Identifying User Strategies in Exploratory Learning with Evolving Task Modelling

Mihaela Cocea and George D. Magoulas
London Knowledge Lab, Birkbeck College, University of London
Email: {mihaela, gmagoulas}@dcs.bbk.ac.uk

***Abstract*—In this paper we present work on adaptive identification of learners' strategies, gradually developing a higher level of adaptation based on evolving models of mathematical generalisation tasks in an Exploratory Learning Environment. A similarity-based classification approach is defined for the identification of strategies, using an initially small number of classes (i.e. strategies). A strategy is composed of several patterns with relations between them. An evolution monitor component observes changes in the environment and triggers a mechanism that builds-up the task model. The task model evolves when new relevant information becomes available by adding a new strategy (class) or a new inefficient pattern, i.e. patterns that make it difficult for the learner to generalise.**

I. Introduction

Adaptive on-line systems are characterised by their ability to modify themselves to account for new data at the time of encounter with it [1]. This property would be highly desirable for adaptive educational systems that aim to deliver personalised support by diagnosing the learner based on the knowledge stored in a predefined model. Adaptivity in the context of educational systems refers to modifications in content, presentation or feedback depending on learners' characteristics. These characteristics could be knowledge/skills, preferences, goals, etc. Here we focus on adaptation of learning to the knowledge/skills level of the user. For this situation, the predefined model describes the educational domain and sometimes the tasks as well [2], [3]. Some topics allow a comprehensive definition of a domain and/or task model, while for others such a definition is not possible. The later ones are usually referred to as ill-defined domains and these would particularly benefit from a mechanism that updates the domain/task model on encounter with new relevant data.

In this paper we present our approach to on-line adaptation of task models in *eXpresser*, an exploratory learning environment for the ill-defined domain of mathematical generalisation. As learners solve a task in *eXpresser*, they can follow different strategies and personalised support is dependent on the identification of the particular strategy followed by the learner.

This is a challenging task as not all strategies are known in advance; extracting strategies from teachers is one way of defining the task model; collecting data from pupils is another. These, however, allow the collection of only limited amounts of data whereas other applications of adaptive modelling, such as intelligent control or system identification, have access to hundreds or thousands of pieces of data.

Also, the way learners use the system changes over time. Thus, as they acquire more knowledge of mathematical generalisation, they tend to use more neat and elegant approaches. Also, as they are taught other mathematical topics they use that knowledge in solving tasks, e.g. use of areas to find out the perimeter. Moreover, besides solving tasks individually, *eXpresser* is used for collaborative activities, one of which challenges learners to find a different strategy than the ones used by one or several of their peers, leading to creative and unpredictable approaches. Besides the diversity of the strategies, learners also use inefficient approaches that would make it difficult for them to generalise and that create additional problems for personalised support as these inefficient approaches may hinder the recognition of the strategy they follow. To address these issues evolving task modelling is proposed in this paper with the aim to add new relevant information to the task model as it becomes available.

The rest of the paper is organised as follows. The next section briefly describes *eXpresser* and two generalisation tasks. Section III presents the formulation of features that describe user behaviour on the task and the mechanism for similarity-based classification of user strategies. Section IV presents the components and explains the phases of the mechanism for evolving the task models. Section V presents the algorithms for on-line structural adaptation of the task models, Section VI presents experiments and results obtained using these algorithms and, finally, Section VII concludes the paper.

II. eXpresser

eXpresser [4] is an exploratory learning environment for the domain of mathematical generalisation. It enables constructions of patterns, creating dependencies between them, naming properties of patterns and creating algebraic-like rules with either names or numbers. It is designed for classroom use and targets pupils of 11-14 year-olds. Each task involves two main phases: constructing a model and deriving an algebraic-like rule from it.

Fig. 1 illustrates the system, the *properties list* of a pattern (linked to another one) and an example of a rule. The screenshot on the left includes two windows: (a) the students' world, where the students build their constructions and (b) the general world that displays the same construction with a different value for the variable(s) involved in the task, and where students can check the generality of their construction by animating their pattern (using the Play button).

