

Enhancing Modelling of Users' Strategies in Exploratory Learning through Case-base Maintenance

Mihaela Cocea, Sergio Gutierrez-Santos and George 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. Exploratory learning allows learners to solve a problem using different strategies. However, only a limited number of these strategies is known in advance and can be introduced by the designer and/or teacher. In previous work we have proposed a case-based knowledge representation, modelling the learner behaviour as simple cases and collection of cases (i.e. strategies). In this paper we enhance this approach through case-base maintenance with mechanisms that identify and store inefficient cases (i.e. cases that pose additional difficulty to the learning process), and enrich the case-base by adding new strategies.

Key words: case-based reasoning, case maintenance, exploratory learning environments, user modelling

1 Introduction

Educational research has shown there are several domains in which young students learn more when they have the possibility of engaging in meaningful activities by actively *constructing* entities to guide their learning, instead of passively accepting information [1]. Several initiatives have appeared in recent years to assist this active kind of learning processes. These software tools, in which learners are allowed to explore a broad set of possibilities and construct models with them, are called Exploratory Learning Environments (ELE). ELEs can arguably have a more positive impact on the user's learning than guided environments but, at the same time, an ELE without any support or guidance can actually hinder learning [2]. Therefore, there is a need to complement ELEs with some degree of intelligence to support the user, despite the open nature of these environments that makes it very challenging to do so because the possibilities of action are broad and usually unstructured.

To address this problem, we have proposed a learner modelling mechanism for monitoring learners' actions based on Case-Based Reasoning (CBR) that allows identification of different strategies that learners may use in solving a task [3]. An important problem remains, however: the knowledge about a task evolves over time - students may discover different ways of approaching the same task, rendering the case-base suboptimal for generating proper feedback. In this paper, we focus on the latter problem of enriching the knowledge in the case-base, i.e

case-base maintenance. This work is done using *eXpresser* [4], an exploratory learning environment for mathematical generalisation.

The rest of the paper is structured as follows. The next section briefly introduces *eXpresser* and the problem of mathematical generalisation. Section 3 describes the CBR cycle for *eXpresser*, the knowledge representation and the similarity metrics employed. Section 4 presents our proposed approach for case-base maintenance and, finally, Section 5 concludes the paper and presents some directions for future work.

2 Exploratory Learning for Mathematical Generalisation

Mathematical generalisation is at the core of mathematical thinking. Usually some generalisation tasks are given in the context of algebra, as “algebra is, in one sense, the language of generalisation of quantity. It provides experience of, and a language for, expressing generality, manipulating generality, and reasoning about generality” [5]. However, algebra is perceived as separate from what it represents [6] and students do not associate it with generalisation.

To address this issue, *eXpresser* [4] aims to link the visual with the algebraic-like representation of rules. It enables constructions of patterns, creating dependencies between them, naming properties of patterns and creating algebraic-like rules with either names or numbers. It targets pupils of 11 to 14 year-olds and it is meant for classroom rather than independent use.

Some screenshots are displayed in Fig. 1, illustrating the system, the *properties list* of a pattern (i.e. iterations, move-right, move-down, and colouring) that is dependent on another one and an example of a rule. The screenshot on the left displays two patterns: a red (darker colour) pattern, having five tiles and a green (lighter colour) one, having six tiles. In the property list of the green pattern displayed in the top right corner, it can be seen that this pattern depends on the red one by the fact that the number of iterations of green tiles is set to “the number of tiles of the red pattern plus one”. The second property in the list establishes the units for *move-right* on the horizontal axis - in the current case it is set to 2, which makes green tiles appear with gaps in between them; these are filled by red tiles, which also have *move-right* property set to 2. The following property sets the units for *moving down* on the vertical axis - in the current case it is set to 0. The last property establishes the number needed to colour all the tiles in the pattern; in the current case it is the same as the number of iterations in the pattern. However, if a pattern is a group of several tiles, this would not be true anymore; for example, if a pattern is a group of three tiles and is repeated/iterated five times, the number required to colour it would be three times five. The bottom right screenshot displays a simple example of a rule for the number of green tiles, which is the number of red tiles plus 1.

