately model the learner's cognitive processes. ASTUS's knowledge representation system also offers multiple execution types, but it differs from Sierra in two main aspects. First, ASTUS separates the goals and the procedures applied to achieve them. Thus, the execution behavior is associated to procedures rather than to goals. Second, ASTUS's execution types were designed to model the teacher's instructions. ASTUS offers different types of procedures such as sequences, equivalent to Sierra's AND goals; selections, equivalent to OR goals; iterations, analogous to FOR-EACH goals; and primitives that represent atomic actions in the learning environment's user interface. Since a procedure's type is explicitly defined, it can be used when disrupting a procedure to generate its possible incorrect executions.

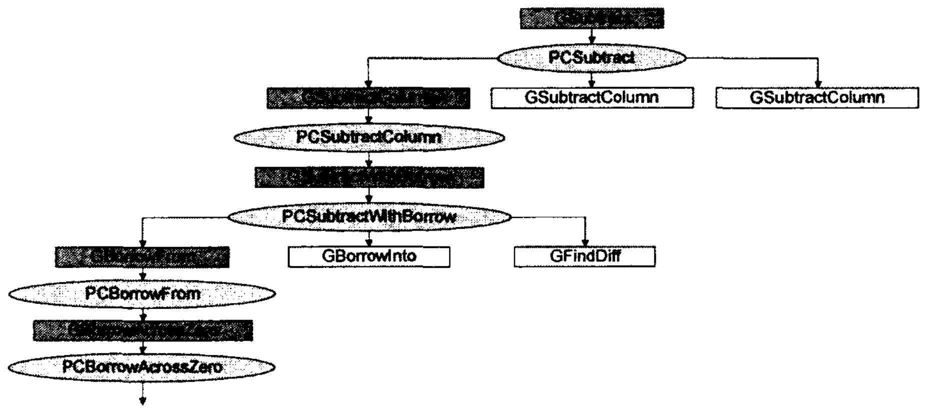


Figure 14 - Part of an episodic tree for a subtraction tutor, rectangles are goals and ovals are procedures. Only the currently active goals are expended

3. Modeling Erroneous Knowledge

Sierra [8] simulates learning and generates knowledge models that learners could have acquired after a set of lessons. Those models produce the correct solution for problems covered by the lessons used to generate it, but might produce erroneous behaviors when applied to problems from subsequent lessons. The errors might be the result of overly specific patterns or of attempting to solve problems that requires procedures introduced only in subsequent lessons. Sierra also simulates incomplete learning by generating models resulting from the removal of sub-goals in AND goals.

ASTUS's knowledge representation system can be used to model erroneous behaviors by disrupting the expert's model of the task to produce incomplete and incorrect models. This objective is achieved by applying principles analogous to Sierra's but adapted to a knowledge representation system designed for MTTs. We first defined, for each type of procedural units, the knowledge required for their correct execution. Then, we enumerated all the disruptions that can be applied to each unit and we described the resulting erroneous behaviors.

This process can be illustrated using conditional procedures which are selection-based procedures where each goal is paired with a logical condition. To successfully execute a conditional procedure, one must know each of these goals and also know the conditions specifying when to use each one of them. During the execution of the procedure, the conditions are evaluated and the goal associated to the first condition that evaluates to true is activated.

Conditional procedures can be disrupted in two ways to represent erroneous behaviors: 1) each sub-goal can be removed from the procedure and 2) each condition can be disrupted. When removing a sub-goal, its associated condition will also be removed thus preventing it from being selected. There are two different ways to disrupt a condition. First, the disrupted condition can be too specific. In this case, the conditional procedure might not have any active condition for some of the problems. Second, the disrupted condition can be too generic. In this case, the conditional procedure might have more than one active condition at the same time. In both cases, if the number of active conditions is not exactly one, the execution of the procedure will need to be repaired by choosing one goal to achieve.

