Exploiting Partial Problem Spaces Learned from Users' Interactions to Provide Key Tutoring Services in Procedural and Ill-Defined Domains

Philippe Fournier-Viger[1], Roger Nkambou[1] and Engelbert Mephu Nguifo[2]
[1]*Université du Québec à Montréal (Canada),* [2]*Université Blaise Pascal (France),*
{fournier-viger.philippe@courriel.uqam.ca, nkambou.roger@uqam.ca }

Abstract. In previous works, we showed how sequential pattern mining can be used to extract a partial problem space from logged user interactions for a procedural and ill-defined domain where classic domain knowledge acquisition approaches don't work well. In this paper, we describe in details how such a problem space can support important tutoring services such as (1) recognizing the plan of a learner, (2) providing hints and (3) estimating the profile of a learner including its expertise level and missing or misunderstandood skills.

Keywords. intelligent tutoring systems, domain knowledge mining, tutoring services

1. Introduction

Domain experts should provide relevant domain knowledge to an Intelligent Tutoring System (ITS) so that it can assist a learner during problem-solving activities. There are three main approaches for providing such knowledge. The first one is cognitive task analysis, which aims at producing 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 [1] and it is not always possible to define a satisfying complete or partial task model, in particular when a problem is ill-structured. According to Simon [2], an ill-structured problem is one that is complex, with indefinite starting points, multiple and arguable solutions, or unclear strategies for finding solutions. Domains that include such problems and in which, tutoring targets the development of problem-solving skills are said to be ill-defined (within the meaning of [3]). Constraint-based modeling (CBM) was proposed as an alternative [4]. It 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. The third approach consists in integrating an expert system in an ITS. However, developing an expert system can be difficult and costly, especially for ill-defined domains. Contrarily to these approaches where domain experts have to provide domain knowledge, a promising approach is to use knowledge discovery techniques for automatically learning a partial problem space from logged user interactions in an ITS. We did an initial research in this direction [5] by proposing a framework to learn a knowledge base from user interactions in procedural and ill-defined domains. The framework was applied in a tutoring system to extract a partial

problem space that is used to guide users, and thus showed to be a viable alternative to the specification of a problem-space by hand for the same domain [5, 6]. The framework differs from other works that attempt to construct a task model from logged student interactions such as [7], [8] and [9], since these latter are devoid of learning or have been designed for being applied in well-defined domains. Recently [10], we improved our framework by developing a sequential pattern mining algorithm to extract a problem space that is richer. But the tutoring services were limited in our first implementation. In this paper, we address this concern by presenting more tutoring services that can be supported. The paper first introduces RomanTutor [11], the tutoring system in which this work is applied. Then, it presents our framework for learning domain knowledge and supported tutoring services. Finally, it presents an evaluation of its application in RomanTutor and a conclusion.

2. The RomanTutor Tutoring System

RomanTutor [11] is a simulation-based tutoring system to teach astronauts how to operate Canadarm2, a 7 degrees of freedom robotic arm deployed on the International Space Station (ISS). The main learning activity in RomanTutor is to move the arm from a given configuration to a goal configuration. During the robot manipulation, operators do not have a direct view of the scene of operation on the ISS and must rely on cameras mounted on the manipulator and at strategic places in the environment where it operates. To move the arm, an operator must select at every moment the best cameras for viewing the scene of operation among several cameras mounted on the manipulator and on the space station.

To provide domain expertise to RomanTutor, we initially applied the expert system approach by integrating a special path-planner in RomanTutor [11]. The path-planner can generate a path avoiding obstacles between any two arm configurations. The path-planner makes it possible to track a learner solution step by step, and generate demonstrations. However, the generated paths are not always realistic or easy to follow, as they are not based on human experience, and they do not cover other aspects of the manipulation task such as selecting cameras and adjusting their parameters. Also, it cannot support important tutoring services such as estimating knowledge gaps of learners.

In a second project [6], we attempted to model the Canadarm2 manipulation task by applying the cognitive task analysis approach with a rule-based representation model. Although, we described high-level rules, it was not possible to go in finer details of the manipulation task. The reason is that for a given robot manipulation problem, there are too many possibilities for moving the robot to a goal configuration and thus, it is not possible to define a complete and explicit task model. In fact, there is no simple ‘legal move generator’ for finding all the possibilities at each step. Hence, RomanTutor operates in an ill-defined-domain.

The CBM approach [4] may represent a good alternative to the previous approaches. However, in the RomanTutor context, it would be very hard for domain experts to describe a set of constraints that accepts all good solutions and no buggy solutions. In fact, there would be too many constraints; the domain is too complex for this approach. Moreover, the CBM approach is useful for validating solutions. But it is not appropriate for suggesting next problem-solving steps to learners. As a solution, we propose a data mining approach for learning a partial problem space from recorded user interactions. The next sections present the three main phases of our approach.

