Un modèle pour les algorithmes avec retour sur trace dans les tuteurs par traçage de modèles

par

Gabriel Beaulieu

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, 26 Otcobre 2016

Le 26 Octobre 2016-10-27

le jury a accepté le mémoire de Monsieur Gabriel Beaulieu dans sa version finale.

Membres du jury

Professeur André Mayers
Directrice / Directeur de recherche
Département d'informatique

Professeur Sylvain Gyroux
Membre interne
Département d'informatique

Professeur Roger Nkambou
Membre externe
Département d'informatique
Université du Quebec à Montréal (UQAM)

Professeure Hélène Pigot
Président-rapporteur
Département d'informatique

Sommaire

La présente thèse décrit des travaux de recherches effectués sur des systèmes tutoriels intelligents (STI) et plus précisément sur les tuteurs par traçage de modèle (MTT). Les MTTs sont des logiciels qui assurent le tutorat d'étudiants au cours de la résolution de tâches pour lesquelles il existe des algorithmes afin de parvenir à une solution. Cependant, les coûts de développement freinent leur essor. Ces coûts peuvent être réduits en utilisant des plateformes de conception mais les système de représentation de connaissances de ces plateformes limitent parfois l'étendue des domaines pour lesquels il est possible de concevoir un MTT. De plus, les MTT produisent parfois des interventions pédagogiques inappropriées lorsque le modèle de la tâche est complexe.

Les travaux de recherche présentés ici s'intéressent à la conception de MTT pour des domaines dans lesquels les étudiants peuvent résoudre la tâche qui leur est assignée de plusieurs façons. Ces domaines comportent parfois des algorithmes avec retour sur trace lorsque l'étudiant ne sait pas forcément quelles sont les alternatives qui feront progresser correctement l'état de la tâche. En effet, ces algorithmes décrivent comment construire une solution à un problème en associant des décisions à des données, puis éventuellement en révisant les décisions au fur et à mesure de la progression de la tâche.

Cette thèse présente dans un premier temps un système de représentation de connaissances pour les algorithmes avec retour sur trace qui rend les connaissances de cet algorithme exploitables par des agents logiciels. Elle présente dans un second temps un ensemble de processus qui exploitent ces connaissances dans le cadre de MTT pour assurer automatiquement le suivi de l'étudiant et ainsi que la production d'interventions pédagogiques. Ces interventions consistent à fournir à l'étudiant:

- de l'aide pour la prochaine étape qui lui explique quelles sont les possibilités dont il dispose et comment déterminer laquelle est la meilleure;
- des rétroactions stratégiques qui lui confirment que son action est valide tout en l'informant de l'existence d'une meilleure alternative le cas échéant;
- des rétroactions négatives qui lui apprennent dans quelles situations les actions invalides qu'il vient d'effectuer s'appliquent.

Une expérimentation a été réalisée avec des étudiants de biologie de l'Université de Sherbrooke pour évaluer les effets de ces interventions sur les choix des étudiants au cours de la résolution de la tâche. Les résultats de cette expérience montrent que les étudiants bénéficiant de ces interventions effectuent plus souvent des choix optimaux, et démontrent ainsi une plus grande maîtrise du domaine [19].

Mots clés: Systèmes tutoriels intelligents, tuteurs par traçage de modèle, interventions pédagogiques, buts de succès, algorithmes avec retour sur trace.

Remerciements

J’aimerais remercier mon directeur de recherche, André Mayers, pour son énorme contribution à ma formation, sa disponibilité, et son aptitude à m’avoir amené jusqu’à la production de ce doctorat.

Merci à tous les membres d’ASTUS, à Luc Paquette et Jean-François Lebeau pour avoir contribué à créer la plateforme sur laquelle j’ai pu travailler et pour les nombreuses et longues discussions sur comment interpréter certaines situations, et comment en gérer d’autres.

Merci à Claude Dery, pour m’avoir autorisé à expérimenter avec les étudiants de son cours, pour m’avoir expliqué les problèmes rencontrés par les étudiants et pour m’avoir décrit ses attentes concernant les étudiants.

Merci à ma famille pour son support tout au long de ma thèse, tant financier que moral : à ma mère Annette, à mon père Cesca, à mon frère Simon.

Merci aussi à ma femme Annabelle Chevrette, sans qui je ne serai pas venu au Québec, et qui m’a soutenu tout au long de ma thèse.

Table des matières

Liste des abréviations

Astus	Apprentissage par Système Tutoriel de l'Université de Sherbrooke
STI	Système Tutoriel Intelligent
MTT	Tuteur par Traçage de Modèle (Model Tracing Tutor)
CT	Tuteur Cognitif (Cognitive Tutor)
CTAT	Cognitive Tutor Authoring Tool
KB	Base de connaissance (Knowledge Base)
PC	Procédure Complexe
PP	Procédure Primitive

Introduction

Contexte

Les systèmes tutoriels intelligents (STI) sont des logiciels qui aident des étudiants à acquérir les connaissances nécessaires à la résolution de problèmes. Ils sont utilisés avec succès dans le cursus de nombreux étudiants, que ce soit pour l'enseignement de requêtes SQL [8], pour l'apprentissage de la physique [21], ou encore pour l'enseignement des mathématiques [2]. Les plateformes de conception telles que TDK [3], CTAT [2] ou Astus [15, 16, 17] réduisent les couts de développement des STI en permettant d'exploiter les connaissances qu'ils contiennent pour automatiser certains de leurs rôles (suivi de l'étudiant, production d'interventions pédagogiques…).

Le point de départ de cette thèse était la volonté de concevoir un tuteur par traçage de modèle (MTT, une catégorie des STI, voir sous-section état de l'art) pour le domaine GNT404 d'analyse d'ADN enseigné à l'Université de Sherbrooke. Ce domaine consiste à localiser l'endroit appelé site de coupure où des enzymes coupent une molécule d'ADN pour l'identifier par comparaison avec une base de donnée de sites de coupure. Le problème qui se pose est qu'une fois la molécule coupée par des enzymes, la position des fragments obtenus sur la molécule de départ n'est pas connue. Il faut donc émettre des hypothèses, les examiner et réaliser d'autres combinaisons d'enzymes pour localiser l'ensemble des sites de coupure. Le défi ici n'était pas de concevoir un MTT mais d'identifier les problèmes de représentation et d'exploitation de connaissances posés des domaines qui offrent une grande liberté à l'étudiant dans la résolution de la tâche. Dans notre cas, l'exploitation des connaissances prend la forme de l'utilisation du modèle de la tâche par le MTT pour accomplir ses rôles intelligemment.

Après étude du domaine GNT404, nous avons identifié que les problèmes rencontrés sont dus à l'absence de construits qui :

1) sont nécessaires à la représentation des algorithmes de retour sur trace. Ce problème se décompose en les sous-problèmes suivants :
    a. L'absence de construits représentant les but du succès. L'objectif des buts de succès est d'atteindre un état de la tâche qui satisfait des conditions. Par conséquent, ces buts peuvent être validés sans efforts ;
    b. L'absence de construits qui simulent le comportement d'un apprenant lorsqu'il fait de multiples tentatives pour atteindre un but de succès. Ces situations se produisent lorsqu'il n'est pas possible de prévoir les conséquences d'une tentative avec des capacités cognitives normales.
2) représentent des connaissances stratégiques, c'est-à-dire de connaissances permettant de choisir parmi un ensemble d'options lesquelles sont les plus pertinentes.

Dans la plateforme Astus, la seule qui offre un modèle de la tâche exploité par le MTT pour générer des interventions pédagogiques [15], la modélisation des buts de succès (problème 1a) aboutit à des interventions redondantes n'apportant aucune information pertinente. Le problème 1b rend ardue la modélisation d'algorithmes de retour sur trace car il est difficile de générer une intervention qui explique à l'apprenant sous quelles conditions poursuivre la construction de la solution ou revenir sur les étapes passées. Enfin, le problème 2 empêche la production d'interventions qui contribuent à l'identification de situations particulières dans lesquelles certaines méthodes plus efficaces[1] s'appliquent. Or, favoriser l'utilisation des méthodes les plus efficaces incite l'étudiant à faire preuve de flexibilité, ce qui caractérise une bonne maitrise du domaine [19].

[1] La notion d'efficacité dépend du domaine modélisé. Il peut s'agir de la méthode nécessitant le moins d'actions de l'apprenant, ou de celle qui minimise son risque d'erreur.

Etat de l'art : Plateformes de conception

Parmi tous les STI qui existent, on distingue deux grands courants : les tuteurs par contraintes [9] (CBT) pouvant être conçus à partir de la plateforme ASPIRE [10, 11], ainsi que les tuteurs par traçage de modèle (MTTs) pouvant être conçus à partir de la plateforme CTAT [2] ou Astus [15, 16, 17]. Dans cette section, nous comparons la plateforme Astus aux plateformes ASPIRE et CTAT pour expliquer d'une part notre choix de travailler avec cette plateforme et d'autre part en quoi nos objectifs poursuivent les précédents travaux effectués sur Astus.

Plusieurs travaux [5, 12] comparent les tuteurs cognitifs (CTs) (sous le terme « model tracing ») produits par CTAT aux CBTs. Mitrovic et al. [12] montrent que les deux approches sont viables en fonction du domaine à modéliser, et présentent un tableau (Tableau 2) qui résume les caractéristiques de ces approches. Kodaganallur et al. [5] pensent que les CBT sont plus simples à concevoir, mais que les tuteurs cognitifs offrent des rétroactions ciblées et de haut niveau (« targeted, high-quality remediation »). Cette conclusion a été critiquée par Mitrovic et Ohlsson [13], essentiellement à cause de la modélisation du domaine ainsi que pour l'utilisation d'un domaine simpliste. Kodaganallur et al. ont répondu point par point à cette critique [6], arguant principalement que leurs conclusions ne dépendent pas de la taille du domaine modélisé et qu'ils ont fourni du matériel qui justifie leur implémentation. Ils ajoutent également que les exemples qu'ils donnent illustrent les problèmes des CBTs quant aux rétroactions. D'une façon assez similaire à celle de Kodaganallur et al. [5] (via la modélisation d'un domaine simple), la plateforme Astus a été comparée aux tuteurs cognitifs [14]. Ces résultats restent d'actualité même si de nombreux développements ont été effectués depuis.

Afin de positionner la plateforme Astus parmi les plateformes de conception, nous avons repris le tableau de Mitrovic et al. [12] et nous y avons ajouté une colonne décrivant les caractéristiques de la plateforme Astus (Voir Tableau 1).

Tableau 1. Comparaison des approches des plateformes de conception en fonction des STI produits

Caractéristique	CTs	CBMs	Tuteurs Astus
Représentation de connaissances	Règles de production (procédural)	Contraintes (déclaratif)	Procedures et buts (procédural)
Fidélité cognitive	Plutôt haute (apprenant)	Plutôt basse (apprenant)	Plutôt haute (expert)
Ce qui est évalué	Actions	Etat de la tâche	Actions
Stratégies de résolution	Celles implémentées	Toutes les stratégies	Celles implémentées (ou autorisées[2])
Solutions	Calculées (peuvent être stockées)	Stockées (peuvent être calculées)	Calculées (peuvent être stockées)
Rétroactions	Immédiates (peuvent être reportées)	Reportées (peuvent être immédiates)	Immédiates (peuvent être reportées)
Aide pour la prochaine étape	Oui	Sur ce qui manque, pas sur la stratégie	Oui
Diagnostic si l'évaluation n'aboutit pas	Solution incorrecte	Solution correcte	Solution incorrecte (mais un diagnostic des actions hors-trace est possible)
Erreurs modélisées explicitement	Oui	Non	Non (mais peuvent l'être)
Implémentation	Plutôt dure mais peuvent être réduits en sacrifiant des avantages	Plutôt simple mais des avantages peuvent être obtenus en complexifiant	Plutôt dure, mais offre de nombreux avantages

[2] Ceci fait l'objet du Chapitre 2

La représentation de connaissance des tuteurs Astus est une représentation hiérarchique de type but-procédure [15] proche des réseaux procéduraux de Sacerdoti [18]. Cette structure procédurale peut être introduite dans les CTs en introduisant la notion de buts, mais les CTs n'exploiteraient pas automatiquement cette hiérarchie pour autant. Contrairement aux CTs et aux CBTs, les tuteurs Astus ne modélisent pas les connaissances de l'apprenant, mais les connaissances d'un tuteur humain qui maitrise le domaine, et ceci afin de pouvoir générer automatiquement les interventions pédagogiques :

- Ils tracent les actions effectuées par l'étudiant [7] ;
- Ils génèrent des indices pour la prochaine étape [14, 15] ;
- Ils diagnostiquent les erreurs de l'apprenant [17] en utilisant une méthode de perturbation des erreurs inspirée de Sierra [20].

Comme les CTs et contrairement aux CBTs, les tuteurs Astus sont des tuteurs par traçage de modèle. Cela implique que les stratégies de résolutions autorisées sont celles modélisées et que de façon générale, les solutions aux problèmes sont calculées grâce à ces stratégies et non pas stockées comme dans les CBT. Cela permet d'avoir de l'aide pour la prochaine étape centrée sur la stratégie de résolution de la tâche ainsi que des rétroactions immédiates dès que l'étudiant fait une erreur.

Les CBT acceptent les solutions de l'étudiant dès que celles-ci ne violent aucune contrainte. Par conséquent, toutes les stratégies de résolution sont autorisées. Inversement, CTs et les tuteurs Astus limitent l'étudiant aux stratégies implémentées. Lorsqu'une action hors-trace est commise, les CBTs la considèrent comme valide (innocent jusqu'à preuve du contraire), CTs et les tuteurs Astus la considèrent comme invalide (coupable jusqu'à prouvé innocent). En revanche, les tuteurs Astus disposent en plus d'un processus de diagnostic des erreurs les plus fréquentes [17] qui perturbe le modèle de la tâche pour expliquer à l'étudiant son erreur si celle-ci est identifiée. Ainsi, comme les CBTs, les tuteurs Astus épargnent à leur concepteur l'encodage explicite des erreurs les plus fréquentes comme les CTs.

De façon générale, les CBTs ont une approche plus simple de la modélisation de domaines. En revanche, ils offrent moins d'atouts que les CTs et les tuteurs Astus, mais ces atouts

peuvent être gagnés en complexifiant l'encodage. Les CTs offrent des outils plus centrés sur les stratégies de résolution de problèmes mais leur conception peut être simplifiée (via les tuteurs par traçage d'exemple) en sacrifiant une partie de ces avantages. Les tuteurs Astus sont plus complexes à produire mais grâce à une modélisation du domaine dont la sémantique est accessible, ils sont en mesure d'automatiser de nombreux processus (interventions pédagogiques, diagnostic des erreurs…) qui doivent être assurés manuellement dans les CBTs et les CTs.

Objectifs

Notre objectif consiste à étendre la représentation de connaissances de la plateforme Astus (appelée dans la suite représentation IJAIED) pour supporter des algorithmes avec retour sur trace. Ainsi, les MTT conçus à l'aide de la plateforme Astus pourront enseigner aux étudiants comment utiliser des algorithmes avec retour sur trace et comment faire de meilleurs choix à l'aide d'interventions adaptées à ces nouvelles situations d'apprentissage.

Il s'agit donc de concevoir:

a) Un schéma procédural permettant la modélisation d'algorithmes de retour sur traces dans les plateformes de conception. Ce schéma doit s'appliquer à l'ensemble des algorithmes avec retour sur trace et sa sémantique doit être accessible aux modules des MTT ;
b) Un ensemble de processus qui exploitent la sémantique de ce schéma pour assurer le suivi de l'étudiant, la production de nouvelles interventions pédagogiques et la détection des situations dans lesquelles ces interventions sont nécessaires.

Bien que nos travaux soient centrés sur la plateforme Astus et la production facilité de MTT, les extensions proposées à la représentation IJAIED sont applicables à tous les systèmes de représentation de connaissances qui disposent d'une structure hiérarchique. Ceci est rendu possible parce que les modélisations proposées dans ces travaux nécessitent uniquement une structure procédurale de type but-procédure pour représenter les algorithmes avec retour sur trace.

Méthodologie

La méthode choisie pour atteindre les objectifs de cette thèse comporte trois phases: modélisation des connaissances, exploitation de ces connaissances pour la génération d'interventions pédagogiques et application de ces processus aux algorithmes avec retour sur trace. Chacune de ces phases sera également validée en réalisant un tuteur complet pour le cours d'analyse d'ADN GNT404 enseigné à l'Université de Sherbrooke.

1. Modélisation des connaissances: cette phase consiste à identifier et concevoir une représentation pour les connaissances nécessaires à la modélisation de domaines contenant des algorithmes avec retour sur trace. Plus précisément il s'agit de représenter les objectifs et les sous-objectifs de ces domaines, mais aussi les procédures qui décrivent comment résoudre les tâches de ces domaines.
2. Exploitation du système de connaissances: cette phase consiste dans un premier temps à enrichir la représentation conçue durant la première phase avec les connaissances qui définissent le caractère optimal des choix de l'étudiant. Dans un second temps, elle consiste à mettre au point les processus qui exploitent ces connaissances pour produire des interventions qui favorisent l'acquisition de ces connaissances. Ces interventions:
    - Décrivent comment déterminer la meilleure alternative et la désignent en cas de besoin dans l'aide pour la prochaine étape;
    - Informent un étudiant qui a fait un choix non-optimal de l'existence d'une meilleure alternative à l'aide d'un nouveau type de rétroaction (rétroaction stratégique);
    - Enseignent à un étudiant qui a fait un choix invalide dans quelles situations son choix est acceptable.

    Enfin, dans un troisième temps, l'influence de ces interventions sera évaluée auprès des étudiants du cours GNT404 au cours d'une expérience comparant les étudiants ayant reçu de telles interventions à des étudiants ne les ayant pas reçues.

3. Application aux algorithmes avec retour sur trace: cette phase consiste à représenter ces algorithmes dans le système de représentation de connaissances IJAIED enrichi grâce aux phases 1 et 2. Grâce à l'utilisation de ce système, les MTT qui utilisent cette représentation pourront produire les interventions pédagogiques conçues durant la phase 2 pour enseigner correctement les algorithmes avec retour sur trace.

Résultats

Cette thèse décrit un système de représentation pour les connaissances contenues dans les algorithmes avec retour sur trace ainsi qu'un ensemble de processus qui exploitent ces connaissances pour automatiser les rôles des MTT. Plus précisément, ces processus permettent la production d'interventions pédagogiques destinées à enseigner ces connaissances. Les contributions apportées par ces travaux sont réparties entre contribution informatique et contribution empirique.

La contribution informatique de cette thèse bénéficie aux plateformes de conception et aux MTT qu'elles produisent. Les MTT produits par les plateformes qui utilisent la représentation de connaissance et les processus élaborés au cours de cette thèse favorisent l'apprentissage des étudiants de plusieurs façons:

- Ils contribuent passivement à l'apprentissage des étudiants qui maitrisent le domaine en assurant leur suivi lorsqu'ils font preuve de flexibilité en s'adaptant à l'état de la tâche;
- Ils contribuent activement à l'apprentissage des étudiants qui ne maîtrisent pas assez le domaine en utilisant des gabarits d'intervention complétés par des éléments de la tâche pour enseigner à l'étudiant comment faire des choix optimaux dans l'aide pour la prochaine étape;
- Ils détectent les erreurs communes des étudiants et produisent des rétroactions négatives qui lui expliquent son erreur et dans quelles situations l'action qu'il vient de faire est justifiée.

- Ils autorisent les étudiants à choisir le sous-objectif de la tâche qu'ils vont poursuivre tout en empêchant au besoin la poursuite simultanée de deux d'entre eux.

Le processus à l'origine de la production des interventions pédagogique décrite dans cette thèse étend le processus de génération des interventions de Luc Paquette [15] pour l'appliquer à de nouvelles situations d'apprentissage afin de générer des interventions pédagogiques encore plus riches.

D'un point de vue empirique, l'expérience réalisée avec des étudiants en biologie à l'Université de Sherbrooke montre que les nouvelles interventions produites automatiquement par le MTT grâce aux travaux de cette thèse contribuent à un meilleur apprentissage: les étudiants soumis à ces interventions font de meilleurs choix. Ils font donc preuve de plus de flexibilité, ce qui caractérise une meilleure maîtrise du domaine [19].

Structure de la thèse

La présente thèse est composée de trois articles qui constituent les trois phases décrites dans la méthodologie. Le chapitre 1 est un article publié dans *Environnements Informatiques pour l'Apprentissage Humain* (EIAH2015). Il décrit dans un premier temps la phase 1 de la méthodologie avec une représentation des buts de succès et une procédure pour les atteindre. Dans un second temps, il débute la phase 2 en présentant un processus de diagnostic des erreurs relatives à la nouvelle procédure pour produire automatiquement des rétroactions négatives qui expliquent à l'apprenant son erreur.

Le chapitre 2 est un article soumis à *IEEE Transactions on Learning Technologies* (TLT). Il décrit comment enrichir la représentation de connaissances présentée dans le chapitre 1 afin de définir l'optimalité des choix dont dispose un étudiant ainsi qu'un ensemble de processus assurant la production d'interventions qui guident l'étudiant dans ses choix (phase 2). Enfin, il termine avec une expérience évaluant les effets de ces interventions sur les choix d'un étudiant au cours de sa réalisation d'une tâche.

Le chapitre 3 est un article soumis à *International Journal of Artificial Intelligence in Education* (IJAIED). Il décrit un schéma pour représenter les algorithmes avec retour sur trace qui fait intervenir les modèles conçus durant les chapitres 1 et 2. Grâce à cela, les connaissances qui forment l'algorithme sont exploitées par les MTT (phase 3) pour assurer le suivi de l'étudiant et la production d'interventions pédagogiques appropriées.

Chapitre 1

Gestion automatisée des buts de succès dans les MTT

Ce chapitre décrit une représentation des buts de succès qui en facilite la gestion dans les MTT. Elle épargner au concepteur du MTT le recours à plusieurs structures procédurales distinctes pour modéliser un processus unique. Cette représentation facilite la modélisation de domaines confrontant l'étudiant à des choix et comportant des procédures employées entre autres dans les algorithmes avec retour sur trace. De plus, elle permet l'automatisation de certains des rôles des MTT pour simplifier leur conception en exploitant le modèle de connaissances.

Cette automatisation évite au concepteur du MTT l'étiquetage systématique de tous les buts du domaine. Les étiquettes des buts sont utilisées dans des interventions pédagogiques pour informer l'étudiant des objectifs qu'il doit accomplir. En outre, la nouvelle procédure présentée dans ce chapitre permet d'automatiser la détection des erreurs fréquentes ainsi que la production des interventions pédagogiques associées.

L'auteur (Gabriel Beaulieu) a conçu la représentation des buts de succès et de la procédure de tâtonnement tout en permettant une gestion automatisée des erreurs dans le cadre de ces nouvelles structures. Il a également contribué à créer le tuteur pour le cours GNT404 dont sont issus les exemples présentés les articles de cette thèse. La plateforme Astus et sa gestion des erreurs sur les structures déjà disponibles furent mises au point et implémentés par Jean-Francois Lebeau et par Luc Paquette. L'auteur a contribué à 80% au travail lié à la rédaction du chapitre.

Gestion automatisée de buts de succès dans les MTT

Gabriel Beaulieu[1], Luc Paquette[2], André Mayers[1]
[1] UdeS, Université de Sherbrooke, 2500, boul. de l'Université, Sherbrooke (Québec) CANADA J1K 2R1
[2]Teachers College, Columbia University, 525 West 120th Street New York, NY, 10027

Résumé. Les coûts de développement ainsi que la pertinence des interventions pédagogiques freinent l'essor des tuteurs par traçage de modèle (MTT). L'utilisation d'un système hiérarchique de représentation de connaissances permet une génération automatisée d'interventions pédagogiques contextualisées. En revanche, les systèmes hiérarchiques rendent difficile la modélisation d'une catégorie d'intentions (les buts de succès) dont l'atteinte est déterminée par l'état du problème et non pas par une série d'actions de l'étudiant. Cet article présente une modélisation des buts de succès et des comportements liés à ces buts dans un système de représentation de connaissance hiérarchique. Cette modélisation, lorsqu'elle est intégrée à une plateforme de MTT, épargne à l'auteur d'un MTT une grande partie de l'effort manuel nécessaire pour encoder le contenu pédagogique. Un MTT pour l'analyse de l'ADN illustre les précédents propos.

Mots-clés. Tuteur par traçage de modèle, Outil auteur, Intervention pédagogique, But de succès, Tâtonnement

Abstract. Programming costs and pedagogical relevance of interventions reduces model tracing tutors (MTT) usage. The use of a hierarchical knowledge modeling allows MTT to generate automatically contextualized pedagogical interventions. However hierarchical systems complicate the conception of some student intents (achievement goals) which are achieved by reaching a specified problem state rather than by accomplishing a set of actions. This article presents a representation for achievement goals and related student behavior in a hierarchical knowledge representation. The MTT is able to use this representation to prevent manual conception of pedagogical content. The result of these improvements is shown by several examples taken from a MTT used for DNA analyze.