We illustrate the process of disrupting a procedure with an example from a subtraction tutor. The task of subtraction was chosen since it was used to develop Sierra, but the same process can be applied to any task modeled with ASTUS. The procedure taken as an example is the one that determines how to proceed in order to find the difference of a specific column. Three sub-goals are available: find the difference of the column's terms, borrow from the next column before finding the difference or copy the minuend as the difference.

```
Conditional 'PCSubtractColumn' achieves 'GSubtractColumn' {
  if 'c' instanceOf 'TopGreaterColumn'
    goal 'GFindDiff' with 'c'
  if 'c' instanceOf 'TopSmallerColumn'
    goal 'GSubtractWithBorrow' with 'c'
  if 'c' instanceOf 'NoSubtrahendColumn'
    goal 'GCopyMinuend' with 'c'
}
```

Any of the sub-goals from this procedure can be removed to produce erroneous behaviors. For example, the 'GSubtractWithBorrow' goal can be removed to produce:

```
Conditional 'PCSubtractColumn' achieves 'GSubtractColumn' {
  if 'c' instanceOf 'TopGreaterColumn'
    goal 'GFindDiff' with 'c'
  if 'c' instanceOf 'NoSubtrahendColumn'
    goal 'GCopyMinued' with 'c'
}
```

When using this version of the procedure, any subtraction problem that requires borrowing will be executed incorrectly. Indeed, there are no conditions that will be evaluated to true when the subtrahend of a column is greater than its minuend. This can lead to errors such as *doesn't-borrow* (described in [8]).

In addition to removing goals, the procedure could be disrupted by modifying its conditions. For instance, the condition for the 'GFindDiff' goal could be made too specific by adding the restriction that the column must also be the units' column:

```
(c isA TopSmallerColumn) and (c isA UnitsColumn)
```

This disturbed condition can lead to the *copy-top-except-units error* [8] by causing erroneous behaviors when subtracting any column, except the units, if the subtrahend is not blank.

We applied a similar process for each type of procedures available in ASTUS's knowledge representation system. The result is a specification of how each one can be disrupted in order to model the learners' erroneous behaviors.

The execution of a disrupted task's model can lead to two kinds of erroneous behaviors. A model can be executed apparently without any issue but resulting in an incorrect step sequence or its execution might lead to impasses (situations for which the knowledge model can't be executed furthermore) that will need to be repaired.

4. Repairs

The occurrence of impasses blocks the execution of the knowledge model. In order to resume problem solving when faced with an impasse, learners will try to repair it by applying known procedural knowledge. The Sierra theory [8] includes three repair strategies that learners might use:

- No-op: the goal causing the impasse is not executed. This is achieved by popping the first goal from the goals stack.
- Back-up: very similar to no-op, but pops the goals stack more than once.
- Barge-on: alter the results of patterns in order to obtain a result that enables the execution of the knowledge model.

Each of these repair strategy can be adapted to ATSUS's knowledge representation system. The no-op and the back-up strategies can be simulated by going back to a previous goal in the episodic tree (figure 14). The barge-on strategy requires the modification to the results of queries and to the evaluation of conditions in order to produce results compatible with the execution of the knowledge model.

Examples taken from our subtraction tutor can be used to illustrate how each of the three repair strategies can be applied in ASTUS. If a learner does not know how to borrow from a column containing a zero as its top digit and the current problem is 203-107, the resulting erroneous behaviors will vary depending on the applied repair strategy. Figure 14 shows part of the episodic tree for this problem's initial state.

If the no-op strategy is used, the learner will omit borrowing from the tens column (GBorrowAcrossZero), but will still add ten to the units column (GBorrowInto) and find the column's difference correctly (GFindDiff). The result of the subtraction will be 203 - 107 = 106. This erroneous behavior is an instance of the *borrow-no-decrement* error described in [8].

If the applied repair strategy is back-up, the learner can go back to any of the previous goal contained in the episodic tree. He might back-up as far as the initial goal (GSubtract) of subtracting the whole problem and skip the units' column. This behavior would produce the answer 203-107 = 10_, an example of the *blank-instead-of-borrow-from-zero* [8] error. He might also back-up to the earlier goal deciding whether he should borrow or not (GSubtractColumn), and try to solve the problem without borrowing for the units' column. This will cause a second impasse since the difference for 3-7 is a negative number.