3. Phase 1: Recording Users' Solutions

In the first phase, the tutoring system records the solutions of users that attempt an exercise. In RomanTutor, an exercise is to move the arm from an initial configuration to a goal configuration. For each attempt, a *sequence of events* (or *plan*) is created in a database. We define an event $X = (i_1, i_2, \ldots i_n)$ as a set of actions $i_1, i_2, \ldots i_n$ done by a learner that are considered simultaneous, and where each action can be annotated with an integer value. In RomanTutor, we defined 112 such actions that can be recorded including (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. Formally, we define a sequence of events (based on [12]) as $s = < (t_1, X_1), (t_2, X_2), \ldots, (t_n, X_n) >$ where each event X_k is associated to a timestamp t_k indicating the time of the event. In RomanTutor, timestamps of successive events are succesive integers (0, 1, 2…). An example of a partial action sequence recorded for an user in RomanTutor 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), 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 [13]. A sequence database having a set of dimensions $D = D1, D2, \ldots Dn$ is called a MD-Database. Each sequence of a MD-Database (a MD-Sequence) possesses a symbolic value for each dimension or the value "*", which means any values. A set of dimension values is called a MD-Pattern and is denoted $d1, d2 \ldots dn$. 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 RomanTutor, values for this dimension are produced by the tutoring system. The four other dimensions of Table 2 are example 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 that 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 RomanTutor, we used 10 skills and the "solution state" dimension to annotate sequences.

4. Phase 2: Extracting Partial Task Models from Users' solutions

In the second phase, the tutoring system applies the data mining framework to extract a partial problem space from users' plans. In this work we chose to mine sequential patterns [14], as we are interested in finding relationships between occurrences of events in users' solutions. To mine sequential patterns, we developed a custom algorithm [10] to handle the type of data to be recorded in a tutoring system such as RomanTutor. The reader can refer to [10] for a technical description of the algorithm and can download a Java implementation by accessing http://www.philippe-fournier-viger.com/spmf/.

The algorithm takes as input a MD-Database and some parameters and find all MD-sequences occuring frequently in the MD-Database. Here, sequences are

sequences with timestamps as in [12]. A sequence $sa = < (\mathrm{ta}_1, \mathrm{A}_1), (\mathrm{ta}_2, \mathrm{A}_2, \ldots, (\mathrm{ta}_\mathrm{n}, \mathrm{A}_\mathrm{n}) >$ is said to be contained in another sequence $sb = < (tb_1, B_1), (tb_2, B_2), \ldots, (tb_n, B_m) >$, if there exists integers $1 = k1 < k2 < \ldots < kn \leq m$ 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 $j \in 1..\, m$. Similarly for MD-Patterns, a MD-Pattern $Px = dx1, dx2 \ldots dxn$ is said to be contained in another MD-Pattern $Py = dy1, dy2 \ldots dym$ if $dx1 \subseteq dy1, dx2 \subseteq dy2, \ldots, dxn \subseteq dyn$ [13]. 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. The problem of mining frequent MD-sequences is to find all the MD-sequences such that their support is higher or equal to $minsup$ for a MD-database D, given a support threshold $minsup$. 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 plan(s) that succeeded.

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)

Table 2. Some frequent patterns extracted from the dataset of Table 1 with a minsup of 33 %

Id	Dimensions					Sequence	Supp.
	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 %

In addition, as [12] we have incorporated in our algorithm the possibility of specifying time constraints on mined sequences [10]. In RomanTutor, we setup the algorithm to mine only sequence 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 succesive events of two time units. 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 patterns less useful for tracking a learner's actions.

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 as it is included in the pattern P3 and it has the same support. To eliminate this type of redundancy, we have adapted our algorithm based on [15] and [16] to mine closed MD-Sequences. Closed MD-sequences are MD-sequences that are not

contained in another sequence having the same support. Mining frequent MD-closed sequences has the advantage of greatly reducing the size of patterns found without information loss [15]. 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 ressources, such as textual hints.

5. Phase 3 : Offering Key Tutoring Services

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 RomanTutor, 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 the plan recognition algorithm significantly. 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.