Keywords. Model tracing tutors, Authoring tools, Pedagogical intervention, Achievement goals, Searching procedure

1 Introduction

Les tuteurs par traçage de modèle (MTT) sont des logiciels favorisant l'acquisition de connaissances ou d'habiletés de résolution de problèmes dont l'efficacité a été démontrée à plusieurs reprises [7, 19]. Les rôles d'un MTT sont répartis entre quatre modules : expert, communication, pédagogique et modèle de l'étudiant [10]. Le module expert évalue les étapes[3] [18] de l'étudiant, et en transmet le résultat aux modules pédagogique et modèle de l'étudiant. Le module modèle de l'étudiant met à jour ses informations sur l'étudiant tandis que le module pédagogique détermine une rétroaction, qui sera transmise par le module communication.

Le principal frein à l'essor des MTT provient des coûts de développement, évalués entre 100 et 300 heures pour une heure de matériel pédagogique [6, 10, 21]. Les plateformes de conception de MTT, comme CTAT [1, 2], et Astus [8, 9, 11], remédient à ce problème tout en assurant une pédagogie proche de celle d'un tuteur humain. Néanmoins, ces dernières effectuent des compromis entre la complexité des domaines modélisés, la polyvalence du modèle de représentation des connaissances, et les efforts d'encodage. CTAT, qui produit des tuteurs cognitifs (CT), s'appuie sur des théories cognitives simulant les connaissances procédurales par des règles de production. Les règles de production sont expressives et polyvalentes, mais les interventions pédagogiques, telles que les patrons des messages d'aide, sont spécifiques à une règle et doivent être écrites par le concepteur du MTT. De même, afin d'offrir une rétroaction pédagogique par rapport aux erreurs, ces dernières sont modélisées explicitement à l'aide de règles erronées, ce qui implique de modéliser tous les raisonnements et toutes les étapes pour assurer le suivi de l'étudiant et lui expliquer ses erreurs.

Contrairement à CTAT, le système de représentation de connaissances hiérarchiques[4] d'Astus, similaire à celui proposé par [16] élimine la conception manuelle des interventions. Avec cette représentation, le module pédagogique génère des interventions [14] et le module

[3] Une étape est un ensemble d'action(s) de granularité pédagogiquement intéressante, consistant à créer, modifier ou supprimer des éléments de la tâche.

[4] Ce terme sera abrégé en systèmes hiérarchiques dans la suite.

expert identifie les erreurs fréquentes [12]. La conception de contenu pédagogique est donc plus ardue dans des systèmes à base de règles que dans les systèmes hiérarchiques. Notre travail étant centré sur ces derniers nous désignons sous le terme **efforts de conception supplémentaires,** les efforts supplémentaires qu'effectue le concepteur d'un MTT pour écrire du contenu pédagogique ou pour détecter des erreurs par rapport à ceux nécessaires avec une plateforme comme Astus.

Les intentions d'un étudiant, modélisées à l'aide de buts, peuvent appartenir à plusieurs catégories parmi lesquelles les buts de succès et les buts d'actions [5, 20]. Les buts de succès sont satisfaits par l'état du problème, et peuvent donc être atteints sans intervention de la part de l'étudiant, alors qu'un but d'action est satisfait lorsqu'une séquence d'action appropriée est effectuée[5]. Ainsi dans le cadre d'un déplacement en voiture « se rendre à destination » est un but de succès satisfait lorsqu'on atteint la destination, alors que « faire 20h de conduite » pour passer son permis est un but d'action. De la même façon, dans le cadre de la manipulation de structures de données, « Équilibrer l'arbre AVL » est un but de succès parfois atteint sans actions, alors que « Insérer le nœud dans l'arbre AVL » est un but d'action.

Les systèmes hiérarchiques (dont la plateforme Astus) ne tiennent habituellement pas compte des buts de succès, parce que la majorité des buts d'un domaine s'interprètent comme des buts d'actions ; en effet, ils ne sont pas satisfaits lorsqu'on amorce la résolution du problème et les étapes prescrites par le modèle sont toujours suffisantes pour les atteindre. De plus, si nécessaire, l'ajout d'une procédure conditionnelle, qui ne stipule aucune action si le but de succès est déjà atteint permet de simuler le comportement associé à un but de succès. Cependant, en plus du code supplémentaire, il est souvent difficile de produire des interventions pédagogiques adéquates.

Il est possible d'intégrer et de gérer les buts de succès dans les systèmes hiérarchiques tels que celui de la plateforme Astus. Les objectifs de cet article sont de montrer comment modifier :

1. le module expert pour modéliser des buts de succès ;
2. le module pédagogique pour générer un texte désignant le but ;

[5] Une grande variété de modélisations est présentée dans Bordini & al.[3]

3. le module pédagogique pour générer de l'aide pour les buts de succès ;
4. le module expert pour interpréter le comportement (model tracing) de tâtonnement d'un étudiant essayant d'atteindre un but de succès ;
5. le module pédagogique pour générer des interventions appropriées aux comportements de tâtonnement.

Afin de maintenir la philosophie de la plateforme Astus, nous nous sommes donné deux défis. Le premier : la plateforme Astus ainsi modifiée doit permettre de créer des MTT sans exiger des efforts de conceptions non requis dans un système non hiérarchique. Pour atteindre ce premier défi, nous avons introduit les buts de succès ainsi qu'une nouvelle procédure, la procédure de tâtonnement. À ce nouveau type de procédure s'associent des erreurs génériques qui doivent être détectées par le module expert ; le module pédagogique doit générer des rétroactions pour aider l'apprenant qui fait ce type d'erreurs. Ce sera le second défi.

Comme la conception de MTT-GNT, un MTT pour une partie du cours GNT404, un cours d'analyse d'ADN enseigné à l'Université de Sherbrooke a motivé nos modifications, nous l'utiliserons pour illustrer les bénéfices de ces modifications.

2 Intégration des buts de succès dans un modèle hiérarchique

2.1 Modélisation des buts de succès

Dans la plateforme Astus, les connaissances procédurales sont encodées à l'aide de procédures et de buts. Les buts représentent des intentions, tandis que les procédures les décomposent en un ensemble de buts moins abstraits. Les procédures sont des structures de contrôles (séquence, itération, boucle, conditionnelle) définissant quels sont les buts issus de la décomposition et dans quel ordre l'étudiant doit les atteindre. ces structures de contrôle, qui sont aussi celles de plusieurs langages de programmation, correspondent selon nos observations aux types d'instructions données par l'enseignant qui expliquent comment réaliser une tâche.

Les buts et les procédures pour réaliser une tâche forment le **graphe procédural** de la tâche. À la tête du graphe se trouve le but racine, modélisant l'intention d'effectuer la tâche, et les

feuilles sont des procédures primitives. Ces dernières ne spécifient aucun sous-but et correspondent aux étapes pour accomplir la tâche. Entre le but racine et les feuilles se trouvent les procédures complexes qui spécifient les sous-buts à atteindre et ces derniers spécifient récursivement des procédures pour les atteindre.

L'arbre épisodique résulte de l'instanciation progressive et récursive du but racine par des procédures. Lorsqu'une confusion est possible, nous qualifions les buts et les procédures du graphe procédural de procéduraux et ceux de l'arbre épisodique d'épisodique. Le concepteur d'un MTT définit, dans le graphe procédural, une ou plusieurs procédures pour atteindre un but. Une procédure sous un but est instanciée dans l'arbre épisodique si les arguments que lui passe le but sont appropriés ou s'ils sont accessibles dans la base de connaissances[6]. Les sous-buts définis par une procédure sont instanciés sauf s'il existe des contraintes (le plus souvent d'ordre) spécifiées par la procédure. Les buts épisodiques qui ne sont pas instanciés sont dits *en attente*, et ceux qui sont instanciés sont dits *en cours*, si une étape pour l'atteindre a été effectuée ; *atteint* si toutes les étapes ont été effectuées ; ou, *disponible* dans les autres cas. De façon similaire, les procédures peuvent être dites soit *en attente*, soit *en cours*, soit *exécutée* ou soit *disponible*. Une procédure primitive *disponible* passe à l'état *exécuté* lorsque l'apprenant (ou le MTT, dans le cas d'une démo) effectue l'étape correspondante. Il s'ensuit alors une mise à jour de l'arbre épisodique: les buts *en attente* dont les contraintes sont satisfaites deviennent *disponibles* et les procédures dont tous les buts enfants sont *atteints* deviennent *exécutées*. Noter qu'un même but du graphe procédural peut être instancié plusieurs fois, par exemple, le but défini par une procédure *tant que* va être instancié tant que la condition de la procédure *tant que* est vraie. Inversement, un but *disponible* peut ne jamais devenir *atteint* si ce but est une des alternatives d'une procédure *choix*. L'arbre épisodique est la principale structure sur laquelle se base le MTT pour intervenir auprès de l'apprenant. Elle est la représentation que le MTT se fait des actions antérieures de l'apprenant, et une anticipation par le MTT des prochaines actions de l'apprenant.

---

[6] La base de connaissance est la représentation que se fait le MTT des faits et données immédiatement accessibles à l'apprenant, soit parce qu'ils sont dans la mémoire de travail de l'apprenant ou qu'ils sont visibles dans l'interface.

Les buts définis ainsi sont dits d'action parce qu'ils sont atteints uniquement par une séquence d'étapes définies par l'arbre épisodique qui résulte de son instanciation. Pour simuler un but à succès avec cette représentation, le concepteur crée une procédure conditionnelle où aucun sous-but n'est spécifié lorsque la condition est vraie. Donc, si sa condition est vraie, cette procédure passe instantanément à l'état *exécuté*, et le but est immédiatement déclaré *atteint*. L'ajout d'une procédure conditionnelle rend le modèle plus complexe, mais son inconvénient majeur est que le message généré pour cette procédure n'est pas pédagogique. Il a la forme « puisque *condition*, vous devez *but* » où condition et but contiennent la même information.

Pour éviter ces inconvénients, nous avons modifié le système hiérarchique de représentation des connaissances d'Astus en ajoutant l'attribut condition à la syntaxe des buts. Pour les buts de succès, cet attribut est instancié par une formule logique du premier ordre (nommé formule de but dans la suite). Le processus de mise à jour de l'arbre épisodique a aussi été modifié de telle façon que si, suite à une étape, un but de succès devient *disponible* et que sa formule est vraie alors ce but passe immédiatement à l'état *atteint*.

Avec MTT-GNT, l'étudiant apprend à localiser les sites de coupure d'enzymes sur une molécule d'ADN. Pour comprendre la tâche de l'apprenant, nous introduisons deux notions de base : 1) un enzyme coupe toujours une chaîne d'ADN aux mêmes endroits pour obtenir des fragments mesurables ; 2) une digestion est le processus pour obtenir des fragments à partir d'un ou plusieurs enzymes. Par abus de langage, nous appelons aussi digestion, le résultat du processus, i.e. l'ensemble des fragments.

L'étudiant au cours de sa tâche effectue plusieurs digestions, et des imprécisions surviennent toujours lors de la mesure des fragments. L'une des sous-tâches de l'étudiant est de s'assurer que la somme des mesures de fragments pour chaque digestion soit égale à la mesure de la molécule initiale. Ce but est un but de succès parce que si les fragments sont mesurés correctement (ou si les erreurs de mesures se compensent de façon à rendre ces erreurs indétectables) alors le but est atteint sans effort de la part de l'étudiant. La formule pour ce but est *isInstance(D, AdjustedDigestion)*. *IsInstance* est un prédicat qui est vrai lorsque l'objet sur lequel il porte (ici, la digestion *D*) est du type spécifié (*AdjustedDigestion*). Plusieurs instances de ce but deviennent *disponibles* lorsque l'étudiant termine la mesure des fragments

des digestions, et si la somme des fragments pour une digestion est acceptable alors le but d'ajuster les fragments passe à l'état *atteint*.

Avec les buts de succès, l'arbre épisodique reflète bien la tâche. Le module expert devient ainsi capable de suivre le comportement d'un étudiant qui s'adapte à la tâche en sautant les étapes superflues et qui démontre ainsi une bonne compréhension du domaine [16]. Ce dernier point peut être pris en compte par le module « modèle de l'apprenant » afin d'avoir un modèle de l'apprenant plus précis. Si la formule est fausse, alors le but est décomposé par des procédures comme n'importe quel but d'action.

2.2 Génération automatique de l'étiquette d'un but

La formule de but contient l'information qui décrit l'intention sous-jacente au but, et comme sa syntaxe est comprise par les modules du MTT, le module pédagogique l'utilise pour désigner le but à l'étudiant. Les buts d'action n'ont pas de formule de but définie, et pour communiquer le sens de ces buts à l'apprenant, le module pédagogique utilise une chaîne de caractère, nommée étiquette et encodée avec le but par le concepteur du MTT.

Génération du message. Lorsque le module pédagogique génère de l'aide sur une procédure, il construit son intervention à partir du patron associé au type de la procédure et ce patron spécifie les éléments (buts générés, paramètres ...) pour le compléter. Récursivement, à chacun de ces éléments est associé un patron ou une étiquette créée par le concepteur du MTT. S'il y a un patron de défini pour un élément et que le concepteur lui ajoute une étiquette alors ce dernier a préséance. La partie droite de la Figure 1 est un exemple de message généré pour une procédure séquence.

```
goal('GSelectHyp', contexts:[Hyptest], , readables:[en:'Select the hypothesis']) {
    parameter('Hypothese', id:'H', readables:[en:'hypothesis to select'])
    }
goal('GExamineHyp', contexts:[Hyptest]) {
    parameter('Hypothese', id:'H', readables:[en:'selected hypothesis'])
    objective {known(H.validity)}
}
```

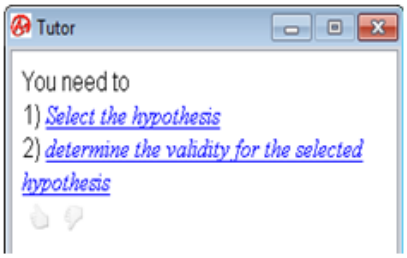

Fig. 1. À gauche un but d'action, GSelectHyp, pour sélectionner une hypothèse, et un but de succès, GexamineHyp, pour tester cette hypothèse. À droite, le message d'aide généré sur la procédure séquence, consistant à sélectionner puis tester une hypothèse.

« You need to [X]*»* est le patron pour une procédure séquence ordonnée et il doit être complété par la liste des buts de la séquence. *« Select the hypothesis »* est l'étiquette de *GselectHyp*, le premier but de la séquence. Le second but, *GexamineHyp* n'est pas étiqueté, mais possède une formule de but dont le prédicat est *known.* Il possède aussi un argument *H* auquel le concepteur a associé l'étiquette *« selected hypothesis »*.

Le patron de *known* est « Determine the [*X*] », et il doit être complété par l'argument de ce prédicat, ici, l'attribut *validity* de l'hypothèse *H*. L'attribut *validity* est étiqueté par *« validity »*, et permet, avec l'étiquette de *H*, de compléter « [X] for the [Y] », le patron utilisé pour désigner l'attribut d'un objet. Il s'ensuit que l'étiquette pour ce second but est « Determine the *validity* for the *selected hypothesis* ».

Dans la pratique, si un but de succès n'est atteint que suite à l'exécution d'une procédure et que la procédure choisie mène toujours à la satisfaction du but alors il n'y a plus de différence dans le comportement entre un but d'action et un but de succès. Le cas échéant, il est souvent plus simple de générer une étiquette appropriée qu'une formule adéquate. Il s'agit du cas du but *GselectHyp*. Ce dernier est un but de succès encodé comme but d'action.

Prédicats et messages générés. Un concepteur de MTT utilisant la plateforme Astus dispose de plusieurs prédicats pour concevoir une formule de but et à chacun de ces prédicats est associé un patron pour générer automatiquement les étiquettes pour ces buts. Nous avons construit de façon progressive cet ensemble des prédicats et il est toujours possible d'en ajouter d'autres. Cependant ceux que nous avons sont suffisants pour générer les étiquettes de

buts de succès pour plusieurs domaines. Voici deux exemples de prédicats et les étiquettes de buts de succès générées, pour les MTT GNT404 et construction d'un arbre AVL.
Le prédicat *known(x)*, utilisé dans la Figure 1, est vrai si *x* (souvent l'attribut d'un objet) est connu. Le patron de message correspondant est *«* Determine the [*X*] *»*. Voici des exemples d'utilisations de ce prédicat, ainsi que les messages correspondants :

- *known(Hypothesis.validity)* génère le message *«* Determine the *validity for the selected hypothesis »*;
- *known(Fragment.length)* génère le message *«* Determine the *length for the studied fragment »*;
- *known(Tree.height)* génère le message *«* Determine the *height for the AVL tree».*

Le prédicat *isInstance(x, t)* est vrai si l'objet *x* est instance du concept *t*. Le patron de message correspondant est *«* Ensure that the [*X*] is a [*T*] *».* Voici des exemples d'utilisations de ce prédicat, ainsi que les messages correspondants :

- *isInstance(doubledigestion, AdjustedDigestion)* génère le message *«* Ensure that the *double digestion* is a *digestion with placed fragments »*;
- *isInstance(SelectedEnzym, PlacedEnzym)* génère le message *«* Ensure that the *selected enzym* is a *enzym with placed sites »*;
- *isInstance(AVLtree, BalancedAVLtree)* génère le message *«* Ensure that the *the working tree* is a *balanced AVL tree »*;

Les expériences passées nous ont montré que le mécanisme utilisé pour décrire des formules logiques est bien compris par les étudiants en sciences, et que les interventions associées sont jugées utiles [15].

Qualités des étiquettes générées. Les étiquettes des buts de succès sont générées à partir de la formule de but. Par exemple, pour le but *GexamineHyp* de la figure 1, la formule est *known(Hypothesis.validity)* et génère l'étiquette *«* Determine the *validity for the selected hypothesis »*, mais si la formule avait été *validity(Hypothesis)* ≠ *unknown* alors l'étiquette aurait été *«* Ensure that the *validity for the selected hypothesis* is not *unknown »*. Ici les

étiquettes sont compréhensibles, mais on peut se demander quels sont les critères à respecter pour que le terme désignant un but soit compréhensible, tout en minimisant la création manuelle d'étiquettes.

Le message généré à partir du patron de formule de but est toujours une phrase. Et, plus une phrase générée automatiquement est complexe ou longue, plus il y a de chances qu'elle soit mal construite, grammaticalement incorrecte, ou éloignée d'une intervention humaine. Il s'ensuit que plus la formule de but est longue et complexe, plus le message généré risque d'être incompréhensible. Une formule de but se représente sous une forme d'arbre dont les nœuds internes sont les patrons et les feuilles sont les étiquettes proposées par le concepteur.
Nos expériences, lors de la modélisation de GNT404, ont montré que la somme de la largeur et de la profondeur de l'arbre constitue un indicateur de clarté. Souvent, lorsque cette somme dépasse 5, le message devient difficile à comprendre et éloigné d'une intervention humaine.
Nous avons aussi constaté que, souvent les experts d'un domaine ont créé un terme pour définir un concept qui se représente autrement par une expression logique complexe. Si ce terme est connu des apprenants, la solution est d'ajouter ce terme au modèle du domaine et de définir des fonctions ad hoc pour ce concept ; le cas échéant, le MTT pourra communiquer à l'apprenant les caractéristiques de ce concept, par exemple il pourra dire qu'il s'agit d'un arbre dont la hauteur du sous-arbre gauche est égale à la hauteur du sous-arbre droit, mais il ne pourra expliquer comment obtenir la hauteur d'un sous arbre, ni justifier l'appartenance d'un objet au concept.
Si le concept n'est pas connu de l'étudiant alors il doit être l'objet de buts et de procédures pour l'enseigner et les messages d'aide pour ces buts et procédure seront plus élaborés. S'il n'existe pas de terme pour un concept et il faut le définir et lui faire correspondre une étiquette simple. Par exemple, pour le concept d'arbre AVL avec des sous-arbres de même hauteur, nous pouvons l'étiqueter « *AVL tree with subtrees of the same height* » et le message associé au but qui vérifie si un arbre est un instance de ce concept est « Ensure that the *working tree* is an *AVL tree with subtrees of the same height* » au lieu de « Ensure that the *height for the left subtree for the working tree* is the same as *the height for the right*

subtree for the working tree ». L'ajout d'un nouveau concept à un modèle, si ce dernier intervient à plusieurs endroits, en simplifie la modélisation, et améliore les interventions.

3 Gestion des comportements de tâtonnement

3.1 Description de la procédure de tâtonnement

Un but de succès induit parfois un comportement de tâtonnement, s'il n'est pas cognitivement possible pour l'apprenant d'entrevoir au préalable la solution au but poursuivi. Pour suivre ce comportement et procurer à l'apprenant une aide optimale, nous avons conçu un nouveau type de procédure, la procédure de tâtonnement. Par exemple, dans la Figure 2, la procédure de tâtonnement nommé "Fouille" montre que pour déterminer la validité d'une hypothèse, l'apprenant a le choix entre trois sous-buts.

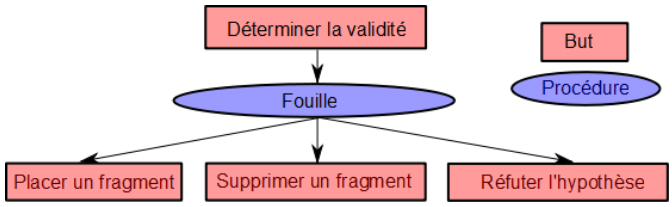

Fig. 2. Arbre procédural pour tester une hypothèse dans GNT404, à l'aide d'une procédure de tâtonnement, intitulé *Fouille*.

Une procédure de tâtonnement spécifie comment atteindre un but par essais successifs. Plus précisément, aux sous-buts spécifiés par la procédure de tâtonnement sont associées soit des procédures correspondant à une façon potentielle d'atteindre le but parent, soit une procédure pour remettre l'interface et la base de connaissance dans un état acceptable pour entreprendre un autre sous-but, soit une procédure qui constate qu'il n'y a pas de solution positive. Cette dernière alternative dans le cas de la figure 2 signifie que l'hypothèse est déterminée fausse.
Lorsqu'une procédure de tâtonnement est instanciée, elle rend disponible un ensemble de sous-buts et, ce sont les futures étapes de l'apprenant qui déterminent lequel de ces sous-buts il a choisi s'il ne fait pas d'erreur. Lorsque le sous-but choisi est atteint, la procédure de tâtonnement vérifie si son but parent a été atteint. Si oui, ce dernier est marqué *atteint* et la

tâche se poursuit normalement ; sinon, la procédure de tâtonnement se poursuit, et instancie un nouvel ensemble de sous-buts. Ces sous-buts, bien qu'instanciés à partir des mêmes buts procéduraux ne sont pas les mêmes puisque, suite au précédent essai l'interface et la base de connaissance ont été modifiées.

Dans le cas de la Figure 2, une hypothèse consiste à dire qu'une enzyme coupe la molécule à une position donnée. Une hypothèse est valide si l'étudiant a trouvé un positionnement pour tous les fragments en respectant son hypothèse et les sites de coupures déjà déterminés. Le tâtonnement consiste à positionner un fragment et, récursivement à voir s'il peut positionner les autres fragments, sinon il choisit un autre fragment. La procédure se termine lorsque tous les fragments sont placés ou que l'étudiant constate qu'au moins un fragment ne peut être placé. Dans le premier cas, la procédure de tâtonnement est terminée, et dans le second, il doit choisir entre retirer le fragment choisi ou réfuter l'hypothèse. Pour éviter des abus, ce dernier choix est considéré comme valide uniquement si la réfutation est triviale ou que l'étudiant a essayé toutes les solutions.

3.2 Message pour la procédure de tâtonnement

Pour la procédure de tâtonnement, deux patrons (voir Figure 3) rappelant à l'étudiant qu'il effectue ce type de procédure ont été conçus. Le module pédagogique choisit le patron en fonction du contexte, i.e. l'arbre épisodique. Si l'étudiant débute la procédure, le patron est « In order to [X], you need to try one of the following options: [Y]. », autrement, il est « You are trying to [X]. Since your last try did not succeed, you need to try one of the following options: [Y]. ». Ces deux patrons sont complétés par l'étiquette du but parent, ainsi que les étiquettes des buts enfants de la procédure.

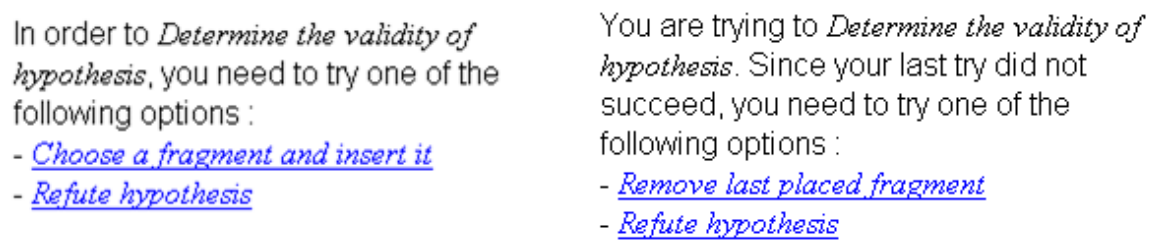

Fig. 3. À gauche, un message pour l'étudiant qui amorce la procédure. À droite, le message pour l'étudiant qui a déjà entrepris la procédure.