To repair this second impasse, the learner could use the barge-on repair strategy. This strategy can be used to modify the result of the queries used to find the difference 3-7 by inverting its argument. The new query would then find the difference 7-3, thus subtracting the top number from the bottom one. The use of this repair would produce 203-107 = 104 as its answer and correspond to the *smaller-from-larger-instead-of-borrow-from-zero* error [8].

5. Conclusion

In this paper, we showed how an MTT authoring framework can model the learners' erroneous behaviors using task independent processes. This is achieved by specifying how ASTUS's procedural knowledge can be disrupted and how impasses can be repaired to produce behaviors analogous to those generated by Sierra.

The model presented in this paper will have multiple applications in ASTUS. For example, it could be used to help assess the learners' knowledge mastery or to facilitate the development of a simulated learner.

Our next objective will be to validate our model by using it to diagnose the erroneous behaviors of a leaner while solving a problem. It will allow us to evaluate the pedagogical relevance of our approach by providing negative feedback on errors, one of Ohlsson's nine learning mechanisms [4].

References

[1] Aleven, V.: Rule-Based Cognitive Modeling for Intelligent Systems, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 33-62.

[2] Fortin, M., Lebeau, J.-F., Abdessemed, A., Courtemanche, F., Mayers, A.: A Standard Method of Developing User Interfaces for a Generic ITS Framework, *Proceedings of ITS 2008*, Woolf, B.P., Aïmeur, E., Nkambou, R., Lajoie, S. (Eds), Lecture Notes in Computer Science 5091, Springer-Verlag, 2008, pp 312-322.

[3] Corbett, A.T., Anderson, J.R.: Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge, *User Modeling and User-Adapted Interaction*, 4, 1995, pp 253-278.

[4] Ohlsson, S.: *Deep Learning: How the Mind Overrides Experience*, Cambridge University Press, 2011.

[5] Mitrovic, A.: Modeling Domains and Students with Constraint-Based Modeling, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 63-80.

[6] Baffes, P., Mooney, R.: Refinemet-Based Student Modeling and Automated Bug Library Construction, *Journal of Artificial Intelligence in Education*, 7(1), pp 75-116.

[7] Mitrovic, A., Koedinger, K., Martin, B.: A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modeling, *Proceedings of User Modelling 2003*,

Brusilovsky, P., Corbett, A., de Rosis, F. (Eds), Lecture Notes in Artificial Intelligence 2702, Springer-Verlag, 2003, pp 313-322.

[8] VanLehn, K.: Mind Bugs: The Origin of Procedural Misconceptions, MIT Press, 1990.

[9] Paquette, L., Lebeau, J.-F., Mayers, A.: Authoring Problem-Solving Tutors: A Comparison Between CTAT and ASTUS, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 377-405.

Conclusion

Contributions

La principale contribution de la présente thèse est la création d'une méthode permettant aux MTTs créés avec Astus de générer automatiquement des indices par rapport à la prochaine étape et des rétroactions négatives. Cette génération a l'avantage de permettre au tuteur de fournir des interventions pédagogiques sans que son auteur ait à inclure de message d'aide dans le modèle de la tâche enseignée.

Les contributions informatiques de la thèse incluent des méthodes et algorithmes reliés aux interventions pédagogiques produites par Astus. L'article présenté au chapitre 2 décrit un ensemble de caractéristiques du système de représentation des connaissances utilisé par Astus qui facilite la génération d'interventions. De même, ce chapitre décrit un modèle permettant d'exploiter le contenu du modèle de la tâche enseignée afin de générer des séquences d'indices expliquant à l'apprenant la prochaine étape qu'il doit effectuer afin de résoudre la tâche.

