Exploiting Sequential Patterns Found in Users' Solutions and Virtual Tutor Behavior to Improve Assistance in ITS

Philippe Fournier-Viger, Usef Faghihi, Roger Nkambou and Engelbert Mephu Nguifo*
Department of Computer Sciences, University of Quebec in Montreal, Montreal, Canada
fournier_viger.philippe@courrier.uqam.ca // faghihi.usef@courrier.uqam.ca // nkambou.roger@uqam.ca
*Department of Computer Sciences, Université Blaise-Pascal, Clermont-Ferrand, France mephu@isima.fr

ABSTRACT

We propose to mine temporal patterns in Intelligent Tutoring Systems (ITSs) to uncover useful knowledge that can enhance their ability to provide assistance. To discover patterns, we suggest using a custom, sequential pattern-mining algorithm. Two ways of applying the algorithm to enhance an ITS's capabilities are addressed. The first is to extract patterns from user solutions to problem-solving exercises for automatically learning a task model that can then be used to provide assistance. The second way is to extract temporal patterns from a tutoring agent's own behavior when interacting with learner(s). In this case, the tutoring agent reuses patterns that brought states of "self-satisfaction." Real applications are presented to illustrate the two proposals.

Keywords

Temporal patterns, Sequential pattern mining, Educational data mining, Intelligent tutoring systems

Introduction

Using knowledge discovery techniques to uncover useful knowledge hidden in a massive amount of educational data has been the subject of much recent research (Baker, Barnes, & Beck, 2008). However, no research has considered mining temporal patterns in Intelligent Tutoring Systems (ITSs) and employed this knowledge to improve their ability to provide assistance. In this paper, we propose two ways of improving the behavior of ITSs by exploiting temporal patterns. Those are to (1) automatically learn task models from recorded novice and expert users' solutions to provide assistance to learners, and (2) building tutoring agents that can adapt their behavior to learners and situations by reusing previously successful patterns of tutoring actions. Our hypothesis is that temporal patterns found in ITSs constitute useful knowledge that can be exploited to improve their ability to provide relevant and adaptive assistance.

The paper is organized as follows. First, it introduces an algorithm for mining temporal patterns. Next, the paper describes two proposals based on this algorithm and describes how they are integrated in an ITS. Finally, the last section presents our conclusions and previews our future work.

Mining temporal patterns from sequences of events

According to Han & Kamber (2006), there are four main kinds of patterns that can be mined from temporal data. These are trends, similar sequences, sequential patterns, and periodical patterns. In this work we chose to mine sequential pattern (Agrawal & Srikant, 1995), as we are interested in finding relationships between occurrences of events that are logged in ITSs. To mine sequential patterns, several algorithms have been proposed (Han & Kamber, 2006). While classical sequential pattern-mining algorithms have for their only goal to discover sequential patterns that occur frequently in several transactions of a sequence database (Agrawal & Srikant, 1995), other algorithms have proposed numerous extensions to the problem of mining sequential patterns (Han & Kamber, 2006). For this work, we chose a sequential pattern-mining algorithm that we have developed (Fournier-Viger, Nkambou, & Mephu Nguifo, 2008a), as it provides several more features than classical sequential pattern algorithms, such as accepting symbols with numeric values, eliminating redundancy, and handling time constraints and contextual information. For a technical description of the algorithm, the reader can refer to Fournier-Viger et al. (2008a). Moreover, a Java implementation of the algorithm can be downloaded from http://www.philippe-fournier-viger.com/spmf/.

The algorithm takes as input a database D of sequences of events. An event $X = (i_1, i_2, \ldots i_n)$ contains a set of items $i_1, i_2, \ldots i_n$, that are considered simultaneous, and where each item can be annotated with an integer value. Formally, a

sequence is denoted $s = <(t_1, X_1), (t_2, X_2), \ldots (t_n, X_n)>$ where each event X_k is associated to a timestamp t_k indicating the time of the event. For example, the sequence *S1* of figure 1 (left) contains two events. It indicates that item *a* appeared with a value of 2 at time 0 and was followed by items *b* and *c* with a value of 0 and 4, respectively, at time 1. An event sequence $sa = <(ta_1, A_1), (ta_2, A_2), \ldots (ta_n, A_n)>$ is said to be contained in another event sequence $sb = <(tb_1, B_1), (tb_2, B_2), \ldots (tb_m, B_m)>$, if there exist integers $1 \leq k1 < k2 < \ldots < kn \leq m$ such that $A_1 \subseteq B_{k1}$, $A_2 \subseteq B_{k2, \ldots,} A_n \subseteq B_{kn}$, and that $tb_{kj} - tb_{k1}$ is equal to $ta_j - ta_1$ for each $j \in \{1 \ldots m\}$. The relative support of a sequence *sa* in a database *D* is defined as the percentage of sequences $s \subseteq D$ that contain *sa* and is denoted by *supD(sa)*. The problem of mining frequent sequences is to find all the sequences *sa* such that $supD(sa) \geq minsup$ for a sequence database *D*, given a support threshold *minsup*, and optional time constraints. The optional time constraints are the minimum and maximum time intervals required between the head and tail of a sequence and the minimum and maximum time intervals required between two adjacent events of a sequence.

As an example, figure1 illustrates a database of six sequences (left) and the corresponding patterns found for a *minsup* of 33% (right). Consider pattern M5. This pattern appears in sequences S4 and S5, respectively. It has a support of 33% (two out of six sequences) and is frequent. Now consider patterns M1 and M2. Because item *a* appears in sequences S1, S2, S3, and S4, with values 2, 2, 5, and 6, respectively, the algorithm separated these values into two groups to create patterns M1 and M2 instead of creating a single pattern with a support of 66 %. For each of these groups, the median (2 and 5) was kept as an indication of the values grouped. This clustering of similar values only occurs when the support is higher or equal to 2 * *minsup* (see Fournier-Viger et al., 2008a, for details).

