

A Data Mining Framework for the Acquisition of Procedural Knowledge in Intelligent Tutoring Systems

Philippe Fournier-Viger^a, Roger Nkambou^a, Engelbert Mephu Nguifo^b

^a*Department of Computer Science, University of Quebec at Montreal, Canada*

^b*Department of Computer Science, Université Lille-Nord de France, France*

Abstract: Domain knowledge acquisition is a very important and time-consuming process in intelligent tutoring system design. Furthermore, for many ill-defined domains, domain knowledge is hard to define explicitly. To address this problem, we proposed a data mining framework for extracting partial task models from recorded user interactions. This article describes improvements made to the framework and its application to extract a knowledge base that supports useful tutoring services in a tutoring system. A preliminary experiment with the tutoring system is presented.

Keywords: intelligent tutoring systems, domain knowledge acquisition, data mining, procedural knowledge

Introduction

Domain knowledge acquisition is a very important process in intelligent tutoring system design. One common way of acquiring this knowledge is the method of cognitive task analysis, which consists of observing expert and novice users for capturing different ways of solving problems. However, this process is very time-consuming and it is not always possible to define a satisfying task model, in particular when a domain is ill-defined. As an alternative to specifying a complete task model, it was proposed to define sets of constraints on what is a correct behavior [1]. Though this approach was shown to be effective for some ill-defined domains, a domain expert still has to design and select the constraints carefully. Contrarily to these approaches where domain experts have to provide the domain knowledge, we proposed a framework for automatically learning a problem space from logged user interactions for procedural and ill-defined domains, and to use this knowledge base to support tutoring services [3]. The paper is organized as follows. First, it introduces RomanTutor, the tutoring system in which this work was applied. Then, the paper describes the data mining framework, several limitations and improvements, preliminary results of applying the framework in RomanTutor, and a conclusion.

1. The RomanTutor Tutoring System

RomanTutor [2] is a simulation-based tutoring system to teach astronauts how to operate the Canadarm2, a 7 degrees of freedom robotic arm deployed on the International Space Station (ISS). 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. Furthermore, for a given robot manipulation problem, there are many possibilities for moving the robot to a goal position and thus, it is not possible to define a complete and explicit task model. As a solution, we identified 155

atomic actions that a learner can carry, which are (1) selecting a camera, (2) performing a small/medium/big increase or decrease of the pan/tilt/zoom of a camera and (3) applying a small/medium/big positive or negative rotation value to an arm joint. We then logged sequences of user actions for two manipulations scenarios. Then, we defined the problem of mining procedural domain knowledge as the problem of finding relevant patterns among recorded user interactions, to build a knowledge base that can support tutoring services [3].

2. The Data Mining Framework

The main component of our framework is sequential pattern mining (see [3] for other parts of the framework). The problem of mining sequential patterns is stated as follows [5]. Let D be a transactional database containing a set of transactions (here also called plans) and a set of sequence of items (called actions in our context). An example of D is depicted in figure 1.a. Let $A = \{a_1, a_2, \dots, a_n\}$ be a set of actions. We call a subset $X \subseteq A$ an actionset and $|X|$, its size. A sequence $s = (X_1, X_2, \dots, X_m)$ is an ordered list of actionsets, where $X_i \subseteq A$, $i \in \{1, \dots, m\}$, and where m is the size of s (also noted $|s|$). A sequence $s_a = (a_1, a_2, \dots, a_n)$ is contained in another sequence $s_b = (b_1, b_2, \dots, b_m)$ if there exists integers $1 \leq i_1 < i_2 < \dots < i_n \leq m$ such that $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, \dots , $a_n \subseteq b_{i_n}$. The relative support of a sequence s_a is defined as the percentage of sequences $s \subseteq D$ that contains s_a , and is denoted by $\text{supD}(s_a)$. The problem of mining sequential patterns is to find all the sequences s_a such that $\text{supD}(s_a) \geq \text{minsup}$ for a database D , given a support threshold minsup .