Le chapitre 4 discute d'un modèle qui a été conçu afin de faire un parallèle entre les caractéristiques du système de représentation des connaissances procédurales définies par Sierra [20] et celles de la représentation utilisée par Astus. Ce modèle décrit aussi comment les connaissances procédurales d'Astus peuvent être perturbées afin de modéliser les erreurs des apprenants. Ce processus est analogue au processus d'impasses et de réparations décrit par Sierra [20]. Ce modèle a permis l'élaboration d'un algorithme permettant à Astus de diagnostiquer les erreurs des apprenants à partir de leurs actions hors trace (chapitre 2). Une méthode a été conçue afin d'utiliser ce diagnostic pour générer une rétroaction négative par rapport aux erreurs des apprenants.

Les contributions empiriques de la présente thèse découlent des cinq expérimentations qui ont été réalisées à l'aide de la plateforme Astus. Les trois premières expériences s'intéressaient à la génération d'indices par rapport à la prochaine étape et ont permis de montrer que 1) l'utilisation des indices générés par Astus permet d'obtenir des gains d'apprentissage plus importants que l'utilisation d'un tuteur sans indices et que 2) il est possible de générer des indices qui sont aussi efficaces et aussi appréciés que ceux écrits par un enseignant.

Les deux dernières expérimentations s'intéressaient à la génération de rétroactions négatives. Malgré le fait que ces expérimentations n'ont pas montré que les rétroactions négatives ont un impact positif statistiquement significatif, elles ont néanmoins montré que 1) les tuteurs développés à l'aide d'Astus sont en mesure de diagnostiquer un grand nombre d'erreurs, 2) les étudiants qui ont reçu des rétroactions négatives ont commis moins d'erreurs que ceux qui n'en ont pas reçu et 3) les étudiants qui ont reçu des rétroactions négatives avaient, en moyenne, un plus grand gain d'apprentissage (cette différence n'est pas statistiquement significative).

Critique du travail

Malgré que les indices générés par les méthodes développées lors de la présente thèse ont été bénéfiques pour les étudiants, peu d'efforts ont été faits dans le but de déterminer de manière formelle le contenu optimal que devraient avoir les interventions générées. De même, peu d'efforts ont été faits dans le but d'utiliser des techniques avancées de génération d'indices. Les indices sont générés à partir de simples gabarits qui sont remplis à partir du contenu du modèle de la tâche. Ceci fait en sorte que certains messages peuvent être difficiles à lire et à comprendre.

L'algorithme de diagnostic des actions hors trace permet de détecter un grand nombre d'erreurs. Par contre, la version actuelle de ce diagnostic gère de manière très naïve les cas où une action hors trace peut être interprétée de plusieurs façons différentes. Dans ce cas, l'algorithme choisi le diagnostic relié à la procédure qui est la plus proche d'une feuille dans l'arbre épisodique. Des méthodes probabilistes auraient pu être utilisées afin d'estimer la probabilité de chacun des diagnostics en fonction d'une évaluation des connaissances de l'apprenant.

Les cinq études empiriques ont été réalisées par rapport à seulement deux tâches différentes. Il n'est pas certain que les résultats obtenus puissent être généralisés à l'ensemble des tâches pouvant être modélisé à l'aide d'Astus. Il est possible que les interventions générées par Astus soient seulement efficaces lorsqu'elles sont utilisées dans un contexte spécifique. D'ailleurs, les tâches modélisées sont toutes deux reliées à l'informatique. Les tuteurs ont donc été utilisés par des étudiants en informatique qui sont très compétents pour interpréter des messages générés par un ordinateur. Cela réduit l'impact de la lisibilité des messages générés par Astus sur les gains d'apprentissage. De plus, les expérimentations effectuées lors de la présente thèse avaient une faible puissance statistique puisque les cours donnés au Département d'informatique à Sherbrooke contiennent peu d'étudiants.

Travaux futurs de recherche

Les travaux futurs peuvent être séparés en trois catégories : l'amélioration des interventions générées, l'amélioration de la validation empirique et la génération de types d'intervention supplémentaires. Les indices par rapport à la prochaine étape pourraient être améliorés en développant un modèle permettant d'adapter les indices générés aux caractéristiques de l'apprenant et aux situations d'apprentissages. La lisibilité des indices pourrait être grandement améliorée par l'utilisation de techniques de génération de langage naturel.