We illustrate here a task called 'stepping stones' (see Fig. 1) displayed in the students' world with a number of 3 red (lighter colour) tiles and in the general world with a number of 8 red tiles; the task requires to build such a construction and to find a general rule for the number of blue (darker colour) tiles needed to surround the red ones. The construction for this task can be built in several ways that we call *strategies*. Here we illustrate the 'C strategy', named after the shape of the building-block, i.e. the basic unit of a pattern. The components of this strategy are displayed separately in the students' world for ease of visualisation: a red pattern with 3 tiles, a blue one made of a C-shape pattern repeated 3 times, and 3 blue tiles.

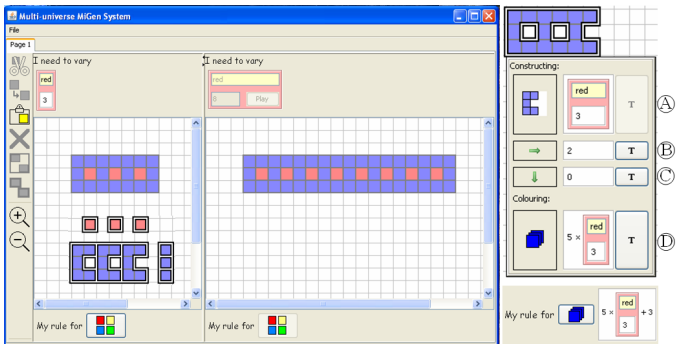


Fig. 1. *eXpresser* screenshots. The left screenshot includes a toolbar, the students' world and the general world. The top right screenshot shows the property list of a pattern. The bottom right screenshot illustrates a rule.

The property list of the C-shape pattern is displayed in the top right screenshot. The first property (Ⓐ) specifies the number of iterations of the building-block; the value for this attribute is set to the value of the iterations of the red pattern by using a T-box (that includes a name and a value); by using a T-box, the two (or more) properties are made dependent, i.e. when the value in the T-box changes in one property, it also changes in the other one(s). The next properties are *move-right* (Ⓑ), which is set to 2, and *move-down* (Ⓒ), which is set to 0. The last property (Ⓓ) establishes the number needed to colour all the tiles in the pattern - in our case 5 times the iterations of the red pattern. The bottom right screenshot displays the rule for the number of blue tiles: 5xred + 3, where red stands for the T-box displayed in Ⓐ (a T-box can be displayed with name only, value only or both).

The construction in Fig. 1 and the rule in the bottom-right corner constitute one possible solution for the 'stepping stones' task. Although in its simplest form the rule is unique, there are several ways to build the construction and infer a rule from its components. Thus, there is no unique solution and students follow various kinds of strategies to construct their models - two examples are illustrated in Fig. 2.

Another task example for *eXpresser* is the 'pond tiling' task which requires to find the number of tiles needed to surround any rectangular pond. Fig. 3 illustrates the construction for this problem and two strategies frequently used by students to solve this task. Compared to the 'stepping stones' task, 'pond tiling' is more complex, requiring the students to generalise using two variables rather than one. The next section presents the strategy modelling and classification mechanisms.

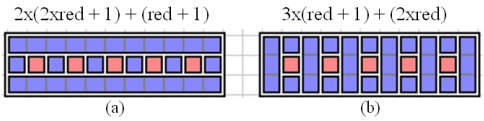


Fig. 2. (a) 'HParallel' Strategy; (b) 'VParallel' Strategy.

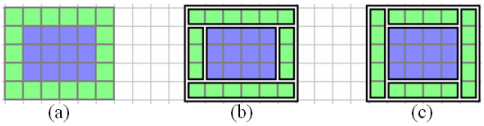


Fig. 3. (a) the construction for the 'pond tiling' task ; (b) 'I Strategy'; (c) 'H strategy'.

III. Task Model and Strategy Classification

The Task Model (TM) includes strategies that learners could follow when solving a task. A strategy is defined as $S_i = \{P_i, R_i\}$, where P_i represents a set of patterns and R_i represents a set of relations between patterns of P_i.

Each pattern of P_i is defined as an attribute-value vector and includes three types of attributes: (a) numeric, (b) variables and (c) binary. The numeric attributes correspond to the values in the property list and the variables correspond to the type of those properties: number, T-box, expression with number(s) or expression with T-box(es). The binary attributes refer to the membership of a pattern to a strategy and is defined as a $PartOfS$ function which returns 1 if the pattern belongs to the strategy and 0 if it does not. There are S binary attributes, where S is the number of strategies in the task model.