ID	Sequences of actions		ID	Seq. patterns	Support
1	1 2 25 46 48 {9 10 11 31}	→	S1	1 46 48	66 %
2	1 25 46 54 {10 11 25} 48		S2	1 25 46 48	50 %
3	1 2 3 {9 10 11 31} 48		S3	1 25 46 {10 11}	33 %
4	2 3 25 46 11 {14 15 48} 74		S4	1 {9 10 31}	33 %
5	4 1 25 27 46 48		S5	1 {9 11 31}	33 %
6	1 3 44 45 46 48	

Fig. 1. (a) A Data Set of 6 Plans (b) Example of Sequential Patterns Extracted

Consider the dataset of figure 1.a. The size of the plan 2 is 6. Suppose we want to find the support of S2. From figure 1.a, we know that S2 is contained in plan 1, 2 and 5. Hence, its support is 3 (out of a possible 6), or 0.50. If the user-defined minimum support value is less than 0.50, then S2 is deemed frequent. To mine sequential patterns several algorithms have been proposed [5], [6]. In a first experiment in RomanTutor, we chose PrefixSpan [6] because it can be easily extended to mine sequential patterns with user-specified constraints. Figure 1.b shows some sequential patterns extracted by PrefixSpan from the data in figure 1.a using a minsup of 25%. However, using PrefixSpan to extract frequent action patterns we faced several limitations.

A first concern that we encountered is that extracted patterns very often contain “gaps” with respect to their containing sequences. For instance, in the example of figure 1, the action “2” of plan P1 has not been kept in the sequential pattern S1. A gap of a few actions is ok because it eliminates “noisy” (non-frequent) learners’ actions. But when sequences contain many gaps or when a gap is too big, it is difficult to use this sequence to track a learner’s actions and suggest a next step. Thus, there should be a way of limiting the size of the gaps in mined sequences. A second concern with SPM is that it can mine sequential patterns that are too short to be useful in a tutoring context (for example, sequences of size 1 or 2). To address these issues, we replaced PrefixSpan by an extended version proposed by Hirate & Yamana [7], which can mine patterns from a database with time information. A time-extended database is defined as a set of time-extended sequences $s = \langle (t_1, X_1), (t_2, X_2), \dots, (t_n, X_n) \rangle$, where each actionset X_x is annotated with a timestamp t_x . Each

timestamp represents the time elapsed since the first actionset of the sequence. Actions within a same actionset are considered simultaneous. For example, one time-extended sequence could be $\langle (0, a), (1, b\ c), (2, d) \rangle$, which represent that action “d” have been done one time unit after “b” and “c”, and two time units after action “a”. The time interval between two actionsets (t_x, X_x) and (t_y, X_y) is calculated as $|t_x - t_y|$. Note that if the time interval is the same between any adjacent actionsets, then $|t_x - t_y|$ become a measure of the number of actionsets between (t_x, X_x) and (t_y, X_y) . The Hirate-Yamana algorithm extract all time-extended sequences s from a time-extended dababase, such that $\text{supD}(s) \geq \text{minsup}$ and that s respect all time constraints. The time constraints are the minimum and maximum time interval required between two adjacent actionsets of a sequence (gap size), and the minimum and maximum time interval required between the head and tail of a sequence. For example, for the sequence $\langle (0, a), (1, b\ c), (2, d) \rangle$, the time interval between the head and the tail is 2 and the time interval between the actionset $(0, a)$ and $(1, b\ c)$ is 1.

A second limitation encountered when applying PrefixSpan to extract a problem space is that it relies on a finite set of actions, for a given domain. As a consequence, if some actions were designed to have parameters or values, they have to be defined as one or more distinct actions. In RomanTutor, for example, learners have to perform joint rotations by applying rotation values to joints. In our initial work, we categorized the joint rotations as small, medium and big, which correspond respectively to less than 60 degrees, 60 to 100 degrees, and more than 140 degrees. The disadvantage with this categorization is that it is fixed. In order to have dynamic categories of actions, we have extended the definition of time-extended sequence database, to have sequence that contains valued actions. A valued action $a\{value\}$, is defined as an action a that has a value $value$. In this work, we consider that a value is an integer. For example, the sequence $\langle (0, a\{2\}), (1, b), (2, bc\{4\}) \rangle$ contains the valued action a , and b with values 2 and 4, respectively. To mine patterns from a valued database, we modified the Hirate-Yamana algorithm, so that when it counts the frequency of an action, it automatically groups similar values and treat these groups as different actions (see [4] for details). For each group, the median value is kept as an indication of the values grouped. Note that it would be possible to keep more information in the mined sequences such as the minimum and maximum values for each group.