Les rétroactions négatives pourraient être améliorées par une meilleure gestion des cas où plusieurs diagnostics sont disponibles pour une même action hors trace. Pour y arriver, Astus devra estimer la maîtrise de la tâche par l'apprenant afin d'évaluer quelle unité de connaissance a la plus grande probabilité d'être la source de son erreur. Il serait aussi possible de modifier l'algorithme de diagnostic des erreurs afin d'être capable de diagnostiquer un plus grand nombre des actions hors trace. Cet objectif peut être atteint en permettant les diagnostics partiels. Actuellement, Astus diagnostique une étape comme étant une erreur seulement si celle-ci correspond de manière exacte (même procédure primitive et mêmes valeurs d'arguments) à une étape interpolée par Astus. Cette restriction pourrait être allégée en permettant le diagnostic lorsque l'étape commise est très proche de celle interpolée. Par exemple, si elles correspondent à la même procédure primitive, qu'elles diffèrent par seulement un argument et que l'argument qui diffère correspond à une valeur primitive, alors Astus pourrait considérer que l'apprenant a tenté de réaliser cette procédure primitive, mais qu'il a fait une seconde erreur lors du choix de la valeur. Dans ce cas, Astus pourrait fournir une rétroaction négative appropriée. Un tel diagnostic serait utilisé seulement lorsqu'aucune autre perturbation de l'arbre épisodique ne peut expliquer l'erreur de l'apprenant.

Les résultats empiriques obtenus pourraient être validés davantage en étant reproduits à l'aide de tuteurs enseignant une plus grande variété de tâches. Entre autres, ces expérimentations supplémentaires permettraient d'évaluer si les interventions générées par Astus sont aussi efficaces lorsqu'elles sont fournies à des étudiants qui n'ont pas de connaissance en informatique.

Il serait intéressant pour le groupe Astus d'étudier comment les méthodes de génération d'interventions développées lors de la présente thèse peuvent être appliquées à d'autres types d'interventions pédagogiques. Par exemple, il serait possible d'évaluer si Astus est en mesure de générer des exercices de type « étude d'exemple » [11], des demandes d'auto-explication [10, 3] et des explications par analogies à d'autres instances de la tâche [17].

Perspective

Malgré que la recherche sur les MTTs a considérablement diminué au profit d'autres types de STI [4, 7, 14] dans les dernières années, les *Cognitives Tutors* demeurent les tuteurs les plus utilisés en classe. Améliorer l'efficacité des MTT a donc un grand potentiel pour l'amélioration de l'apprentissage des étudiants qui utilisent ces tuteurs de façon régulière. C'est pourquoi le groupe Astus s'est donné l'objectif de faire évoluer les MTT.

Les méthodes et algorithmes présentés dans la présente thèse sont un premier pas vers la création de tuteurs capables d'intervenir de façon personnalisée et adaptée aux situations d'apprentissages auxquelles font face les apprenants. Plus les tuteurs seront en mesure de générer des interventions de type, de format et de contenu différents, plus ils seront en mesure de mettre en œuvre des stratégies pédagogiques sophistiquées maximisant l'apprentissage.

Bibliographie

[1] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R, The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains, *Proceedings of ITS 2006*, Ikeda, M., Ashlay, K., Chan, T.-W. (Eds), Lecture Notes in Computer Science 4053, Springer-Verlag, 2006, pp 61-70.

[2] Aleven, V., Sewall, J., McLaren, B.M., Koedinger, K.R., Rapid Authoring of Intelligent Tutors for Real-World and Experimental Use, *Proceedings of ICALT 2006*, 2006, pp 847-851.

[3] Aleven, V., Koedinger, K.R., An Effective Metacognitive Strategy: Learning by Doing and Explaining with a Computer-Based Cognitive Tutor, *Cognitive Science*, 26(2), 2008, pp 147-179.

[4] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K.R., Example-Tracing Tutors: A New Paradigm for Intelligent Tutoring Systems, *International Journal of Artificial Intelligence in Education*, Special Issue on "Authoring Systems for Intelligent Tutoring Systems", 2009, pp 105-154.

