Mining Temporal Patterns to Improve Agents Behavior: Two Case Studies

Philippe Fournier-Viger[1], Roger Nkambou[1], Usef Faghihi[1], and Engelbert Mephu Nguifo[2]

[1] Université du Québec à Montréal, Montréal, Canada
{fournier_viger.philippe, nkambou.roger, faghihi.usef}@courrier.uqam.ca
[2] Université Blaise Pascal, Clermont-Ferrand, France
mephu@isima.fr

Abstract. We propose two mechanisms for agent learning based on the idea of mining temporal patterns from agent behavior. The first one consists of extracting temporal patterns from the perceived behavior of other agents accomplishing a task, to learn the task. The second learning mechanism consists in extracting temporal patterns from an agent's own behavior. In this case, the agent then reuses patterns that brought self-satisfaction. In both cases, no assumption is made on how the observed agents' behavior is internally generated. A case study with a real application is presented to illustrate each learning mechanism.

Key words: Data-mining driven agents, agent learning, temporal patterns

1 Introduction

Logging information about agents' behavior and analyzing it could provide useful knowledge for improving the behavior of agents. However, the amount of data to be recorded and analyzed can be huge to learn interesting information. As a solution, we propose to rely on data mining algorithms to analyze agent behaviors, and discover temporal regularities. More precisely, in line with researches indicating that data mining has the potential to improve the learning and reasoning capabilities of agents (data-mining driven agents) [1, 2], we propose two learning mechanisms that are integrated in agents deployed in real applications. The two learning mechanisms are based on the three same operation phases: (1) recording behaviors, (2) extracting temporal patterns from this data and (3) using this knowledge to improve agents' behavior. The first learning mechanism is for agents that learn a task by observing other agents performing it. In that case an agent records the behavior of other agents to discover patterns among this data that will then constitute its knowledge base. The second learning mechanism is for agents that learn from their own behavior. In this case, an agent records its behavior to then reuse patterns that led to self-satisfaction.

This chapter first introduces the problem of mining temporal patterns and an algorithm for mining temporal patterns, and discusses related works in agent

learning. Next, the second and third sections present the two learning mechanisms based on this algorithm and describe how they are integrated in virtual agents. Finally, the last section present conclusions and preview our future work.

2 Mining Temporal Patterns from Sequences of Events

According to [10], there are four main kinds of patterns that can be mined from time-series data. These are trends, similar sequences, sequential patterns and periodical patterns. In this work we chose to mine sequential pattern [3], as we are interested in finding relationships between occurrences of events in agents' behavior. To mine sequential patterns, several algorithms have been proposed [10]. But to our knowledge, few works have been published on using sequential pattern mining for agent learning. For example, [14] proposed to implement sequential pattern mining in a robot playing soccer. In this case, sequential patterns are used to derive prediction rules about what actions or situations might occur if some preconditions are satisfied. This is different from the two forms of learning that we consider in this chapter.

While classical sequential pattern mining algorithms have for only goal to discover sequential patterns that occurs frequently in several transactions of a database [3], other algorithms have proposed numerous extensions to the problem of mining sequential patterns [10]. For this work, we chose a sequential pattern mining algorithm that we have developed [8], as it combines several features from other algorithms such as accepting time constraints [11], processing databases with dimensional information [17], eliminating redundancy [20, 18], and also because it offers some original features such as accepting symbols with numeric values [8]. We describe next some basic features of the algorithm. Other features will be presented through the chapter with detailed explanations about why they are important for this work. For a technical description of the algorithm the reader can refer to [8]. Moreover, the reader can download a Java implementation of the algorithm by accessing http://www.philippe-fournier-viger.com/spmf/.