A third limitation that we encountered when applying the PrefixSpan algorithm is that it does not consider the context of each sequence. In a tutoring system context, it would be useful, for instance, to annotate sequences with success information and the expertise level of a user and to mine patterns containing this information. Our solution to this issue is to add dimensional information to sequences. Pinto et al. [8] originally proposed Multi-dimensional Sequential Pattern Mining (MDSPPM), as an extension to SPM. A Multidimensional-Database (MD-Database) is defined as a sequence database having a set of dimensions $D = \{D_1, D_2, \dots, D_n\}$. Each sequence of a MD-Database (an MD-Sequence) possesses a symbolic value for each dimension. This set of value is called an MD-Pattern and is noted $\{d_1, d_2, \dots, d_n\}$. For example, consider the MD-Database depicted in the left part of figure 2. The MD-Sequence 1 has the MD-Pattern $\{\text{“true”}, \text{“novice”}\}$ 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 $P_x = \{d_{x1}, d_{x2}, \dots, d_{xn}\}$ is said to be contained in another MD-Pattern $P_y = \{d_{y1}, d_{y2}, \dots, d_{ym}\}$ if there exists integers $1 \leq i_1 < i_2 < \dots < i_n \leq m$ such that $d_{x1} \subseteq d_{y1}, d_{x2} \subseteq d_{y2}, \dots, d_{xn} \subseteq d_{yn}$. The problem of MDSPPM is to find all MD-Sequence appearing in a database with a support higher than minsup. The right part of figure 2 shows some patterns that can be extracted from the MD-Database of figure 2, with a minsup of 2 sequences.

To achieve MDSPPM, we applied SeqDim [8] and tested it with our extended version of the Hirate-Yamana algorithm. In our experiment with RomanTutor, MDSPPM showed to be useful as it allowed to successfully identify patterns common to all expertise level that

lead to failure (“*, failure”), for example. Currently, we have encoded two dimensions: expertise level and success. But additional dimensions can be easily added. In future works, 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 by finding common skills demonstrated by users who used that pattern. This will allow a thorough cognitive diagnosis of missing and misunderstanding skill for the users who demonstrated part of that pattern.

An MD-Database			Mined MD-Sequences	
ID	Dimensions	Sequences	Dimensions	Sequences
1	true, novice	<(0,a),(1,bc)>	*, novice,	<(0,a)>
2	true, expert	<(0,d) >	*, *	<(0,a)>
3	false, novice	<(0,a),(1,bc)>	*, novice	<(0,a), (1,b)>
4	false, interm.	<(0,a),(1,c), (2,d)>	true, *	<(0,d)>
5	true, novice	<(0,d), (1,c)>	true, novice	<(0,c)>
6	true, expert	<(0,c), (1,d)>	true, expert	<(0,d)>

Fig. 2. An Example of SPM with Dimensions and Time Information

The last improvement that we made is not to the SPM algorithm. But rather to how we apply the algorithm to extract a partial problem space. Originally [3], we recorded sequences of user actions for a whole problem-solving exercise in RomanTutor. Then from the sequences recorded, we mined frequent patterns. But, we noticed that this approach is problematic when an exercise is long. For example, we noticed that in general, after 6 or more actions performed by a learner, it becomes hard to tell which pattern the learner is doing. For this reason, we added the definition of “problem states”. For example, in the RomanTutor, where an exercise consists of moving a robotic arm to attain a specific arm configuration, a problem state is defined as the set of 3D zones containing the arm joints. An exercise is then viewed as going from a problem state P_1 to a problem state P_F . Two types of sequences are logged when a learner do an exercise. First, we log the sequence of problem states visited by the learner $A=\{P_1, P_2 \dots P_n\}$. Second, we log 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 performed the same exercise, we extract sequential patterns from (1) the sequences of problems states visited, and (2) from the sequences of actions performed for going from a problem state to another. Dividing long problems into sub-problems allow a better guidance of the learner, because at any moment, only the patterns starting from the current problem state are considered.