Le module pédagogique produit les messages précédents dynamiquement puisque, ce n'est qu'au cours de l'exécution qu'il connaît le but parent, les essais déjà effectués et les sous-buts disponibles. Pour produire de tels messages dynamiquement avec un système de production, il faudrait le modifier pour intégrer les structures syntaxiques d'Astus. Sans ces structures, il est difficile pour un système de production de déterminer si un ensemble de règles modélise une itération, une boucle, ou, dans le cas présent à un comportement de tâtonnement. De plus, si les messages sont conçus manuellement alors en plus de détecter les essais faits et les sous-buts disponibles, il faudrait que le concepteur crée autant de messages qu'il y a de sous-ensembles de sous-buts disponibles.

Il est possible de simuler une procédure de tâtonnement pour un MTT basé sur un système de représentation hiérarchique (sans procédure de tâtonnement) à l'aide d'une procédure de boucle. Cependant, le patron d'une procédure de boucle est de la forme *« Until [X] you have to [Y] »,* complété par le message relatif à la condition de terminaison de la procédure, ainsi que par l'étiquette de son but enfant. Nous retrouvons un message redondant puisque le message associé au sous-but et celui associé à la condition contiennent une information similaire.

3.3 Gestion des erreurs fréquentes

Construire un MTT en mesure d'apporter une aide spécifique aux erreurs de l'étudiant demande beaucoup de temps. En particulier, s'il faut modéliser à la main toutes les erreurs pour les reconnaître, et s'il faut pour chacune de ces erreurs construire les interactions pédagogiques. Certains (e.g. [4]) doutent que les gains pédagogiques contrebalancent ces efforts. Cependant, les systèmes hiérarchiques épargnent ces coûts, car ils permettent une gestion automatisée des erreurs liées au type de procédure. Pour la procédure de tâtonnement, les deux erreurs classiques sont : 1) choisir un but non disponible ; et 2) poursuivre le tâtonnement alors que le but est atteint. La gestion des erreurs associées à la procédure de tâtonnement se fait comme pour les autres types d'erreurs [13], et avec les mêmes

contraintes : les rétroactions doivent être proches de celles d'un tuteur humain, et ne pas exiger d'efforts spécifiques à chaque procédure.

Les rétroactions liées aux erreurs sont formées à partir de patrons. Pour l'erreur consistant à choisir un but non disponible, le module pédagogique choisit entre deux patrons : 1) : *«* In order to try to [X], you choose to [Y]. You already tried to do it and you are not allowed to retry. » ; 2) *«* In order to try to [X], you choose to [Y] . You are allowed to make this choice only under the following conditions: [Z] *».* La première partie de ces patrons est identique. Elle est complétée par les étiquettes du but parent de la procédure ainsi que du but choisi par l'étudiant. Elle a pour but d'attirer l'attention de l'étudiant sur le but parent et la façon dont il a choisi de l'atteindre. La seconde partie du message explique à l'étudiant son erreur. Pour le premier patron (1), l'erreur consiste à refaire un but qui ne peut être refait. Pour le second patron (2), l'erreur est de choisir un but alors qu'il n'est pas disponible, et dans ce cas, le patron est complété par la condition associée à ce but puisque l'étudiant semble l'ignorer. C'est le cas dans GNT404 si l'étudiant tente par exemple de valider l'hypothèse sans l'avoir complétée (voir Figure 4). Le processus pour détecter quel but non disponible a été choisi par l'étudiant a déjà été décrit [13] et il s'appuie sur la théorie de la réparation [17] qui propose des types d'erreurs cognitivement plausibles par un étudiant. Brièvement, ce processus consiste à instancier des buts non disponibles suite à la modification des arguments du but, ou des conditions et de voir quelles combinaisons de ces modifications expliquent de façon la plus cognitivement plausible l'étape non prévue de l'étudiant.

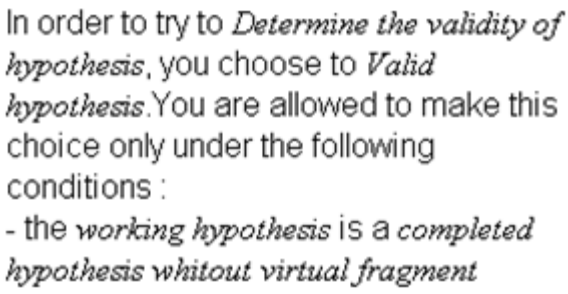

Fig. 4. Exemple d'intervention si l'étudiant choisit un sous-but dont la condition n'est pas valide.

Enfin, la dernière erreur consiste à poursuivre la procédure de tâtonnement alors que le but de succès parent est satisfait. Dans ce cas, la rétroaction est proche à celle qui se produit lorsqu'un étudiant continue une procédure de boucle terminée : *«* You already reached *[X].* You do not have to try again. *».* Ce patron est complété par l'étiquette du but parent de la procédure de tâtonnement.

4 Conclusion

L'intégration des buts de succès dans un système de représentation de connaissance hiérarchique améliore les performances pédagogiques des MTT produits et épargne l'encodage manuel de contenu pédagogique. Les bénéfices observés dans MTT-GNT sont : 1) le suivi d'un étudiant qui fait preuve d'adaptation en sautant les étapes non nécessaires ; 2) une amélioration des interventions générées ; 3) la génération des messages pour décrire les buts de succès ; 4) l'identification des erreurs liées à la procédure de tâtonnement, et la génération d'un message d'aide approprié. Bien que centré sur Astus, notre modèle des buts de succès et des procédures de tâtonnement est compatible avec tous systèmes hiérarchiques de représentation des connaissances.

La modélisation de buts de succès à l'aide d'une formule logique est nécessaire pour interpréter correctement l'étudiant qui exécute une procédure spécifiant des buts de succès ou pour une exploitation de cette formule par les modules du MTT. Dans les autres situations, il est plus simple de modéliser un but de succès sous forme de but d'action étiqueté

manuellement puisqu'une intervention en langage naturel est plus facile à concevoir qu'une formule logique du premier ordre.

Bien que les messages générés soient bien compris des étudiants et que ceux-ci les trouvent utiles, il est possible de les améliorer dans deux directions. La première est celle de l'utilisation des connaissances en linguistique computationnelle et en base de données pour améliorer la qualité de la langue. Il s'agit de modifier dynamiquement les patrons pour 1) utiliser la forme active lorsque celle-ci existe et qu'elle est appropriée ; et 2) de produire plusieurs phrases lorsque la formule de but est complexe ou trop longue. La seconde est de modifier le ton du message en fonction du profil de l'étudiant et de l'état du problème. Par exemple, le module pédagogique pourrait inciter un étudiant à demander de l'aide s'il s'avère que la tâche est ardue, ou au contraire, à réfléchir davantage, s'il adopte un comportement qui laisse croire qu'il utilise l'aide pour arriver rapidement à la solution.

Références

1. Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K. R.: The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In: M. Ikeda, K. D. Ashley, T. W. Chan (Eds.): Proceedings of the 8th International Conference on Intelligent Tutoring Systems (2006) 61-70
2. Aleven, V.: Rule-Based Cognitive Modeling for Intelligent Tutoring Systems. In: Nkambou, R., Mizoguchi, R. Bourdeau, J. (Eds.): Advances in Intelligent Tutoring Systems, Springer (2010)
3. Bordini, R.H., Dastani, M., Dix, J., Elfallahseghrouchni, A.: Multi-Agent Programming: Languages, Platforms and Applications. Number 15 in Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer (2005)
4. Corbett, A.T., Koedinger, K.R., Anderson, J.R.: Intelligent Tutoring Systems. Handbook of Human-Computer Interaction (2nd ed.) (1997) 849-874
5. Dastani, M., van Riemsdijk, M. B., & Meyer, J.-J. C.: Goal types in agent programming. Proceedings of the 17th European conference on artificial intelligence (ECAI'06) (2006)
6. 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) 153-178
7. Koedinger, K. R., Anderson, J. R., Hadley, W. H., Mark, M. A.: Intelligent tutoring goes to school in the big city. International Journal of Artificial Intelligence in Education, 8 (1997) 30-43
8. Lebeau J-F., Fortin M., Abdessemed A., Mayers, A.: Ontology-based knowledge representation for a domain-independent problem-solving ITS framework. The Sixth

International Workshop on Ontologies and Semantic Web for E-Learning (in conjunction with ITS 2008) (2008)
9. Lebeau, J.-F., Paquette, L., Fortin, M., Mayers, A.: An Authoring Language as a Key to Usability in a Problem-Solving ITS Framework. In: Aleven, V., Kay, J., Mostow, J. (Eds): Proceedings of ITS 2010 Part II (2010) 236-238
10. Murray T.: An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In: T. Murray, S. Blessing, S. Ainsworth (Eds.): Authoring tools for advanced learning environments Kluwer Academic Publishers , Dordrecht (2003)
11. Paquette, L., Lebeau, J.-F., Mayers, A.: Integrating Sophisticated Domain-Independent Pedagogical Behaviors in an ITS Framework. In: Aleven, V., Kay, J., Mostow, J. (Eds): Proceedings of ITS 2010 (2010) 236-238
12. Paquette, L., Lebeau, J.-F., Mayers, A.: Authoring Problem-Solving Tutors: A Comparison Between Astus and CTAT. Advances in Intelligent Tutoring Systems, (2010) 377-405
13. Paquette, L., Lebeau, J.F, Mayers, A.: Modeling Learners Erroneous Behaviours in Model Tracing Tutors. Proceedings of UMAP 2012 (2012)
14. Paquette, L., Lebeau, J.F, Mayers, A.: Une alternative aux systèmes de production pour la création de tuteurs par traçage de modèle. Actes du colloque Environnements informatiques d'apprentissage humain (EIAH 2013) (2013) 115-126
15. Paquette, L., Lebeau, J.F, Beaulieu, G., Mayers, A.: Designing a Knowledge Representation Approach for the Generation of Pedagogical Interventions by MTTs. International Journal of Artificial Intelligence in Education (IJAIED) (2014)
16. Sacerdoti, E.D.: A Structure for Plans and Behavior. Elsevier, New York (1975)
17. VanLehn, K.: Mind Bugs. Cambridge, MA: MIT Press (1989).
18. VanLehn, K.: The behavior of tutoring systems. International Journal of Artificial Intelligence in Education, 16 (2006) 227-265
19. VanLehn, K.: The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. Educational Psychologist, 46 (4) (2011) 197-221
20. VanRiemsdijk, M.B., Dastani, M., Winikoff, M.: Goals in Agent Systems: A Unifying Framework. In: Padgham, Parkes, Müller and Parsons(eds.): Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems(AAMAS2008) (2008) 713-720
21. Woolf, B.P., Cunningham, P.: Building a community memory for intelligent tutoring systems. In: K. Forbus, H. Shrobe (Eds.): Proceedings of the sixth national conference on artificial intelligence. AAAI Press , Menlo Park, CA (1987)

Chapitre 2

Enseignement des stratégies de résolution de problèmes à l'aide de MTT

Ce chapitre décrit l'intégration de connaissances stratégiques dans le système de représentation de connaissances de la plateforme Astus ainsi qu'un processus de génération d'interventions pédagogiques (interventions stratégiques) utilisant ces connaissances. Ces interventions enseignent aux étudiants comment faire des choix et ont pour objectif d'augmenter la maîtrise du domaine de ces étudiants en les poussant à faire de meilleurs choix. Ils feront ainsi preuve de plus de flexibilité en résolvant les tâches qui leurs sont assignées.

L'expérience décrite dans ce chapitre évalue les effets des interventions stratégiques. Le MTT conçu pour les exemples du chapitre 1 fut enrichi avec des connaissances stratégiques puis utilisé par des étudiants pour mesurer l'impact des interventions stratégiques sur leurs choix.

L'auteur (Gabriel Beaulieu) a contribué à cet article en mettant au point la représentation des connaissances stratégiques, en identifiant les situations dans lesquelles ces interventions bénéficient aux étudiants et en renforçant les capacités de la plateforme Astus pour que les MTT conçus génèrent automatiquement les interventions stratégiques appropriées grâce à des gabarits de messages. Il a aussi mis en place l'expérimentation et assuré le support des étudiants utilisant le logiciel. Comme mentionné auparavant, la plateforme Astus fut conçue par Jean-François Lebeau, et la gestion automatisée des interventions pédagogiques fut

introduite pour la première fois par Luc Paquette. La contribution de l'auteur à la charge de travail de la rédaction de l'article est de 80%.

Learning Problem-Solving Strategies in Model-Tracing Tutors

G. Beaulieu, L. Paquette, A. Mayers, *Member, IEEE*

Abstract — Model Tracing Tutors (MTTs) are designed to teach students procedural knowledge about how to solve problems, but generally offer little support in terms of teaching them how to compare problem-solving procedures to decide which one is better. Past studies show that students who receive strategic interventions comparing multiple procedures are more flexible and efficient and solve problems with fewer errors. This paper describes how to enhance a MTT with a hierarchical knowledge representation to support strategic knowledge and generate strategic interventions to improve student flexibility. This approach allows the MTT to automatically generate pedagogical content related to strategic knowledge without requiring this content to be manually encoded by the MTT's author. The benefits of these automated interventions were evaluated in a pilot study conducted with MTT-GNT, a tutor for a lesson on DNA analysis taught at Université de Sherbrooke.

Index Terms— Automated interventions, Flexibility, Knowledge representation, Model Tracing Tutors, Strategic interventions

---

- *G. Beaulieu is with Université de Sherbrooke, 2500 Boulevard de l'université, Sherbrooke, QC J1K 2R1.*
- *L. Paquette is with the University of Illinois at Urbana-Champaign,1310 South 6th Street, Champaign, IL 61820.*
- *A. Mayers is with Université de Sherbrooke, 2500 Boulevard de l'université, Sherbrooke, QC J1K 2R1.*

1 Introduction

MODEL-tracing tutors (MTTs) are software designed to support students as they solve problems, while providing scaffolding such as immediate correctness feedback and hints about what should be done next. Their benefits for the tutoring of well-defined problems have been measured through multiple studies. For example, Koedinger et al. [1] conducted an experiment with 9th-grade algebra students. They found that students using a tutor outperformed others by 15%, for tests focusing on general skills, to 100%, for tests focusing on the skills taught by the tutor. VanLehn [2] studied the effect of human tutoring versus intelligent tutoring systems and other tutoring systems like computer-aided instruction. This study does not confirm the common belief that human tutoring improves learning by an effect size of 2.0 standard deviations [3], when compared to no tutoring. It rather found an effect size of about 0.79, whereas intelligent tutoring systems were found to have an effect size of

0.76 (rather than 1.0 as commonly believed [3]) compared to no tutoring.

Problem solving in an MTT is accomplished through a series of steps (a set of interface actions with an interesting pedagogical granularity) which modify the task state. Each step performed by the student is evaluated by the MTT in order to provide hints and feedback that guide the student in improving his or her mastery of the task. The different functionalities of an MTT [4] are usually provided by four modules [5]: expert, communication, pedagogical, and student model. The communication module interprets the student's actions as steps, which are evaluated by the expert module. Given this evaluation, the student model module updates the student's skill levels, and the pedagogical module builds the feedback that will be sent to the student by the communication module.

The use of MTTs has been limited by their development costs [6, 7]. In order to decrease these costs, authoring tools such as CTAT [8] and Astus [9, 10] were developed. In the CTAT platform, production rules, used to model procedural knowledge [11], are employed to support the authoring of cognitive tutors (CTs) by modeling the procedural knowledge required to execute the tutored task. CTAT also provides tools for building example-tracing tutors [12] in which tutors are created by demonstrations of the problem-solving procedure rather than by building a model of the knowledge required to solve the task. However, although they can evaluate students' strategies and provide hints on optimal steps, they suffer from the same limitations as cognitive tutors: the need for manually written interventions, the absence of feedback indicating that students' steps are non-optimal, and the impossibility of providing an explanation of why a step is optimal. Our emphasis is not only on providing the learner with rich pedagogical interventions. We wanted to focus on allowing the MTT to use its knowledge model to assume some of its roles. More specifically, we intend to make a MTT which use the knowledge modeled to automatically generate interventions and trace learner steps. Thus, we do not talk about example-tracing tutors here, as their knowledge representation cannot include the task principles that explain why a step is optimal.

In comparison, the Astus framework [9] uses a knowledge representation similar to hierarchical task networks [9], designed to be closer to teacher-to-student instructions. In

Astus's representation [9, 10], a procedure is a structure that describes how to reach a goal (named the parent goal) by defining a set of subgoals that need to be achieved in order to validate the parent goal. By using a hierarchical knowledge representation system (abbreviated here by the term "hierarchical system"), Astus's MTTs are able to automatically create pedagogical interventions in natural language and interpret students' procedural errors without requiring additional development costs [10].

Even if a MTT can demonstrate how to solve a task through production rules (CTAT) or procedures (Astus), there may be multiple valid ways to solve it, represented by multiple alternative rules or procedures. When this case arises, an expert student demonstrates something called flexibility [13]. According to Verschaffel [14], flexibility describes the student's ability to use multiple procedures when solving a task. Rittle-Johnson describes flexibility as the student's ability to select the best procedure (the one requiring the fewest steps or the one with the lowest error risk) to solve the task [15]. In this paper, we use Rittle-Johnson's definition, as knowing how to optimally solve a problem requires knowledge of how to solve it using multiple methods. We suggest that acquiring flexibility starts with learning different procedures, as proposed by Verschaffel, and is mastered by learning when it is suitable to use each procedure [16, 17, 18].

Studies demonstrate that students who are taught multiple ways to solve a problem often show evidence of flexibility [15, 17, 19, 20]. They will choose the most efficient procedures [20] or the ones with lower risk of errors [15, 17]. Well-performing students often come from countries like Japan and Hong Kong where students are taught how to follow procedures but also how to compare multiple procedures for the same problem [20]. In Canada, teaching by comparison and demonstration of problem-solving procedures is considered as an important approach in mathematics [21]. The effect of teaching using comparison has also shown benefits in preschoolers' spatial mapping [22] and in children's development of abstract thought, relational thought and object categorization [23, 24].

Flexibility is closely related to strategic knowledge, the knowledge of how to choose which facts and principles to apply [25]. Weber [25] explains that undergraduate students difficulties

in constructing mathematical proofs come from shortcomings in their strategic knowledge.

It is important to make the difference between strategic knowledge and domain-independent higher-level skills. For example, a recent work by Matsuda et al. [26] demonstrated that supporting metacognitive scaffolding, (which explains to the learner how to teach to a virtual student) improves learners procedural skill for the domain they are teaching to a virtual student (equation solving).

Our work is not about metacognitive scaffolding. It is more related to what Matsuda et al. [26] call "cognitive scaffolding" because it is strongly related to the domain and how to solve tasks.[7] We want here to improve student flexibility: the MTT has to help the student realizing that his steps are not optimal and that even if his strategy is correct, better ways exist.

Some previous work by Anderson et al. [27] evaluated how problem solving strategies were affected by low-level steps. More specifically they compared the strategies choosed by the learner when the MTT was doing the simple mathematical operations versus when the leaner did the operations himself/herself. They showed that MTT-based calculations biased the learner toward an inefficient strategy even if it allowed them to advance further in the curriculum. The difference with our work is that we aimed here a higher level knowledge (strategic knowledge) to help them choose the appropriate strategy.

In order to teach strategic knowledge, the MTT should provide students with three types of pedagogic interventions designed to improve strategic knowledge. These interventions are designed to help students by explaining how to make an optimal choice.

The first type is **next-step hints**. Next-step hints are traditionally provided by MTTs [4] and can be provided on demand when the student does not know how to continue solving the task. The hint informs the student about the goals he or she may follow. However, when multiple goals are available to the student, MTTs traditionally supply hints related to one of

[7] Even though Matsuda et al. found no improvement when supporting cognitive scaffolding, this may not be the case in model tracing tutors. Matsuda et al. pedagogic choice was about teaching learners "teaching skills" and not directly "problem-solving skills" as in MTTs.

the goals rather than providing the student with information about how to determine which goal is the best one.

We propose that MTTs should also be able to provide feedback to the student when a non-optimal step is taken (**strategic feedback**), which is equivalent to VanLehn's [4] category of "correct but non-optimal" feedback. This feedback explains to the student that the choice they just made is correct but not optimal and how to choose the best of multiple valid options.

Strategic feedback (VanLehn's [4] "non-optimal" feedback) and positive feedback (feedbacks validating learner's steps) differs by their pedagogical objectives [4]. It has been shown that students often prefer well-known procedures and procedures they are used to follow even if they admit afterward that these are not the optimal ones [17]. Unlike positive feedback, strategic feedback does not only confirm that the student's steps are correct, it also teaches students how to compare different procedures, select the optimal one, and improve their overall flexibility.

The third type of intervention we propose is a particular case of **negative feedback**. Its role is to explain to a student who has executed a known procedure why this procedure cannot be applied in the current context. We consider that strategic knowledge includes the conditions in which a procedure is applicable. Indeed, before selecting the optimal procedure, the student needs to be able to evaluate which ones are applicable in the current context.

The research presented in this paper has two main objectives: 1) allow MTTs to automatically generate interventions that enhance the student's learning of strategic knowledge; 2) conduct an evaluation of the effectiveness of these interventions in the context of MTT-GNT, a tutor implementing our strategic knowledge representation. Our research was divided into three steps:

1. creating a strategic knowledge model and integrating it into a MTT's hierarchical knowledge representation;
2. allowing the MTT to evaluate students' choices;
3. using the strategic knowledge model to automatically create pedagogical

interventions in natural language when needed.

In the work described here, we used the Astus framework to achieve our objectives. In this paper, we explain how Astus's knowledge representation structures can be adapted to support the authoring of strategic knowledge. MTTs created using these structures have the capability to automatically generate strategic interventions in quasi-natural language. These interventions can provide explanations about how to select the optimal options, why a selected option is not optimal, and why a selected option cannot be applied in a specific context. We also describe our experiment to evaluate if learners benefits from these interventions.

2 Related Works

There are multiple authoring frameworks. We first detail their characteristics and then we describe one of these frameworks, CTAT. This section objective is to explain limitations of previous works to justify our choice to work with Astus framework for modeling strategic knowledge explicitely. Some previous works [28, 29] compare MTTs (with CTAT) and contrainst-based modeling (CBM), the two main authoring framework approaches. Mitrovic et al. [28] comparison suggests that both approaches are viable depending on the domain to be modeled. Kodaganallur et al. [29] work explains that CBM are easier to create but that model-tracing offer more precise feedbacks (called "remediation"). This work was criticized by Mitrovic and Ohlsson [30] (especially their CBM implementation and the toy-domain modeled), even though the main claims got an answer in [31].

Another work by Paquette et al. [32] compared cognitive tutors (CTs) made by CTAT and Astus tutors to explain their main differences. Even though lots of work (error diagnostic, intervention generation …) were done on Astus framework since this article was published, it gives enough information to see the main differences between the two approaches.

In order to provide the reader with information about authoring frameworks, we used here Mitrovic's Table 2 [28] (See Table 1) which sums up differencies between CBM and model

tracing approaches. We added to this table a column about Astus in order to make clear what are the pro and cons of each authoring framework. Then, as Astus tutors are model-tracing tutors, we will describe further the CT approach to explain why we chose to work on Astus instead of CTs.

Table 1. Mitrovic et al. [28] Table comparing CTs et CBM with a Astus column.

Property	CTs	CBMs	Astus tutors
Knowledge Representation	Production rules (procedural)	Constraints (declarative)	Procedures (procedural) [9]
Cognitive Fidelity	Tends to be higher (learner)	Tends to be lower (learner)	Tends to be higher (expert)
What is evaluated	Actions	Problem state	Actions
Problem solving strategies	Implemented ones	Flexible to any strategy	Implemented or allowed ones[1]
Solutions	Computed (can be stored)	Stored (can be computed)	Computed (can be stored)
Feedback	Tends to be immediate (can be delayed)	Tends to be delayed (can be immediate)	Immediate or delayed[1]
Problem solving hints	Yes	Only missing elements, but not strategy	Yes [9]
Diagnostic if no match	Solution is incorrect	Solution is correct	Solution is incorrect (error-diagnostic available [10])
Bugs represented	Yes	No	Yes (but sometimes unnecessary thanks to the error diagnostic [10])
Implementation efforts	Tends to be harder, but can be made easier with loss of advantages	Tends to be easier but can be made harder to gain advantages	Tends to be harder (but offers advantages)

Cognitive Tutors are widely used in the American education system [1]. A CT interprets student problem-solving behavior using a cognitive model that captures, in the form of production rules [8] (a representation of the base unit of procedural memory), the skills that the student is expected to learn [33]. Using such models, the CT is able to trace a student who is solving a problem and provide pedagogical interventions close to those of a human.

Semantic knowledge in CTs is modeled using working memory elements (WMEs) whose

attributes model an object's main characteristics. These attributes may be links to other WMEs or primitive data (numbers, strings, etc.). When solving a problem, the set of all instantiated WMEs is a representation of the student's interpretation of the task state.