The algorithm takes as input a database D of sequences of events. An event $X = (i_1, i_2, ...i_n)$ contains a set of items $i_1, i_2, ...i_n$ that are considered simultaneous, and where each item can be annotated with an integer value. Formally, a sequence is denoted $s =< (t_1, X_1), (t_2, X_2), ..., (t_n, X_n) >$, where each event X_k is associated to a timestamp t_k indicating the time of the event. The size, n, of a sequence is the number of events in the sequence, i.e. $|s|$. For example, the size of sequence $S1$ of Fig. 1 (left) contains two events. The sequence $S1$ 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. A sequence $sa = < (ta_1, A_1), (ta_2, A_2, ..., (ta_n, A_n) >$ is said to be contained in another sequence $sb = < (tb_1, B_1), (tb_2, B_2), ..., (tb_n, B_m) >$, if there exists integers $1 = k1 < k2 < ... < kn \leq m$ such that $A_1 \subseteq B_{k1}, A_2 \subseteq B_{k2}, ..., A_n \subseteq B_{kn}$, and that $tb_{kj} - tb_{k1}$ is equal to $ta_j - ta_1$ for each $j \in 1...m$. The relative support of a sequence sa in a sequence database D is defined as the percentage of sequences

$s \subseteq D$ that contains sa, and is denoted by $sup_D(sa)$. The problem of mining frequent sequences is to find all the sequences sa such that $sup_D(sa) \geq minsup$ 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, Fig. 1 illustrates a database of 6 sequences (left) and the corresponding patterns found for a minsup of 33% (right). Consider pattern $M5$. This pattern appears in sequence $S4$ and $S5$. The first event of $M5$ $(0, f)$ is contained in the first event of $S4$ $(0, f)$ and the second event of $M5$ $(2, e)$ is contained in the event of $S4$ $(2, a\{6\}e)$ that is occuring two time units after the first event of $S4$. Since the pattern $M5$ is contained in $S4$ and $S5$, it has a support of 33% (2 out of 6 sequences). Now consider patterns $M1$ and $M2$. Because the item a appears in sequence $S1$, $S2$, $S3$ and $S4$ with values 2, 2, 5 and 6 respectively the algorithm separated these values in 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 [8] for more 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	Supp.
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	…	…

Fig. 1. A database of 6 sequences (left) and mined sequences (right)

3 Agents that Learn from other Agents

The first form of learning that we consider for an agent is learning by observing other agents. Researchers have made various proposals for integrating learning-by-observation in agents (see [19] for a brief review). However, many of them rely on strong assumptions. For example, van Lent and Laird [19] propose a framework to learn production rules from recorded human behavior. In this approach, a human has to teach an agent by performing a task. However, this approach is tightly linked to a very specific conception of intelligence, as humans performing a demonstration are required to specify their actions as complex operators organized in a hierarchy and having goal conditions, and they must explicitly state their goals during the demonstrations. Contrarily to this view, we here address the problem of learning-by-observation for an agent by considering that it can only perceive actions of other agent, without additional information.

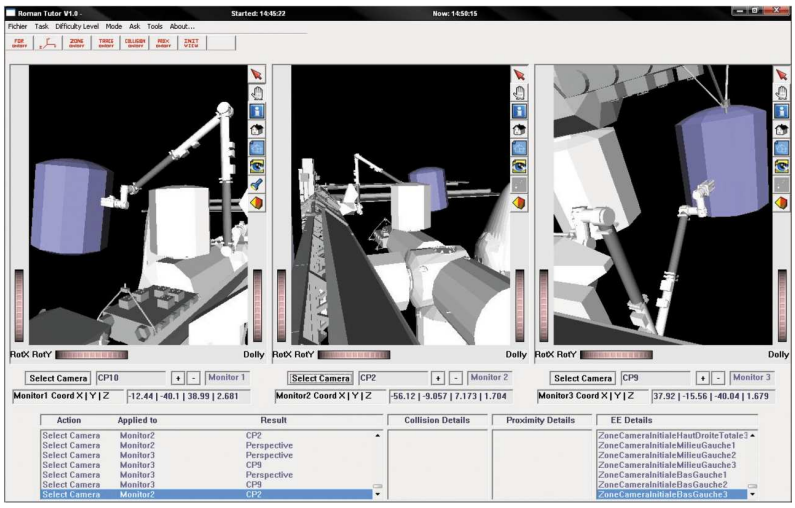

Fig. 2. The RomanTutor user interface

We illustrate this form of learning in the context of RomanTutor [13] (fig. 2), a virtual learning environment for learning how to operate the Canadarm2 robotic arm on the international space station. The main learning activity in RomanTutor is to move the arm from one configuration to another. This is a complex task, as 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 RomanTutor 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 [7]. For this reason, instead of providing domain knowledge to the agent, we implemented a learning mechanism which allows the agent to learn by observing the behavior of other agents performing a task (in this case, humans). The agent 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 this virtual agent, and an experiment.