The set of relations R_i is defined as $R_i = \{RA_i, RP_i\}$. RA_i is a set of relations between attributes of patterns and RP_i is a set of relations between patterns. RA_i includes two types of relations: (a) *dependency relations* such as the one illustrated in Fig. 1 where the number of the iterations of the blue pattern depends on the iterations of the red pattern through the use of a T-box; (b) *value relations* such as the fact that the value of the colouring property of the blue pattern in Fig. 1 is 5 times the value of the iterations of the red pattern. A strategy is specific when it does not have dependency relations and is general when it has all the dependency relations required by the task. RP_i includes ordering relations: $Prev$ relation indicates the previous pattern and $Next$ relation indicates the next pattern, with respect to the current pattern. The structure of strategies is displayed in Fig. 4.

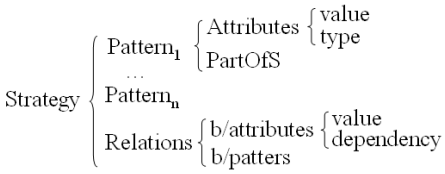


Fig. 4. The structure of strategies.

To provide personalised support while learners are solving a task, it is important to identify what strategies they are using, so that the support is about the particular approach used rather than a predefined strategy that has little to do with the learner's current thinking. The strategy identification is in fact a similarity-based classification problem [5], where the training samples are the strategies stored and each class has only one exemplar. A new construction is classified by

calculating the distance to all training samples and choosing the label of the closest one (k-nearest neighbor with k=1).

The similarity metrics used for patterns, relations and strategies are displayed in Fig. 5; four different metrics are used for two reasons: (a) different types of data, i.e. numeric, string, sets, call for different metrics and (b) the four categories (see Fig. 5) are valuable for providing personalised feedback, e.g. if the learner needs to work on the structure of the strategy or on the relations between patterns. The overall similarity metric for strategies is: $Sim = w_1 * \overline{F_1} + w_2 * F_2 + w_3 * F_3 + w_4 * F_4$, where $\overline{F_1}$ is the normalised value of F_1. To bring F_1 in the same range as the other metrics, i.e. $[0, 1]$, we applied linear scaling to unit range [6] using the function $\overline{F_1} = F_1/z$. Weights are applied to the four similarity metrics to express the central aspect of the construction, the structure. This is mostly reflected by the $\overline{F_1}$ metric and, to a lesser extent, by the F_3 metric. Therefore, we agreed on the following weights: $w_1 = 6, w_2 = 1, w_3 = 2, w_4 = 1$. This leads to the range of $[0, 10]$ for the values of Sim.

Similarity measures	Pattern/Relation	Strategy
Numeric attributes	$D_{IR} = \sqrt{\sum_{j=v+1}^{w} (\alpha_{I_j} - \alpha_{R_j})^2}$	$F_1 = \begin{cases} z / \sum_{i=1}^{z} D_{I_i R_i}, & \text{if } \sum_{i=1}^{z} D_{I_i R_i} \neq 0 \\ z, & \text{if } \sum_{i=1}^{z} D_{I_i R_i} = 0 \end{cases}$
Variables	$V_{IR} = \frac{\sum_{j=1}^{v} g(\alpha_{I_j}, \alpha_{R_j})}{v}$ $g(\alpha_{I_j}, \alpha_{R_j}) = \begin{cases} 1, & \text{if } \alpha_{I_j} = \alpha_{R_j} \\ 0, & \text{if } \alpha_{I_j} \neq \alpha_{R_j} \end{cases}$	$F_2 = (\sum_{i=1}^{z} V_{I_i R_i}) / z$
Relations between attributes	$P_{IR} = \frac{\lvert RA_I \cap RA_R \rvert}{\lvert RA_I \cup RA_R \rvert}$	$F_3 = (\sum_{i=1}^{z} P_{I_i R_i}) / y$
Relations between patterns	$T_{IR} = \frac{\lvert RP_I \cap RP_R \rvert}{\lvert RP_I \cup RP_R \rvert}$	$F_4 = (\sum_{i=1}^{z} T_{I_i R_i}) / z$

I = Input Pattern/Strategy; R = Retrieved Pattern/Strategy
z = minimum number of patterns in strategies I and R
y = the number of relations between attributes in strategy R.