Each production rule has two sides. The left-hand side is a condition describing when the rule is applicable. The right-hand side is the action part of the rule. It models the set of changes applied by the rule to the task state.

Given a task state, there may be a chain of production rules that must be applied successively to model the effects of a single problem-solving step in the user interface. The right-hand side of the last step of the chain is responsible for the actual step in the interface, whereas the other rules represent mental inferences leading to the step. Thus, when a student executes a step, the task evolves according to the rule chain that leads to this step by applying each rule's action part.

Production rules are a powerful tool that offers a lot of freedom when authoring a task's cognitive model. However, although this freedom makes the authoring process easier, it also makes it difficult for the pedagogical module 1) to interpret the content of the rules in order to automatically generate pedagogical content; and 2) to deduce the underlying computational structures, such as sequences of actions and iterations, contained in the model of the task [9].

In order to provide students with pedagogical interventions, the author of a CT is required to label each production rule with intervention templates explaining how to follow the rule. Given a step and a chain of rules leading to it, the CT will provide a set of hints starting with the one associated with the first rule of the chain and ending with the one associated with the last rule. Students' mistakes are traced through production rules marked as invalid or "buggy". Each of these buggy rules is labeled with the corresponding negative feedback that should be provided to students when they make this mistake. This model renders pedagogical experiments time-consuming, as creating a new pedagogical strategy requires the author to manually rewrite each of the model's interventions.

When multiple chains of rules are available, the CT's author can assign a static priority to

each rule. The best chain is obtained by selecting the rule with the highest priority every time multiple rules are available. Thus, static priorities can be seen as a way to model strategic knowledge in CTs. When providing hints to a student, the rule with the highest priority will be deemed the most optimal and next-step hints will be provided for that rule.

The main challenge of representing strategic knowledge in CTs is the replication of the interventions coming from multiple rules. When the learner has to make a choice, his/her options are split into three categories: optimal, non-optimal and invalid. As the three corresponding types of feedback (positive, non-optimal, negative) can be provided for the same step depending on the option chosen by the learner, it means that a single choice is modeled used multiple rules. Therefore the strategic knowledge will be replicated across all rules' interventions. For example, if the learner is doing a goal G and has three possible actions A (optimal), B (non-optimal) and C (non-optimal), it leads to three rules. The rules will have the following interventions (*next-step hint*[8], feedback):

- A: *You have to do G. Your best option is to choose A*; You correctly did G;
- B: You correctly did G, however A was a better option;
- C: You correctly did G, however A was a better option.

In this example, we see that the strategic knowledge stating that A is the best choice is replicated.

The production rules used by CTs to represent procedural knowledge have advantages, but they also have limitations that hierarchical knowledge representations do not have. These limitations make strategic knowledge modeling harder in cognitive tutors:

- The CT does not allow one to model a choice within a single production rule. Thus, each choice type is modeled by one rule, and the CT cannot automatically know that a set of rules belongs to the same procedural process. This makes it hard to use the model of the task to automatically create strategic interventions;
- The strategical knowledge is replicated both between rules for the same choice and

[8] B and C has no next-step hint because they model the same choice. Therefore, there is no need to duplicate the next-step hint between rules.

between the differents types of interventions;

- Pedagogical interventions are written manually by the CT's author. Thus, they cannot automatically use task-state information to provide rich interventions [9], and more precisely interventions explaining why a choice is optimal. In some cases, additional production rules might be required to match specific task-states in order to provide contextualized interventions.

3 Strategic Knowledge Modeling in a Hierarchical System

3.1 DNA analysis and strategic knowledge

This section describes DNA analysis as taught in the course GNT404 at Université de Sherbrooke. In the work described here, this domain was used as a test case to evaluate the effect of interventions generated by our strategic knowledge model. Solving tasks related to this domain requires students to be able to make appropriate choices in order to minimize measurement errors and reduce the number of steps required to solve the task. Flexibility is thus a key notion for solving tasks in this domain. We summarized here the domain and the two main strategies we used during this article.

In GNT404, students learn to identify DNA molecules. To do so, they have to find the location where some enzymes cut the DNA molecule (called "cutting sites"). The challenge of the task in GNT404 is that the learner does not know the original location of the DNA fragment obtained after the digestion of the molecule by enzymes. Therefore, the learner measures fragment lengths and make hypotheses.

DNA fragments are placed in a gel in which they move in exponential relation to the inverse of their length: the longer a fragment is, the less it moves in the gel. The first strategy we studied (see Section 4) is the choice of the measurement procedure. The base (and imprecise) way consists into measuring the fragment motion and converting it to obtain its length. For some fragments, it is possible to notice that an already-measured fragment has exactly the same motion, which indicates that the two fragments have the same size. Thus,

the other way consists into deducing the fragment length if there is a fragment with the same displacement.

The overall strategy consists into minimizing imprecise measures by deducing length as often as possible. This shortens the overall time required to solve the problem as each measure requires multiple steps (recording motion, converting it, and eventually correcting imprecisions) and facilitates the analysis of hypotheses by having coherent fragment length through multiple digestions.

However, even if the overall strategy is relatively simple we (the authors and GNT404's teacher) noticed that many students often do not learn this strategy. These students spend a lot of times doing measures they could avoid, resulting in many further problems when deducing the place of DNA fragments on the molecule.

The DNA molecule is linear and can be read in both directions; thus a cutting site can be in two possible locations. The student defines a reading direction (an orientation) by choosing an enzyme that cuts the molecule only once not in the middle. This enzyme is called the pivot enzyme, and the student places its cutting site at one of the two possible locations. Other enzymes' cutting sites will then be placed on the molecule according to the orientation defined by the student's choice.

Choosing a pivot enzyme close to the middle of the DNA molecule decreases the probability to obtain big fragments during later digestions with multiple enzymes, because if an enzyme cut the molecule close to its extremity, the biggest fragment will be cut by the pivot enzyme. Big fragments are more subject to measurement imprecision as their motion is in an exponential relation to the inverse of their length. Therefore, having a good pivot enzyme helps having precise measures.

The choice of the appropriate measurement procedure and of the pivot enzyme affects the precision of fragment lengths, and thus the cutting site placement. Therefore, a student who demonstrates flexibility in solving the task by choosing the right measurement procedure and/or the ideal pivot enzyme will execute fewer steps and will more easily identify the DNA molecule with a reduced risk of errors. This explains why GNT404 is a good domain for

showing how to model strategic knowledge and evaluating the effects of strategic interventions on students.

3.2 Hierarchical System of the Astus Framework

In this section we describe Astus's knowledge representation and its model tracing processes in order to make our strategic knowledge integration understandable by the reader.

MTTs created using the Astus platform are designed to capture the student's interactions with the user interface and interpret them as pedagogically relevant steps. The sequence of steps executed by the student while interacting with the MTT is then used to trace the student's problem-solving process against an "episodic tree" built using:

- a **knowledge base** (KB), the MTT's representation of the interface's visible objects and the objects that are hypothesized to be in the student's working and long-term memories;
- a **task model**, including a procedural graph and the declarative knowledge for the domain.

The **procedural graph** is simultaneously (1) a formal representation of the task's solving procedures, designed to model a teacher's explanation of how to solve the task; (2) a representation of how the student perceives this knowledge; and (3) a structure used to interpret the student's steps. Some additional informations about procedural graph are presented in [9] (for example, Figure 2 and Figure 3 of [9] are examples of procedural graphs). For some of the student's mistakes, the author may want the MTT to provide the student with specific interventions. For this purpose, procedures marked as invalid can be added to the procedural graph in order to give the student manually encoded negative feedback similar to that found in CTs.

Nodes in the procedural graph represent goals and procedures. **Goals** model the student's intention and **procedures** model how to achieve these goals. Procedures are defined as control structures (sequence, iteration, loop, search…) that divide the parent goal into a set of child goals (called subgoals) and define constraints between subgoal executions. The

procedural graph's root, called the **root goal,** models the student's intention to solve the task. The leaves of the procedural graph are called **primitive procedures** and model the task's steps.

The **episodic tree** is built by instantiating the procedural graph's goals and procedures for a specific task, and labeling them according to their execution state (as above, informations are available in [9], more specifically, Figure 4 of [9] is an example of an episodic tree). Procedures and goals can be in an *available*, *executing*, or *validated* state whereas only goals can be in the *waiting* state. Leaves in an episodic tree can be either primitive procedures or *waiting* goals. A *waiting* goal becomes *available* if it is the root goal or when the constraints defined by its parent procedure are satisfied. When a goal becomes *available*, the MTT instantiates the subtree squared in it (we say that the goal is split by its child procedures). An *available* goal becomes *executing* (resp. *validated*) when one of its child procedures becomes *executing* (resp. *validated*). A procedure is *available* while none of its subgoals are *executing* or *validated*, becomes *executing* when some of them are *executing* or *validated*, and is *validated* when all its subgoals are *validated*. A primitive procedure becomes *validated* as soon as the student executes the step it models.

The episodic tree is updated each time a student executes a step. Thus, at any moment during the solving of the task, the set of *validated* procedures and goals represents the history of the student's steps and intents; the set of *available* procedures and goals is the set of all valid intents and procedures that can be followed; and the set of *waiting* goals models future intents that the student must achieve to solve the task.

Goals and procedures have parameters defined by their parents. A procedure can define its subgoals' parameter values from its own set of parameters or from internal queries. In the Astus framework, the cognitive skills mastered by the student (capacity to recognize a given concept or relation in the interface, or a type of mental inference) which produce a set of objects or primitive data required for the execution of a procedure are modeled by **queries** composed from atomic black-box code. There are two main types of query:

- the selection, using filters modeled by logical formulas, of an object from the KB;

- an inference from black-box combinatory structures (AND, OR, XOR…) applied to a set of objects (obtained from parameters or other queries) in the KB.

As queries model mastered cognitive skills, the pedagogical module does not provide explanations indicating how to execute a query, but it can explain to the student what queries are needed to execute a procedure. In addition, the pedagogical module can call on the expert module to obtain the result of the query, which can then be provided to the student. For example, given two booleans A and B, the pedagogical module can explain to the student that they must calculate "A and B" or "A or B" and provide the result "true" or "false", but it cannot explain that in order for "A or B" to evaluate to true, either A needs to be true or B needs to be true.

3.3 Strategic Procedures in the Astus Framework

The objective of providing strategic interventions is to improve student flexibility or, in other words, to teach students how to choose the optimal object or goal when they solve the task. In the Astus framework's hierarchical system, choices occur when a choice procedure (choice between objects) or a search procedure (choice between goals) becomes *available* in the episodic tree. In the previous iteration of Astus, these two procedures were able to trace a student making the right choice, but they were not designed to provide the MTT with the capability to produce strategic interventions and thus, teach the student strategic knowledge. This led us to change these two structures to allow an MTT's author to model strategic knowledge, and enable the MTT itself to produce strategic interventions.

Choice procedure. The new choice procedure is now composed of the following:

- the set *E* of the elements to choose from;
- a unary predicate (validity condition) *p* defining whether an element *e* from *E* is valid;
- a query *r* modeling the mental inference that determines the best choice from E;
- a subgoal *B* taking an element from E as a parameter, modeling the intention to choose this parameter.

When the choice procedure becomes *available*, each valid choice i (each i in E with $p(i)$ = true) becomes a parameter of a goal B_i, an instance of B in the episodic tree. These goals are then split by procedures to obtain all the new available steps. If a subgoal B_i becomes *executing* or *validated*, the MTT will interpret this event as evidence that the object i was chosen by the student. Therefore, other subgoals of the procedure are removed from the episodic tree, as they model non-chosen options. The procedure becomes *validated* when one of its subgoals (the remaining one) becomes *validated*.

In order to trace student choices and detect non-optimal choices, we adapted the Astus framework's model-tracing process. The MTT now follows this scheme (the figures' fonts are explained in next paragraph):

1. It uses r to calculate and save the best option when the procedure is instantiated (i.e. becomes *available*) in the episodic tree.
2. If asked for help, it produces a next-step hint (Figure 1) explaining that the student has a choice to make and describing the query r. Describing r helps the student make the best choice.
3. If the student chooses an invalid object, it creates a negative feedback (Figure 2) and gives it to the student. If not, it waits for the subgoal and procedure to be *validated* by the student.
4. Otherwise, it compares the optimal object to the chosen one, and if they do not match, it creates and gives the student a strategic feedback (Figure 3) describing the query r.

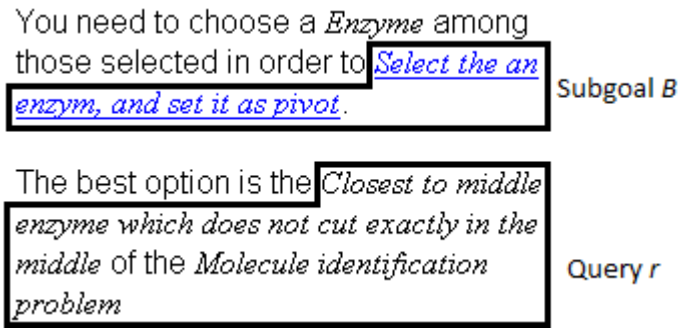

Figure 1. Next-step hint with strategic knowledge example in MTT-GNT: the pedagogical module explains how to choose the best pivot enzyme by saying what to do (subgoal *B*) and describing the query *r*.

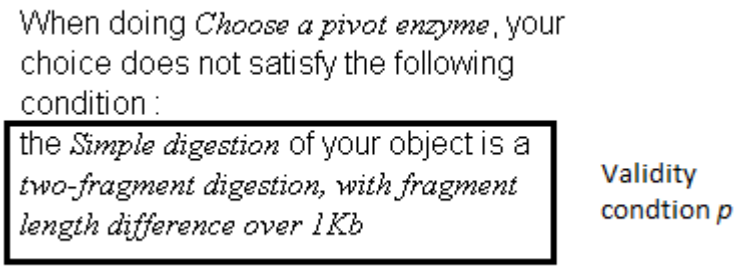

Figure 2. The MTT gives this negative feedback to a student that made the wrong pivot enzyme choice. It explains what the choosen option has to satisfy using validity condition *p*.

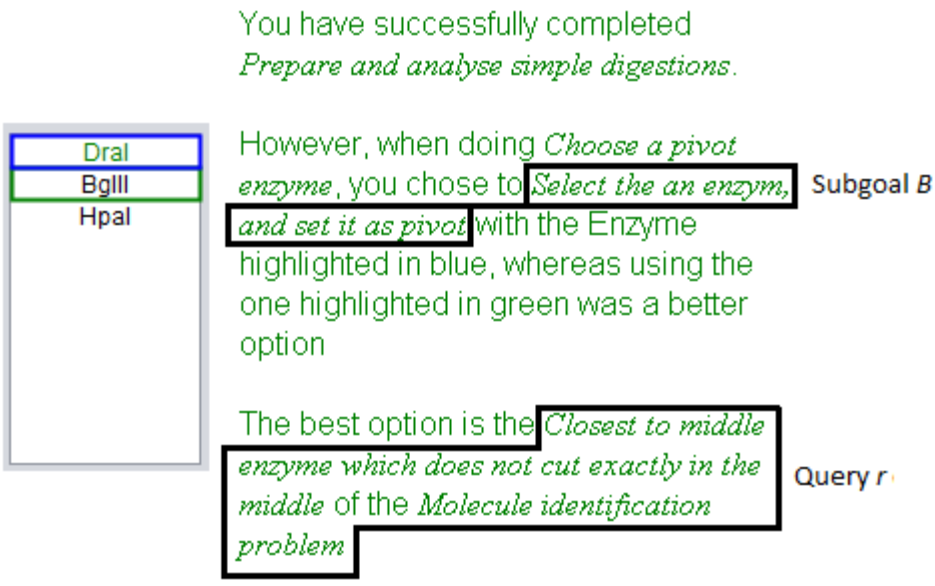

Figure 3. Strategic feedback produced. when the learner chooses a non-optimal pivot enzyme in MTT-GNT. The right side shows interface effects that help contextualize the intervention. It validates the achievement of subgoal *B* and tells the learner how to make a better choice using query *r*.

As we will describe it now, this model allows the MTT to explain to the student how to make a valid and optimal choice. By including strategic information about the query *r* in its next-step hint, the MTT targets flexibility by asking the student to compare all options as soon as possible. These interventions are generated from message templates (standard font in figures), completed by object or goal labels to contextualize the message (*italic* font in figures). Goal labels in next-step hints (*underlined italic blue* text) are also hyperlinks allowing the student to ask for more precise help about how to achieve the goal in question by clicking on it.

The expert module matches the chosen object against the optimal one when the student *validates* the procedure. This prevents the teaching of strategic knowledge from interfering with the teaching of the procedures: the MTT checks whether the student knows domain procedures before providing feedback on strategic knowledge targeting their flexibility. Thus, the MTT adopts the behavior of a teacher who lets students complete their tasks or subgoals

before explaining that there was a better option.

Strategic feedback in Astus is composed of three parts (see Figure 3). The first part is similar to positive feedback and approves the student's step. This prevents the student from thinking their step was erroneous. The second part contextualizes the intervention, by highlighting the chosen and optimal objects in the interface. The third part focuses on teaching strategic knowledge to help students make the optimal choice in the general case.

This new format for the choice procedure allows for the diagnosis of the student's strategic errors since the validity condition is explicit. An MTT created using the Astus framework can thus produce negative feedback (see Figure 2) explaining to a student who made an invalid choice the condition under which this choice would have been valid. This negative feedback is formatted in a way that highlights the positive form of the validity condition as "your object must be p" rather than explaining why the chosen object is not valid "your choice is invalid because your object is $\neg p$". We chose not to include strategic knowledge of how to make an optimal choice (query *r)* into the negative feedback because we wanted students to focus first on learning how to select a valid choice before learning how to make optimal choices.

As shown in this section, the new choice procedure allows the MTT to handle all of the three kinds of strategic interventions listed in the introduction. It provides strategic feedback before a step, in the form of next-step hints, helping students make their choice (Figure 1), and after a step to explain why a choice was not optimal (Figure 3). It also provides negative feedback explaining how to evaluate whether a choice is valid or not (Figure 2).

Search procedure. One of our previous articles [30] describes the introduction of the search procedure in Astus's knowledge representation and the common errors made by learners following it. In this article, we modified the search procedure model to integrate strategic knowledge and allow the MTT to give the learner advices on what steps are optimal. We will now describe how the search procedure works in order to make our improvements understandable.

A search procedure divides a particular kind of goal (called an achievement goal) that is *validated* by reaching a task state that satisfies its constraints rather than by following a predefined step sequence. The search procedure simulates a process in which there are multiple ways to *validate* the parent goal, but a fixed sequence of steps cannot be planned ahead of time. Thus, the student has to search for how to *validate* the parent goal by trying one of its subgoals, verifying whether its completion validated the parent goal and repeating this process until the parent goal is validated.

Each of the search procedure's subgoals models one way to progress through the task that consists in achieving the procedure's parent goal. In some cases, a subgoal may not be applicable to the task. We therefore associate with each subgoal a query describing when this subgoal models a valid way to progress through the task resolution.

As long as a search procedure's parent goal is not *validated,* the procedure will loop, computing the set of valid subgoals and adding them to the episodic tree. The model-tracing process will then wait for the student to execute a step generated by one of the subgoals. Once this first step is executed, the episodic tree is updated, the chosen subgoal becomes *executing* (or *validated*) and the other subgoals are removed from the episodic tree. Once the chosen subgoal is *validated,* the MTT checks whether the achievement goal is satisfied by the state of the task. If the goal is satisfied, the procedure ends and is labeled as *validated*. Otherwise, the process restarts and the MTT computes the new set of valid subgoals.

When the student chooses an invalid subgoal, the MTT uses the query associated with that subgoal to create negative feedback explaining when this subgoal models a valid way to try to achieve the procedure's parent goal [34]. This allows the MTT to produce instruction about how to determine whether a subgoal is valid. However, it does not allow the MTT to explain how to choose the optimal subgoal. The MTT is unable to create next-step hints or strategic feedback explaining what the best subgoal to achieve the parent goal is, because the task model does not contain the strategic knowledge used to determine the optimal way to perform the search.

In order to model the strategic knowledge related to these interventions, we allowed the author to associate with each child subgoal a query used to compute its priority. The usage of a query rather than a static priority allows the MTT's modules to have access to the priority's semantic. With these queries, the pedagogical module is able to produce next-step hints (Figure 4) and strategic feedback (Figure 5) describing the best option in a given task state.

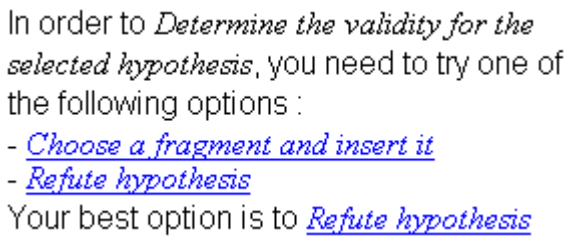

Figure 4. Next step hint with strategic content for a search procedure in MTT-GNT. The learner is analyzing a hypothesis and as it is obviously wrong. His/her best option is to refute it directly, avoiding the steps related to fragment placement.

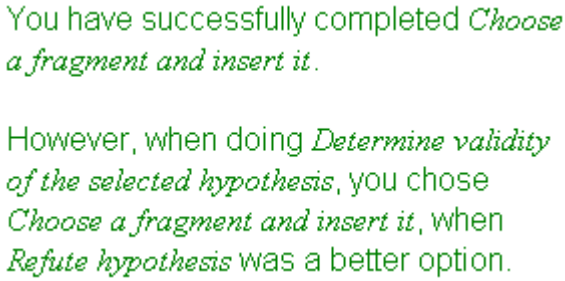

Figure 5. Strategic feedback of a search procedure. The situation is the same as in Figure 4, but the learned did not choose the best option.

As MTTs can access the semantic of a query, the strategic interventions can provide a description of the cognition necessary to calculate the goal's priority. However, so far, no domains requiring the use of dynamic priorities have been modeled using the Astus platform or CTAT. All of the examples studied used static priorities that do not evolve with the task. The only factor influencing which subgoal is optimal is the validity of each subgoal. Some high-priority goals may not be applicable in a specific context and in this context the application of a lower-priority goal will thus be considered optimal. As we couldn't

experiment to verify that pedagogical interventions describing priority calculus are clear, understandable and close to human interventions, we decided not to describe them in this paper.[9] For this reason, the current version of the Astus framework is able to explain which subgoal is the best one but not the cognition behind this assertion.

As we did for the choice procedure, we updated the model-tracing process to allow the MTT to detect non-optimal choices and produce strategic interventions regarding them. The same process is applied: the MTT computes which subgoal is the best one when the procedure starts a new iteration. Then, it determines the subgoal chosen by the student (i.e. the one which just became *validated* or *executing*) when they execute a step arising from the successive decomposition of this subgoal into procedures. As we did for the choice procedure, to prevent the teaching of strategic knowledge from interfering with the learning of procedural knowledge, the MTT compares the optimal subgoal to the one chosen by the student when the student *validates* it. If they are identical, the MTT gives positive feedback to the student. Otherwise, the MTT provides strategic feedback similar to that shown on Figure 5.

By adding queries defining subgoal priorities, the MTT author can now model strategic knowledge. This allows the resulting MTT to create and communicate the three kinds of strategic interventions listed in the introduction.

4 Experimentation

4.1 Working Hypotheses

The previous section describes how we modified Astus's hierarchical knowledge representation to model strategic knowledge, and how MTTs can create strategic interventions using the modeled knowledge. In this section, we describe an experiment that evaluated the effect of the strategic interventions on GNT404 students.

[9] The description of the process leading to the description of priority calculus will be presented in the modeling of a domain which requires dynamic priorities. Such a description will confirm whether the resulting interventions are clear and understandable.

First we conducted an experiment with GNT404 volunteers. Although only 27 students participated, resulting in low statistical power, this experiment suggested that strategical interventions helped students. We therefore reproduced the same experiment in order to get more reliable data and gathered data for 124 students during a first training phase. Out of those 124 students, 118 completed the course exam that we used as a post-test.

As explained in Section 2.1, the GNT404 model presents two main cases of strategic knowledge: the fragment measurement procedure and the pivot enzyme selection. These two cases led us to two working hypotheses. The first, H_1 (the corresponding null hypothesis is H_0) asserted that students in the experimental group, who receive strategic knowledge interventions, learn more quickly how to measure fragments and use the optimal measurement procedure (deduction) more often when it is possible to do so. The second hypothesis, denoted by H_3 (the corresponding null hypothesis is H_2) supposed that students from the experimental group make better choices when selecting a pivot enzyme.

4.2 Experiment

Each student followed these steps:

1. Attend a presentation describing the task, including a description of MTT-GNT and of the software (used in previous years to evaluate learners on the DNA analysis) that would be used to evaluate the students' knowledge of DNA analysis;
2. Use MTT-GNT to train on 4 simple problems and one complex problem;
3. Send the data recorded during the training phase;
4. Use the evaluation software to get their grade (exam, executed in the same conditions as the training phase because of its length).

Four simple problems were designed to teach students how to deal with task-specific problems they may encounter during their exam. The complex problem was isomorphic to the one presented in the exam. During the final exam, students were allowed to ask the teacher questions. This was done in order to prevent them from having a significant advantage depending on which group they were assigned to.