3.1 The observing phase

In the *observing phase*, the virtual agent observes and records the behavior of users that attempt an arm manipulation exercise (moving the arm from an initial configuration to a goal configuration). For each attempt, the virtual agent 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 RomanTutor, 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. Defining actions with integer values is beneficial because, as mentioned, our algorithm can group automatically actions with similar values an treat these groups as different actions. An example of a partial sequence of actions recorded for an user is $< (0, 6\{2\}), (1, 63), (2, 53\{4\}), (3, 111\{2\} >$ which represents decreasing the rotation value of joint SP (action 6) by 2 units, selecting camera CP3 (action 63), increasing the pan of camera CP2 (action 53) by 4 units and then its zoom (action 111) by 2 units.

A problem that we faced when designing the virtual agent's observing phase is that it would also be useful to annotate sequences with contextual information such as success information and the expertise level of a user, to then mine patterns containing this information. Our solution to this issue is to take advantage of an extra feature of our algorithm (based on [17]), which is to add dimensional information to sequences. A database having a set of dimensions $D = D1, D2, ...Dn$ is called an MD-Database. Each sequence of a MD-Database (an MD-Sequence) possesses a symbolic value for each dimension. This set of values is called an MD-Pattern and is denoted $d1, d2...dn$. In the context of our virtual agent, we defined two dimensions, "success" and "expertise level", which are added manually to each sequence recorded. The left side of Fig. 3 shows an example of an MD-Database having these two dimensions. As an example, the MD-Sequence B1 has the MD-Pattern "true", "novice" for the dimensions "success" and "expertise level". The symbol "*", which means any values, can also be used in an MD-Pattern. This symbol subsumes all other dimension values. An MD-Pattern $Px = dx1, dx2...dxn$ is said to be contained in another MD-Pattern $Py = dy1, dy2...dym$ if $dx1 \subseteq dy1, dx2 \subseteq dy2, ..., dxn \subseteq dyn$. The problem of mining frequent sequences with dimensional information is to find all MD-Sequence appearing in an MD-Database with a support higher or equal to *minsup*. As an example, the right part of Fig. 3 shows some patterns that can be extracted from the MD-Database of Fig. 3, with a *minsup* of 2 sequences. In our virtual agent, dimensional information is very important, as it allowed to successfully identify patterns common to all expertise levels that lead to failure ("false, *"), for example.

3.2 The learning phase

In the *learning phase*, the virtual agent applies the algorithm to extract frequent sequences that build its domain knowledge. For mining patterns, we setup the algorithm to mine only sequences of size two or greater, as sequence shorter 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. The benefits of accepting a gap of two is that it eliminates some "noisy" (non-frequent) learners'

ID	Dimensions	Sequences
B1	true, novice	<(0,a),(1,bc)>
B2	true, expert	<(0,d)>
B3	false, novice	<(0,a),(1,bc)>
B4	false, interm.	<(0,a),(1,c), (2,d)>
B5	true, novice	<(0,d), (1,c)>
B6	true, expert	<(0,c), (1,d)

->

Dimensions	Sequences
*, novice,	<(0,a)>
*, *	<(0,a)>
*, novice	<(0,a), (1,b)>
true, *	<(0,d)>
true, novice	<(0,c)>
true, expert	<(0,d)>

Fig. 3. An example of sequential pattern mining with contextual information

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.