After performing preliminary tests with *RecognizePlan*, we noticed that in general, after more than 6 actions performed by a learner, it becomes hard to tell which pattern the learner is doing. For that reason, we made improvement to how the RomanTutor 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 RomanTutor, where an exercise consists of moving a robotic arm to attain a specific arm configuration, the 3D space was 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 P1 to a problem state PF. For each attempt at solving the exercise, RomanTutor 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 $P_3, \ldots P_{n-1}$ to P_n). After many users performed the same exercise, RomanTutor extracts sequential patterns from (1) sequences of problems states visited, and from (2) 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 more 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 RomanTutor can assert with confidence that the learner expertise level is "intermediate". In the same way, RomanTutor can

diagnose mastered and missing/misunderstanding 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 in determining the possible actions from the current problem state and proposing one or more actions to the learner. In RomanTutor, this functionality is triggered when the student selects "What should I do next?" in the interface menu. RomanTutor 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 patterns contains skills that are not considered mastered by the learner, RomanTutor can use tutoring resources to explain them. If no actions can be identified, RomanTutor can use the aforementioned path planner to generate approximate solutions. In this current version, RomanTutor only interacts with the learner upon request. But it would be possible to program RomanTutor 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 RomanTutor is to let learners explore patterns to learn about possible ways of solving problem. 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 which could prompt them on their goals and helps them go through the patterns to achieve these goals.

6. Experimentation

We conducted a preliminary 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 Canadarm2 robotic arm to one of the two cubes (figure 1.A). We asked 12 users to record plans for these exercises. The average length of plans was 20 actions. From this data, RomanTutor extracted a partial problem space. In a subsequent work session, we asked the users to evaluate the tutoring services provided by the virtual agent. All users agreed that the assistance provided was helpful. We also observed that the virtual agent 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 1.B illustrates a hint message given to a learner upon request during scenario 1. The guiding tutoring service selected 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 finally increase the rotation value of joint “WE”. The values on the right column indicate the values associated to the action. In this context, the values “1” and “3” means to rotate the joints 10 º and 30 º, respectively (1 unit equals 10º). By default, three steps are showed to the learners in the hint window depicted in figure 2.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.

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

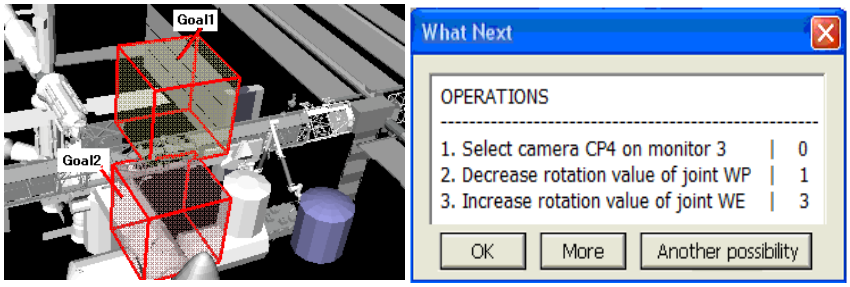

Fig. 1. (A) The two scenarios (B) A hint generated by the virtual agent

7. Related work

Lastly, we mention a work that is close to ours [17]. This work was applied for the well-defined domain of logic proofs. The solution proposed by Barnes et al. consists of building a Markov Decision Process containing learner solutions for a problem. This is mainly a graph where each state represents a correct or erroneous state and each link an action to go from a state to another. Then, given a state, an optimal path can be calculated to reach a goal state according to constraints such as frequency, lowest probability of errors or shortest number of actions. An optimal path found in this way is then used to suggest to a learner the next actions to perform. As for our framework, this approach has to be applied for each problem. But the proposal of Barnes et al. differs from ours in several ways. The first important difference is that we extract partial problem spaces from user solutions. Thus, our framework ignores parts of learner solutions that are not frequent. This strategy allows coping with domains where the number of possibilities is very large and user solutions do not share many actions. In fact, our framework build approximations of learners' solutions, where the frequency threshold *minsup* control what will be excluded from these approximations.

A second important difference is that the approximations created by our framework are generalizations as they consist of subsequences appearing in several learner solutions. This property of problem spaces produced by our framework is very useful as it allows finding patterns that are common to several profiles of learners or contexts (for example, patterns common to expert users that succeed to solve a problem, or common to users possessing/lacking one or more skills that succeeded/failed). Conversely, the proposal of Barnes et al. doesn't take into account the profile of learners who recorded solutions and other contextual information to learners' plans. Therefore, their proposal cannot support more elaborated tutoring services such as estimating the profile of a learners by looking at the actions that a learner applies (e.g. expertise level), and hints cannot be suggested based on an estimated profile of a learner. We believe this to be a major limitation of their proposal, as in many cases an ITS should not consider the "optimal" solution as being the best solution for a learner. An ITS should instead select sucessful solutions that are adapted to a learner profile, to make the learner progress along the continuum from novice to expert.

