Adaptive Modelling of Users' Strategies in Exploratory Learning using Case-based Reasoning

Mihaela Cocea, Sergio Gutierrez-Santos and George D. Magoulas

London Knowledge Lab, Birkbeck College, University of London,
23-29 Emerald Street, London, WC1N 3QS, UK
{mihaela, sergut, gmagoulas}@dcs.bbk.ac.uk

Abstract. In exploratory learning environments, learners can use different strategies to solve a problem. To the designer or teacher, however, not all these strategies are known in advance and, even if they were, introducing them in the knowledge base would involve considerable time and effort. In previous work, we have proposed a case-based knowledge representation, modelling the learners behaviour when constructing/exploring models through simple cases and sequences of cases, called strategies. In this paper, we enhance this approach with adaptive mechanisms for expanding the knowledge base. These mechanisms allow to identify and store inefficient cases, i.e. cases that pose additional difficulty to students in their learning process, and to gradually enrich the knowledge base by detecting and adding new strategies.

Key words: user modelling, knowledge base adaptation, exploratory learning environments, case-based reasoning

1 Introduction

Exploratory learning environments (ELEs) provide activities that involve constructing [1] and/or exploring models, varying their parameters and observing the effects of these variations on the models. When provided with guidance and support ELEs have a positive impact on learning compared with other more structured environments [2]; however, the lack of support may actually hinder learning [3]. Therefore, to make ELEs more effective, intelligent support is needed, despite the difficulties arising from their open nature.

To address this, we proposed a learner modelling mechanism for monitoring learners' actions when constructing/exploring models by modelling sequences of actions reflecting different strategies in solving a task [4]. An important problem, however, remains: only a limited number of strategies are known in advance and can be introduced by the designer/teacher. In addition, even if all strategies were known, introducing them in the knowledge base would take considerable time and effort. Moreover, the knowledge about a task evolves over time - students may discover different ways of approaching the task, rendering the knowledge base suboptimal for generating proper feedback, despite the initially good coverage. To address this, we employ a mechanism for adapting the knowledge base in the context of *eXpresser* [5], an ELE for mathematical generalisation.

The knowledge base adaptation involves a mechanism for acquiring inefficient cases, i.e. cases which include actions that make it difficult for students to create a

generalisable model, and a mechanism for acquiring new strategies. The former could be potentially useful to enable targeted feedback about the inefficiency of certain parts of a construction, or certain actions of the student; this approach could also lead gradually to creating a library of inefficient constructions produced by students that could be analysed further by a researcher/teacher. Without the later a new valid strategy will not be recognised as such, and, consequently, the learner modelling module will diagnose the learner to be still far from a valid solution and any potential feedback will be confusing as it will guide the learner towards the most similar strategy stored in the knowledge base.

The next section briefly introduces *eXpresser* and mathematical generalisation. Section 3 describes the case-based reasoning (CBR) cycle for *eXpresser* and gives a brief overview of the knowledge representation and the identification mechanism employed. Section 4 presents our proposed approach for adapting the knowledge base. Section 5 describes the validation of this approach and, finally, Section 6 concludes the paper and presents some directions for future work.

2 Mathematical Generalisation with *eXpresser*

eXpresser [5] is an ELE for the domain of mathematical generalisation. It is designed for classroom use and targets pupils of 11 to 14 year-olds. Each task involves 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 left screenshot includes two windows: (a) the students' world, where 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 model and to find a general rule for the number of blue (darker colour) tiles needed to surround the red ones. The model 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. Its components are displayed separately in the students' world for ease of visualisation: a red pattern, having 3 tiles, a blue one made of a C-shape pattern repeated 3 times, and 3 blue tiles.

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* (Ⓑ), set to 2, and *move-down* (Ⓒ), set to 0. The last property (Ⓓ) establishes the number for colouring 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: 5 x red + 3, where red stands for the T-box in Ⓐ (a T-box can be displayed with name only, value only or both).

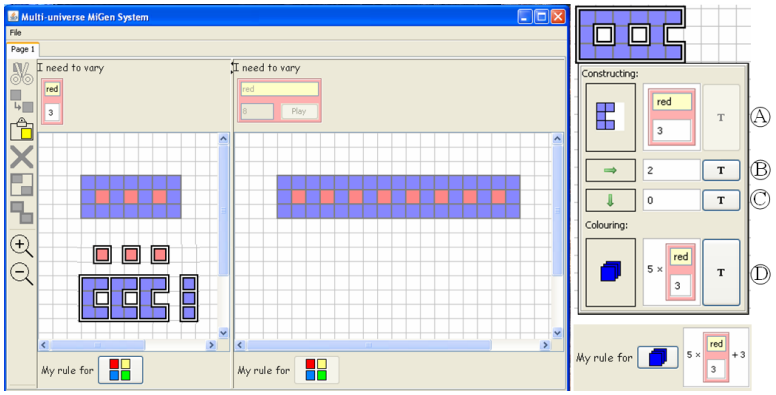

Fig. 1. *eXpresser* screenshots.