A second important consideration in the learning phase is that when applying sequential pattern mining, there can be many redundant frequent sequences found. For example, in Fig. 1 (right), pattern M2 is redundant as it is included in pattern M3 and it has exactly the same support. To eliminate redundancy, we rely on an extra feature of our algorithm which allows mining only closed sequences. "Closed sequences" are sequences that are not contained in another sequence having the same support. Mining frequent closed sequences has the advantage of greatly reducing the size of patterns found, without information loss (the set of closed frequent sequences allows reconstituting the set of all frequent sequences and their support) [20]. To mine only frequent closed sequences, our sequential pattern mining algorithm was extended based on [20] and [18] to mine closed MD-Sequences (see [8]).

3.3 The application phase

In the third phase, the application phase, the virtual agent provides assistance to the learner by using the knowledge learned in the learning phase. Recognizing a learner's plan is the basic operation that is used to provide assistance. This is achieved by the plan recognizing algorithm (algorithm 1), described next.

Algorithm 1 (RecognizePlan Algorithm)
RecognizePlan(Student_trace, Patterns)
  Result := ∅.
  FOR each pattern P of Patterns.
    IF Student_trace is included in P.
      Result := Result ∪ {P}.
  IF Result = ∅ AND size(Student_trace) ≥ 2.
    Remove last action of Student_trace.
    Result := RecognizePlan(Student_trace, Patterns).
  RETURN Result.

The plan recognizing algorithm RecognizePlan is executed after each student action. It takes the sequence of actions performed by the student (Student_trace) for the current problem and a set of frequent actions sequences (Patterns) as

inputs. When the plan recognizing algorithm is called for the first time, the variable *Patterns* is initialized with the whole set of patterns found during the learning phase of the virtual agent. The algorithm first iterates on the set of patterns *Patterns* to note all the patterns that include Student_trace. If no pattern is found, the algorithm removes the last action performed by the learner from Student_trace and searches again for matching patterns. This is repeated until the set of matching patterns is not empty or the size of Student_trace is smaller than 2. In our tests, removing user actions has shown to improve the effectiveness of the plan recognizing algorithm significantly, as it makes the algorithm more flexible. The next time *RecognizePlan* 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.

We describe next the main tutoring services that the tutoring agent provides based on the plan recognizing algorithm.

First, the virtual agent can assess the expertise level of the learner (novice, intermediate or expert) by looking at the patterns applied. If for example a learner applies 80% of the time "intermediate" patterns, then the virtual agent can assert with confidence that the learner expertise level is "intermediate". Second, the agent can guide the learner. This tutoring service consists in determining the possible actions from the current problem state and proposing one or more actions to the learner. This functionality is triggered when the student selects "What should I do next?" in the RomanTutor interface menu. The virtual agent then identifies the set of possible next actions according to the matching patterns found by *RecognizePlan* and 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 of the learner. If no actions can be identified, the virtual agent can use a path planner [13] to generate approximate solutions. In this current version, the virtual agent only interacts with the learner upon request. Nonetheless, it would be possible to program the virtual agent 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 is part of our current work.

3.4 An experiment

We conducted an experiment in RomanTutor with two exercises to qualitatively evaluate the virtual agent's capability to provide assistance. The two exercises consists each of moving a load with the robotic arm to one of the two cubes (figure 4.a). We asked twelve users to record plans for these exercises. The average length was 20 actions. From this data, the virtual agent extracted 558 sequential patterns with the algorithm. In a subsequent work session, we asked the users to evaluate the tutoring services provided by the virtual agent. Users agreed that the assistance provided was helpful. We also observed that the virtual agent correctly inferred the estimated expertise level of learners.

As an example of interaction with a learner, Fig. 4 illustrates a hint message given to a learner upon request during scenario 1. The guiding tutoring service