ID	Sequences
S1	<(0,a{2}), (1,bc{4})>
S2	<(0,a{2}), (1,c{5})>
S3	<(0,a{5}), (1,c{6})>
S4	< (0,f), (1, g),(2,a{6}e)>
S5	<(0, f b{3}), (1,h),(2,ef) >
S6	<(0,b{2}), (1,d)>

→

ID	Mined sequences	Support
M1	<(0,a{2})>	33 %
M2	<(0,a{5})>	33 %
M3	<(0,a{2}), (1, c{5})>	33 %
M4	<(0,c{5})>	50 %
M5	<(0,f), (2, e)>	33 %
M6	…	…

Figure 1. A database of six sequences (left) and mined sequences (right)

First proposal: Automatically learning task models from users' solutions

Our first proposal is for the acquisition of domain knowledge in an ITS. Typically, domain experts have to provide relevant knowledge to an ITS so that it can guide a learner during problem-solving activities. One common way of acquiring such knowledge is to use the method of cognitive task analysis that aims to produce effective problem spaces or task models by observing expert and novice users for capturing different ways of solving problems. However, cognitive task analysis is a very time-consuming process (Aleven, McLaren, Sewall, & Koedinger, 2006), and it is not always possible to define a satisfying complete or partial task model, particularly when a problem is ill-structured. According to Simon (1978), an ill-structured problem is one that is complex, with indefinite starting points, multiple and arguable solutions, or unclear strategies for finding solutions. A domain that includes such problems and in which tutoring targets the development of problem-solving skills is said to be ill-defined (within the meaning of Lynch, Ashley, Aleven, & Pinkwart, 2006). An alternative to cognitive task analysis is constraint-based modeling (CBM) (Mitrovic, Mayo, Suraweera, & Martin, 2001), which consists of specifying sets of constraints on what is a correct behavior, instead of providing a complete task description. Though this approach was shown to be effective for some ill-defined domains, a domain expert has to design and select the constraints carefully, and it cannot support tutoring services such as suggesting next steps to be performed by a learner. Contrarily to these approaches, where domain experts have to provide the domain knowledge, a promising approach is to use knowledge discovery techniques to automatically learn a partial problem space from logged user interactions in an ITS, and to use this knowledge base to offer tutoring services. A few works have been done in this direction in the field of ITS (for example, Riccuci, Carbonaro, & Casadei, 2007; Matsuda, Cohen, Sewall, Lacerda, & Koedinger, 2007; Barnes & Stamper, 2008), but they either (1) require specifying a minimal set of background knowledge, (2) have been applied in well-defined domains, (3) rely on strong assumption such that tasks can be modeled as production rules, or (4) do not take into account learner profiles.

We propose a solution that is not constrained by those limitations. We illustrate this proposal in the context of CanadarmTutor (Kabanza, Nkambou, & Belghith, 2005) (Figure 2) a virtual learning environment for learning how to operate the Canadarm2 robotic arm on the international space station. The main learning activity in CanadarmTutor is to move the arm from one configuration to another. This is a complex task, because the arm has seven joints and the user must chose at any time the three best cameras for viewing the environment from around twelve cameras on the space station and adjust their parameters. We have integrated a tutoring agent in CanadarmTutor to provide assistance to learners during this task. However, there are a very large number of possibilities for moving the arm from one position to another, and because one must also consider the safety of the maneuvers, it is very difficult to define a task model for generating the moves that a human would execute (Fournier-Viger, Nkambou, & Mayers, 2008b). For this reason, instead of providing domain knowledge to the agent, we implemented a learning mechanism that allows the tutoring system to learn by recording the behavior of users performing a task. The tutoring system then uses this knowledge to provide assistance to learners. We describe next the three operation phases of the learning mechanism as they are implemented in CanadarmTutor, and an experiment.