Each of the training exercises consisted of a DNA molecule and a set of enzymes. The first problem (S0) was the one presented to students for them to learn how to use the interface. Other exercises were of increasing difficulty:

- S1 had only 5 enzymes each cutting the molecule only once. It was designed to teach students how to choose the right pivot enzyme, the "easiest" part of the task;
- S2 focused on teaching the best way to obtain the fragment's length by using a rule or deducing it;
- S3 focused on fragment placement. In this exercise, only 3 enzymes cut the fragment once. All others cut it multiple times;
- The complex problem (C) comprised 4 enzymes cutting once, 3 enzymes cutting twice, 2 enzymes cutting three times and one enzyme cutting four times.

As the software used to complete the exam is independent of our MTT, we were not able to obtain trace data on each of the students' steps. We were not able to acquire information about the choice of the measurement procedure or pivot enzyme choices. However, we were able to record how many times the students opened the curve used to convert the fragment's displacement to fragment length. Thus, we collected the following data:

- The number of deductions to obtain fragment length for each training problem (related to H_1 during training);
- The number of times during the exam that the student opened the curve to get a fragment's length from its motion on the gel (related to H_1 after training);
- The chosen pivot enzyme for each training problem (related to H_3 during training).

4.3 Group Formation

The students were divided into two homogeneous groups in regard to their biology grades. Students from the control group trained using MTT-GNT without strategic interventions. Students from the experimental group trained using the full version of MTT-GNT, including strategic interventions.

The first, intuitive, way to get fragment length is to measure the fragment's displacement in

the gel. This is why students often measure fragments even if there are other ways to get fragment length. When possible, the optimal way is always to deduce the fragment's length. Thus, in order to guarantee that students learned all of the measuring procedures, the MTT always told students asking for help to deduce the fragment length if such a deduction was possible. For the control group, however, this option was not labeled as the best one (See Figure 6).

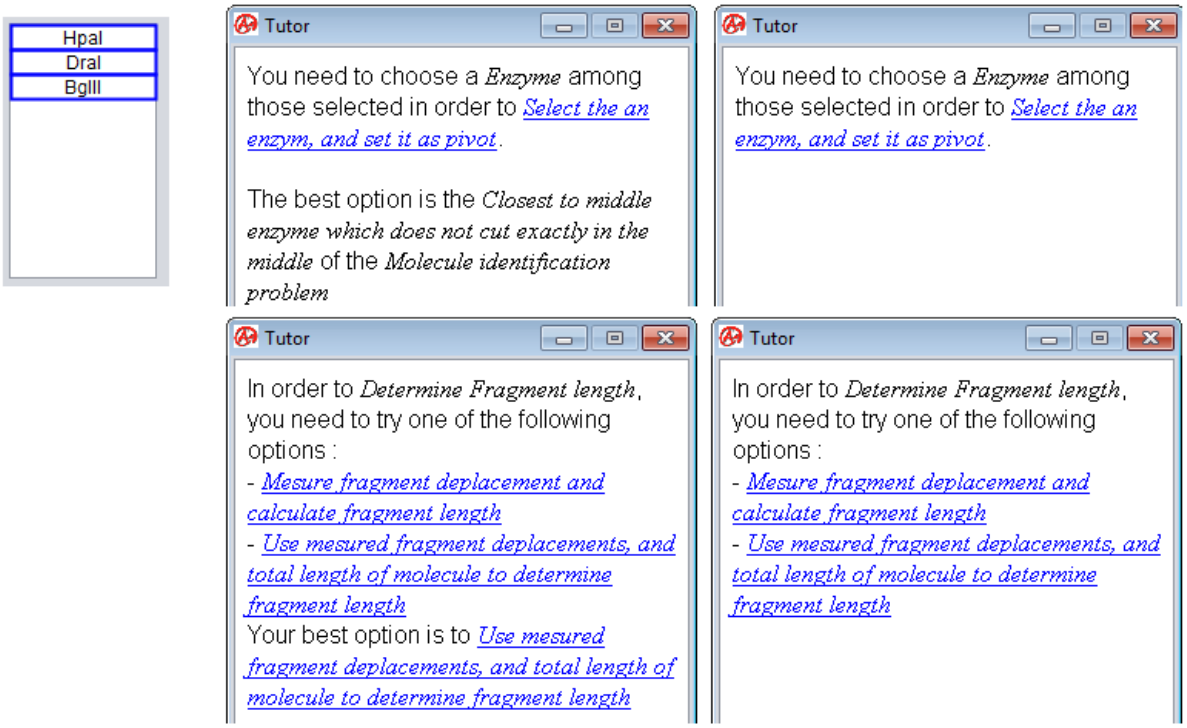

Figure 6. Intervention comparison between groups for next-step hints. Top hints are about choosing a pivot enzym while bottom ones are about measuring fragments. Control group interventions are on the right.

The same amount of feedback was provided for both groups. The only difference is that when the learner does not select the optimal way, the positive feedback (First paragraph of Figure 3) of the control group was replaced by a VanLehn's [4] called "non-optimal" feedback: the same text with strategic information (the last two paragraphs of Figure 3) added.

Solving a GNT404 problem can take up to 15 hours for some students. Therefore, we could not use a controlled environment (e.g., the University's computer labs) to supervise students. They were given a download link and an authenticated access to the software, and were free to train on the problems on their personal computers. We gathered data for 65 students in the control group and 59 students in the experimental group. Only 62 students from the control group and 56 students from the experimental group completed the exam.

4.4 Results

Choice of the optimal measurement procedure. We performed an ANOVA test for both training and exam phases (shown in Table 2). For the training phase, we compared the number of deductions between the control and the experiment group. For the exam phase, we compared the number of curve opening. On average, students in the control group chose the optimal procedure 20.8% (Standard error: 0.202; confidence interval: 0.05) when it was possible to deduce a fragment length whereas the experimental group made the optimal choice 26.7% (Standard error: 0.272; confidence interval: 0.07) of the time. Due to a technical problem (described further in Section 4.5) some data were missing. This problem occurs when the task implies lots of measures (more complex ones S3-C1) and did not affect the groups identically (23 students in the control group and 11 students in the experimental group for C1 problem). We replaced these data by the group mean because replacing it by the minimum value would introduce a bias in favor of the null hypotheses and replacing it by the maximum value would bias in favor of our hypotheses. We thus selected the group mean as a "neutral" value in order to keep student data for other problems. During the exams, students in the control group opened the curve 59.2 (std. err.: 46.8; confidence interval: 11.46) times on average, and students in the experimental group did so 42.8 (std. err.: 36.9; confidence interval: 9.27) times.

TABLE 2

STATISTICAL TESTS BETWEEN BOTH GROUPS FOR TRAINING AND EXAM PHASES

Phase	Stat	*p*	Effect size
Training	$F(1,122) = 1.886$	0.1721	$\omega^2 = 0.0071$

Exam	F(1,112) = 4.237	0.0404(*)	$\omega^2 = 0.0276$

Choice of the optimal pivot enzyme. When grading the choice of the optimal pivot enzyme, each problem was assigned as many points as there were enzymes that could be selected as the pivot. We made this choice to reduce noise due to students who may select the pivot enzyme randomly. Students only earned points for a problem if they selected the right enzyme on the first try. Students in the control group earned a mean of 9.98 points out of 16 (std. err.: 3.55; confidence interval: 1.23) and students in the experimental group earned a mean of 11.44 points (std. err.: 3.65; confidence interval: 1.13). We performed an ANOVA test on these data (results shown in Table 3).

TABLE 3

STATISTICAL TESTS ON PIVOT ENZYME CHOICES

Stat	*p*	Effect size
F(1,122)= 5.309	0.0229(*)	$\omega^2 = 0.0336$

4.5 Interpretation

Choice of the optimal measurement procedure. Strategic interventions seemed to have a small effect on the selection of the optimal measurement procedure. The results for all exercises did not allow us to reject the null hypothesis H_0 ($p = 0.1721$), but the results during the exam did ($p = 0.0404$).

We think the statistical significance of our experiment (and probably our effect size) was impacted by a technical issue that some students encountered. This issue prevented the exercise data from being recorded and occurred mainly when the students were converting fragment displacement to length. Further investigation seemed to indicate that these issues arose when some steps related to displacement measures were executed too fast.

This probably led the control group data to be better than expected as the students who did not master the strategic knowledge measured more fragments. As a result, they were more exposed to this problem, with the issue occurring for 23 students in the control group and 11

students in the experimental group.

However, some students, even in the experimental group, ended up using the optimal measurement procedure only a few times (fewer than 10 times out of 100 possible times) even during the complex problem. We think that these students confused strategic feedback with positive feedback. Both interventions were non-blocking, without any special interface effect and written in green. Thus, given the positive part of the strategic feedback, students may think it is an unconditional validation of the executed step and never read, learn and master the strategic knowledge provided by the intervention.

Results from the exams suggested that the null hypothesis H_0 could be rejected. Although the results obtained from training data did not lead us to reject the null hypothesis, we attribute more importance to the exams results, as students had more freedom when solving the problem. This showed how students behave when they do not receive feedbacks about fragments measuring. Additionally, the results from the exams were not biased by the technical issues encountered during training. The results suggested that the strategic interventions provided by MTT-GNT helped students learn how to choose the optimal measurement procedure. Therefore, according to Rittle-Johnson's [15] definition of flexibility (choosing the right procedure at the right moment), this experiment showed with a small effect size that strategical interventions helped acquiring flexibility. Students who did not master the strategic knowledge of choosing a measurement procedure were more likely to choose the procedure they were used to following instead of the optimal one, as described by Newton et al. [17].

Choice of the optimal pivot enzyme. Data analysis on the choice of a pivot enzyme validated the positive impact of strategic interventions. These results allowed us to confirm hypothesis H_3, and to reject the null hypothesis H_2 ($p < 0.05$). This seemed confirmed by a further look on the complex problem data (the one with the most enzymes) showing that 87.5% of the students (resp. 62.5%) students from the experimental group (resp. control group) chose the optimal pivot.

5 Conclusion

The benefits of learning simultaneous procedures in order to acquire flexibility when performing problem-solving tasks have been studied and demonstrated multiple times [15, 17, 19, 20]. Learning can be enhanced through multiple strategic interventions occurring during problem solving. In this article, we presented a way of modeling and using strategic knowledge to automatically produce strategic interventions within a MTT. In order to achieve this result, we integrated strategic knowledge elements into the hierarchical knowledge representation used by the Astus framework. This allows Astus MTTs to translate strategic knowledge elements into natural language and automatically generate pedagogical content that can be used to produce strategic feedback.

Specifically, we described how the Astus framework's knowledge representation can be modified in order to allow its MTTs to produce three kinds of interventions to help students master strategic knowledge. An Astus MTT can integrate information from its model of strategic knowledge into (1) next-step hints and (2) a new kind of feedback called strategic feedback in order to help students make optimal choices. It can also produce (3) negative feedback designed to help students make valid choices by indicating how to choose a valid object or a valid goal.

Our experiment showed that these strategic interventions have a small but significant effect on students: they chose the optimal measurement procedure and the optimal pivot enzyme more often when solving a DNA analysis problem.

The data on the measurement procedure suggested that, even when given a strategic intervention, some students in the experimental group kept using the same non-optimal measurement procedure. This might be caused by a mix-up between some of our strategic feedback and positive feedback. The small effect size we noticed could come from the fact that students did not receive the feedback provided by the MTT, assuming that green feedbacks are always positive ones. We therefore suggest first to change the color of strategical feedbacks somewhere between positive (green) and negative (red) feedbacks.

Then, future work on the student model module can incorporate information about the students' mastery of the strategies. For example, strategic feedback could block the interface when the student keeps choosing the non-optimal option, in order to force them to read the intervention and learn its strategic knowledge. On the other hand, these interventions could be non-blocking when a student often shows evidence of having mastered the strategic knowledge.

We hypothesize that those improvements will further benefit students who receive strategical interventions. It will help them make optimal choices which will in return reduce measurement errors and decrease the number of steps required to solve problems. Further experiments will test this hypothesis and whether the proposed improvements both increase the overall effect size of the experiment and reduces the mean standard errors by reducing students' progression margins.

Further experiments will help us confirm the effectiveness of strategic interventions. Changes to the exam software would allow us to collect additional data about whether students use the deduction approach when measuring the length of a fragment and to verify the student's choice of a pivot enzyme.

Other work could be conducted to evaluate the effectiveness of our strategic feedback in domains with dynamic priorities. Although our current model supports dynamic priorities and can generate corresponding interventions, tasks requiring dynamic priorities have not been modeled yet. Modeling such domains will allow us to evaluate the clarity and effectiveness of strategic feedback explaining how to determine the best subgoal of a search procedure.

REFERENCES

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

[2] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems", *Educational Psychologist, 46(4)* pp. 197-221, 2011.

[3] B. Bloom, "The 2-sigma problem: The search for methods of group instruction as effective as one-to-one tutoring", *Educational Researcher* 13 (6) pp. 4-16, 1984.

[4] K. VanLehn, "The behavior of tutoring systems" *International Journal of Artificial Intelligence in*

Education, 16, pp. 227-265, 2006.

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

[6] N.T. Heffernan, K.R. Koedinger, L. Razzaq, "Expanding the model-tracing architecture: A 3rd generation intelligent tutor for algebra symbolization" *International Journal of Artificial Intelligence in Education, 18, 2*, pp. 153-178, 2008.

[7] T. Murray, "An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art", *Authoring tools for advanced learning environments,* T. Murray, S. Blessing and S. Ainsworth eds., Dordrecht: Kluwer Academic Publishers, 2003.

[8] V. Aleven, "Rule-Based Cognitive Modeling for Intelligent Tutoring Systems", in *Advances in Intelligent Tutoring Systems*, R. Nkambou, R. Mizoguchi and J. Bourdeau, eds., Springer, vol. 308 pp. 33-62, 2010

[9] L. Paquette, J.F. Lebeau, G. Beaulieu, A. Mayers, "Designing a knowledge representation approach for the generation of pedagogical interventions by MTTs" *International Journal of Artificial Intelligence in Education (IJAIED)*, 2015.

[10] L. Paquette, J.F. Lebeau, A. Mayers, "Modeling learners' erroneous behaviours in model tracing tutors" *Proceedings of UMAP 2012*, vol. 7379 pp. 316-321, 2012.

[11] J.R. Anderson, "ACT: A simple theory of complex cognition." *American Psychologist, 51*, pp. 355–365, 1996.

[12] V. Aleven, B.M. McLaren, J. Sewall, K.R. Koedinger "Example-tracing tutors: A new paradigm for intelligent tutoring systems", *International Journal of Artificial Intelligence in Education* 2009.

[13] M. Schneider, B. Rittle-Johnson, J.R. Star, "Relations among conceptual knowledge, procedural knowledge, and procedural flexibility in two samples differing in prior knowledge" *Developmental Psychology,* Advance online publication, 2011. doi:10.1037/a0024997

[14] L. Verschaffel, K. Luwel, J. Torbeyns, W. VanDooren, "Developing adaptive expertise: A feasible and valuable goal for (elementary) mathematics education?", *Ciencias Psicologicas 2007(1),* pp. 27-35, 2007.

[15] B. Rittle-Johnson, J.R. Star, "Does comparing solution methods facilitate conceptual and procedural knowledge? An experimental study on learning to solve equations" *Journal of Educational Psychology, 99 (3)*, pp. 561-574, 2007.

[16] A.W. Blote, E. VanderBurg, A.S. Klein, "Students' flexibility in solving two-digit addition and subtraction problems: Instruction effects", *Journal of Educational Psychology, 93*, pp. 627-638, 2001.

[17] K.J. Newton, J.R. Star, K. Lynch, "Understanding the development of flexibility in struggling algebra students", *Mathematical Thinking and Learning, vol. 12, no. 4*, pp. 282-305, 2010.

[18] J.R. Star, C. Seifert, "The development of flexibility in equation solving", *Contemporary Educational Psychology, 31*, pp. 280-300, 2006.

[19] J.R. Star, B. Rittle-Johnson, "Flexibility in problem solving: The case of equation solving.", *Learning and Instruction,18*, pp. 565-579, 2008.

[20] J.W. Stigler, J. Hiebert, "The teaching gap: Best ideas from the world's teachers for improving education in the classroom", *NewYork: Free Press*, 1999.

[21] National Mathematics Advisory Panel, "Foundations for success: The final report of the National Mathematics Advisory Panel", *Washington, DC: U.S. Department of Education,* 2008.

[22] J. Loewenstein, D. Gentner, "Spatial mapping in preschoolers: Close comparisons facilitate far mappings", *Journal of Cognition and Development, 2,* pp. 189-219, 2001.

[23] L.L. Namy, D. Gentner, "Making a silk purse out of two sow's ears: Young children's use of comparison in

category learning." *Journal of Experimental Psychology: General, 131*, pp. 5-15, 2002.

[24] L.M. Oakes, R.J. Ribar, "A comparison of infants' categorization in paired and successive presentation familiarization tasks", *Infancy, 7,* pp. 85-98, 2005.

[25] K. Weber, "Student difficulty in constructing proof: The need for strategic knowledge."*Educational Studies in Mathematics, 48*, pp. 101-119, 2001.

[26] N. Matsuda, N. Barbalios, J. Zhao, A. Ramamurthy, G. Stylianides, K.R. Koedinger, "Tell me how to teach, I'll learn how to solve problems." *Proceedings of the International Conference on Intelligent Tutoring Systems.* 2016.

[27] S. Ritter, J.R. Anderson, "Calculation and strategy in the equation solving tutor" *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pp. 413-418, 1995.

[28] A. Mitrovic, K.R. Koedinger, B. Martin, "A comparative analysis of cognitive tutoring and constraint-based modelling" *Proceedings of the Ninth International Conference on User Modeling UM 2003(Vol. LNAI 2702),* P. Brusilovsky & A. Corbett & F. d. Rosis (Eds.), Berlin: Springer-Verlag, pp. 313-322, 2003.

[29] V. Kodaganallur, R.R. Weitz, D. Rosenthal "A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms," *International Journal of Artificial Intelligence in Education*, vol. 15, pp. 117-144, 2005.

[30] A. Mitrovic, S. Ohlsson "A Critique of Koaanallur, Weitz, and Rosenthal, "A Comparison ofModel-Tracing and Constraint-Based Intelligent Tuoring Paradigms"", *International Journal of Artificial Intelligence in Education*, vol. 16, pp. 277-289, 2006.

[31] V. Kodaganallur, R. Weitz, D. Rosenthal, "An Assessment of Constraint-Based Tutors: A Response to Mitrovic and Ohlsson's Critique of "A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms." *International Journal of Artificial Intelligence in Education,* vol. 16, pp. 291–321, 2006.

Paquette, L., Lebeau, J.-F., Mayers, A. "Authoring Problem-Solving Tutors: A Comparison Between Astus and CTAT" *Advances in Intelligent Tutoring Systems*, pp. 377-405, 2010

[32] V. Aleven, B.M. McLaren, J. Sewall, K.R. Koedinger "The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains", *ITS 2006* M. Ikeda, K.D. Ashley, T.W. Chan eds., vol. 4053 pp. 61–70 2006

[33] G. Beaulieu, L. Paquette, A. Mayers, "Gestion automatisée de buts de succès dans les tuteurs par traçage de modèle", *Environnements informatiques pour l'apprentissage humain*, Agadir, 2015

Chapitre 3

Enseigner des algorithmes avec retour sur trace dans les MTT

Ce chapitre présente un schéma pour représenter les algorithmes avec retour sur trace dans un système hiérarchique comportant des structures similaires à celles de la plateforme Astus. Grâce à cette représentation les MTT produits par la plateforme sont capables d'exploiter les éléments clés des algorithmes avec retour sur trace pour générer des interventions pédagogiques qui favorisent l'apprentissage des étudiants lorsqu'ils appliquent ces algorithmes.

En outre, ce chapitre étend les possibilités de modélisation des systèmes hiérarchiques tels que celui de la plateforme Astus en décrivant une gestion des contraintes de concurrence. Cela permet d'une part de modéliser des domaines moins bien définis dans lesquels la façon de résoudre les tâches est moins bien déterminée et d'autre part d'instaurer des contraintes d'un nouveau genre entre les sous-objectifs d'un domaine. Grâce à ces contraintes les étudiants sont libres de choisir quel sous-objectif ils veulent atteindre, mais seront contraints de le valider avant de s'attaquer à un autre sous-objectif indépendant.

L'auteur (Gabriel Beaulieu) a contribué au travail de rédaction de ce chapitre à hauteur de 90%. Il est à l'origine de la représentation des algorithmes avec retour sur trace et des contraintes de concurrence dans le système hiérarchique de la plateforme Astus. Tout comme dans les chapitres 1 et 2, la plateforme Astus fut conçue par Jean-François Lebeau. La

description de la plateforme qui est faite ici est le fait de l'auteur de la thèse mais son implémentation et sa mise au point sont dues à Jean-François Lebeau.

Teaching backtracking algorithms in Model Tracing Tutors

Gabriel Beaulieu[1], Luc Paquette[2], André Mayers[1]

[1]Université de Sherbrooke, Canada
[2]University of Illinois at Urbana-Champaign, USA

Abstract. Model tracing tutors (MTTs) teach learners how to perform problem-solving tasks. Authoring frameworks can be used to reduce the effort associated with the creation of MTTs by implementing general solutions for common MTT features such as tracing learners' actions, producing pedagogical interventions and diagnosing errors. MTTs generated with the Astus framework use a hierarchical knowledge representation (Paquette et al. 2015), allowing them to automatically generate pedagogical content. However, generating proper pedagogical interventions for tasks involving backtracking algorithms is difficult and they can generate inappropriate interventions in the case of parallelism constraints (i.e., in tasks where some parts can be solved in any order but not simultaneously). This article describes how Astus's hierarchical knowledge representation can be modified to allow for the modeling of backtracking algorithms and parallelism constraints, thus enabling MTTs created using Astus to automatically trace the learner's backtracking steps and generate pedagogical interventions related to backtracking and parallelism. These advances have been incorporated into an Astus MTT for DNA analysis used at Université de Sherbrooke.

Keywords. dynamic programming, model-tracing tutors, hierarchical knowledge representation, constraints between goals

1 Introduction

Model tracing tutors (MTTs) teach learners how to perform problem-solving tasks. They select the task, trace the steps[10] taken by the learner in solving it, produce pedagogical interventions and diagnose the learner's mistakes (VanLehn, 2006). They can describe the next step, show the learner how to make this step and even produce feedback which validates the step, explains his/her errors to the learner or proposes a better alternative (VanLehn, 2006).

The different roles of MTTs have classically been attributed to 4 modules (Wenger, 1987):

- The communication module checks that the interface reflects the task states, identifies the learner's steps and displays interventions.
- The expert module evaluates the learner's steps and diagnoses errors.
- The pedagogical module selects tasks and decides which intervention should be given to the learner.
- The "learner model" module evaluates the learner's skill level.

MTT development costs, estimated at between 100 and 300 hours (Heffernan et al., 2008; Murray, 2003; Woolf et al., 1987), can be lowered using authoring frameworks like the Cognitive Tutor Authoring Tool (CTAT; Aleven et al., 2006; Aleven et al., 2009; Aleven, 2010) and Astus (Paquette et al., 2010; Paquette et al., 2012b). Authoring frameworks reduce development costs by implementing general solutions to common MTT features.

While authoring frameworks are effective at reducing MTT development costs, they provide no tools to help model domains that involve backtracking algorithms. Specifically, they do not offer knowledge representation structures that help trace learners' steps or give them interventions close to those authored by a domain expert.[11] This makes these domains harder to model than others, and MTT authoring does not save as much time as for other domains using authoring frameworks.

---

[10] A step is a set of actions that have some pedagogical interest.

[11] It is possible to trace a learner applying a backtracking algorithm. However, this will require an incorrect use of knowledge structures, which may lead to inappropriate interventions.

A backtracking algorithm's objective is to maximize or satisfy an objective function by making decisions for a set of data.[12] The algorithm describes the order in which to consider the data and how to make decisions for the datum currently under consideration (see section 3.1). For example, in the *knapsack* problem, we have to decide whether (decision) we put objects (data) in the knapsack or not in order to maximize the total interest of knapsack objects. As decisions are made successively, without considering the remaining data, it may happen that previously made decisions do not lead to the best solution. In such cases, we have to cancel previous decisions (backtrack) before making others.

Another problem for authoring frameworks is detecting and producing pedagogical interventions when the learner is trying to achieve two subtasks he/she is not allowed to pursue simultaneously (we say that the subtasks have a parallelism constraint). It is possible to do this by ordering the subtasks, which will prevent the learner from reaching them simultaneously. However, this artificially fixes the order in which the learner has to accomplish them, and the MTT may generate confusing interventions describing inappropriate constraints.

Providing an authoring framework capable of correctly modeling backtracking algorithms and parallelism constraints is important because it increases the framework expressivity and allows the MTT author to model more domains. Moreover, it allows the resulting MTTs to use the task model to trace learners' steps and produce clear pedagogical interventions close to those of a human expert.