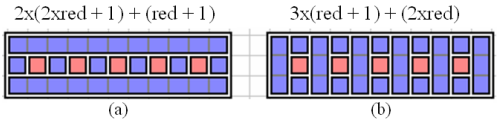

Fig. 2. (a) 'HParallel' Strategy; (b) 'VParallel' Strategy.

The construction and the bottom-right rule in Fig. 1 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 strategies to construct their models. Two such examples are illustrated in Fig. 2.

3 Modelling Learners' Strategies Using CBR

In case-based reasoning (CBR) [8] knowledge is stored as cases, typically including a problem and its solution. When a new problem is encountered, similar cases are retrieved and the solution is used/adapted from them. Four processes are involved [8]: (a) *Retrieve* similar cases to the current problem; (b) *Reuse* the cases (and adapt them) to solve the current problem; (c) *Revise* the proposed solution if necessary; (d) *Retain* the new solution as part of a new case.

In exploratory learning a problem has multiple solutions and it is important to identify the one the learner used or if the learner produced a new valid solution. To this end, *eXpresser* has a case-base (knowledge base) of solutions (i.e. strategies) for each task. When a learner builds a construction, it is transformed into a sequence of simple cases (i.e. strategy) and compared to all strategies in the case-base for the task the learner is working on; the case-base consists of strategies, i.e. composite cases, rather than simple cases. To *retrieve* the most similar strategies to the one used by the learner, appropriate similarity metrics are employed (see below). Once the most similar strategies are identified, they are used in a scaffolding mechanism that implements a form of *reuse* by taking this information into account along with other information, such as learner characteristics (e.g. knowledge level, spatial ability), completeness of solution and state within a task. The *reuse*, *revise* and *retain* steps are part of the knowledge base adaptation described in Section 4: simple cases are modified and then stored in a set of inefficient cases; new strategies are stored without modifications.

We use the term knowledge base adaptation in the sense that the knowledge base changes over time to adapt to new ways in which learners approach tasks - ways that could be either efficient or inefficient. This is referred to as 'adaptation to a changing environment' [9]. It is not, however, the same as adaptation in the CBR sense (case adaptation), which involves the *reuse* and *revise* processes [10]. These processes, though, are included in the acquisition of inefficient cases, and thus, case adaptation is partly present in our mechanism. The acquisition of new strategies corresponds to case-base maintenance in CBR terminology [8], as it involves adding a new case for which no similar case was found.

Knowledge Representation. Strategies for building constructions are represented as series of cases with relations between them. A case is defined as $C_i = \{F_i, RA_i, RC_i\}$, where C_i is the case and F_i is a set of attributes. RA_i is a set of relations between attributes and RC_i is a set of relations between C_i and other cases.

The set of attributes of C_i is defined as $F_i = \{\alpha_{i_1}, \alpha_{i_2}, \ldots, \alpha_{i_N}\}$. It includes three types: numeric, variables and binary. Numeric attributes correspond to the values in the property list and variables correspond to the type of those properties: number, T-box, expression with number(s) or expression with T-box(es). Binary attributes refer to the membership of a case to a strategy, defined as a $PartOfS$ function returning 1 if it belongs to the strategy and 0 otherwise; there are S binary attributes (S=number of strategies in the knowledge base).

The set of relations between attributes of a case C_i and attributes of other cases (including C_i) is represented as $RA_i = \{RA_{i_1}, RA_{i_2}, \ldots, RA_{i_M}\}$, where at least one of the attributes in each relation $RA_{i_m}, \forall m = \overline{1, M}$, is from F_i, the set of attributes of C_i. Two types are used: (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 value of the colouring property of the blue pattern in Fig. 1 is 5 times the iterations of the red pattern'. The set of relations between cases is represented as $RC_i = \{RC_{i_1}, RC_{i_2}, \ldots, RC_{i_P}\}$, where one of the cases in each relation $RC_{i_j}, \forall j = \overline{1, P}$ is C_i. Two time-relations are used: (a) $Prev$ relation indicates the previous case and (b) $Next$ relation indicates the next case, with respect to the current case.