Fig. 5. Similarity metrics for patterns/relations and strategies.

In classroom sessions with pupils we have observed that some pupils tend to build some patterns inefficiently - mainly by grouping several individual tiles and using that group as the building block. This information is useful for teachers and, therefore, it should be included in the TM; it could also be incorporated in the strategy classification mechanism when we have knowledge about how these inefficient patterns affect the recognition of strategies. However, inefficient constructions that are likely to be used by learners are hard to identify. Therefore, initially the TM includes only strategies and inefficient patterns are added into a separate set on-line, when such patterns are identified.

IV. Evolving Task Modelling

The evolution process is displayed in Figure 6. As the learner is solving a task and making a construction using *eXpresser*, the Strategy Identification Module, using the input from the Task Model (i.e. the strategies) identifies the strategy used by the learner through the similarity-based approach presented in the previous section. However, sometimes the learners' constructions, either partially or as a whole, are dissimilar to the information stored in the TM. If that is the case, the Evolution Monitor Module verifies if the new situations are relevant for the task and if they should be added to the TM. If the verification is successful, the Adaptation Mechanism updates the TM, adjusting its structure and the structure of its patterns. The process includes five phases:

Phase 1. Start with a TM that has some strategies introduced by the researcher/teacher.

Phase 2. Collect new data on-line.

Phase 3. Calculate similarity between new data and stored strategies (Strategy Identification Module) using the metrics from Section III. If the similarity is between certain boundaries specified in the Evolution Monitor Module and checked by Algorithms 1 and 4 (Section V), go to next phase. Else, stop.

Phase 4. Using Algorithms 2 and 5 (Section V), perform controls on new data to verify whether these represent a new valid strategy or a new inefficient approach that can be considered part of an existing strategy. If satisfied, go to next phase. Else, stop.

Phase 5. Using Algorithms 3 and 6 (Section V), adapt the structure of the TM by adding the new strategy and updating the structure of patterns, or by adding a new inefficient pattern (Adaptation Module).

Two examples together with the algorithms for the TM on-line structural adaptation are presented in the next section.

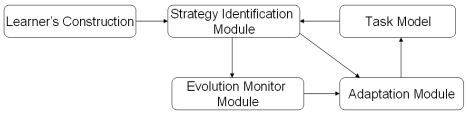


Fig. 6. The evolution process consists of structure adaptation and features adaptation.

V. Structure Adaptation

For certain applications, like educational systems, it is highly desirable to be able to evolve the structure of the task models used in the similarity-based identification mechanism without having to rebuild the system. On-line algorithms have been developed to identify and add new strategies and new inefficient patterns to the task model structure, without affecting the recognition of the ones already in the TM. Therefore, adding new strategies and new inefficient patterns does not require modifications of the similarity-based identification mechanism to deal with the new data.

To add a new strategy to the TM structure, a new set of patterns and their relations needs to be added and then each pattern (either efficient or inefficient) needs a new $PartOfS$ attribute that reflects whether it belongs to the newly introduced strategy or not. Therefore, the structure of TM and of the patterns are modified. Similarly, when a new inefficient pattern is added the structure of TM is modified.

Fig. 7 shows two examples from the 'stepping stones' task introduced previously for an inefficient pattern and a new strategy; the constructions in Fig. 7a and 7c have been broken down into the individual components used by the students for ease of visualisation. These examples and the corresponding algorithms are detailed in the following subsections.

A. Inefficient patterns acquisition mechanism

The goal of this mechanism is to identify patterns of strategies constructed in inefficient ways and store them in

a library of 'inefficient patterns', i.e. constructions that pose difficulties for the learners in their process of generalisation. The library could be further used for automatic generation of feedback, or could be analysed by a researcher or teacher. The results of such an analysis could be then used to design better interventions or make other design decisions for the current system, could be presented as a lesson learned to the scientific community of mathematics teachers and researchers, or even discussed further in class (e.g if an inefficient construction is frequently used by the pupils of that class).