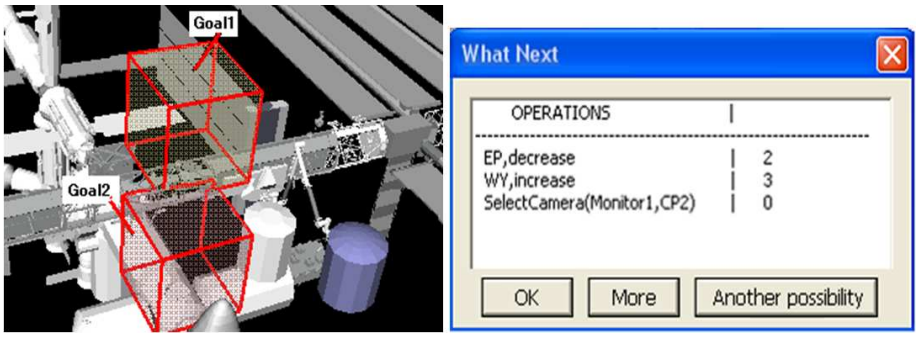

Fig. 4. (a) The two scenarios (b) A hint generated by the virtual agent

selected the pattern that has the highest support value, matches the last student actions, is marked "success" and corresponds with the estimated expertise level of the learner. The given hint is to decrease the rotation value of the joint "EP" (20°), increase the rotation value of joint "WY" (30°), and finally to select camera "CP2" on "Monitor1". By default, three steps are showed to the learners in the hint window depicted in Fig. 4.b. 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. The description of actions depicted in Fig. 4.a are an example of resources that can be used to annotate patterns.

Although the pattern mining algorithm was applied once in this experiment, it would have been possible to make the agent apply it periodically, so that the agent would continuously update its knowledge base while interacting with learners. Moreover, we have encoded only two dimensions: expertise level and success. However additional contextual information could easily be added. In future work for example, we plan to encode skills involved as dimensional information (each skill could be encoded as a dimension). This will allow computing a subset of skills that characterize a pattern. This will allow diagnosing missing and misunderstanding skill for users who demonstrated a pattern.

4 Agents that Learn from Their Own Behavior

The second form of learning that we consider for an agent is to learn from its own behavior. Unlike the learning mechanism implemented in the previous agent, this learning mechanism is not designed for learning new behaviors or procedural knowledge, but for making an agent reuse previously self-satisfying behaviors. We integrated this mechanism in a virtual agent named CTS [5] that we have also tested in RomanTutor to provide assistance to learners. The following subsections describe CTS, the three operation phases of the learning mechanism that was integrated in CTS, and two experiments carried in RomanTutor to validate (1) the behavior of the new CTS and (2) the behavior of the data mining algorithm with large data sets.

4.1 The CTS Cognitive Agent

CTS (Conscious Tutoring System) is a generic cognitive agent, whose architecture (fig. 5) is inspired by neurobiology and neuropsychology theories of human brain function. It relies on the functional "consciousness" [9] mechanism for much of its operations. It also bears some functional similarities with 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 [12], a codelet is a very simple agent, "a small piece of code that is specialized for some comparatively simple task". As in Baars theory's [4], these simple processors do much of the processing in the CTS architecture.

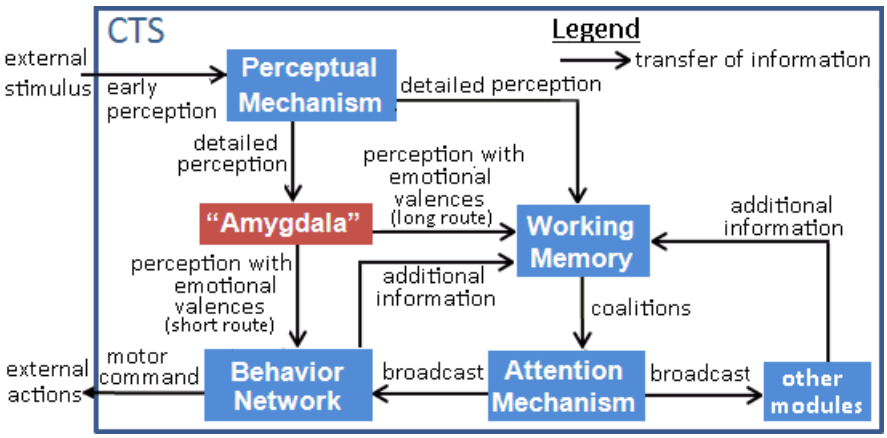

Fig. 5. A simplified view of the CTS architecture (see [6] for more details)

CTS possess two routes for processing external stimuli (cf. fig. 5). 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 [6]. In both cases, the stimuli processing begins with percept codelets [12] 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 it 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 ar-

chitecture. 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 [4]) 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) [15]. 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 form of learning that we consider in this article. This learning mechanism is implemented in CTS by the three operation phases, described next.