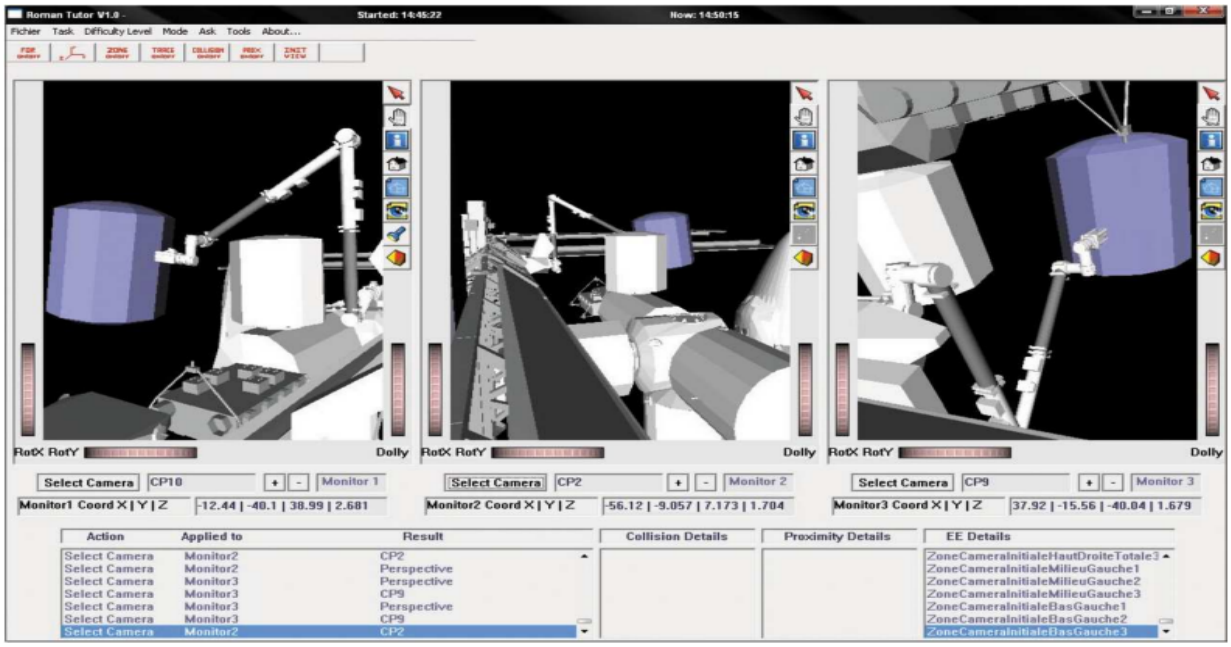

Figure 2. The CanadarmTutor interface

The observing phase

In the first phase, the tutoring system records the solutions of users that attempt an exercise. In CanadarmTutor, an exercise is to move the arm from an initial configuration to a goal configuration. For each attempt, the tutoring system logs a sequence of events. In this context, an event is a set of actions (items) that are considered unordered temporally. We defined 112 primitive actions that can be recorded in CanadarmTutor, which are (1) selecting a camera, (2) performing an increase or decrease of the pan/tilt/zoom of a camera, and (3) applying a rotation value to an arm joint. Actions of types (2) and (3) are annotated with integer values that indicate, respectively, the number of increments/decrements applied to a camera parameter and the number of degrees that a joint is rotated. An example of a partial action sequence recorded for an user in CanadarmTutor is <(0,6{2}),(1,63),(2,53{4}),(3,111{2})> , which represents decreasing the rotation value of joint SP (action 6) by two units, selecting camera CP3 (action 63), and increasing the pan of camera CP2 (action 53) by four units and then its zoom (action 111) by two units.

To annotate sequences with contextual information, we have extended the notion of sequence database with dimensional information, as suggested by Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., & Dayal, U. (2001). A sequence database having a set of dimensions $D = D_1, D_2, \ldots D_n$ is called an MD-Database. Each sequence of an MD-Database (an MD-Sequence) possesses a symbolic value for each dimension or the value "*", which means any value. A set of dimension values is called an MD-Pattern and is denoted *d1, d2, ... dn*. An MD-Pattern $Px = \{dx_1,$

$dx_2, \ldots dx_n\}$ is said to be contained in another MD-Pattern $Py = \{dy_1, dy_2, \ldots dy_m\}$ if $dx_1 \subseteq dy_1, \ldots dx_n \subseteq dy_m$. The relative support of a sequence (or MD-Pattern) in a sequence database *D* is defined as the percentage of sequences (or MD-Pattern) that contains it. Table 1 shows an example of MD-Database containing six learner plans annotated with five dimensions. The first dimension "Solution state" indicates if the learner plan is a succesful or buggy solution. In the case of CanadarmTutor, values for this dimension are produced by the tutoring system. The four other dimensions of Table 2 are examples of dimensions that can be added manually. Here, whereas the dimension "Expertise" denotes the expertise level of the learner that performed a sequence, "Skill_1", "Skill_2", and "Skill_3" indicate, respectively, if three specific skills were shown by the learner who performed the sequence. This example includes only five dimensions of three main types (skills, expertise level, and solution state). However, our framework can accept any kind of learner information or contextual information encoded as dimensions. In fact, in CanadarmTutor, we used 10 skills and the "solution state" dimension to annotate sequences.

Table 1. An example database containing 6 user solutions

ID	Dimensions					Sequence
	Solution state	Expertise	Skill_1	Skill_2	Skill_3	
S1	successful	expert	yes	yes	yes	<(0, a),(1, bc)>
S2	successful	novice	no	yes	no	<(0, d)>
S3	buggy	expert	yes	yes	yes	<(0, a),(1, bc)>
S4	buggy	intermediate	no	yes	yes	<(0, a), (1, c), (2, d)>
S5	successful	expert	no	no	yes	<(0, d), (1, c)>
S6	successful	novice	no	no	yes	<(0, c), (1, d)>

The learning phase

In the learning phase, the virtual agent applies the algorithm to find all MD-Sequence with a support higher or equal to *minsup*. For mining patterns, we set up the algorithm to mine only sequences of size two or greater, as shorter sequences would not be useful in a tutoring context. Furthermore, we chose to mine sequences with a maximum time interval between two adjacent events of two actions. The benefits of accepting a gap of two is that it eliminates some "noisy" (non-frequent) learners' actions, but at the same time it does not allow a larger gap size that could make it less useful for tracking a learner's actions. As an example, Table 2 shows some patterns that can be extracted from the MD-Database of Table 1, with a *minsup* of two sequences (33%). Consider pattern P3. This pattern represents doing action *b* one time unit (immediately) after action *a*. The pattern P3 appears in MD-sequences S1 and S3. It has thus a support of 33% or two MD-sequences. Because this support is higher or equal to *minsup*, P3 is deemed frequent. Moreover, the dimension values for P3 tell us that this pattern was performed by expert users that possess skills "Skill_1", "Skill_2", and "Skill_3", and that P3 was found in plan(s) that failed, as well as in plan(s) that succeeded.

Another important consideration is that when applying sequential pattern mining, there can be many redundant frequent sequences found. For example, in Table 2, the pattern P1 is redundant because it is included in the pattern P3 and has the same support. To eliminate this type of redundancy, we have adapted our algorithm based on Wang, Han, & Li (2007) and Songram, Boonjing, & Intakosum (2006) to mine closed MD-sequences. Closed MD-sequences are MD-sequences that are not contained in another sequence having the same support. Mining frequent closed MD-sequences has the advantage of greatly reducing the size of patterns found without information loss (Wang et al., 2007). Once patterns have been mined by our sequential pattern-mining algorithm, they form a partial problem space that can be used directly to provide tutoring services. However, one can also edit the patterns or annotate them with tutoring resources, such as textual hints.