A strategy is defined as $S_u = \{N_u(C), N_u(RA), N_u(RC)\}$, $u = \overline{1, r}$, where $N_u(C)$ is a set of cases, $N_u(RA)$ is a set of relation between attributes of cases and $N_u(RC)$ is a set of relations between cases. The structure of a strategy is displayed in Fig. 3.

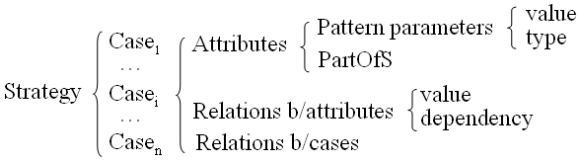

Fig. 3. Schematic structure of strategies.

Similarity Metrics. Strategy identification is based on scoring elements of the strategy followed by the learner according to the similarity of their attributes and their relations to strategies previously stored. The similarity metrics used

for cases and strategies are displayed in Fig. 4. 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 [11] using the function $\overline{F_1} = F_1/z$.

Similarity measures	Case	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 cases	$T_{IR} = \frac{\lvert RC_I \cap RC_R \rvert}{\lvert RC_I \cup RC_R \rvert}$	$F_4 = (\sum_{i=1}^{z} T_{I_i R_i}) / z$

I = Input Case/Strategy; R = Retrieved Case/Strategy
z = minimum number of cases of strategies I and R;
y = the number of cases in strategy R that have relations between attributes

Fig. 4. Similarity metrics for cases and strategies.

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. The metrics have been tested for several situations of pedagogical importance: identifying complete strategies, partial strategies, mixed strategies and non-symmetrical strategies. The similarity metrics were successful in identifying all these situations (details can be found in [4]).

4 Adaptation of the Knowledge Base

Our approach for adapting the knowledge-base of *eXpresser* includes acquiring inefficient simple cases, acquiring new strategies and deleting redundant ones. In this paper we focus on the first two, for which some examples from the 'stepping stones' task are displayed in Fig. 5; the constructions are broken down into individual components used by the students for ease of visualisation. These examples, with the adaptation rationale and mechanism are discussed below.

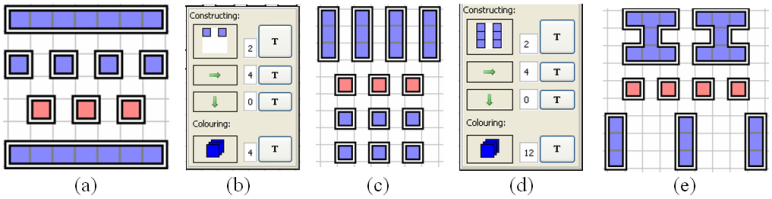

Fig. 5. (a)-(b) HParallel strategy with an inefficient component (blue middle row) and its property list; (c)-(d) VParallel strategy with an inefficient component (blue vertical bars) and its property list; (e) a new strategy

Acquiring inefficient simple cases. The goal of this mechanism is to identify parts of strategies constructed in inefficient ways and store them in a set of 'inefficient constructions', i.e. constructions that pose difficulties for learners in their process of generalisation. The set could be used for automatic generation of feedback or analysed by a researcher/teacher. The results could inform the design of better interventions, could be presented as a lesson learned to the scientific community of mathematics teachers/researchers, or even discussed in class (e.g. if an inefficient construction is often chosen by pupils of that class).

The construction in Fig. 5a illustrates an inefficient case 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 - see its property list in Fig. 5b. The efficient way to construct this component is one tile repeated 4 times or, to make it general, one tile repeated the number of red tiles plus 1. Both approaches, i.e. the efficient and the inefficient, 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 of the blue middle row (C_i) to the ones of the red middle row (C_j). If the learner relates value 2 of iterations of C_i to value 3 of iterations of C_j, i.e. value 2 is obtained by using the iterations 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. Another example of an inefficient case is given in Fig. 5c, with its property list presented in Fig. 5d. These inefficient cases are not related to mathematical misconceptions, but seem to be a consequence of the system's affordances combined with little experience of generalisation tasks, and come from learners' wish to make there patterns bigger without considering the generality of their approach. However, they are pedagogically important, offering learners the possibility to reflect on their actions in relation to generalisation.