[5] Anderson, J.R., Pelletier, R., A Development System for Model-Tracing Tutors, *Proceedings of the International Conference of the Learning Sciences*, 1991, pp 1-8.

[6] Anderson, J.R., Corbett, A.T., Koedinger, K.R., Pelletier, R., Cognitive Tutors: Lessons Learned, *The Journal of the Learning Sciences*, 4(2), 1995 pp 167-207.

[7] Barnes, T., Stamper, J., Croy, M., Lehman, L., A Pilot Study on Logic Proof Tutoring Using Hints Generated from Historical Student Data, *Proceedings of EDM 2008*, Baker, R., Barnes, T., Beck, J. (Eds.), 2008, pp 197-201.

[8] Chi, M., Jordan, P., VanLehn, K., Litman, D, To Elicit or to Tell: Does it Matter?, *Proceedings of the 2009 Conference on Artificial Intelligence in Education*, Dimi-

trova, V., Mizoguchi, R., Du Boulay, B., Graesser, A. (Eds.), IOS Press, 2009, pp 197-204.

[9] Chi, M., VanLehn, K., Litman, D., Do Micro-Level Tutorial Decisions Matter: Applying Reinforcement Learning to Induce Pedagogical Tutorial Tactics. *Proceedings of ITS 2010 Part I*, Aleven, V., Kay, J., Mostow, J. (Eds), Lecture Notes in Computer Science 6094, Springer-Verlag, 2010, pp 224-234.

[10] Conati, C., VanLehn, K., Teaching Meta-Cognitive Skills: Implementation and Evaluation of a Tutoring System to Guide Self-Explanation while Learning from Examples, *Proceedings of AIED 1999*, Lajoie, S., Vivet, M. (Eds), IOS Press, 1999, pp 297-304.

[11] McLaren, B.M., Isotani, S., When is it Best to Learn with All Worked Examples?, *Proceedings of AIED 2011*, Biswas, G., Bull, S., Kay, J., Mitrovic, A. (Eds), Lecture Notes in Artificial Intelligence 6738, Springer-Verlag, 2011, pp 222-229.

[12] Mitrovic, A., A Knowledge-Based Teaching System for SQL, *Proceedings of ED-MEDIA '98*, Ottmann, T., Tomek, I. (Eds.), 1998, pp 1027-1032.

[13] Mitrovic, A., Weerasinghe, A., Revisiting Ill-Definedness and the Consequences for ITSs, *Proceedings of AIED 2009*, Dimitrova, V., Mizoguchi, R., Du Boulay, B., Graesser, A. (Eds.), IOS Press, 2009, pp 375-382.

[14] Mitrovic, A.: Modeling Domains and Students with Constraint-Based Modeling, in *Advances in Intelligent Tutoring Systems*, Nkambou, R., Bourdeau, J., Mizoguchi, R. (Eds), Studies in Computational Intelligence 308, Springer-Verlag, 2010, pp 63-80.

[15] Murray, T., An Overview of Intelligent Tutoring System Authoring Tools: Updated Analysis of the State of the Art, in *Authoring Tools for Advances Learning Environment*, Murray, T., Blessing, S., Ainsworth, S. (Eds), Kluwer Academic Publishers, 2003, pp 491-544.

[16] Nkambou, R., Bourdeau, J., Mizoguchi, R., Advances in Intelligent Tutoring Systems, Studies in Computational Intelligence 208, Springer-Verlag, 2010.

[17] Ohlsson, S., Computational Models of Skill Acquisition, in *The Cambridge Handbook of Computational Psychology*, Sun, R. (Ed.), Cambridge University Press, 2008, pp 359-395.

[18] Ohlsson, S.: *Deep Learning: How the Mind Overrides Experience*, Cambridge University Press, 2011.

[19] VanLehn, K., Lynch, C., Schultz, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., Wintersgill, M., The Andes Physics Tutoring System: Lessons Learned, *International Journal of Artificial Intelligence in Education*. 15, 2005, pp 147-204.

[20] VanLehn, K., *Mind Bugs: The Origin of Procedural Misconceptions*, MIT Press, 1990.