8. Conclusion

In this paper, we have presented a fourth approach for domain knowledge acquisition in ITS that has shown to be a viable alternative to classic domain knowledge acquisition approaches, particularly for procedural and ill-defined domains where classical approaches failed. Since the proposed data mining framework and its inputs/outputs are domain independent, it can be potentially applied to any procedural ill-defined domains where the problems can be stated in the same way. We described how the approach can support key tutoring services. Results of our experiment with RomanTutor showed an improvement over previous versions of RomanTutor in terms of tracking learners' behavior and providing hints. In future works, we will perform further experiments to measure empirically how the tutoring services influence the learning of students. We also plan to experiment with different tutoring strategies.

References

[1] V. Aleven, B.M. McLaren, J. Sewall, K. Koedinger, The Cognitive Tutor Authoring Tools (CTAT): Preliminary evaluation of efficiency gains. *Proc. Intelligent Tutoring Systems 2006*, 61-70, 2006.

[2] H. A. Simon, Information-processing theory of human problem solving". In W.K. Estes (Ed.), *Handbook of learning and cognitive processes: Vol. 5. Human information*, 1978.

[3] C. Lynch, K. Ashley, V. Aleven, N. Pinkwart, Defining Ill-Defined Domains; A literature survey. *Proc. of the Intelligent Tutoring Systems for Ill-Defined Domains Workshop (ITS06)*, 1-10, 2006.

[4] A. Mitrovic, M. Mayo, P. Suraweera, B. Martin, Contraint-based tutors: a success story. *Proc. of the Industrial & Engineering Application of Artificial Intelligence & Expert Systems*, 931-940, 2001.

[5] R. Nkambou, E. Mephu Nguifo, P. Fournier-Viger, Using Knowledge Discovery Techniques to Support Tutoring in an Ill-Defined Domain. *Proc. of ITS 2008*, 395-405, 2008.

[6] P. Fournier-Viger, R. Nkambou, A. Mayers, Evaluating Spatial Representations and Skills in a Simulator-Based Tutoring System. *IEEE Transactions on Learning Technologies*, **1**:1(2008): 63-74.

[7] B. M. McLaren et al, Bootstrapping Novice Data: Semi-Automated Tutor Authoring Using Student Log Files, *Proceedings of the Workshop on Analyzing Student-Tutor Logs (ITS'2004)*, 2004.

[8] S.B. Blessing, A Programming by Demonstration Authoring Tool for Model-Tracing Tutors. In *Authoring Tools for Advanced Technology Learning Environments: Toward Cost-Effective Adaptive, Interactive and Intelligent Educational Software*, 93-119. Kluwer Academic Publishers, 2003.

[9] M. Jarivs, G. Nuzzo-Jones, N.T. Heffernan, Applying Machine Learning Techniques to Rule Generation in Intelligent Tutoring Systems. *Proc. Intelligent Tutoring Systems 2006*, 541-553. Springer, 2006.

[10] P. Fournier-Viger, R. Nkambou, E. Mephu Nguifo, A Knowledge Discovery Framework for Learning Task Models from User Interactions in Intelligent Tutoring Systems. *Proc. 6th Mexican International Conference on Artificial Intelligence*, 765-778. LNAI 5317, Springer, 2008.

[11] F. Kabanza, R. Nkambou, K. Belghith, Path-Planning for Autonomous Training on Robot Manipulators in Space. *Proc. IJCAI 2005*, 1729-173, 2005.

[12] Y. Hirate, H. Yamana, Generalized Sequential Pattern Mining with Item Intervals, *Journal of Computers*, **1**:3(2006), 51-60.

[13] H. Pinto et al, Multi-Dimensional Sequential Pattern Mining, *Proc. Int. Conf. Information and Knowledge Management (CIKM2001)*, 81-88, 2001.

[14] R. Agrawal, R. Srikant. Mining Sequential Patterns. *Proc. Int. Conf. on Data Engineering*, 3-14, 1995.

[15] J. Wang, J. Han, C. Li, Frequent Closed Sequence Mining without Candidate Maintenance, *IEEE Trans. on Knowledge and Data Engineering*, **19**:8 (2007),1042-1056,

[16] P. Songram, V. Boonjing, S. Intakosum, Closed Multidimensional Sequential Pattern Mining, *Proc. 3rd Int. Conf. Information Technology: New Generations*, 512-517, 2006.

[17] T. Barnes, J. Stamper, Toward Automatic Hint Generation for Logic Proof Tutoring Using Historical Student Data, *Proc. 9th Int. Conf. Intelligent Tutoring Systems (ITS 2008)*, 373-382, 2008.

Acknowledgment. Our thanks go to the FQRNT and NSERC for their logistic and financial support. The authors would like to thank Severin Vigot and Mikael Watrelot for integrating the framework in RomanTutor, and all members of the GDAC/PLANIART teams who participated in the development of RomanTutor.