Table 2. Some frequent patterns extracted from the dataset of Table 2 for *minsup* = 33%

ID	Dimensions					Sequence	Support
	Solution state	Expertise	Skill_1	Skill_2	Skill_3		
P1	*	expert	yes	yes	yes	<(0, a)>	33%
P2	*	*	*	yes	yes	<(0, a)>	50%
P3	*	expert	yes	yes	yes	<(0, a), (1, b)>	33%
P4	successful	*	no	*	*	<(0, d)>	50%
P5	successful	expert	*	*	yes	<(0, c)>	33%
P6	successful	novice	no	*	no	<(0, d)>	33%

The application phase

In the third phase, the tutoring system provides assistance to the learner by using the knowledge learned in the second phase. The basic operation that is used for providing assistance is to recognize a learner's plan. In CanadarmTutor, this is achieved by the plan recognition algorithm RecognizePlan, which is executed after each student action. When RecognizePlan is called for the first time, it iterates on the whole set of patterns found during the learning phase to note all the patterns that include the sequence of actions performed by the learner. If no pattern is found, the algorithm ignores the last action performed by the learner and searches again. This is repeated until the set of matching patterns is not empty or the size of the sequence of student actions is smaller than 2. In our test, removing user actions has shown to improve the effectiveness of RecognizePlan significantly. The next time it is called, it will be called with the set of matching patterns found by its last execution. This ensures that the algorithm will not consider patterns that have been previously rejected.

After performing preliminary tests with RecognizePlan, we noticed that, in general, after more than six actions performed by a learner, it becomes hard to tell which pattern the learner is doing. For that reason, we improved how the CanadarmTutor applies the sequential pattern-mining algorithm to extract a knowledge base. Originally, it mined frequent patterns for a whole problem-solving exercise. We modified our approach to add the notion of "problem states." In the context of CanadarmTutor, where an exercise consists of moving a robotic arm to attain a specific arm configuration, the 3D space is divided into 3D cubes, and the problem state at a given moment is defined as the set of 3D cubes containing the arm joints. An exercise is then viewed as going from a problem state P_1 to a problem state P_f. For each attempt at solving the exercise, CanadarmTutor logs (1) the sequence of problem states visited by the learner $A = P_1, P_2, \ldots P_n$ and (2) the list of actions performed by the learner to go from each problem state to the next visited problem state (P_1 to P_2, P_2 to P_3, … $P_{(n-1)}$ to P_n). After many users perform the same exercise, CanadarmTutor extracts sequential patterns from sequences of problem states visited and from sequences of actions performed for going from a problem state to another. To take advantage of the added notion of problem states, we modified RecognizePlan so that every time the problem state changes, RecognizePlan will be called with the set of patterns associated to the new problem state. Moreover, at a coarse grain level, a tracking of the problem states visited by the learners is also achieved by RecognizePlan. This allows connecting patterns for different problem states. We describe next the main tutoring services that a tutoring agent can provide based on the plan-recognition algorithm.

First, a tutoring agent can assess the profile of the learner by looking at the patterns applied. If, for example, a learner applies 80% of the time patterns with value "intermediate: for dimension "expertise," then CanadarmTutor can assert with confidence that the learner expertise level is "intermediate." In the same way, CanadarmTutor can diagnose mastered and missing/misunderstood skills for users who demonstrated a pattern by looking at the "skills" dimensions of patterns applied, and can estimate other aspects of a learner's profile. This results in rich information that can be used in various ways by a tutoring system. An example is given by the next tutoring service.

Second, a tutoring agent can guide the learner. This tutoring service consists of determining the possible actions from the current problem state and proposing one or more actions to the learner. In CanadarmTutor, this functionality is triggered when the student selects "What should I do next?" in the interface menu. CanadarmTutor then identifies the set of possible next actions according to the matching patterns found by RecognizePlan. The tutoring service then selects the action among this set that is associated with the pattern that has the highest relative support and that is the most appropriate for the estimated expertise level and skills of the learner. If the selected pattern contains skills that are not considered mastered by the learner, CanadarmTutor can use tutoring resources to explain them. If no actions can be identified, CanadarmTutor can rely on a special path planner to generate approximate solutions (see Kabanza et al., 2005 for details). In this current version, CanadarmTutor interacts with the learner only upon request. But it would be possible to program CanadarmTutor so that it can intervene if the learner is following an unsuccessful pattern or a pattern that is not appropriate for its expertise level. Testing different tutorial strategies with learners is part of our current work.

Finally, a tutoring service that has been implemented in CanadarmTutor is to let learners explore patterns to learn about possible ways of solving problems. Currently, the learners can explore patterns with a very simple interface. However, the learner could be assisted in this exploration by using an interactive dialog with the system that could prompt them on their goals and help them go through the patterns to achieve these goals.

Experiment

We conducted a preliminary experiment in CanadarmTutor with two exercises to qualitatively evaluate the virtual agent's capability to provide assistance. The two exercises each consist of moving a load with the Canadarm2 robotic arm to one of the two cubes (figure 3a). We asked 12 users to record plans for these exercises. The average length of a plan was 20 actions. From this data, CanadarmTutor extracted a partial problem space. In a subsequent work session, we asked users to evaluate the tutoring services provided by this version of CanadarmTutor. All users agreed that the assistance provided was helpful. We also observed that CanadarmTutor correctly inferred the expertise level of all the learners and thus provided hints that were adapted to the user profile. As an example of interaction with a learner, Figure 3b shows a hint message given to a learner upon request during scenario 1. The guiding tutoring service selects the pattern that has the highest support value, matches the last student actions, is marked "successful," and corresponds with the estimated expertise level of the learner. The given hint is to select camera CP4 on Monitor3, decrease the rotation value of the joint WP, and increase the rotation value of joint WE. The values on the right column indicate the values associated to the action. In this context, values 1 and 3 mean to rotate the joints 10º and 30º, respectively (1 unit equals 10º). By default, three steps are showed to the learners in the hint window depicted in figure 3b. However, the learner can click on the "more" button to ask for more steps or click on the "another possibility" button to ask for an alternative.

It should be noted that, although we applied the sequential pattern algorithm only one time after recording the learners plan, it would be possible to make CanadarmTutor apply it periodically to update its knowledge base, while interacting with learners.