Algorithm 1 Verification(*StrategiesCaseBase*, *InputStrategy*)

```
Find most similar strategy to InputStrategy from StrategiesCaseBase
StoredStrategy ← most similar strategy;
if similarity > θ then
   Find cases of InputStrategy that are not an exact match to any case of StoredStrategy
   for each case that is not an exact match do
      InputCase ← the case that is not an exact match
      Compare InputCase to all cases of the set of inefficient cases;
      if no exact match then
         Find the most similar case to InputCase from the cases of StoredStrategy
         StoredCase ← the most similar case
         if Conditions(StoredCase, InputCase) returns true then // see Alg. 2
            InefficientCaseAcquisition(StoredCase, InputCase) // see Alg. 3
         end if
      end if
   end for
end if
```

Algorithms 1, 2 and 3 illustrate how inefficient simple cases 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 cases; for each of these cases, several checks are performed (Alg. 2). Upon satisfactory results and if the cases are not already in the set of inefficient cases, they are then stored (Alg. 3). What is stored is actually a modification of the most similar (efficient) case, in which only the numerical

values of *iterations*, *move-right* and/or *move-down* are updated together with the value and dependency relations. These are the only modifications because, on one hand, they inform the way in which the pattern was built, including its non-generalisable relations, and, on the other hand, it is important to preserve the values of $PartOfS$ attributes, so the researcher/teacher knows in which strategies these can occur. The colouring attributes and the relation between cases are not important for this purpose and, therefore, they are not modified. This has also the advantage of being computationally cheaper.

Algorithm 2 Conditions($C1$, $C2$)

if ($MoveRight[C1] \neq 0$ and $Iterations[C1]*MoveRight[C1] = Iterations[C2]*MoveRight[C2]$) or ($MoveDown[C1] \neq 0$ and $Iterations[C1] * MoveDown[C1] = Iterations[C2] * MoveDown[C2]$) **then**
  return true
else
  return false
end if

Algorithm 3 InefficientCaseAcquisition($StoredCase$, $InputCase$)

$NewCase \leftarrow StoredCase$
for $i = 4$ to $v - 1$ **do** // attributes from *iterations* to *move-down*
  if value of attribute i of $NewCase$ different from that of $InputCase$ **then**
    replace value of attribute i of $NewCase$ with the one of $InputCase$
  end if
end for
for all relations between attributes **do** // value an dependency relations
  replace relations of $NewCase$ with the ones of $InputCase$
end for
add $NewCase$ to the set of inefficient cases

New strategy acquisition. The goal is to identify new strategies and store them for future use. New strategies could be added by the teacher or could be recognized 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 for the detected new strategy.

Fig. 5e illustrates the so-called "I strategy", due to its resemblance to letter I. When compared to all strategies, it is rightly most similar to the 'VParallel' one (see Fig. 2b), as some parts correspond to it. However, the similarity is low, suggesting it may be a new strategy. Without the adaptation mechanism, the learner modelling module will infer the learner is using the 'VParallel' strategy, but is still far from completion. This imprecise information is potentially damaging as it could, for example, lead to inappropriate system actions, e.g. providing confusing feedback by guiding the learner towards the 'VParallel' strategy. Conversely, the adaptation mechanism will prevent generating such confusing feedback, and storing the new strategy will enable appropriate feedback in the future - automatically or with input from the teacher/researcher.

Algorithms 4, 5 and 6 illustrate how an input strategy is identified and stored as a new strategy (composite case). If the similarity between the input strategy and the most similar strategy from the case-base 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 case-base (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 very similar to an existing one.

Algorithm 4 NewStrategyVerification(*StrategiesCaseBase*, *InputStrategy*)

```
Find most similar strategy to InputStrategy from the StrategiesCaseBase
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 cases 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 the strategies case-base
adjust values of PartOfS
```

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 simple cases in the *InputStrategy*. Good solutions are characterized by a relatively small number of simple cases; therefore, we propose for the value of θ_2 the maximum number of cases 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 cases and checks its correspondence to the rule solution of the task. For example, in the construction of Fig. 5c, 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 case-base and the $PartOfS$ values are adjusted.

5 Validation

The validation of our proposed mechanisms includes: (a) identifying the boundaries of how far a pattern can be modified and still be recognised as similar to its original; (b) correct identification of inefficient cases within these boundaries and (c) correct identification of new strategies. This low-level testing of the system shows how the adaptation of the knowledge-base and the learner modelling module function together to improve the performance of the system.

To this end, experiments have been conducted using real data produced from classroom use of *eXpresser* as well as artificial data that simulated situations observed in the classroom sessions. Simulated situations were based on varying parameters of models produced by learners in order to provide more data.