We describe next how the main tutoring services are implemented. To recognize a learner’s plan, the system proceeds as follow. The first action of the learner is compared with the first action of each frequent pattern for the current problem state. If the actions do not match for a pattern, the system discards the pattern. Each time the learner makes an action, the system repeats the same process. It compares the actions done so far by the learner with the remaining patterns. When the problem-state changes, the system considers the set of patterns associated to the new problem state. If at any given moment a user action does not match with any patterns, the algorithm ignores the last user action or the current action to match for each pattern. This makes the plan recognizing algorithm more flexible and has shown to improve its effectiveness. At a more coarse grain level, a tracking of the problem states visited by the learners is achieved similarly as the tracking for actions. One utility of the plan recognizing algorithm for actions/problem states is to assess the expertise level of the learner (novice, intermediate or expert) by looking at the patterns applied.

The plan recognizing algorithm also plays a major role in the RomanTutor tutoring service for guiding the learner. It allows determining the possible actions from the current state according to the problem space. This functionality is triggered when the student selects

“What should I do next?” in the interface menu. The algorithm returns the set of possible actions with the associated patterns. 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 of the learner. In the cases where no actions can be identified, RomanTutor use a path planner [2] to generate an approximate solution.

3. A Preliminary Experiment

We conducted a preliminary experiment in RomanTutor for two scenarios depicted in [3]. We asked 12 users who participated in the first experiment to record new plans. The expertise level and success failure was added manually to sequences. From this data, we extracted sequential patterns for problem-states and actions with the extended SPM algorithm. In a subsequent work session, we asked the users to compare the tutoring services offered in [3] with those offered with the newly extracted knowledge base. On the whole, users preferred the newer version, as the hints offered were generally more precise and more appropriate, and help could be provided in more situations. We also observed that the system inferred more often correctly the estimated expertise level of learners.

4. Conclusion

In this paper, we reported several limitations encountered when applying SPM to extract procedural domain knowledge from logged user interaction. To overcome these limitations, we presented an extended SPM framework that combines (1) time intervals, (2) multi-dimensional pattern mining and (3) the automatic clustering of valued actions. We also suggested dividing problems into problem states to enhance the relevance of the tutoring services. The proposed framework has been implemented and was used to extract a problem space, and support tutoring services in RomanTutor. We are working on including skills as dimensional information, and on conducting a larger experiment.

Acknowledgements

Our thanks go to the FQRNT and NSERC for their logistic and financial support. We also thank the members of GDAC/PLANIART teams who have participated in the development of RomanTutor, and specially Severin Vigot for integrating the algorithm in RomanTutor.

References

- [1] Mitrovic, A., Mayo, M., Suraweera, P. & Martin, B. (2001). “Constraint-based tutors: a success story”. *Proc. Industrial & Engineering Application of Artificial Intelligence & Expert Systems* (pp. 931-940).
- [2] Kabanza, F., Nkambou, R. & Belghith, K. (2005), Path-planning for Autonomous Training on Robot Manipulators in Space, *Proc. of IJCAI 2005*.
- [3] Nkambou, R, Mephu Nguifo, E. & Fournier-Viger, P. (2008), Using Knowledge Discovery Techniques to Support Tutoring in an Ill-Defined Domain. *Proceedings of the 9th International Conference on Intelligent Tutoring Systems (ITS 2008)*. LNCS 5091, Springer (pp. 395-405).
- [4] Fournier-Viger, P., Nkambou, R. & Mephu Nguifo, E. (2008). A Sequential Pattern Mining Algorithm for Extracting Partial Problem Spaces from Logged User Interactions. *Proc. 3rd Intern. Workshop on Intelligent Tutoring Systems in Ill-Defined Domain (in conjunction with ITS2008)* (pp. 46-55).
- [5] Agrawal, R. & Srikant, R.: (1995). Mining Sequential Patterns. *Proc. Int. Conf. on Data Engineering* (pp. 3-14).
- [6] Pei, J., Han, J. et al. (2004). Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Trans. Knowledge and Data Engineering*, 16(10), 1-17.
- [7] Hirate, Y., Yamana, H. (2006). Generalized Sequential Pattern Mining with Item Intervals, *Journal of Computers*, 1(3), pp. 51-60.
- [8] Pinto, H et al. (2001), Multi-Dimensional Sequential Pattern Mining, *Proc. Int. Conf. Information and Knowledge Management* (pp. 81-88).