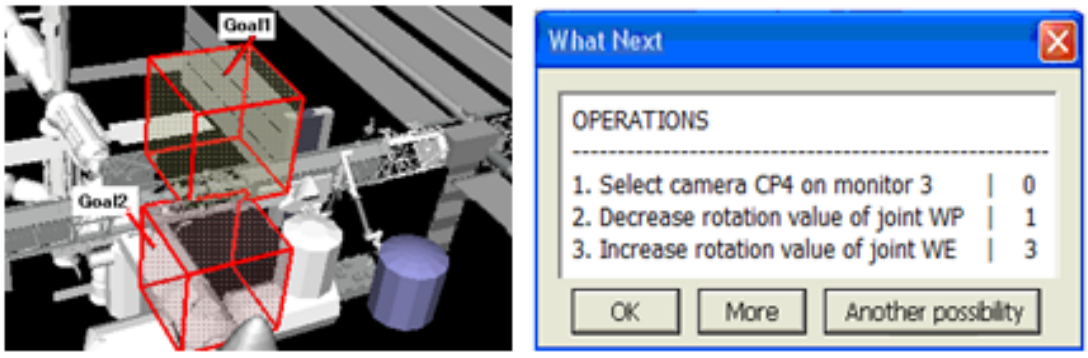

*Figure 3*a. The two scenarios; 3b. A hint generated by the virtual agent

Second proposal: A tutoring agent that learns from its own behavior

Our second proposal is to build tutoring agents that can learn from their own behavior by reusing previously successful patterns of tutoring actions. We illustrate this proposal with a virtual agent named CTS (Dubois, Poirier, & Nkambou, 2007) that we have also tested in CanadarmTutor to provide assistance to learners. The following subsections describe CTS, the three operation phases of the new learning mechanism that was integrated in CTS, and two experiments carried in CanadarmTutor to validate (1) the behavior of the new CTS and (2) the behavior of our sequential pattern-mining algorithm with large data sets.

The CTS cognitive agent

The Conscious Tutoring System (CTS) is a generic cognitive agent whose architecture (fig. 4) is inspired by neurobiology and neuropsychology theories of human brain function. It relies on the functional "consciousness" (Franklin & Patterson, 2006) mechanism for much of its operations. It also bears some functional similarities to the physiology of the nervous system. Its modules communicate with one another by contributing information to its working memory (WM) through information codelets. Based on Hofstadter et al's idea (Hofstadter & Mitchell, 1992), a codelet is a very simple agent, "a small piece of code that is specialized for some comparatively simple task." As in Baars's theory (Baars, 1997), these simple processors do much of the processing in the CTS architecture.

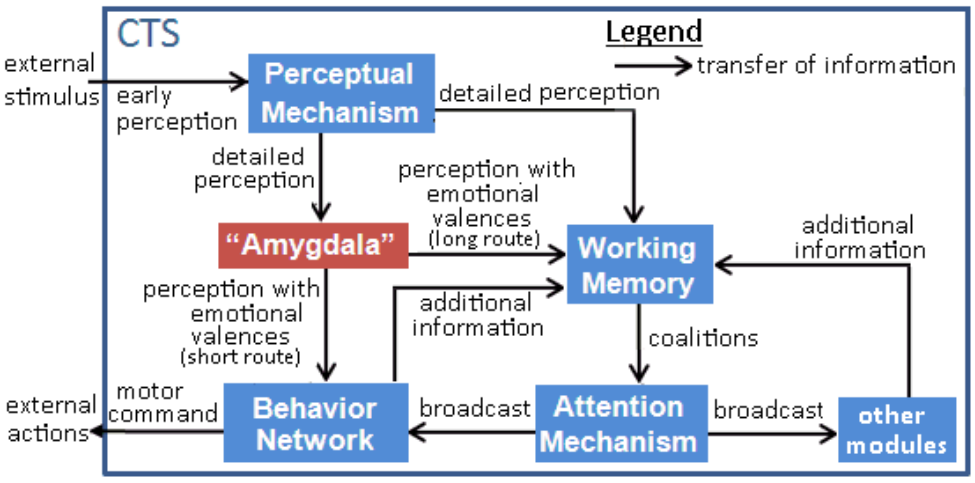

Figure 4. A simplified view of the CTS architecture (see Faghihi et al., 2008 for more details)

CTS possesses two routes for processing external stimuli (cf. fig. 4). Whereas the "long route" is the default route, the "short route" (which will not be described here) allows quick reactions when received information is deemed important by the pseudo-amygdala, the module responsible for emotional reactions in CTS (Faghihi, Poirier, Dubois, Gaha, & Nkambou., 2008). In both cases, the stimuli processing begins with percept codelets (Hofstadter & Mitchell, 1992) that perform collective interpretations of stimuli. The active nodes of the CTS's perception network constitute percepts. In the long route, these percepts enter WM as a single network of codelets, annotated with an activation value. These codelets create or reinforce associations with other already present codelets and create a coalition of information codelets. In parallel, the emotional codelets situated in the CTS's pseudo-amygdala inspect the previously mentioned coalition's informational content, and if is deemed important, infuse it with a level of activation proportional to its emotional valence. During every cognitive cycle, the coalition in the WM that has the highest activation is selected from the WM by the "attention mechanism" and is broadcast to all the modules in the CTS architecture. This selection process ensures that only the most important, urgent, or relevant information is broadcast in the architecture. Following a broadcast, every subsystem (module or team of codelets) that recognizes the information may react to the coalition by appending additional information to it. This process of internal publications (as suggested by Baars, 1997) can continue for many cognitive cycles before an action is executed by CTS. The module responsible for action planning, selection, and execution is the behavior network (BN) (Maes, 1989). When the BN receives a broadcast coalition, it selects the appropriate action to execute. In the current CTS version, we have designed the BN using a graphical authoring tool. We have implemented in CTS, the second proposal that we consider in this article. This learning mechanism is implemented in CTS by the three operation phases, described next.