First, a preliminary experiment using classroom data was conducted to identify possible values for the threshold θ in Algorithm 1 and threshold θ_1 in Algorithm 4. Since our main aim was to test the adaptive modelling mechanism we decided not to seek optimal values for these thresholds, but only to find a good

enough value for each one. Two possibilities were quickly identified - 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 boundaries 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 limits of changes that a pattern can undergo without losing its structure. We used 34 artificial inefficient cases from two tasks: (a) 'stepping stones' that was defined earlier and (b) 'pond tiling' which requires to find the number of tiles needed to surround any rectangular pond. Fig. 6 illustrates the construction for the 'pond tiling' problem and two strategies frequently used by students to solve this task. Compared to 'stepping stones', pond tiling is more difficult, requiring the use of two variables (the width and height of the pond) rather than one.

From the 34 cases, 47% were from the 'stepping stones' task and 53% were from the 'pond tiling' task. Using these cases, the following boundaries 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.

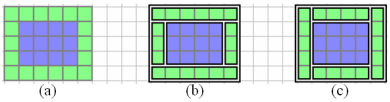

Fig. 6. (a) the construction for the 'pond tiling' task ; (b) 'I Strategy'; (c) 'H strategy'.

Experiment 2: correct identification of inefficient cases within the previously identified boundaries. From the 34 inefficient cases of *Experiment 1*, 13 were outside the identified boundaries and 21 were within. From the 21 cases within the boundaries, 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 cases, 52.48% had the overall similarity greater than 4.00 and 100% had the numeric similarity above 1.00. These results indicate that a small modification of a pattern can drastically affect the identification of the strategy the learner is following; hence almost half the cases have an overall similarity less than 4.00. The results obtained using the numeric similarity are much better and consistent with the fact the modifications are just numerical.

Experiment 3: correct identification of 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 knowledge base 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. As opposed to Experiment 2, the overall similarity performs better, being consistent with the fact that the overall similarity reflects better the resemblance with the stored strategies than the numeric similarity alone. Given the range that the overall similarity has, i.e. from 0 to 10, values below 3.50 indicate a very low similarity and therefore, rightly suggest that the learner's construction is considerably different from the ones in the knowledge base.

6 Conclusions

In this paper we presented research on modelling users' behaviour in *eXpresser*, an exploratory learning environment for mathematical generalisation. We adopted a case-based formulation of strategies learners follow when building constructions in *eXpresser* and employed similarity metrics to identify which strategy is used by each learner. Due to the open nature of the environment, however, not all strategies are known in advance. Moreover, learners use the system in inefficient ways that lead to difficulties in solving the given tasks. To overcome these problems, we developed an adaptive modelling mechanism to expand an initially small knowledge base, by identifying inefficient cases (i.e. cases that pose additional difficulty to the user's learning process) and new strategies. Future work includes using the information on inefficient cases and new strategies in an automatic way to either incorporate it in the feedback and/or inform the teachers and allow them to author feedback.

References

1. Piaget, J.: The Psychology of Intelligence. New York: Routledge (1950).
2. de Jong, T., van Joolingen, W.R.: Scientific discovery learning with computer simulations of conceptual domains. Review of Educational Research, 68, 179-202, (1998).
3. Kirschner, P., Sweller, J., Clark, R.E.: Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-based, Experiential and Inquiry-based Teaching. Ed. Psych., 41(2), 75–86 (2006).
4. Cocea, M., Magoulas, G.: Task-oriented Modeling of Learner Behaviour in Exploratory Learning for Mathematical Generalisation. In Proceedings of the 2nd ISEE workshop, AIED'09, 16–24 (2009).
5. Pearce, D., Mavrikis, M., Geraniou, E., Gutierrez, S.: Issues in the Design of an Environment to Support the Learning of Mathematical Generalisation. In Proceedings of EC-TEL08, 326–337 (2008).
6. Mason, J.: Generalisation and Algebra: Exploiting Children's Powers. In L.Haggarty (ed.), Aspects of Teaching Secondary Mathematics: Perspectives on Practice, 105–120 (2002).
7. Kaput, J.: Technology and Mathematics education. In D. Grouws (ed.) Handbook of Research on Mathematics Teaching and Learning, NY: Macmillan, 515–556 (1992).
8. Kolodner, J.L.: Case-Based Reasoning, Morgan Kaufmann Publishers, Inc.(1993).
9. Anguita D.: Smart adaptive systems: State of the art and future directions of research. In Proceedings of EUNITE 2001, 1–4 (2001).
10. Mitra, R., Basak, J.: Methods of case adaptation: A survey. International Journal of Intelligent Systems, 20(6), 627–645 (2005).
11. Aksoy, S., Haralick, R.M.: Feature Normalisation and Likelihood-based Similarity Measures for Image Retrieval. Pattern Recognition Letters 22, 563–582 (2001).