Our objectives are as follows:

- Create a procedural scheme that allows the modeling of backtracking algorithms in authoring frameworks. This scheme must fit most backtracking algorithms and its semantics must be accessible to software agents (here, MTT modules).
- Allow authoring frameworks to model domains that use parallelism constraints and to exploit these constraints to produce interventions that teach learners the constraints.

[12] If achieving the objective only requires ordering decisions, the data are the decision order.

Our first objective is to offer a structure for domains whose solving processes involve a backtracking algorithm. The backtracking algorithm is merely a procedural abstraction used by MTTs that will be instantiated by MTT authors to model the task. The learner simply learns to solve the task, and uses a backtracking algorithm in doing so.

Our objectives are not only knowledge modeling but also allowing software agents (here MTT modules) to utilize the modeled knowledge.

Two main authoring frameworks are suitable for our objectives. The first, the CTAT framework, produces cognitive tutors (CTs) that use production rules to simulate task resolution. A production rule describes both the conditions that need to be met in order for the rule to be applied and its effects on the task state when applied. This is a simple and expressive knowledge representation system (KRS) but the black box nature of production rules makes it difficult for a CT to access the semantics of the knowledge modeled. Therefore, the CT author writes the interventions manually when designing the CT.

The second framework, Astus, associates simple instructions (sequence, choice, iteration, etc.) and behaviors with domain-independent message templates describing how to solve the task (Lebeau et al., 2008; Paquette et al., 2010; Paquette et al., 2012a; Paquette et al., 2015). When an intervention is needed, these message templates are instantiated depending on the model of the task, resulting in the automatic generation of rich pedagogical interventions similar to those authored by a domain expert (Paquette et al., 2015; Paquette et al., 2012b).

We chose to work on the Astus framework because we intended to allow the framework to use a backtracking algorithm model to trace learner steps and produce pedagogical interventions. The Astus framework was more suitable for this, because Astus already proposes both an explicit KSR (referred to here as pre-sys for previous system) and processes for generating pedagogical interventions. The general process is described by Paquette et al. (2012a) and Paquette et al. (2015); an extension to the KSR (here called new-sys) is described by Beaulieu et al. (2015) and the support strategic interventions are described by Beaulieu et al. (submitted).

We used Alekhnovich et al.'s (2005) model of backtracking algorithms. We provide a summary of this model to help readers who are unfamiliar with backtracking algorithms. We then present a procedural graph containing the key elements of the model. This graph is made from new-sys's procedural structures rather than a new structure to help MTT authors adapt the graph to the algorithms they are modeling.

Our parallelism constraint syntax is similar to the order constraint syntax. We have adapted the model tracing process to these new constraints. As a result, Astus MTTs now check whether the learner is complying with the constraints and describe these constraints to learners who make errors.

Our work on the Astus framework helps create MTTs for domains involving backtracking algorithms and/or parallelism constraints. In particular:

- The expert module traces a learner who is using a backtracking algorithm to solve the task and checks whether he/she respects the task parallelism constraints.
- The pedagogical module uses the model of a backtracking algorithm. It produces interventions and teaches learners the key elements of backtracking algorithms. Moreover, it also produces feedbacks when a learner does not respect task parallelism constraints. These feedbacks explain what subgoals cannot be achieved simultaneously and what subgoal the learner has to complete first.

These results were demonstrated using MTT GNT, an MTT developed for the course GNT404 at Université de Sherbrooke. An experiment described in Beaulieu et al. (submitted) uses this MTT to show that strategic interventions help learners make optimal decisions.

Section 2 presents the KRS of the Astus platform (new-sys). This is required to understand Section 3. Section 3 first describes the key elements of Alekhnovich et al.'s (2005) model for backtracking algorithms. Then, Section 3 presents a procedural graph containing these elements and ends by illustrating the model by means of two examples: one teaches function integration and the other comes from MTT GNT. Section 4 describes new-sys modifications that make it possible to model parallelism constraints in the Astus KRS.

The conclusion in Section 5 sums up our work and results and proposes avenues for future work.

2 Astus's hierarchical knowledge representation

The Astus framework proposes a KRS designed to be interpretable by domain-independent processes in order to generate pedagogical content automatically. To this end, Astus's KRS is composed of procedural structures called complex procedures (CPs) and goals that are used to model the tutored task. The resulting model of the task is similar to the way an expert would describe the task to a learner. Astus's various modules know these structures and can thus use this knowledge to trace learners' steps against the model of the task, demonstrate how to solve the task and produce pedagogical interventions (Paquette et al., 2012a; Paquette et al., 2012b).

This section presents the classic Astus knowledge representation structure (pre-sys) as well as some modification from new-sys we used to model backtracking algorithms and parallelism constraints. All our examples come from the knapsack and *WordResearch* problems. The knapsack problem involves filling a knapsack with objects without exceeding a given weight, while maximizing the total value of the picked objects. The *WordResearch* problem involves finding the longest word over the alphabet {*a,b,c*} that does not contain any repeating sub-string *vv*, where *v* is a non-null character chain. For example, *abac* is a valid word, whereas *abacbabac* is not, because of the substring *baba.*

2.1 Declarative knowledge

Task elements used by the learner are modeled in the Astus framework as **objects** and typed using **concepts** (an object is an instance of a concept). Each object has a set of attributes composed of primitive data (numbers, strings, data sequences) or links to other objects. For example, in the *WordResearch* problem, *a*, *b* and *c* are objects that instantiate the concept "`letter`".[13] Objects that instantiate the concept "`word`" have a unique attribute, a sequence of links to objects that are instances of the concept "`letter`". During problem

[13] This format is used for interventions produced by the MTT or element labels. Italic text means that this is an author label, while normal text comes from generic message templates associated with Astus knowledge structures. The pedagogical module replaces the character "_" by task element labels.

solving, all the instantiated objects are contained in the knowledge base (KB), which represents the MTT's interpretation of how the learner perceives the task.

When the learner solves the task, the MTT can retrieve specific objects from the KB using **queries**. Queries simulate mental inferences and are made by combining atomic elements that are assumed to be mastered by the learner. For example, the expert module can access the set of all the letters in the *WordResearch* problem using the query All(concept:letter), whose result is the set {*a, b, c*}, i.e., the set of all the objects that instantiate the concept "*letter*".

2.2 Procedural knowledge

Procedural knowledge describes how to solve a task. In Astus, this knowledge is modeled using goals and procedures. **Goals** model intents completed by parameters. For example, the goal "*add a letter to the word _*" is parameterized by the word to which a letter should be added. The two goals "*add a letter to the word ab*" and "*add a letter to the word abc*" are two instances of the same goal, but do not model the same intent. Thus, a goal is defined by a generic intent but also by its parameters.

A **procedure** for a given goal (called the parent goal) describes how to achieve it. For example, the goal "*add a letter to the word abc to make a valid word*" can be decomposed into "*choose a letter to form a valid word*" and "*add the letter at the end of the word*". We say that a goal B is a subgoal of a procedure *P* (*P* generates *B*) if *P*'s decomposition of its parent goal may lead to an instance of *B*.[14]

There are two procedure types: primitive procedures and complex procedures. **Primitive procedures** (PPs) model task steps. They are defined by two scripts: 1) a sequence of actions to realize in the MTT's interface in order to execute the step; and 2) the KB modifications induced by the step.

[14] Procedures can decompose goals in multiple ways. Therefore, a procedure may not always generate some of its subgoals.

Complex procedures (CPs) decompose their parent goal into subgoals. Complex procedures can be of multiple different types, with each type defining different ways the subgoals are generated and parameterized: *sequence, iteration, choice, loop,* and *search*.[15]

Goals and procedures form a graph called the **procedural graph** that models the knowledge required to solve a task. Figure 1 illustrates part of MTT GNT's procedural graph.

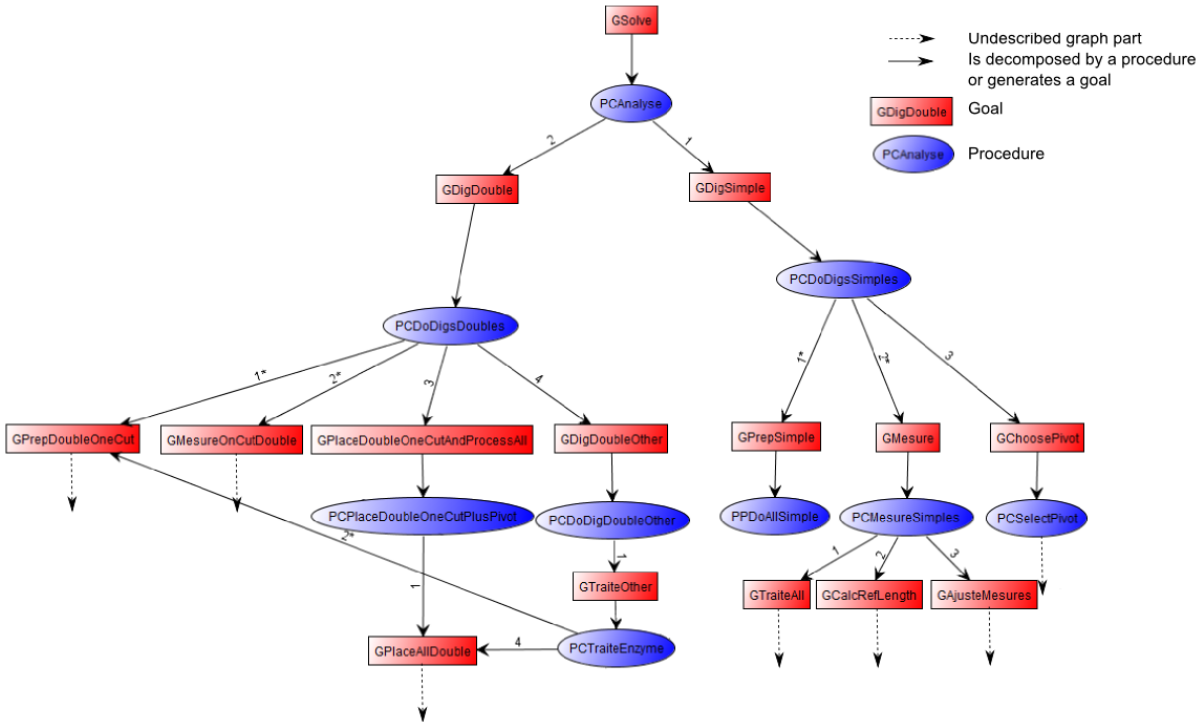

Fig. 2. Sub-graph of MTT GNT procedural graph for GNT404 lesson.

Definition 1. A procedural graph is an oriented graph defined by the following principles:

- The graph starts with the root goal, which models the intent to solve the task.
- Graph nodes are goals or procedures.
- If a node is a goal, it has exiting edges to the CP that decomposes it or the PP that achieves it.
- If a node is a CP, it has exiting edges to its subgoals.
- Graph leaves are PPs.

[15] Some of these CP types are described later in this section.

In order to trace learner steps, the expert module builds an **episodic tree** by instantiating the procedural graph for a specific problem. In such a tree, goals and procedures are labeled depending on their execution state: *waiting*, *available*, *executing* or *validated* (only goals can be *waiting*). Goals and procedures are instantiated[16] and parameterized with KB objects. Thus, a task's episodic tree represents the history of the task (i.e., *validated* goals and procedures achieved by the learner), a set of all valid subtasks that may be achieved by the learner (i.e., *available* or *executing* goals and procedures) and future goals that must be achieved when their constraints are relaxed (i.e., *waiting* goals).

Definition 2. The **episodic tree** defines the current state of the task. It is defined as follows:

- The tree starts with the root goal of the task.
- Nodes are goals or procedures.
- Procedures are labeled using a state from the range {*available*, *executing*, *validated*}.
- Goals are labeled using a state from the range {*waiting*, *available*, *executing*, *validated*}.
- The subgoals of a procedure are its children in the tree.
- The procedures that decompose a goal are its children in the tree.
- Leaves are *waiting* goals or PPs.

A *waiting* goal becomes *available* and is decomposed by procedures when the constraints from its parent procedure are satisfied (i.e., when all the subgoals that have higher priorities according to the parent procedure are *validated*). An *available* goal becomes *executing* (resp. *validated*) when one of its child procedures becomes *executing* (resp. *validated*). A complex procedure is *available* when none of its children are *executing* or *validated*. It becomes *executing* when at least one of its children becomes *executing* (or *validated*) and then *validated* when all of its children are *validated*. A primitive procedure becomes *validated* when the learner executes the step it models. Astus MTTs use *executing* and *available* nodes to generate help on goals and procedures started by the learner (Paquette et al., 2012b).

[16] When we are talking about a goal (or a procedure) from the episodic tree, we are referring to its parameterized instance and not to the goal (or procedure) from the procedural graph.

When the learner executes a step, the communication module determines what PP matches this step. Then, the expert module compares this PP to the set of all *available* PPs to identify the one executed by the learner. When it finds a match, the episodic tree's PP becomes *validated* and all node labels are updated.

We will now describe the complex procedures used in sections 3 and 4 to model the backtracking algorithm and parallelism constraints. Sequence and iteration procedures are described first, followed by the search procedure.

A **sequence procedure** defines a set of subgoals and a set of order constraints between them. When the procedure becomes *available*, all its subgoals are instantiated and added to the episodic tree. Subgoals are labeled *available* if they are free and *waiting* otherwise. A *waiting* subgoal becomes *available* when all subgoals that must be executed before it are *validated*.

An **iteration procedure** defines a subgoal *b* and a set *O* of KB objects. When the procedure becomes *available*, it generates a set of instances of *b* parameterized using each of the objects contained in *O*. For example, tracing a learner who is reducing a set of fractions can be modeled using an iteration procedure. This procedure's subgoal *b* is "`reduce the fraction _`", parameterized by the fraction to reduce. The set *O* of this procedure is the set of fractions to reduce. Therefore, the learner has to "`reduce the fraction _`" for each fraction in *O*.

The syntax of sequence and iteration procedures allows for the modeling of order constraints. However, neither iteration procedures nor sequence procedures allow for the modeling of parallelism constraints, as they offer no mechanisms to prevent the learner from simultaneously executing two or more of the *available* subgoals.

Now we will describe the search procedures used in Section 3 to model backtracking algorithms. Search procedures decompose particular subgoals called achievement goals. An **achievement goal** is *validated* when a proper task state is reached, rather than after a specific sequence of steps (Dastani et al., 2006). In some cases, an achievement goal may be *validated* before any steps are taken by the learner (for example, depending on the initial state of the

problem), whereas, in other cases, the *validation* of the achievement goal may require multiple attempts. In the Astus framework, achievement goals are modeled using a first-order formula (called a condition) which describes the states of the task that satisfy it (Beaulieu et al., 2015). A task state *validates* an achievement goal if and only if the evaluation of the goal condition is positive.

A **search procedure** *T* decomposes an achievement goal *p* when it is not possible to predict with absolute certainty how to *validate p*. The procedure *T* allows the learner to try multiple methods (modeled by the procedures decomposing its subgoals) or multiple variants of the same method to *validate p*. The behavior of the search procedure is described in more detail in Beaulieu et al. (2015), and pedagogical interventions associated with this type of procedure are described in Beaulieu et al. (submitted).

Subgoals of a search procedure *T* differ semantically from other goals: they model an attempt to achieve *T's* parent goal *p*, whereas other goals model the intent to reach a task state or to perform specific steps. It is possible for a subgoal *b* of the procedure *T* to be correctly *validated* without achieving its parent goal *p*. When this occurs, *b* is *validated*, because the learner has correctly achieved it, but *T* starts a new iteration allowing the learner to try another method to achieve *p*. In such a situation, the learner could, for example, continue to order data and make decisions according to a backtracking algorithm or backtrack and cancel some of his or her decisions.

Each subgoal of a search procedure is combined with a condition describing when it models a valid intent. Multiple subgoals may be *available* at the same time, meaning that the learner will need to choose which subgoal to achieve (i.e., which method to apply). Once this choice has been made, the other subgoals are removed from the episodic tree, as they model unchosen alternatives.

An iteration of a search procedure *T* starts when its parent's achievement goal *p* becomes *available* or when one of its subgoals becomes *validated*. This iteration begins by evaluating *p'*s condition. If the evaluation is positive, *p* is *validated* and the procedure ends. If not, the procedure generates a new instance of each subgoal whose condition is positively evaluated.

This means that at the beginning of an iteration, the set of *available* subgoals of T is the set of subgoals modeling the intent to apply an appropriate method.

Sections 3 and 4 demonstrate that new-sys, the updated (Beaulieu et al., 2015; Beaulieu et al., submitted) version of pre-sys (Paquette et al., 2015) is expressive enough to model backtracking algorithms and can be extended to support parallelism constraints. This allows MTTs created via the Astus framework to use the model of the task to trace a learner and provide pedagogical interventions during tasks involving backtracking algorithms or parallelism constraints.

3 Modeling backtracking algorithms in Astus

This section describes how we achieved our first objective, which was to allow the modeling and use of backtracking algorithms in MTT. Specifically, Astus MTT modules use the generic procedural graph for backtracking we describe here to trace learners' steps and generate interventions.

When creating an MTT for a task that involves backtracking, its author can instantiate this graph into a subgraph appropriate for the task. Section 3.1 discusses Alekhnovich et al.'s (2005) model of backtracking algorithms. The key elements of this model are described in Section 3.2. Section 3.3 presents our procedural graph for backtracking algorithms. Section 3.4 instantiates this graph for two tasks: function integration and DNA analysis.

3.1 Discussions about the model

Optimization problems, search problems and backtracking algorithms. Search problems involve finding a satisfying solution to a problem, whereas an optimization problem involves finding one of the best solutions (also called satisfying solutions: the definition of a satisfying solution depends on the problem type). A backtracking algorithm describes how to build a set of solutions to a problem that must be explored to find a satisfying one.

The algorithm extends a current solution by ordering the data, then making a decision for the first datum according to that order. After that, the resulting solution is evaluated to decide whether to pursue extending the solution or to modify past decisions. For example, the *knapsack* problem involves filling a knapsack with objects without exceeding a maximum

weight W. In this problem, data are available objects and decisions consist of adding an object to the knapsack or not. The *knapsack* problem is as an optimization problem when the objective is to maximize the total value of the knapsack's content. It is a search problem when the objective is to find a solution satisfying a constraint.

The algorithm A_{sac} for the knapsack problem examines the objects successively according to their value/weight ratios and decides whether to add each object or not. This algorithm iterates until one of the following occurs:

- The resulting solution is satisfying;
- A decision was made for all the objects;
- Both decisions were studied for an object.

In the last two cases, the algorithm backtracks to search for alternative (and better) solutions and modifies decision.

Teaching how to solve a problem using a backtracking algorithm requires three steps:

1. Explaining how to order data;
2. Explaining how to decide for the selected datum;
3. Explaining how to decide between ending the algorithm, taking new objects/decisions or backtracking.

For A_{sac}, these generic steps can be defined as follows:

1. Order objects by decreasing value/weight ratio and take the first.
2. Add or do not add the object to the knapsack.
3. End the algorithm if the current solution is a satisfying solution. If not, evaluate whether an additional object can be added to the knapsack. Otherwise backtrack on past decisions.

Modeling backtracking algorithms. Multiple models for backtracking algorithms exist: Borodin et al. (2003) study fixed and adaptive priority algorithms; Woeginger (2000) presents a model to determine whether or not a problem has a fully polynomial time approximation schemes (FPTAS); and Alekhnovich et al. (2005) describe a model generalizing these definitions.

Our generic procedural graph is based on Alekhnovich et al.'s (2005) model for three reasons:

- The Borodin et al. (2003) and Woeginger (2000) models are for particular problems and thus less powerful. Priority algorithms, proposed by Borodin et al. (2003), consider only one decision per datum. DP[17]-simple algorithms, proposed by Woeginger (2000), consider that data ordering does not depend on any past decision or data.
- This model includes elements corresponding to the three steps involved in teaching a backtracking algorithm.
- Despite its limitations, this model allows the representation of most of the backtracking algorithms we can teach learners.

Alekhnovich et al.'s (2005) model supposes that the decision whether to extend a solution or abort (backtrack) is made using only the information provided by the current partial solution. This does not allow for the simultaneous exploration of multiple decisions or for the decision to abort based on elements unrelated to the current solution. This prevents the modeling of some *branch-and-bound* algorithms that take the whole set of solutions into account before deciding which one to extend or to backtrack. It is probably possible to build the branch-and-bound algorithms the same way we built the backtracking algorithms. We say 'probably' here because we have not found a domain in which to test it. The principal issue is not building the algorithm but determining whether the MTT will generate useful interventions.

Buresh-Oppenheim et al. (2011) extended the Alekhnovich et al. (2005) model to allow it to take advantage of memoizing. Memoizing reduces the global complexity of finding a satisfying solution to a problem by preventing the algorithm from solving the same subproblem multiple times. However, the use of memoization is not required when solving problems. It is only a process used by computers to increase the performance of their algorithms. Hence, this extension was not implemented in our research.

[17] For Dynamic Programming

3.2 Formal model

This section aims to identify key elements of Alekhnovich et al.'s (2005) model for optimization/research problems and backtracking algorithms. These elements were included in our procedural graph using Astus's new-sys procedural structures, thereby allowing MTTs created using Astus to automatically trace learners' steps and produce pedagogical interventions for tasks that require backtracking algorithms.

Definition 3. An **optimization/search problem** P is a triplet (D, H, f_P), where D is a data set, H a decision set for each datum and f_P a set of objective functions. A solution $S_i \in D^i \times H^i$ to P is a sequence of i data (the order in which elements are selected) combined with decisions from H. In a search problem, the objective function f^i: $D^i \times H^i \rightarrow \mathrm{R}$ applied to S_i returns *valid* if S_i is a **satisfying** solution and *invalid* otherwise. In an optimization problem, the objective function returns a real number and a **satisfying** solution is a solution that returns the maximum possible value for the objective function.

In the knapsack problem, D is the set of objects $d_j\ (x_j, w_j)$, where x_j is the object's value and w_i its weight. H is $\{0, 1\}$: *0* if the object is not put in the knapsack, 1 otherwise. A solution $S_i = (d_1...d_i, h_1...h_i)$ is a sequence of i objects $(d_1...d_i)$ in the order they are selected and the corresponding decisions $(h_1...h_i)$ about whether the object should be put in the knapsack or not. The objective function f^i is:

$$f^i(S_i)=0 \text{ if } \Sigma^i_{k=1}\ h_k * w_k > W$$

$$f^i(S_i) = \Sigma^i_{k=1}\ h_k * x_k \text{ otherwise}$$

We say that S_i is **partial** if it contains decisions for a subset of the problem data. Thus, S_i is:

- partial if $i<|D|$;
- satisfying if it maximizes the total value, i.e., $\forall\ S_j\ f^i(S_j) \leq f^i(S_i)$.

Let A be a backtracking algorithm for a problem $P=(D, H, f_P)$. A describes how to extend a partial solution to P using two function sets: ordering functions $\{r^i_A\}$ and choice functions $\{c^i_A\}$ for i in $[1\ldots|D|]$. Ordering functions order data, and are used to select which datum

should be selected next. Choice functions define the set of possible decisions for the selected datum. Let $S_k=(d_1...d_k, h_1...h_k)$ be a partial solution; then:

- $r^k_A(S_k)$ is an ordering (a sequence) of elements from D, where d_{k+1} is the next datum to select . For example, $r^k_{Asac}(S_k)$ sorts the objects by decreasing valuue/weight ratio where e, the object that should be selected next, is the unselected object with the best ratio. Here we have simplified Alekhnovich et al.'s (2005) model to make it more understandable. Indeed, in the original model, functions r^k_A order all data that could exist and not only the problem data.
- c^k_A takes a solution S_k and an object e. It evaluates which decisions can be made for e given the current partial solution, returning an ordering of H decisions from best to worst. It partitions decisions that lead to incorrect (or non-optimal) solutions from others, using the symbol $\perp$. For example, in the knapsack problem, $c^k_A(S, e)$ would return $(0, \perp, 1)$ if adding e to the knapsack exceeds the maximum weight and, as such, the only allowed decision is 0: exclude e from the knapsack.

From an instructional perspective, teaching a learner how to determine what datum e to select, given a partial solution S, entails teaching how to evaluate the ordering function $r^k_A(S)$. In most cases, this function can be evaluated only once and this evaluation is then used for all future decisions, thus making the decision process easier for the learner as he/she will not need to evaluate the function every time a decision is made. For example, in the knapsack problem A_{sac}, since the value of each object is static, the order of the objects doesn't change as the problem is being solved.

Similarly, teaching the learner what decision to make for an object e entails teaching how to evaluate the choice function $c^k_A(S, e)$:

- which decisions are allowed (the ones before $\perp$);
- which decisions are the best ones (the first ones according to the order);
- which decisions are not allowed (the ones after $\perp$).

Backtracking algorithms define how to extend a partial solution by selecting a new datum and what decisions to make for that datum. The next datum is selected according to the

ordering function r^k_A, and the decision for this datum is made according to the ordering from the choice function c^k_A. If this combination of datum and choice does not lead to a satisfying solution or an extendable one, the algorithm will backtrack to a previous partial solution and consider the next decision according to the choice function.