The observation phase

In the first phase, the observation phase, CTS records a sequence of events (as defined in the second section of this article) for each of its executions. Each event $X = (t_i, A_i)$ represents one cognitive cycle. While the timestamp t_i of an event indicates the cognitive cycle number, the set of items A_i of an event contains (1) an item that represents the coalition of information-codelets that was broadcast during the cognitive cycle and (2) four optional items with numeric values indicating the four emotional valences (high threat, medium fear, low threat, compassion) associated with the broadcast coalition. CTS actually incorporates four emotions inspired by the OCC model of emotions (Ortony, Clore, & Collins, 1988). See Faghihi et al. (2008) for in-depth details about the emotional mechanism of CTS. An example of partial sequence recorded during our experiment was <(1, *c2*), (2, *c4*), (3, *c8 e2*{-0.4})>. This sequence shows that during cognitive cycle 1 the coalition *c2* was broadcast, followed by the broadcast of *c4* during cognitive cycle 2. Furthermore, it indicates that coalition *c8* was broadcast during the third cognitive cycle and that it generated a negative emotional valence of −0.4 for emotion *e2* (medium fear).

The learning phase

The second operation phase consists of mining frequent patterns from the sequences of events recorded for all executions of CTS by applying our sequential pattern-mining algorithm. This process is executed at the end of each CTS execution, from the moment where five sequences are available (five CTS executions). Currently, we have set up the sequential pattern-mining algorithm to mine only closed sequences with more than three events and with a support higher than 25%. Applying the algorithm results in a set of frequent sequential patterns.

The application phase

The third operation phase consists of improving CTS behavior by making CTS reuse relevant patterns that carry positive emotions. This is done by intervening in the coalition selection phase of CTS. The idea is here to find, during each cognitive cycle, the patterns that are similar to CTS's current execution, then to select as the next coalition to be broadcast the one most probable of generating positive emotions for CTS according to these patterns. Influencing the coalitions that are broadcast will then directly influence the actions that will be taken by the CTS behavior network. This adaption of CTS could be implemented in different ways. We used the SelectCoalition algorithm (figure 4), which takes as parameters (1) the sequence of previous CTS broadcasts (Broadcasts), (2) the set of frequent patterns (Patterns), and (3) the set of coalitions that are candidates to be broadcast during a given cognitive cycle (CandidateCoalitions). This algorithm first sets to zero a variable *min* and a variable *max* for each coalition in CandidateCoalitions. Then, the algorithm repeats the following steps for each pattern *p* of Patterns. First, it computes the strength of *p* by multiplying the sum of the emotional valences associated with the broadcasts in *p* with the support of *p*. Then, it finds all the coalition c ∈ CandidateCoalitions that appear in *p* after the sequence of the last *k* broadcasts of Broadcasts for any $k \geq 2$. For each such coalition *c*, if the strength of *p* is higher than *c.max, c.max* is set to that new value. If that strength is lower than *c.min, c.min* is set to that new value. Finally, when the algorithm finishes iterating over the set of patterns, the algorithm returns to CTS's working memory the coalition *c* in CandidateCoalitions having the highest positive value for the sum *c.min* + *c.max* and where *c.max* > 0. This coalition will be the one that will be broadcast next by CTS's attention mechanism. In the case of no coalition meeting these criteria, the algorithm will return a randomly selected coalition from CandidateCoalitions to CTS's working memory.

```
SelectCoalition(Patterns, Broadcasts, CandidateCoalitions)
   FOR each pattern c ∈ CandidateCoalitions
         c.min := 0. c.max := 0.
   FOR each pattern P of Patterns.
         Strength := CalculateSumOfEmotionalValences(P) * Support(P).
         FOR k := 2 to |P|.
               Sa := last k Broadcasts of Broadcasts.
               IF (Sa ⊆ P)
                  FOR each coalition c ∈ CandidateCoalitions appearing
                              after Sa in P
                        c.max := maxOf(Strength, c.max).
                        c.min := minOf(Strength, c.min).
   RETURN c ∈ CandidateCoalitions with the largest positive
             (c.max + c.min) and such that c.max > 0.
```

Figure 5. The SelectCoalition algorithm

The *c.max* > 0 criterion is included to ensure that the selected coalition appears in at least one pattern having a positive sum of emotional valences. Moreover, we have added the *c.min* + *c.max* criterion to make sure that the patterns with a negative sum of emotional valences will decrease the probability of selecting the coalitions that it contains. In our experiments, this criterion has proved to be very important as it can cause CTS to quickly stop selecting a coalition appearing in positive patterns if it becomes part of negative patterns. The reader should note that algorithms relying on other criteria could be used for other applications.

Testing the new CTS in CanadarmTutor

To test CTS's new learning mechanism, users were invited to perform arm manipulations using CanadarmTutor with integrated CTS. These experiments aimed at validating CTS's ability to adapt its behavior to learners. During these experiments, we qualitatively observed that CTS adapted its behavior successfully to learners. Two experiments are described here. The first describes in detail a situation that occurred with User 3 that illustrates well how CTS adapts its behavior thanks to the new learning mechanism. The second experiment describes how our sequential pattern-mining algorithm behaves when the number of recorded sequences increases.

User 3 tended to make frequent mistakes when he was asked to guess the arm's distance from a specific part of the space station. Obviously, this situation caused collision risks between the arm and the space station and was thus a very dangerous situation. This situation was implemented in the CTS's Behavior Network. In this situation, CTS has to make a decision between (1) giving a direct solution such as "You should move joint SP" (Scenario 1) or giving a brief hint such as "This movement is dangerous. Do you know why?" (Scenario 2).

During the interaction with different users, the learning mechanism recorded several sequences of events for that situation, each of them carrying emotional valences. The average length of the stored sequences was 26 events. For example, one partial trace saved when CTS gave a hint (scenario 2) to User 2 was <(13, *c11*), (14, *c14*), (15, *c15*), (16, *c18*), (17, *c19 e4*{0.8})>. In this trace, the positive valence 0.8 for emotion *e4* (compassion) was recorded because the learner answered an evaluation question correctly after receiving the hint. In another partial trace saved by CTS <(16, *c11*), (17, *c14*), (18, *c16*), (19, *c17*), (20, *c20 e2*{−0.4})>, User 2 received a direct solution from CTS (Scenario 1), but failed to answer correctly an evaluation question. This resulted in the valence −0.4 being associated to emotion *e2* (medium fear). After five executions, the learning mechanism extracted ten frequent sequences from the recorded sequences, with a minimum support (*minsup*) higher than 0.25.