The construction in Fig. 1 and the rule in the bottom-right corner constitute one possible solution to the following generalisation problem: how many green tiles are required for any given number of red tiles in order to produce the displayed construction? Although for this particular problem the rule is unique, there are several ways to build the construction. For example, another way of

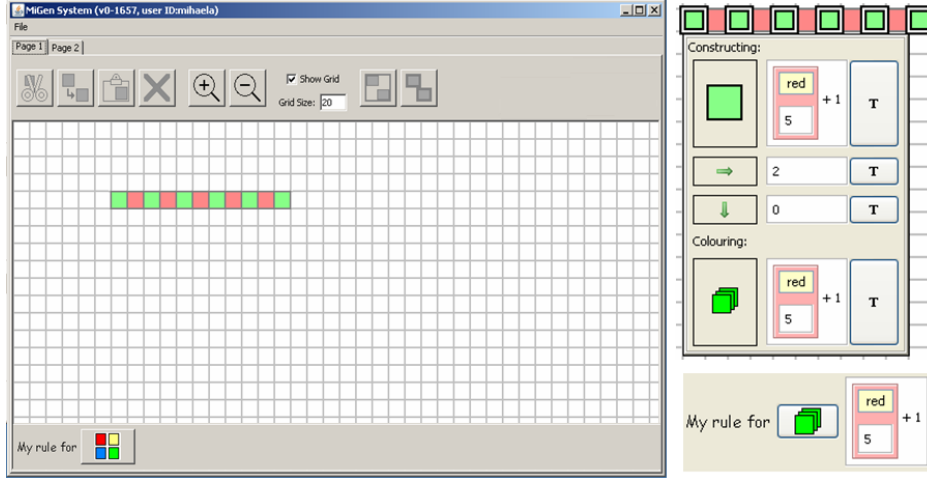


Fig. 1. *eXpresser* screenshots. The screenshot on the left includes a toolbar, an area for pattern construction and an area for defining rules; the main area has two patterns: one composed of 5 red (darker colour) tiles and another of 6 green (lighter colour) tiles. The screenshot on the top right shows the property list of the green pattern: iterate as many times as "red" plus 1, move 2 tiles right and 0 tiles down after every iteration, use "red + 1" units of green colour. The bottom right screenshot illustrates a rule for the number of green tiles.

constructing it would be to define a pattern as a group of a green and a red tile and repeat it five times (i.e. equal to the number of red tiles) along the horizontal axis and then add an extra green tile. Therefore, there are several *strategies* that one could follow when building a particular construction and the *components* of these strategies are *patterns* with certain *attributes* or properties (e.g. the ones defined in the property list) and linked with certain *relations*, such as the dependency relation illustrated in Fig. 1 (i.e. the number of green tiles depends on the number of red ones). In the following section, a formalisation for knowledge representation is presented that covers both the components of a strategy and the strategy as a whole.

3 CBR for Learner Modelling

In CBR [7] the knowledge is stored as cases, typically including the description of a problem and its solution. When a new problem is encountered, similar cases are retrieved and the solution is used or adapted from one or more of the most similar cases. The CBR cycle typically includes four processes [7]: (a) *Retrieve* cases that are similar to the current problem; (b) *Reuse* the cases (and adapt) them in order 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 the same problem has multiple solutions and it is important to identify which one is used by the learner. To address this for *eXpresser* each task has a case-base of solutions (i.e. strategies).