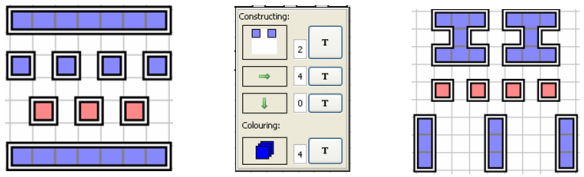


Fig. 7. (a) HParallel strategy with one inefficient component (blue middle row) ; (b) property list of the inefficient component; (c) a new strategy

The construction in Fig. 7a illustrates an inefficient pattern within the "HParallel" strategy of the 'stepping stones' task: the middle bar of blue tiles is constructed as a group of two tiles repeated twice - this can be seen in the property list of this pattern displayed in Fig. 7b. The efficient way to construct this component is one tile repeated four times or, to make it general, one tile repeated the number of red tiles plus one. The efficient and the inefficient way of constructing the middle row of blue tiles lead to the same visual output, i.e. there is no difference in the way the construction looks like, making the situation even more confusing. The difficulty lies in relating the values used in the construction of the middle row of blue tiles (P_i) to the ones used in the middle row of red tiles (P_j). If the learner would relate the value 2 of iterations of P_i to the value 3 of iterations of P_j, i.e. the value 2 is obtained by using the number of red tiles (3) minus 1, this would work only for a 'stepping stones' task defined for 3 red tiles. In other words, this will not lead to a general model.

Algorithms 1, 2 and 3 illustrate how inefficient patterns are identified and stored. First, the most similar strategy is found. If there is no exact match, but the similarity is above a certain threshold θ, the process continues with the identification of the inefficient patterns; for each of these patterns, several checks are performed (Alg. 2). Upon satisfactory results and if the patterns are not already in the set of inefficient patterns, they are then stored (Alg. 3).

B. New strategies acquisition mechanism

The goal of this mechanism is to identify new strategies and store them for future use. New strategies could be added by the teacher or could be recognized as new from the learners' constructions. In the later case, after the verification checks described below, the decision of storing a new strategy is left with the teacher. This serves as another validation step.

Fig. 7c illustrates the so-called "I strategy", as some of its building blocks resemble the letter I. When compared to all stored strategies, this strategy is rightly most similar to the 'VParallel' one (see Fig. 2b), as some patterns correspond to it. However, the similarity is low, suggesting it may be a new strategy. Without any form of TM adaptation, the learner modelling module will infer that the learner is using the 'VParallel' strategy, but is still far from having completed it. This imprecise information could be potentially damaging as it could, for example, lead to inappropriate system actions, e.g. providing confusing feedback that would guide the learner towards the 'VParallel' strategy. Conversely, identifying the new construction as a new valid strategy will prevent generating potentially confusing feedback, and storing the new strategy will enable producing appropriate feedback in the future - automatically or with input from the teacher/researcher.

Algorithm 1 Verification($Strategies$, $InefficientPatterns$, $InputStrategy$)

Find most similar strategy to $InputStrategy$ from $Strategies$
$StoredStrategy \leftarrow$ most similar strategy;
if $similarity > \theta$ **then**
  Find patterns of $InputStrategy$ that are not an exact match to any pattern of $StoredStrategy$
  for each patterns that is not an exact match **do**
    $InputPattern \leftarrow$ the pattern that is not an exact match
    Compare $InputPattern$ to all patterns in $InefficientPatterns$;
    if no exact match **then**
      Find the most similar pattern to $InputPattern$ from the patterns of $StoredStrategy$
      $StoredPattern \leftarrow$ the most similar pattern
      if Conditions($StoredPattern$, $InputPattern$) returns **true** **then** {see Alg. 2}
        InefficientPatternAcquisition($StoredPattern$, $InputPattern$) {see Alg. 3}
      end if
    end if
  end for
end if

Algorithm 2 Conditions($P1$, $P2$)

if ($MoveRight[P1] \neq 0$ and $Iterations[P1] * MoveRight[P1] = Iterations[P2] * MoveRight[P2]$) or ($MoveDown[P1] \neq 0$ and $Iterations[P1] * MoveDown[P1] = Iterations[P2] * MoveDown[P2]$) **then**
  return true
else
  return false
end if