Now turning back to User 3, during the coalition selection phase of CTS, the learning mechanism evaluated all mined patterns to detect similar patterns having ended by self-satisfaction. The learning mechanism chose the pattern <(0, *c11*), (1, *c14*), (3, *c18*), (4, *c19 e4*{0.8})> because it contained the most positive emotional valence, had the highest frequency, and the events (0, *c11*), (1, *c14*) matched with the latest events executed by CTS. Therefore, CTS decided that it was better to give a hint (Scenario 2) than to give the answer (Scenario 1) to User 3. This was achieved by broadcasting coalition *c18* (Scenario 2) instead of coalition *c16* (Scenario 1). If the emotional valence had not been as positive as was the case for previous users, CTS might have chosen Scenario 1 rather than Scenario 2. It should be noted that because the set of patterns is regenerated after each CTS execution, some new patterns can be created, while others can disappear, depending on the new sequences of events that are stored by CTS. This ensures that CTS behavior can change over time if some scenarios become less positive or more negative, and more generally that CTS can adapt its behavior to a dynamic environment. In this experiment, the learning mechanism has shown to be beneficial by allowing CTS to adapt its actions to learners by choosing between different scenarios based on its previous experience. This feature is very useful, as it allows the designers to include many alternative behaviors but to let CTS learn by itself which ones are the most successful.

We performed a second experiment with the learning mechanism, but this time to observe how our sequential pattern-mining algorithm behaves when the number of recorded sequences increases. The experiment was done on a 3.6 GHz Pentium 4 computer running Windows XP, and consisted of performing 160 CTS executions for a situation similar to the previous one where CTS had to choose between scenario 1 and scenario 2. In this situation, CTS conducts a dialogue with the student that includes from two to nine messages or questions (an average of six) depending on what the learner answers and the choices CTS makes (similar to choosing between scenarios 1 and 2). During each trial, we randomly answered the questions asked by CTS, and took various measures during CTS's learning phase. Each recorded sequence contained approximately 26 broadcasts.

Figure 6 presents the results of the experiment. The first graph shows the time required for mining frequent patterns after each CTS execution. From this graph, we see that the time for mining frequent patterns was generally short (less than 6 seconds) and increased linearly with the number of recorded sequences. In our context, this performance is very satisfying. However, the performance of the data-algorithm could still be improved as we have not yet fully optimized all of its processes and data structures. In particular, in future works we will consider modifying the algorithm to perform incremental mining of sequential patterns as some other sequential pattern-mining algorithms

do. This would improve performance, as it would not be necessary to recompute from scratch the set of patterns for each new added sequence.

The second graph shows the average size of patterns found for each execution. The size ranges from 9 to 16 broadcasts. The third graph depicts the number of patterns found. It remains low and stabilized at around 8.5 patterns during the last executions. The reason why the number of patterns is small is that we mined only closed patterns (see definition in the third section). If we had not mined only closed patterns, all the sub-sequences of each pattern would have been included in the results. Mining closed patterns is also much faster because, during the search for patterns, large parts of the search space that are guaranteed not to lead to close patterns are pruned. For example, for mining non-closed patterns from the first four sequences only, it took more than one hour (we stopped the algorithm after one hour), while mining closed patterns took only 0.558 seconds. The reason for this is that the four sequences share more than 15 common broadcasts. Therefore, if the pruning of the search space is not done, the algorithm has to consider all combinations of these broadcasts, which is computationally very expensive. This demonstrates that it is beneficial to mine closed patterns. Finally, the average time for executing the SelectCoalition algorithm at each execution was always less than 5 milliseconds. Thus, the costliest operation of the learning mechanism is the learning phase.

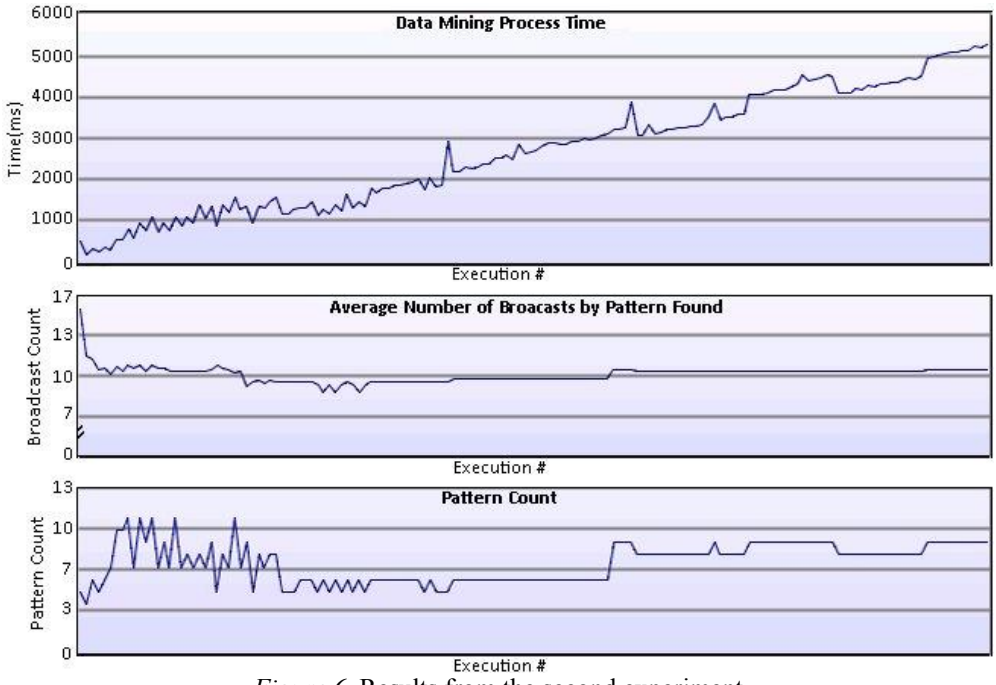

Figure 6. Results from the second experiment

Conclusion

In this article, we presented the idea of exploiting temporal data found in intelligent tutoring system logs to improve their capability to provide relevant and adaptive assistance. To demonstrate this idea, we described two proposals. While the first one is designed to learn task models from recorded novice and expert solutions to provide assistance to learners in problem-solving activities, the second one is aimed at building tutoring agents that can adapt their behavior to learners and situations by reusing previously successful patterns of tutoring actions. The two proposals should be reusable in other tutoring agents and domains, as the format for encoding behaviors is fairly generic.