4.2 The observation phase

In the first phase, the *observation phase*, CTS records a sequence of events (as defined in section 2) 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 having numeric values indicating the four emotional valences (high threat, medium fear, low threat, compassion) associated with the broadcast coalition[3]. For example, one partial sequence recorded during our experimentation 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).

4.3 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 setup 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.

4.4 The application phase

The third operation phase consists in improving CTS behavior by making CTS reuse relevant patterns that carry positive emotions. This is done by intervening

[3] CTS actually incorporates four emotions inspired by the OCC model of emotions [16]. See [6] for in-depth details about the emotional mechanism of CTS.

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 to then select the next coalition to be broadcast that is the 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 (algorithm 2), 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 $c \in$ *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.

Algorithm 2 (SelectCoalition Algorithm)

```
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.
```

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 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 make 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 have been used for other applications.

4.5 Testing the new CTS in RomanTutor

To test CTS's new learning mechanism, users were invited to perform arm manipulations using RomanTutor with integrated CTS. These experiments aimed at validating CTS ability to adapt its behavior to learners. During these experiments, we qualitatively observed that CTS adapted its behavior successfully to learners. Two experiments are here described. The first describes in details one 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 the data 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 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 of 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 to 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 chose that it is better to give a hint (Scenario 2) than to give the answer (Scenario 1) to User 3. Concretely, 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 other 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 in the context of a tutoring agent, 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 the data 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 has 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. Fig. 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 s) and increased linearly with the number of recorded sequences. In our context, this performance is very satisfying. The second graph shows the average size of patterns found for each execution. It ranges from 9 to 16 broadcasts. The third graph depicts the number of patterns found. It remained 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 (c.f. section 3.2). If we had not mined only closed patterns, all the subsequences of each pattern would have been included in the results. Finally, the average time for executing the *SelectCoalition* algorithm at each execution. This time was always less than 5 milliseconds. Thus, the costliest operation of the learning mechanism is the learning phase.

5 Conclusion

In this chapter, we presented the idea of building agents that learn by extracting temporal patterns from their own behavior or the behavior of other agents. To demonstrate this idea, we proposed two learning mechanisms. While the first learning mechanism is aimed at learning new procedural knowledge by observing other agents performing a task, the second one is designed for making an

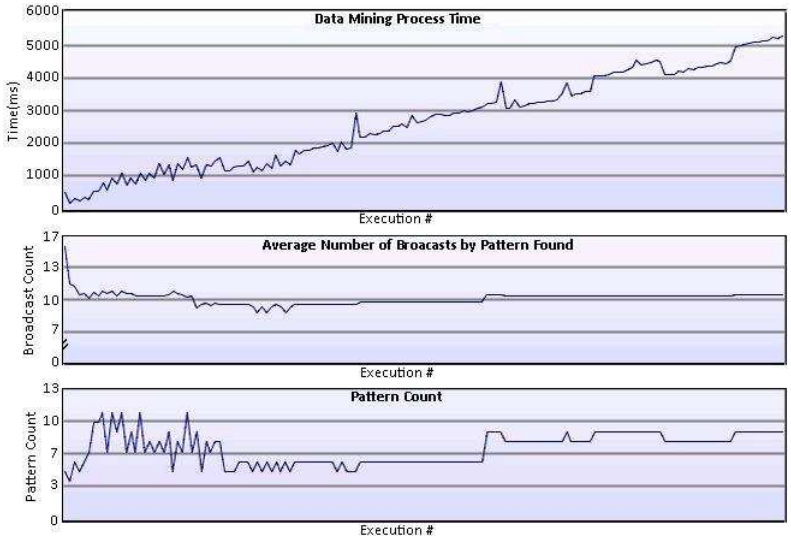

Fig. 6. Results from second experiment

agent reuse its self-satisfying behaviors. We presented each learning mechanism through the case study of a virtual agent. The two virtual agents were tested in real applications and experiments have shown positive results, as regards the capability of the agents to adapt their behavior and the performance of the learning mechanisms. The two learning mechanisms should be reusable in other agents and contexts, as they make little assumption on the architectures of the observed agents and their decision making processes, and the format for encoding behaviors is fairly generic.