Algorithm 3 InefficientPatternAcquisition($StoredPattern$, $InputPattern$, $InefficientPatterns$)

$NewPattern \leftarrow StoredPattern$
for $i = 1$ to w **do** {all attributes}
  if value of attribute i of $NewPattern$ different from that of $InputPattern$ **then**
    replace value of attribute i of $NewPattern$ with the one of $InputPattern$
  end if
end for
add $NewPattern$ to $InefficientPatterns$

Algorithms 4, 5 and 6 illustrate the process by which an input strategy is identified and stored as a new strategy. If the similarity between the input strategy and the most similar strategy in the TM is below a certain threshold θ_1 (Alg. 4), some validation checks are performed (Alg. 5) and upon satisfaction, the new strategy is stored in the TM (Alg. 6). If the input strategy has been introduced by a teacher and the similarity is below θ_1, the teacher can still store the new strategy, even if it is very similar to an existing one in the TM.

In Algorithm 5 the SolutionCheck($InputStrategy$) function verifies whether $InputStrategy$ 'looks like' a solution by examining if the mask of $InputStrategy$ corresponds to the mask of the task. The following check takes into consideration the number of patterns in the $InputStrategy$. Good solutions are characterized by a relatively small number of patterns; therefore, we propose for the value of θ_2 the maximum number of patterns among all stored strategies for the corresponding task, plus a margin error (such as 3). If this check is satisfied, the RelationVerification($InputStrategy$) function derives a rule from the value relations of the $InputStrategy$ and checks its correspondence to the rule solution of the task. For example, in the construction of Fig. 7c, the rule derived is $3 * (\frac{red}{2} + 1) + 7 * \frac{red}{2}$ which corresponds to the solution $5 * red + 3$. If all checks are satisfied, the new strategy is stored in the TM and the $PartOfS$ values are adjusted.

Algorithm 4 NewStrategyVerification($Strategies$, $InputStrategy$)

```
Find most similar strategy to InputStrategy from Strategies
if similarity < θ1 then
    if ValidSolution(InputStrategy) returns true then {see Alg. 5}
        NewStrategyAcquisition(InputStrategy) {see Alg. 6}
    end if
end if
```

Algorithm 5 ValidSolution($InputStrategy$)

```
if SolutionCheck(InputStrategy) returns true then {checks if
InputStrategy 'looks like' a solution}
    if the number of patterns of InputStrategy < θ2 then
        if InputStrategy has relations between attributes then
            RelationVerification(InputStrategy) {verifies that the numeric
            relation corresponds to the task rule solution}
            if successful verification then
                return true
            end if
        end if
    end if
end if
```

Algorithm 6 NewStrategyAcquisition($NewStrategy$)

```
add NewStrategy to Strategies
adjust values of PartOfS
```

VI. Experiments

The similarity-based classification has been tested using 36 constructions from classroom and individual sessions, resulting in 100% accuracy for identification of complete strategies, partial strategies (strategies with missing patterns) and mixed strategies (combinations of two or more strategies). As in this paper we focus on the evolving mechanism, in the rest of the section we present experiments that test its behaviour.

Three experiments were designed to test the evolving mechanism. The purpose of the first one was to identify the bounds for the inefficient modifications of a pattern that still allows its recognition as similar to the original efficient form. The second experiment looks at the correct identification of inefficient patterns within the bounds previously identified. The third experiment examines the correct identification of new strategies.