In future work, we will perform further experiments to measure empirically how the different versions of CanadarmTutor influence the learning of students. We will investigate different ways of improving the performance of our sequential pattern-mining algorithm, including modifying it to perform an incremental mining of sequential patterns. We also plan to integrate the two proposals into other tutoring systems.

Acknowledgements

The authors thank the Canadian Space Agency, Fonds Québécois de la Recherche sur la Nature et les Technologies, and the Natural Sciences and Engineering Research Council for their logistic and financial support. The authors also thank members of GDAC/PLANIART teams who have participated in the development of CanadarmTutor.

References

Agrawal, R., & Srikant, R. (1995). Mining Sequential Patterns. *Proceedings of the International Conference on Data Engineering* (pp. 3–14), Los Alamitos, CA: IEEE Computer Society Press.

Aleven, V., McLaren, B. M., Sewall, J., & Koedinger, K. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. *Proceedings the 8th International Conference on Intelligent Tutoring Systems* (pp. 61–70), Berlin: Springer.

Baars, B. J. (1997). *In the theater of consciousness.* New York: Oxford University Press.

Baker, R., Barnes, T., & Beck, J. E. (2008). *Proceedings of Educational Data Mining 2008: 1st International Conference on Educational Data Mining*. Montreal, Quebec, Canada. June 20-21, 2008.

Barnes, T. & Stamper, J. (2008). Toward automatic hint generation for logic proof tutoring using historical student data, *Proceedings of the 9th International Conference on Intelligent Tutoring Systems* (pp. 373–382), Berlin: Springer.

Dubois, D., Poirier, P., & Nkambou, R. (2007). What does consciousness bring to CTS? *Proceedings of the 2007 AAAI Fall Symposium* (pp. 55–60), Menlo Park, CA: AAAI Press.

Faghihi, U., Poirier, P., Dubois, D., Gaha, M., & Nkambou, R. (2008). How emotional mechanism learn and helps other types of learning in a cognitive agent, *Proceedings of the 2009 IEEE/WIC/ACM Conference on Intelligent Agent Technology*, Los Alamitos, CA: IEEE Computer Society Press.

Fournier-Viger, P., Nkambou, R., & Mephu Nguifo, E. (2008a). A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems. *Proceedings of the 7th Mexican Conference on Artificial Intelligence* (pp. 765–778), Berlin: Springer.

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

Franklin, S., & Patterson, F.G.J. (2006). The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *Proceedings of 9th World Conference on Integrated Design & Process Technology*. San Diego: Society for Design and Process Science.

Han, J., & Kamber, M. (2006). *Data mining: Concepts and techniqu*es (2nd ed.). San Francisco: Morgan Kaufmann.

Hofstadter, D. R., & Mitchell, M. (1992). The copycat project: A model of mental fluidity and analogy-making. In K. Holyoak, J. & Barnden, J. A. (Eds.) *Advances in connectionist and neural computation theory 2* (pp. 31–113). Norwood, NJ: Ablex.

Kabanza, F., Nkambou, R., & Belghith, K. (2005). Path-planning for autonomous training on robot manipulators in space. *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (pp. 1729–173), Denver, CO: Professional Book Center.

Lynch, C., Ashley, K., Aleven, V. & Pinkwart, N. (2006). Defining ill-defined domains: A literature survey. *Proceedings of the Intelligent Tutoring Systems for Ill-Defined Domains Workshop at the 8th International Conference on Intelligent Tutoring Systems* (pp. 1–10), Jhongli, Taiwan, June 27, 2006.

Maes, P. (1989). How to do the right thing. *Connection Science, (1),* 291–323.

Matsuda, N., Cohen, W. Sewall, J., Lacerda, G., & Koedinger, K. (2007). Predicting students' performance with SimStudent: Learning cognitive skills from observation. *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 467–478), Amsterdam: IOS Press.

Mitrovic, A., Mayo, M., Suraweera, P., & Martin, B. (2001). Constraint-based tutors: A success story. *Proceedings of the 14th Industrial & Engineering Application of Artificial Intelligence & Expert Systems* (pp. 931–940), Berlin: Springer.

Ortony, A., Clore, G., & Collins, A. (1988). *The cognitive structure of emotions*. Cambridge University Press.

Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q., Dayal, U. (2001). Multi-Dimensional Sequential Pattern Mining, *Proceedings of the International Conference of Information and Knowledge Management* (pp. 81–88), New York: ACM.

Riccuci, S., Carbonaro, A., & Casadei, G. (2007). Knowledge acquisition in intelligent tutoring system: A data mining approach. *Proceedings of the 6th Mexican Conference on Artificial Intelligence* (pp. 1195–1205), Berlin: Springer.

Simon, H. A. (1978). Information-processing theory of human problem solving. In W. K. Estes (Ed.), *Handbook of learning and cognitive processes: Vol. 5. Human information*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Songram, P., Boonjing, V., Intakosum, S. (2006). Closed multidimensional sequential pattern mining. *Proceedings of the 3rd International Conference on Information Technology: New Generations* (pp. 512–517), Los Alamitos, CA: IEEE Computer Society Press.

Wang, J., Han, J., Li, C. (2007), Frequent closed sequence mining without candidate maintenance, *IEEE Transactions on Knowledge and Data Engineering, 19*(8), 1042–1056.