In future work, we will perform further experiments to measure empirically how the virtual agents 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 compare the two learning mechanisms with others agent learning mechanisms, and to integrate the two virtual agents in other tutoring systems.

Acknowledgements

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

References

1. Cao, L., Luo, C., Zhang, C.: Agent-Mining Interaction: An Emerging Area. Proc. AIS-ADM 2007, LNAI 4476, pp. 60-73, 2007. Springer-Verlag, Berlin.
2. Symeonidis, A. L., Athanasiadis, I. N., Mitkas, P. A.: A retraining methodology for enhancing agent intelligence. Knowledge-Based Syst. **20**(4), 388-396 (2008)
3. Agrawal, R., Srikant, R.: Mining sequential patterns. Proc. Int. Conf. Data Eng., pp. 3-14 (1995)
4. Baars, B.J.: In the theater of consciousness. Oxford Univ. Press, Ofxord. (1997)
5. Dubois, D., Poirier, P., Nkambou, R.: What does Consciousness bring to CTS?. Proc. 2007 AAAI Fall Symp., pp. 55-60, AAAI Press (2007)
6. Faghihi, U., Poirier, P., Dubois, D., Gaha, M., Nkambou, R.: How emotional mechanism learn and helps other types of learning in a cognitive agent. Proc. WI-IAT 2008 (2008)
7. Fournier-Viger, P., Nkambou, R., Mayers, A.: Evaluating spatial representations and skills in a simulator-based tutoring system. IEEE Trans. Learn. Technol. **1**, pp. 63-74 (2008)
8. Fournier-Viger, P., Nkambou, R., Mephu Nguifo, E.: A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems. Proc. 7th Mex. Int. Conf. Artif. Intell., LNAI 5317, pp. 765-778. Springer (2008)
9. Franklin, S., Patterson, F.G.J.: the LIDA architecture: adding new modes of learning to an intelligent, autonomous, software agent. Proc. IDPT-2006 (2006)
10. Han, J., Kamber, M.: Data mining: concepts and techniques, Morgan Kaufmann Publ., San Franc. (2000)
11. Hirate, Y., Yamana, H.: Generalized sequential pattern mining with item intervals. J. **1**(3), 51–60 (2006)
12. Hofstadter, D.R., Mitchell, M.: The Copycat project: a model of mental fluidity and analogy-making. In: Barnden, J., Holyoak, K., (eds.) Advances in connectionnist and neural computation theory, pp. 31-113. Lawrence Erlbaum Associates (1992)
13. Kabanza, F., Nkambou, R., Belghith, K.: Path-planning for autonomous training on robot manipulators in space. Proc. 19th Int. Joint. Conf. Artif. Intell., pp. 1729-1731 (2005)
14. Lattner, A.D., Miene, A., Visser, U., Herzog, O.: Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. Proc. Robocup 2005 Conf., pp. 118-129 (2005)
15. Maes, P.: How to do the right thing. Connect. Sci. **1**, 291–323 (1989)
16. Ortony, A., Clore, G.: cognitive structure of emotion. Cambridge Univ. Press, Camb. (1988)
17. Pinto, H., Han, J., Pei, J., Wang, K., Chen, Q.: Multi-dimensional sequential pattern mining. Proc. 10th Int. Conf. Inf. Knowl. Manag., pp. 81-88 (2001)
18. Songram, P., Boonjing, V., Intakosum, S.: Closed multidimensional sequential pattern mining. Proc. 3rd Int. Conf. Inf. Techn.: New Gener., pp. 512-517 (2006)
19. van Lent, M., Laird, J.E.: Learning procedural knowledge through observation. Proc. K-CAP 2001, pp. 179-186 (2001)
20. Wang, J., Han, J., Li, C.: Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng. **19**(8), 1042–1056 (2007)