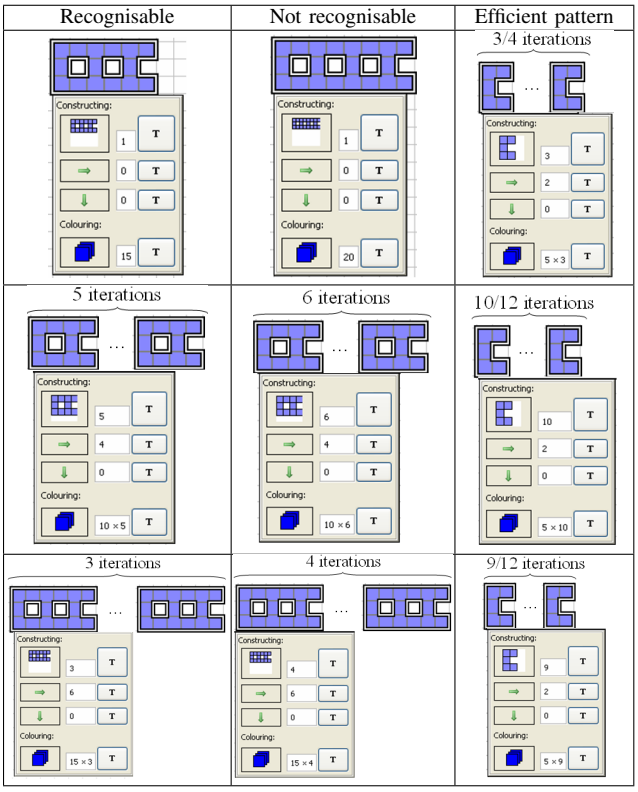
The data used in these experiments is a combination of real data from classroom sessions with *eXpresser* and artificial data that simulated observed situations from classroom trials. The simulated situations were obtained by varying parameters of patterns/strategies produced by learners and were used to provide us with more data.

Prior to the three experiments a small study was conducted on classroom data to identify possible values for the threshold θ in Algorithm 1 and threshold θ_1 in Algorithm 4. We aimed at identifying good enough values rather than seeking optimal ones, as our main goal was to test the evolving mechanism. Two possibilities were established for each threshold: for θ - the minimum *overall similarity* (4.50) minus an error margin (0.50) or value 1.00 for the *numerical similarity*; for θ_1: the maximum *overall similarity* (3.20) plus an error margin (0.30) or value 1 for the *numeric similarity*.

Experiment 1: identifying the bounds of how far a pattern can be inefficiently modified and still be recognised as similar to its original efficient form. As mentioned previously, we consider changes in a pattern that can lead to the same visual output as the original one but use different building-blocks. More specifically, these building-blocks are groups of two or more of the original efficient building-block. This experiment looks for the changes that a pattern can undergo without losing its structure so that it can still be considered the same pattern.

For this experiment we used 34 artificial inefficient patterns from the two tasks introduced in Section II: 'stepping stones' and 'pond tiling'. From the 34 patterns, 47% were from the 'stepping stones' task and 53% were from the 'pond tiling' task. Using these patterns, the following bounds were identified: (i) groups of less than 4 building-blocks; (ii) groups of 2 building-blocks repeated less than 6 times and (iii) groups of 3 building-blocks iterated less than 4 times.

TABLE I
BOUNDS FOR RECOGNITION OF INEFFICIENT PATTERNS



To illustrate these bounds the 'C Strategy' of the 'stepping stones' task is used in Table I. The building block of this strategy is a 'C shape' pattern. The 1st column illustrates an inefficient pattern that is still recognisable as similar to the efficient form displayed in the 3rd column and the 2nd column displays an inefficient pattern that is not recognisable anymore as similar to the efficient form. The three rows correspond to the three bounds mentioned above. The iterations of the efficient pattern from the 3rd column refer to the number of iteration needed to obtain the same shapes as the ones from the 1st and the 2nd column. For example, in the 1st row, the '3/4 iterations' denotes that 3 iterations are needed to obtain the same shape as the one in the 1st column and that 4 iterations are needed to obtain the same shape as the one in the 2nd column; the property list displays the first value.

Although to the human eye, the differences between the recognisable and not recognisable inefficient patterns seem very small because of the awareness of the sub-pattern that is repeated, the metrics at the moment do not incorporate that knowledge. In the future, we plan to develop a mechanism that detects sub-patterns and to integrate that information in the metrics used for the similarity-based classification.

Experiment 2: correct identification of inefficient patterns within the previously identified bounds. From the total of 34 inefficient patterns used in *Experiment 1*, 13 were outside the identified bounds and 21 were within. From the 21 patterns within the bounds, 62% were from the 'stepping stones' task and 38% were from the 'pond tiling' task. Using the previously identified values for θ, we obtained the following results: out of these 21 patterns, 52.48% had the overall similarity greater than 4.00 and 100% had the numeric similarity above 1.00.