These elements, once incorporated into the Astus KRS, will be used to teach the learner how to apply a backtracking algorithm. Specifically, they will allow the MTT to generate strategic interventions (Beaulieu et al., submitted) explaining how to order data and how to make good decisions.

The process of exploring the solution set is captured in Alekhnovich et al.'s (2005) computation tree, a tree that defines all the ways to build solutions starting from the empty one.

Definition 4. A **computation tree** for a problem $P = (D, H, f_P)$ and a backtracking algorithm A is a tree whose nodes are partial solutions to P:

- Its root is the empty solution.
- Each internal node n of depth k models a partial solution S_k where decisions have been made for k data. Each child of n is a solution S' defined as the solution S_k to which a new datum e, chosen according to $r^k_A(S_k)$, has been added. Each child node models a different decision for e, chosen according to $c^k_A(S_k, e)$[18], where the j^{th} child of n selects the j^{th} decision from $c^k_A(S_k, e)$.

For example, in the knapsack problem, the computation tree of A_{Sac} is rooted in the empty solution (where no object is selected). Each node n of depth k models the partial solution $S_k = (d_1 \ldots d_k, h_1 \ldots h_k)$. The new object $e = d_{k+1}$ to select is the object with the best value/weight ratio that is not already in $\{d_1 \ldots d_k\}$. If putting e into the knapsack is allowed (i.e. $c^k_A(S_k, e) = (1, 0, \perp)$), n has two children $S' = (d_1 \ldots d_k\ e, h_1 \ldots h_k\ 1)$ and $S'' = (d_1 \ldots d_k\ e, h_1 \ldots h_k\ 0)$. Otherwise, it has only one child, S''.

[18] This decision must be located before $\perp$.

The algorithm's solutions to a problem (the potentially satisfying solutions)[19] are the leaves of its computation tree. The search for a satisfying solution using the algorithm consists of a deep-first search over the computation tree.

In summary, the task model for an MTT teaching a learner how to apply a backtracking algorithm to solve an optimization/search problem must include the following:

- the data set D and decisions from H;
- the problem's objective, which is to find a satisfying solution;
- ordering functions r^k_A;
- choice functions c^k_A;
- deep-first search in the algorithm's computation tree.

3.3 Modeling a backtracking algorithm in Astus

This section describes our procedural graph for a generic problem $P = (D = \{d_i\}, f_P, H)$[20] and a generic backtracking algorithm $A = (\{r^k_A\}, \{c^k_A\})$ for this problem. This graph can be adapted to author an MTT for a specific task that involves a backtracking algorithm by instantiating the parts of the graph corresponding to the task-specific ordering function r^k_A and choice function c^k_A. Two instantiation examples are detailed in Section 3.4.

The sets D and H and the partial solution S' the learner builds are declarative knowledge and thus located in the KB. In an optimization problem, the KB also contains the best solution S built by the learner so far. Object attributes and concepts are defined by the MTT's author depending on the modeled task.

In the Astus framework it is important to note that the interface will often have visual matches of S and S'. These visual representations help the learner:

- visualize the current solution and decide how to extend it;
- compare the current solution to the best one (for optimization problems);
- memorize the best solution without confusing it with other solutions.

---

[19] Most of the solutions built using an algorithm will not be satisfying. This is why we have used the term "potentially" satisfying.

[20] The distinction between optimization and search problems will be made when required.

For example, an MTT for the knapsack problem will have a visual representation of every object, a visual effect applied to the objects that have been added to the knapsack, and a representation of the best knapsack configuration encountered so far.

The problem's objective, the ordering function r^k_A, the choice function c^k_A and the deep-first search in the computation tree define how to solve the task. They are thus procedural knowledge and are included in the procedural graph (see Figure 2).

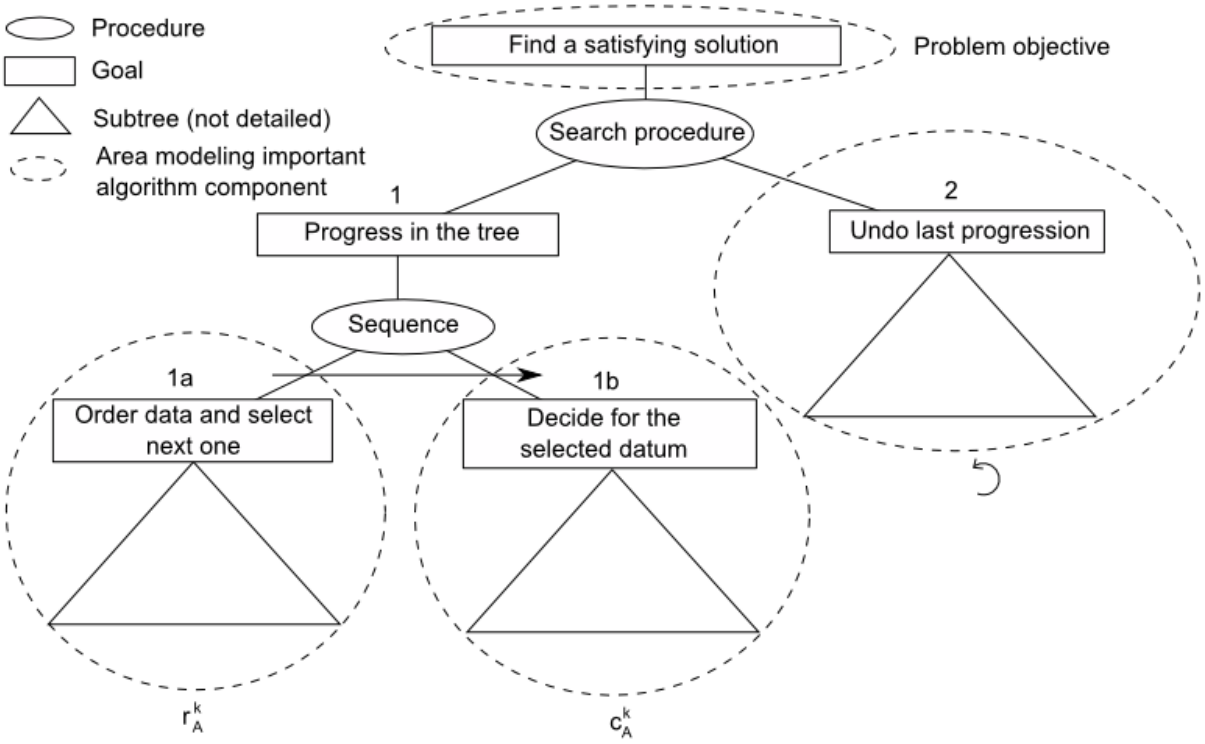

Fig. 3. Procedural graph of a generic backtracking algorithm. The subgraphs generated under the subgoals 1a, 1b and 2 must be instantiated by the MTT's author according to the task.

Goal labels in Figure 2 are generic and illustrate the abstract intent they model. In the context of a specific task, these labels should be more precise to help learners achieve these goals.

Root goal. The root of the graph is an achievement goal modeling the task's objective "`find a satisfying solution`". This goal is *validated* if and only if there is a satisfying solution in the KB (i.e., the learner has found a satisfying solution and knows that it is satisfying). Pedagogical interventions about this goal will be given using its label

(specified by the MTT's author). In A_{sac}, for example, this label could be "`find the best content for the knapsack`".

The MTT examines the root goal's condition *Cond* to evaluate whether it is *validated* or not. In a search problem, *Cond* checks whether *S'* is evaluated by the objective function as *valid.* In an optimization problem, *Cond* checks whether *S'* is the last solution to build (i.e, is a computation tree leaf and all decisions are less desirable ones). As *S* is the best solution encountered so far, if *S*' is the last solution to build, the learner has examined all solutions and thus, *S* is the best of all solutions: *S* is satisfying. In other words, *Cond* checks whether the learner has examined all interesting solutions. This ensures that the best solution encountered so far is the best one and thus, is satisfying.

Search procedure. The root goal is decomposed by a search procedure that simulates the learner's behavior when applying the backtracking algorithm. This procedure allows the learner to progress in the tree (subgoal 1 Figure 2) or to backtrack (subgoal 2, Figure 2). The MTT sometimes allows the learner to give a trivial answer to the problem via a third subgoal (see GNT404 example, Section 3.4). This type of subgoal is not included in Figure 2 because such subgoals are not important parts of the algorithm: they merely allow the learner to avoid some optional (and/or redundant) steps.

Each subgoal of the search procedure is associated with a condition that describes when the learner should achieve it. The MTT uses these conditions to trace the learner's steps and produce interventions (Beaulieu et al., submitted) to help him perform the correct steps.

A backtrack (subgoal 2, Figure 2) occurs when (1) decisions have been made for each datum; (2) no decision can be made for the next datum; or (3) all the decisions for the selected datum have been considered. Otherwise, the learner has to extend the solution (i.e., progress in the computation tree: subgoal 1, Figure 2).

In the knapsack problem, case (1) occurs when the learner has already made a decision for every object about whether or not it should be added to the knapsack. Case (3) occurs when he/she has examined all the possible decisions for the current object. Case (2) does not occur in the knapsack problem. It occurs in the *WordResearch* problem when it is not

possible to add a letter to the current word without building an invalid one. For example, given the word *abacaba*:

- *abacabaa* is invalid because of the substring *a*;
- *abacabab* is invalid because of the substring *ab*;
- *abacabac* is invalid because of the substring *abac*.

When modeling a task, it is important that the conditions of the search procedure's subgoals only model reachable situations. This simplifies the process of modeling conditions and prevents the MTT from generating interventions that describe impossible situations. For example, the *knapsack* problem condition for a backtracking goal describes only cases (1) and (3). Thus, the MTT does not explain to the student that he/she also has to backtrack when he/she cannot make a decision, because this does not occur: making a decision is always possible.

Subgoal 1: Progress in the tree. A sequence procedure models how to progress using a deep-first search in the tree. The first subgoal s_{1a} of this procedure is an achievement goal, "`order data and select the next one`" (subgoal 1a, Figure 2). The learner does not need to achieve this goal after completing a backtracking goal: he/she only has to make a new decision. Let $S = (O, X)$ be the current partial solution. The subgoal s_{1a} is *validated*, i.e., the learner is not required to select a new datum, if and only if $|O| > |X|$, indicating that there are is at least one selected datum for which there is no current decision. Thus, the condition for s_{1a}, which describes when it is *validated*, is $|O| > |X|$.

Making a decision for the selected datum is the second subgoal of the procedure, "`decide for the selected datum`" (Subgoal 1b, Figure 2).

Subtrees corresponding to detailing the data selection and the decision making processes are not illustrated in Figure 2 because they depend on the calculation of $\{r^k_A\}$ and $\{c^k_A\}$, which are themselves modeled using Astus's procedural knowledge components. MTTs created with Astus use these sub-trees to trace the learner's steps and to generate interventions when he/she calculates the r^k_A and/or c^k_A he/she needs.

In some cases, it might be reasonable for the MTT's author to assume that the learner already knows how to evaluate $\{r^k_A\}$ and $\{c^k_A\}$. In such cases, the subtrees corresponding to these functions are modeled as PPs and $\{r^k_A\}$ and $\{c^k_A\}$ are modeled by queries used to parameterize their respective PPs. In other cases, these subtrees are defined using CPs that decompose their goals into the required steps.

Subgoal 2: Backtrack. Let *S' = (O', X')* be the learner's current solution. When the learner backtracks, he/she cancels his/her last decision. In other words, he/she deletes the last element *x* of *X'*. Once the backtracking goal is completed for a specific decision, the search procedure generates a new subgoal, either to progress in the tree or to backtrack again, depending on the task's state.

Sometimes, all decisions for the last selected datum have been studied. When faced with such a situation, the learner has to backtrack further and cancel the decision made for the previous datum. In such a case, he/she must also backtrack on the selection of the last object of *O'* (the last selected datum) since this selection was made after the decision he/she is going to cancel.

The subtree corresponding to the backtracking process is not illustrated in Figure 2, as its instantiation depends on the task. For example, in order to backtrack in the *WordResearch* problem, a learner might use a PP that simultaneously cancels the last decision (the last letter added) and the datum selection (position of the letter). In other domains, backtracking may require multiple steps. For example, when integrating mathematical functions using integration by parts (IPP), the MTT may require the learner to first erase the subfunctions chosen during the IPP before canceling the IPP itself, as a way of reminding the learner that other subfunctions may work better.

Overall, modeling backtracking algorithms in Astus can be achieved by instantiating a procedural graph similar to the one shown in Figure 2. This procedural graph contains 4 key elements that are essential to backtracking algorithms:

1. the problem's objective, modeled by the graph's root;

2. the order in which the data must be selected (the ordering function $\{r^k_A\}$), modeled by the procedure decomposing the goal "*order data and select the next one*";
3. the decision-making process for the selected data (the choice function $\{c^k_A\}$), modeled by the procedure decomposing the goal "*decide for the selected data*";
4. the deep-first exploration of the computation tree, modeled using a search procedure and the conditions of its subgoals: "*progress in the tree*" and "*undo last progression*".

New-sys structures make modeling these elements easier and help Astus's modules to access their semantics. Thus, Astus's modules are able to trace the learner's steps and to produce pedagogical interventions as in other domains (Lebeau et al., 2010; Paquette et al., 2010; Paquette et al., 2012a; Paquette et al., 2015). Specifically, the expert module:

- traces the learner's steps when he/she makes his/her deep first search in the computation tree using the search procedure described in Beaulieu et al. (2015);
- checks that the learner selects the data in the correct order and makes the right decisions, using strategic knowledge contained in the $\{r^k_A\}$ and $\{c^k_A\}$ functions. Astus's use of strategic knowledge is described in Beaulieu et al. (submitted);
- diagnoses mistakes. Using this diagnosis, the pedagogical module produces interventions describing when the subgoal attempted by the learner must be achieved (Beaulieu et al., 2015).

3.4 Procedural graph instantiation examples

In order to incorporate backtracking algorithms in MTTs authored using Astus, the procedural graph illustrated in Figure 2 must be instantiated for a specific task. In this section, we discuss two possible instantiations: one for the integration of mathematical functions and the other for the identification of DNA molecules. Of these two tasks, only the DNA identification problem was implemented in a working MTT.

Integrating functions. When solving an integration problem, the learner has to choose an integration method, apply it to the current function, assess whether it successfully integrated the function and backtrack if application of the method did not result in a successful integration. In correctly applying this algorithm, the learner produces this trace:

1. He/she chooses an integration method (i.e., subgoal 1, Figure 2).
2. Depending on the results of the method, the learner:
    a. ends the algorithm if a primitive function is obtained (root goal of Figure 2 *validated*);
    b. applies step 1 again on the functions obtained by decomposing the first one (for example, integrating *fg'* in an IPP);
    c. backtracks and changes integration method (i.e., subgoal 2, Figure 2).

Figure 3 is the procedural graph of this process, derived from Figure 2.

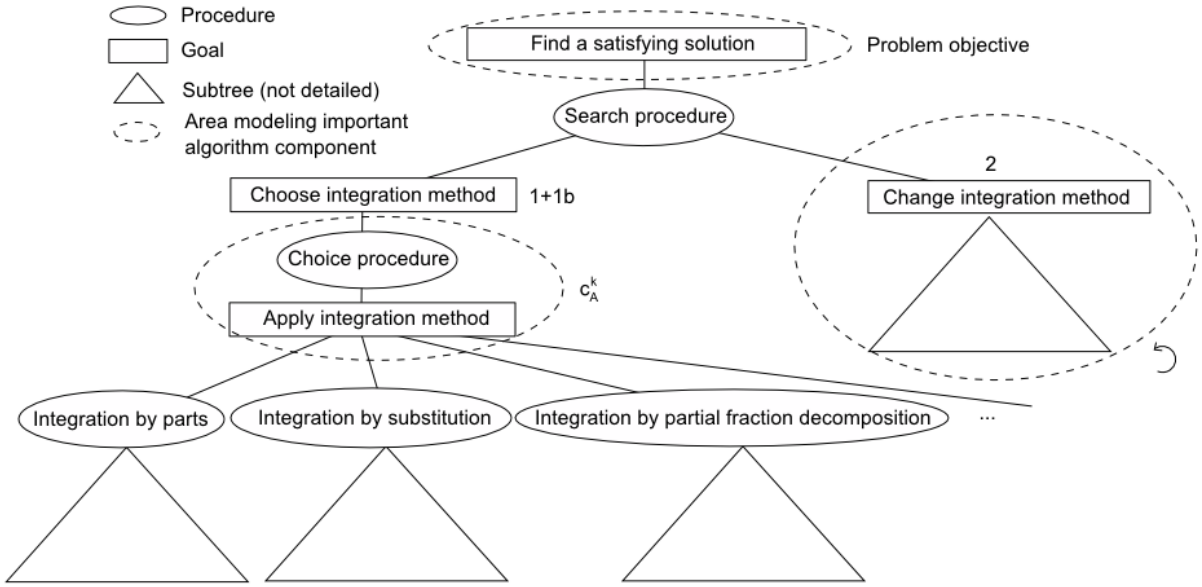

Fig. 4. Procedural graph for function integration. It comes from the procedural graph of Figure 2: the subgoal that orders data and selects one is deleted and the subgoal that consists of making a decision is extended.

In the function integration problem, the data set D is the set of all functions the learner may learn to integrate. The set of decisions H are the integration methods the learner can apply. To solve an integration task, the learner chooses and applies a method and, if needed, solves every subproblem that comes from the application of this method. In other words, the

learner builds a solution $S = (O, X)$ composed of the sequence O of functions that are successively integrated and the sequence X of integration methods used. Thus, S is both a decomposition of the initial function into subfunctions and a list of the methods used to integrate those functions.

Let's analyze what happens to subgoals 1a and 1b from Figure 2. These subgoals consist of selecting a datum (i.e., a function to integrate) and then making a decision for this datum (i.e., choosing a method). When integrating a function, the learner does not explicitly select functions to integrate. Rather, the functions he/she works on are naturally imposed by the integration methods being applied and the way he/she applies the method. For this reason, the integration problem does not instantiate an analogous goal to Figure 2's subgoal 1a: the goal to "`Progress in the tree`" consists solely of selecting and applying an integration method.

When applying an integration method, the learner may decompose the initial function into subfunctions which will need to be integrated (partial fraction decomposition) or into a different function, for example when applying IPP or integration by substitution (IBS). These new functions will be the next "data" the leaner will have to work on.

There are two ways to instantiate the graph in Figure 3, depending on pedagogical objectives. The first way aims to teach specific integration methods. It requires that, once the learner has chosen an integration method, every possible different application of this method be tested before backtracking and choosing a different method. This teaches learners how to apply one specific method to integrate a function. For example, the subgraph modeling integration by parts is also a backtracking algorithm. The objective of this subgraph is to look for the correct subfunctions f and g.

The second way provides the learner with the freedom to choose a new integration method after each application of a method or reapply a previously applied method using different parameters (for example, reapplying IBS after changing the chosen variable). It teaches the learner how to choose the correct integration method and solve the problem. In that case, a decision comprises both the integration method and the subfunctions to use. This

allows the student to change his/her mind and review his/her decision, either by changing the integration method or by swapping the subfunctions.

These possibilities show two things:

- Backtracking algorithms may be nested. As it is made from simple structures, our graph can be adapted to the task to model even when some of its parts are also backtracking algorithms.
- Our model is flexible enough to be adapted to pedagogical objectives.

Identifying DNA molecules. The DNA molecule identification problem was modeled for the creation of an MTT used in a genetics course (GNT404) at Université de Sherbrooke. It teaches learners how to locate the position of DNA fragments resulting from enzymatic digestion of a DNA molecule.

Let *e'* be an enzyme cutting the molecule once at a location (called the cutting site) known to the learner. In order to place the fragments from the digestion d_1 of the DNA by an enzyme *e*, the learner creates a digestion d_2 using both *e* and *e'*. Let f be the fragment from d_1 cut by *e'*. f is in d_1 but not in d_2, where it has been cut into two shorter fragments that can be identified. As the learner knows where the cutting site of *e'* is located, he/she knows that the two small fragments are on either side of that site, yielding only 2 possible placements for *f*. Each of these possibilities leads to one hypothesis whose blanks must be filled with the remaining fragments.

In testing hypotheses to identify the correct one, the learner can be blocked by physical constraints (fragment size or previously located fragment). In that case, he/she backtracks, removing previously placed fragments in order to try another answer. If the hypothesis being tested is valid, he/she will find a correct placement and validate the hypothesis. Otherwise, he/she will reject it and will have to test the other hypothesis. Figure 4 is the procedural graph for this task.

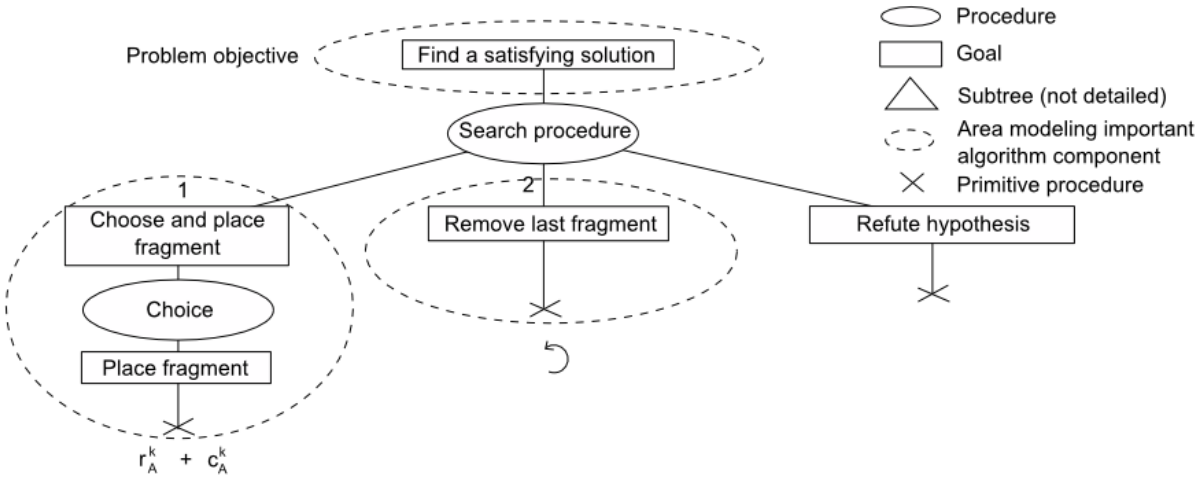

Fig. 4. A subgraph illustrating GNT404's procedural graph for the backtracking process. Data-ordering and decision-making subtrees are combined in one branch. A third branch is added in order to directly refute wrong hypotheses when possible.

In the DNA molecule identification problem, the data are fragment positions that are not directly manipulated by the student (he/she only orders the fragments, their position resulting from the length of previous fragments). Therefore, the instantiation of *"progress in the tree"* and *"undo last progression"* have been modified in a way common to all backtracking algorithms whose data manipulation is implicit (or imposed by the task). Specifically, these modifications are as follows:

- Combine the data-ordering subgoal and the decision-taking subgoal into one (subgoals 1a and 1b in Figure 2 resulting in subgoal 1, Figure 5). This modification is made because the learner is not required to directly manipulate data (fragment positions).
- Use a PP to model the backtracking (subgoal 2, Figure 4). This modification is made because the decision about which position to place a fragment in is implicitly modified when removing another fragment.

Another modification to the graph in Figure 2 is the addition of a third branch (subtree induced by subgoal 3, Figure 4). This branch models the process used by the learner to identify trivial situations (wrong hypotheses), as when the total length of the hypothesized

molecule is greater than the real length of the DNA molecule. This modification is specific to GNT404 and not common to all domains where data manipulation is implicit.

Identifying trivial situations leads to interesting interventions because the learner has two alternatives: identify the situation (optimal option) or solve the problem using the usual method. Thus, the MTT has to explain to the learner what possibilities are available and what the best option is, leading to the intervention in Figure 5. The process for generating these instructions is described in Beaulieu et al. (submitted).

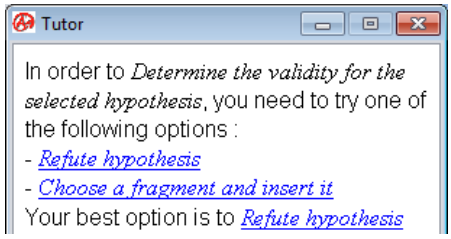

Fig. 5. Strategic next-step hint that explains to the learner he/she can directly refute the hypothesis.

This subgoal may also lead to new mistakes. The process of detecting these and producing negative feedback is described in Beaulieu et al. (2015).

Thus, Astus's MTTs use instantiations of the procedural graph in Figure 2 to teach learners how to solve the task using backtracking algorithms. The Figure 2 graph has to be instantiated and adjusted to the task being modeled. Specifically, these adjustments involve:

- adding new branches to allow the learner to identify particular situations with known outcomes;
- using PPs instead of subtrees when the learner is assumed to have already mastered the steps;
- combining multiple branches into one when the corresponding steps are implicit.