These results indicate that a relatively small modification of a pattern can significantly affect the identification of the strategy the learner is following; hence almost half the patterns have an overall similarity less than 4.00. For example using any of the patters in column two of Table I with the other efficient components of the 'C strategy', leads to a small similarity when compared to the 'C strategy'.

Experiment 3: correct identification of new strategies. The data for this experiment included 10 new strategies: 7 observed in trials with pupils and 3 artificial. Out of the 10 new strategies, 4 were from the 'pond tiling' task; all of them were observed in trials with pupils. The remaining 6 new strategies were from the 'stepping stones' task, with 3 of them observed and 3 artificial. The task models for the two tasks included originally 4 strategies for the 'stepping stones' task and 2 strategies for the 'pond tiling' task. Using the previously identified values for θ_1, we obtained the following results: out of the 10 new strategies, 100% had the overall similarity below 3.50 and 70% had the numeric similarity below 1.00. Given the range that the overall similarity has, i.e. 0 to 10, values below 3.50 indicate a very low similarity and therefore, rightly suggest that the learner's strategy is considerably different from the ones in the task model.

A summary of results is displayed in Table II. They show that the algorithms are capable of recognising new relevant data with an accuracy of 100% using two of the previously identified values for the thresholds.

TABLE II
SUMMARY OF RESULTS

Experiment	Threshold		Accuracy
Identification of inefficient patterns	Overall similarity	4	52.48%
	Numeric similarity	1	100%
Identification of new strategies	Overall similarity	3.50	100%
	Numeric similarity	1	70%

As already mentions in Section III, the numeric similarity ($\overline{F_1}$) reflects the structure of a construction. Therefore, using the numeric similarity for the thresholds θ and θ_1 identifies structural modification - performing very well in *Experiment 2*, where all changes are numerical, but not as well in *Experiment 3*, where the changes are not exclusively numerical. As a strategy is not defined only by patterns and their structure, but by relations as well, the overall similarity performs better in *Experiment 3*.

VII. CONCLUSION

In this paper we have presented an approach for task model evolution in the context of an exploratory learning environment for mathematical generalisation. The task model includes initially a limited number of strategies that learners may adopt when solving a task. Using our proposed evolving mechanism, the task model is gradually enriched with more strategies and with a set of inefficient patterns, i.e. constructions that make the process of generalisation more difficult for the learner. Similarity-based classification is used for strategy identification and the same similarity metrics are employed in the identification of new relevant data. This mechanism enables the addition of new classes without affecting the recognition of the existing ones.

ACKNOWLEDGMENT

This work is partially funded by the ESRC/EPSRC Teaching and Learning Research Programme (Technology Enhanced Learning); Award no: RES-139-25-0381.

REFERENCES

[1] P. Angelov and D. Filev, "An approach to online identification of takagi-sugeno fuzzy models," *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, no. 1, pp. 484–498, 2004.

[2] M. Sasajima, Y. Kitamura, T. Naganuma, S. Kurakake, and R. Mizoguchi, "Oops: User modeling method for task oriented mobile internet services," in *Proceedings of WI '07*. IEEE Computer Society, 2007, pp. 771–775.

[3] P. Brusilovsky and D. W. Cooper, "Domain, task, and user models for an adaptive hypermedia performance support system," in *Proceedings of IUI '02*. ACM, 2002, pp. 23–30.

[4] D. Pearce, E. Geraniou, M. Mavrikis, S. Gutiérrez, and K. Kahn, "Using pattern construction and analysis in an exploratory learning environment for understanding mathematical generalisation: The potential for intelligent support," in *Proceedings of the 1st ISEE Workshop, EC-TEL'08*, 2008, pp. 21–30.

[5] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti, "Similarity-based classification: Concepts and algorithms," *J. Mach. Learn. Res.*, vol. 10, pp. 747–776, 2009.

[6] S. Aksoy and R. M. Haralick, "Feature normalization and likelihood-based similarity measures for image retrieval," *Pattern Recognition Letters*, vol. 22, no. 4, pp. 563–582, 2001.