4 Modeling parallelism constraints

In Astus, when there are no ordering constraints between the subgoals of a sequence or iteration procedure, the learner is always allowed to pursue them simultaneously. Constraints preventing him/her from doing so, called parallelism constraints, are sometimes needed:

- to correctly solve the task;
- to focus the learner's attention on the subgoal he/she is achieving;
- for technical reasons: for example, to improve performance.

This section describes the modifications made to Astus's KRS in new-sys to allow the modeling of parallelism constraints and the generation of pedagogical interventions related to these constraints.

4.1 New-sys modifications

We adopt a syntax similar to that of the order constraints. This not only allows the modeling of parallelism constraints but also the use of similar techniques to generate pedagogical interventions.

Sequence procedure. The sequence procedure now allows the MTT's author to model a set of parallelism constraints $\{(b_i, b_j), (b_l, b_k) \ldots\}$ between goals. A constraint (b_i, b_j) forces the learner who starts the goal b_i (i.e., b_i is *executing*) to *validate* it before performing any step related to b_j. When b_i transitions to *executing*, the subtree rooted in b_j transitions to *waiting*. This subtree is returned to its previous state when b_i is *validated*.

By setting subtrees subject to parallelism constraints to the *waiting* state, the MTT:

- prevents the learner from performing any related step;
- makes error diagnosis easier by comparing the executed step to the *waiting* PP (see Section 4.2).

Iteration procedure. Subgoals generated by an iteration procedure are instances of the same goal, parameterized using an object set that is built during the task-solving process. There are two situations:

- The objects are known when the domain is modeled. In that case, the author can use a sequence procedure to define parallelism constraints precisely.
- If the objects are not known, it is hard to define these constraints because the objects are not generated yet and because the constraints have to be explained automatically to the learner using the task model.

We defined parallelism constraints in the iteration procedures in the simplest possible way, using a boolean variable: all objects can be treated simultaneously depending on the variable value (Section 5 discusses how this model could be improved in future research work). If the learner is allowed to treat objects simultaneously the usual process is applied. Otherwise, when one of the subgoals *b* becomes *executing*, others (and their subtrees) become *waiting*. They will become *available* again once *b* is *validated* by the learner.

4.2 Pedagogical intervention generation

The different procedure types are the ways in which a learner can achieve a goal. When the learner makes an error, he/she has applied the procedure in an incorrect way. Adding parallelism constraints to sequence and iteration procedures modifies the way the learner can achieve his/her goals. Therefore, the MTT has to handle new errors. It must:

- use the parallelism constraints model to detect the learner's errors;
- generate pedagogical interventions explaining what the learner did wrong and what he/she needs to do to solve the task.

Let *b* and *b'* be two goals with a parallelism constraint and assume the learner has started *b* (i.e., *b* is *executing*). If the learner does not respect the constraint (i.e., he/she executes a PP *p* resulting from the decomposition of *b'*), the MTT has to stop him/her and provide an intervention explaining that he/she has already started *b* and is not allowed to start *b'* until *b* has been achieved.

This error can be detected because there is no *available* PP matching *p*. When such a situation occurs, the expert module can check whether there is a match between the executed step and any of the *waiting* PPs. If a match is found (here, *p* matches the executed step) the module looks in the episodic tree for the match's highest *waiting* ancestor *b'*. Then, it looks for the currently *executing* subgoal *b* that forbids the achievement of *b'*. If the parent procedure of *b* and *b'* is an iteration procedure, *b'* is the only *executing* subgoal of *p*. If it is a sequence procedure, *b'* is one of the *executing* subgoals such that *(b, b')* is one of its parallelism constraints.

In order to generate an intervention related to the learner's step, the procedure types of *p*, *b* and *b'* are passed to the pedagogical module. The pedagogical module uses this information to generate feedback explaining to the learner that he/she cannot achieve *b* while executing *b'*. This feedback is instantiated from a domain-independent template that depends on the procedure type and completed using the labels of subgoals *b* and *b'*.

If *p* is an iteration procedure, the feedback is provided with visual effects highlighting the differences between *b* and *b'* and targeting their parameters: the parameter from the set for the procedure associated with *b*' is highlighted in blue whereas the one associated with *b* is highlighted in red (see Figure 6).

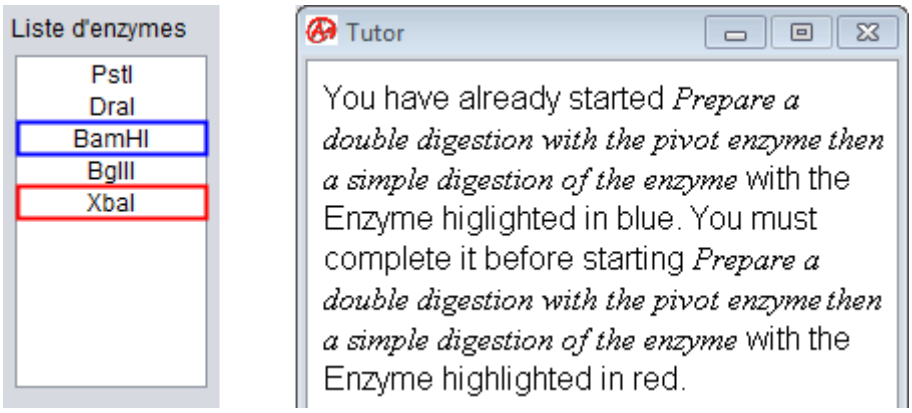

Fig. 6. Feedback and visual effects when the learner executes a step that does not respect a parallelism constraint in GNT404.

In the example illustrated in Figure 6, the task needed a parallelism constraint because it forces the learner to prepare digestions involving the same enzyme side by side. Digestions are placed in a gel where fragments move depending on their length. As digestions involving the same enzymes often have fragments with the same length, placing them side by side will help learners see fragments which behave the same because they have the same length. This:

- spares learners the need to make multiple measurements;
- keeps them from obtaining different sizes for same-length fragments;
- indirectly helps them fill hypotheses later by ensuring they have the right fragment lengths.

If *p* is a sequence procedure, the feedback is simply a text message similar to the one in Figure 7, without any interface effect or parameters.

5 Conclusion

MTT authoring frameworks reduce the development costs of MTTs, using expressive knowledge representation systems to automate some of their roles. We have shown in previous articles that a hierarchical knowledge representation system is advantageous in MTTs in order to reduce development costs related to intervention generation (Paquette et al., 2012a). This article describes our extension to Paquette et al.'s (2015) work by improving the expressivity of Astus's hierarchical knowledge representation. The contributions of this article are as follows:

- We have provided a description of how to combine simple procedural structures to model backtracking algorithms. By using simple procedural structures instead of a new global one, we make instantiating and adapting a backtracking algorithm easier by allowing the addition or removal of appropriate sections depending on the task being modeled.
- We have presented a new model for parallelism constraints, by which the learner can now be prevented from pursuing two goals simultaneously. It was already possible to prevent this by using ordering constraints, but it was an incorrect way to use order constraints and could lead to inappropriate interventions.
    - We have also presented a set of processes that use this model to detect errors related to parallelism constraints and generates interventions to teach the learner which goals are incompatible for simultaneous achievement.

Moreover, the semantics of our knowledge representation are accessible to agents rather than being black-box modeled. This allows agents to create processes using the task model to achieve their objectives, as Paquette et al. (2015) did to generate pedagogical interventions, and as we did here to support parallelism constraints. This is an important advance, because humans naturally organize knowledge in a goal-procedure structure. Giving agents access to

the knowledge semantics will help them teach this knowledge to humans by understanding how this knowledge is organized in the human mind.

These contributions have been integrated into the Astus framework and now allow the modeling of domains and tasks that involve backtracking algorithms or parallelism constraints. Thus, Astus MTTs now use the task model to trace learner steps and produce pedagogical interventions as the learner is applying a backtracking algorithm or when he/she does not respect parallelism constraints. While we have used the Astus KSR, our results can be obtained in any knowledge representation offering similar or equivalent procedural structures.

Allowing authoring frameworks to model tasks that involve backtracking algorithms makes it easier to conduct scientific experiments on these tasks. Indeed, having an expressive knowledge representation makes it possible to modify the MTT's behavior to evaluate pedagogical benefits without changing the whole task model. For example, the MTT for DNA molecule identification has been used to conduct a study measuring the benefits of strategic interventions (Beaulieu et al., submitted).

Note that only one domain was modeled in our work. It would be interesting to ensure that our model effectively supports all tasks involving backtracking, by trying to model other domains. This would also allow us to evaluate whether learners' perceptions of MTT-generated interventions depend on the domain or task modeled.

Two improvements regarding constraints would be interesting:

- Automatically identify a visual effect for feedbacks about parallelism constraints in a sequence procedure, as we did for the iteration procedure, by highlighting their different parameters (Figure 6). Feedback related to a sequence procedure is provided in textual form only, and we think that an associated visual effect could help the learner understand the feedback.
- Improve order and parallelism constraints. For the sequence procedure, constraints are equivalent to a two-variable predicate. For the iteration procedure, parallelism constraints are all-or-nothing and ordering constraints allow only a total order on all

subgoals. We have chosen this model because we anticipate that building a more complete and complex model will also affect order constraints and the MTT pedagogical module that explains the constraints to the learner. Such an endeavor constitutes a full research project impacting both knowledge representation and intervention generation. For example, a learner who has to achieve three subgoals *A*, *B*, *C* could learn that he/she is allowed to do *C* only when *A* or *B* is *validated*, or that two of these subgoals can be pursued simultaneously, but not all of them.

Abbreviations

Astus	Apprentissage par Système Tutoriel de l'Université de Sherbrooke
CTAT	Cognitive Tutor Authoring Tool
KRS	Knowledge Representation System
MTT	Model Tracing Tutor
MTT GNT	Tutor developed for the GNT404 lesson of Université de Sherbrooke
New-Sys	Astus's knowledge representation system modified after the Paquette et al. (2015) article
CP	Complex Procedure
Pre-Sys	Astus's knowledge representation system described in Paquette et al. (2015) article
PP	Primitive Procedure

References

Alekhnovich, M., Borodin, A., Buresh-Oppenheim, J., Impagliazzo, R., Magen, A., Pitassi, T. (2005) Toward a model for backtracking and dynamic programming. *Proceedings of the 20th Annual Conference on Computational Complexity*, (pp. 308-322).

Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K. R. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. In M. Ikeda, K. D. Ashley, T. W. Chan (Eds.) *Proceedings of the 8th International Conference on Intelligent Tutoring Systems* (pp. 61-70), Berlin: Springer.

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

Aleven, V. (2010) Rule-Based Cognitive Modeling for Intelligent Tutoring Systems. In Nkambou, R., Mizoguchi, R. Bourdeau, J. (Eds.). *Advances in Intelligent Tutoring Systems*, Berlin: Springer.

Beaulieu, G., Paquette, L., Mayers, A. (2015). Gestion automatisée de buts de succès dans les tuteurs par traçage de modèle. *Environnements informatiques pour l'apprentissage humain*, Agadir.

Borodin, A., Nielsen, M., Rackoff, C. (2003). (Incremental) priority algorithms. *Algorithmica,* 37(4), 295-326.

Buresh-Oppenheim, J., Davis, S., Impagliazzo, R. (2011) A Stronger Model of Dynamic Programming Algorithms. *Algorithmica*, 60(4), 938–968.

Dastani, M., van Riemsdijk, M. B., Meyer, J.-J. C. (2006). Goal types in agent programming. *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06).*

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

Lebeau, J.F., Fortin, M., Abdessemed, A., Mayers, A. (2008). Ontology-based knowledge representation for a domain-independent problem-solving ITS framework. *The Sixth International Workshop on Ontologies and Semantic Web for E-Learning* (in conjunction with ITS 2008).

Lebeau, J.F., Paquette, L., Fortin, M., Mayers, A. (2010). An Authoring Language as a Key to Usability in a Problem-Solving ITS Framework. In: Aleven, V., Kay, J., Mostow, J. (Eds): *Proceedings of ITS 2010 Part II* (pp. 236-238).

Murray, T. (2003). An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In T. Murray, S. Blessing and S. Ainsworth (Eds.) *Authoring tools for advanced learning environments,* Dordrecht: Kluwer Academic Publishers.

Paquette, L., Lebeau, J.-F., Mayers, A. (2010). Integrating Sophisticated Domain-Independent Pedagogical Behaviors in an ITS Framework. In Aleven, V., Kay, J., Mostow, J. (Eds) *Proceedings of ITS 2010 Part II* (pp. 236-238).

Paquette, L., Lebeau, J.-F., Mayers, A. (2010bis). Authoring Problem-Solving Tutors: A Comparison Between Astus and CTAT *Advances in Intelligent Tutoring Systems,* 377-405.

Paquette, L., Lebeau, J.F., Mayers, A. (2012a). Modeling Learners' Erroneous Behaviours in Model Tracing Tutors. *Proceedings of UMAP 2012.*

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

Paquette, L., Lebeau, J.F, Beaulieu, G., Mayers, A. (2015). Designing a Knowledge Representation Approach for the Generation of Pedagogical Interventions by MTTs. *International Journal of Artificial Intelligence in Education (IJAIED).*

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

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge*, Morgan Kaufmann Press.

Woeginger, G. (2000). When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (FPTAS)? *INFORMS Journal on Computing 12,* 57-75.

Woolf, B.P., Cunningham, P. (1987). Building a community memory for intelligent tutoring systems. In K. Forbus, H. Shrobe (Eds.) *Proceedings of the Sixth National Conference on. Artificial Intelligence*, AAAI Press Menlo Park(CA).

Conclusion

Contributions

Cette thèse présente un ensemble de structures procédurales adaptées à un système hiérarchique, un modèle pour les algorithmes avec retour sur trace et des processus pour exploiter automatiquement ces structures au sein des MTT. Ces derniers assurent alors intelligemment le suivi de l'étudiant et la production d'interventions pédagogiques dans les domaines modélisés. Ainsi, les MTT bénéficiant des travaux de cette thèse offrent des interventions pédagogiques riches et diagnostiquent les erreurs durant la résolution de tâches impliquant des algorithmes avec retour sur trace, tout en épargnant à leurs concepteurs l'encodage de matériel pédagogique.

Les contributions informatiques de cette thèse sont donc une représentation de connaissances pour les algorithmes avec retour sur trace ainsi que l'exploitation automatisée de cette représentation par le MTT pour générer des interventions pédagogiques appropriées durant l'apprentissage. Plus précisément, cette thèse décrit un ensemble de structures procédurales permettant la modélisation de situations d'apprentissage diverses. Ces structures sont :

- Les buts de succès, dont la validation des objectifs dépend de l'état de la tâche et non pas d'un ensemble d'action (voir chapitre 1);
- Les procédures de tâtonnement, qui consistent à effectuer plusieurs tentatives pour atteindre un but de succès (voir chapitre 1);
- Les algorithmes avec retour sur trace. Ces algorithmes décrivent comment construire une solution à un problème tout en révisant régulièrement ses choix pour optimiser la solution construite (voir chapitre 3);

- Les contraintes de concurrence, qui empêchent la poursuite simultanée de deux objectifs à atteindre (voir chapitre 3) mais autorisent l'étudiant à choisir l'ordre dans lequel il les effectue.

Ces structures permettent la modélisation de connaissances dont la sémantique est exploitée par le MTT (voir chapitres 1 et 2). Le MTT génère alors des interventions pour enseigner ces connaissances à des apprenants. Ces interventions sont communiquées à l'apprenant lorsqu'il demande de l'aide ou après une étape pour lui confirmer la validité de l'action qu'il vient de faire. Elles sont enrichies par des effets d'interface et sont précisées à l'aide d'éléments spécifiques à la situation d'apprentissage pour faciliter leur compréhension.

Les contributions empiriques de cette thèse sont liées à la réalisation d'une expérience à l'Université de Sherbrooke. Cette expérience démontre que les interventions stratégiques générées à l'aide des travaux de cette thèse (au chapitre 2) augmentent le nombre choix optimaux effectués par des étudiants, qui démontrent alors une plus grande flexibilité et par conséquent une meilleure maitrise du domaine [19].

Critique du travail

Bien que les connaissances modélisées grâce à cette thèse soient exploitables par des agents logiciels et que les interventions produites par les MTT soient bénéfiques aux étudiants, aucune techniques de langage naturel n'a été employée pour rendre les interventions plus simples, plus courtes et, dans certains cas, plus compréhensibles par les étudiants. En effet, les gabarits utilisés permettent de générer facilement des messages compréhensibles, mais le résultat dépend fortement de la structure des connaissances modélisées. Ainsi, en épargnant l'encodage de matériel pédagogique au concepteur du MTT Astus on l'oblige néanmoins à faire attention à la façon dont il modélise le domaine.

En outre, le modèle des algorithmes avec retour sur trace de [1] à partir duquel à été conçu le schéma pour ces algorithmes est utilisé pour des démonstrations de complexité. Il a été choisi afin de que le modèle des algorithmes dynamiques avec retour sur trace soit le plus général

possible. Par conséquent, le schéma que nous proposons nécessite des ajustements (décrits au chapitre 3) si l'algorithme modélisé est relativement simple. Dans certains cas particuliers (typiquement des algorithmes fixés, voir chapitre 3) il pourrait exister d'autres structures procédurales plus adaptées à l'enseignement de ces algorithmes.

Enfin, d'un point de vue empirique, les étudiants participants à l'expérience n'ont pas été encadrés en laboratoire à cause de la durée de la tâche. D'une part, cela signifie que la résolution de la tâche implique de nombreuses habiletés qui ne sont pas toutes liées à des connaissances stratégiques. D'autre part, cela a conduit à l'absence de certaines données pouvant introduire un biais dans les résultats obtenus. Par exemple, il est possible que l'effet des interventions soit réduit parce que certains étudiants découvrent par eux-mêmes la méthode optimale. Les résultats de l'expérience sont donc potentiellement liés à tâche qui a été modélisée.

Travaux futurs de recherche

Il est possible de séparer les travaux futurs en plusieurs catégories. Il serait souhaitable de poursuivre l'extension des représentations de connaissances hiérarchiques afin de permettre la modélisation de plus de domaines et, idéalement de permettre la modélisation de méthodes de résolution de problèmes, voir même d'aboutir à la réalisation de MTT pour des domaines mal définis. En outre, étendre la gestion automatisée des rôles des MTT conçus par des outils d'auteur favorisera leur essor puisque les coûts de développement en sont le frein principal.

Il y a également un travail à faire sur la génération automatisée des interventions pédagogiques. Ce système, mis au point par Luc Paquette, utilise des gabarits d'interventions dont la pertinence et la clarté dépendent directement du modèle de la tâche. Des techniques de langage naturel peuvent être mises au point afin de réduire la longueur des interventions générées, ou pour les rendre plus claires. Le chapitre 1 présente une discussion au sujet des étiquettes de buts de succès générées automatiquement, mais cette discussion ainsi que les pistes suggérées pour les étiquettes des buts de succès s'appliquent à d'autres interventions produites par le MTT.

Une autre amélioration possible a été soulevée par la modélisation de contraintes de concurrence dans la plateforme Astus. Il est apparu que le modèle actuel des contraintes (d'ordre ou de concurrence) ne permet pas de représenter toutes les contraintes mais uniquement celles qui sont exprimables sous forme de prédicat binaire. En améliorant cette représentation, il serait possible de modéliser des domaines ayant des contraintes plus élaborées tout en assurant une génération automatisée des interventions.

Enfin, du point de vue empirique, il serait intéressant de conduire des expériences sur d'autres domaines que GNT404 pour réduire le nombre de paramètres qui influencent la maitrise du domaine. Cela confirmera l'apport des interventions stratégiques et la polyvalence du système de représentation de connaissance mis au point dans cette thèse.

Perspectives

Les bénéfices des travaux présentés dans cette thèse ont été validés en réalisant un tuteur pour le cours d'analyse d'ADN GNT404. La polyvalence de la représentation de connaissance établie favorisera leur utilisation des plateformes de conception dans plus de domaines. Ils assisteront donc les tuteurs humains en offrant une pédagogie ciblée :

- Grâce à de l'aide pour la prochaine étape ;
- Grâce à des rétroactions positives pour lui donner confiance ;
- Grâce à des rétroactions stratégiques pour lui expliquer comment faire des choix optimaux ;
- Grâce à des rétroactions négatives qui lui expliquent son erreur.

Les contributions apportées par cette thèse à la réalisation de MTT simplifient leur conception dans des domaines impliquant des algorithmes de avec retour sur trace pour lesquels il n'est pas possible de prédire comment résoudre la tâche. En cela, ces travaux constituent un premier pas vers la réalisation de MTT pour des domaines mal définis. Cela ouvre des portes vers la réalisation d'agents logiciels qui s'attaquent seuls à des problèmes mal définis à partir d'un ensemble de connaissances, que ce soit en planification pour

résoudre ces problèmes en examinant les contraintes et les objectifs de la tâche ou pour enseigner ces connaissances à des étudiants.

Bibliographie

[1] Alekhnovich, M., Borodin, A., Buresh-Oppenheim, J., Impagliazzo, R., Magen, A., Pitassi, T.: Toward a model for backtracking and dynamic programming in *Proceedings of the 20th annual conference on computational complexity,* 2005, pp 308-322.

[2] Aleven, V., McLaren, B.M., Sewall, J., Koedinger, K. R.: The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains, in *Proceedings of the 8th International Conference on Intelligent Tutoring Systems,* M. Ikeda, K. D. Ashley, T. W. Chan (Eds.) 2006, pp 61-70.

[3] Aleven, V.: Rule-Based Cognitive Modeling for Intelligent Tutoring Systems. In Nkambou, R., Mizoguchi, R. Bourdeau, J. (Eds.). *Advances in Intelligent Tutoring Systems*, Berlin: Springer, 2010.

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

[5] Kodaganallur, V., Weitz, R.R., Rosenthal, D.: A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms in *International Journal of Artificial Intelligence in Education*, vol. 15, 2005, pp. 117-144.

[6] Kodaganallur, V., Weitz, R.R., Rosenthal, D. : An Assessment of Constraint-Based Tutors: A Response to Mitrovic and Ohlsson's Critique of "A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms in *International Journal of Artificial Intelligence in Education,* vol. 16, pp. 291–321, 2006.

[7] Lebeau, J.-F., Paquette, L., Fortin, M., Mayers, A.: An Authoring Language as a Key to Usability in a Problem-Solving ITS Framework in *Proceedings of ITS 2010 Part II*, Aleven, V., Kay, J., Mostow, J. (Eds), 2010, pp. 236-238.

[8] Mitrovic, A.: A Knowledge-Based Teaching System for SQL in *Proceedings of ED-MEDIA,* 1998, pp 1027-1032.

[9] Mitrovic, A.: Fifteen years of constraint-based tutors: what we have achieved and where we are going in *User Modeling and User-Adapted Interaction* 22(1–2), 2011, pp. 39–72.

[10] Mitrovic, A., Suraweera, P. Martin, B., Zakharov, K., Milik, N., Holland, J.: Authoring Constraint-Based Tutors in ASPIRE in Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006), M. Ikeda, K. D. Ashley, & T. W. Chan (Eds.), 2006, pp. 41- 50 Berlin: Springer Verlag.

[11] Mitrovic, A., Martin, B., Zakharov, K., Milik, N., Holland, J., McGuigan, N.: ASPIRE: an authoring system and deployment environment for constraint-based tutors in *Artificial Intelligence in Education* 19 (2), 2009, pp. 155–88.

[12] Mitrovic, A., Koedinger, K.R., Martin, B.: A comparative analysis of cognitive tutoring and constraint-based modelling in *Proceedings of the Ninth International Conference on User Modeling UM 2003(Vol. LNAI 2702),* P. Brusilovsky & A. Corbett & F. d. Rosis (Eds.), 2003, pp. 313-322, Berlin: Springer-Verlag.

[13] Mitrovic, A., Ohlsson, S.: A Critique of Koaanallur, Weitz, and Rosenthal, "A Comparison of Model-Tracing and Constraint-Based Intelligent Tutoring Paradigms"", *International Journal of Artificial Intelligence in Education*, vol. 16, 2006, pp. 277-289.

[14] Paquette, L., Lebeau, J.-F., Mayers, A.: Authoring Problem-Solving Tutors: A Comparison Between Astus and CTAT in *Advances in Intelligent Tutoring Systems,* 2010, pp 377-405.

[15] Paquette, L., Lebeau, J.-F., Beaulieu, G., Mayers, A.: Designing a knowledge representation approach for the generation of pedagogical interventions by MTTs in *International Journal of Artificial Intelligence in Education (IJAIED)*, 2015, pp. 118-156.

[16] Paquette, L., Lebeau, J.-F., Beaulieu, G., Mayers, A.: Automating Next-Step Hints Generation Using ASTUS in *Proceedings of ITS 2012*, 2012, pp. 201-211.

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

[18] Sacerdoti, E. D.: A Structure for Plans and Behavior, New York: Elsevier.

[19] Schneider, M., Rittle-Johnson, B., Star, J.R.: Relations among conceptual knowledge, procedural knowledge, and procedural flexibility in two samples differing in prior knowledge in *Developmental Psychology,* Advance online publication, 2011,

doi:10.1037/a0024997.

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

[21] 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 in *International Journal of Artificial Intelligence in Education 15*, 2005, pp 147-204.