When a learner is building a construction, it is transformed into a sequence of simple cases (i.e. strategy) and compared with all the strategies in the case-base

for the particular task that the learner is working on (the case-base consists of strategies (composite cases), rather than simple cases). To *retrieve* the strategy or strategies that are most similar to the one used by the learner, appropriate similarity metrics are employed (see Section 3.2). Once the most similar strategy or strategies are identified, they are forwarded to a feedback module that implements a form of *reuse* by taking this information into account along with other information, like the characteristics of the learner (e.g. knowledge level, spacial ability), completeness of solution and state within a task.

The *revise* and *retain* steps are part of the maintenance process described in Section 4: simple cases are modified and then stored in a set of inefficient cases; new strategies are stored without modifications.

The following two subsections present the knowledge representation and the similarity metrics together with an example in the context of *eXpresser*.

3.1 Knowledge Representation

The case-based knowledge representation is part of our previous work [3], but we present it here for completeness. In our approach, strategies for building a construction are represented as a series of cases with certain relations between them.

Definition 1. A case is defined as $C_i = \{F_i, RA_i, RC_i\}$, where C_i represents 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 respectively.

Definition 2. The set of attributes of a given case C_i is defined as $F_i = \{\alpha_{i_1}, \alpha_{i_2}, \dots, \alpha_{i_N}\}$.

The set F_i includes three types of attributes: (a) numeric, (b) variables and (c) binary. Variables refer to different string values that an attribute can take, and binary attributes indicate whether a case can be considered in formulating a particular strategy or not. This is represented as a “part of strategy” function: $PartOfS_u : C_i \rightarrow \{0, 1\}$, $PartOfS_u = 1$ if $C_i \in S_u$ and $PartOfS_u = 0$ if $C_i \notin S_u$, where S_u represents a strategy and is defined further on. The set of attributes of a generic case for *eXpresser* is presented in Table 1. The first v attributes ($\alpha_{i_j}, j = \overline{1, v}$) are variables, the ones from $v + 1$ to w are numeric ($\alpha_{i_j}, j = \overline{v + 1, w}$) and the rest are binary ($\alpha_{i_j}, j = \overline{w + 1, N}$).

Definition 3. The set of relations between attributes of a given case C_i and attributes of other cases (as well as attributes of C_i) is represented as $RA_i = \{RA_{i_1}, RA_{i_2}, \dots, RA_{i_M}\}$, 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 of binary relations are used: (a) a *dependency relation* (D_{i_s}) is defined as $(\alpha_{i_k}, \alpha_{j_l}) \in D_{i_s} \Leftrightarrow \alpha_{i_k} = DEP(\alpha_{j_l})$, where $DEP : \alpha_{i_k} \rightarrow \alpha_{j_l}$ for attributes α_{i_k} and α_{j_l} that are variables of cases i and j (where $i = j$ or $i \neq j$), and means that α_{i_k} depends on (is built upon) α_{j_l} (if $i = j, k \neq l$ is a required condition to avoid circular dependencies) (e.g. the width type of a case is built upon the height type of the same case; the width type of a case is built upon

Table 1. The set of attributes (F_i) of a case.

Category	Name	Attribute	Possible Types/Values
Patterns	Colour	α_{i_1}	Red/Green/Blue/Yellow
properties	Width type	α_{i_2}	Number (n)/Icon variable (iv)/numeric expression (n_exp)/icon variable(s) expression (iv_exp)
	Height type	α_{i_3}	n /iv /n_exp /iv_exp
	\vdots	\vdots	\vdots
	Colour type	α_{i_v}	n /iv /n_exp /iv_exp
	Width value	$\alpha_{i_{v+1}}$	numeric value
	Height value	$\alpha_{i_{v+2}}$	numeric value
	\vdots	\vdots	\vdots
	Colour value	α_{i_w}	n /iv /n_exp /iv_exp
Group flag	isGroup	$\alpha_{i_{w+1}}$	0/1
Part of	$PartOfS_1$	$\alpha_{i_{w+2}}$	0/1
Strategy	$PartOfS_2$	$\alpha_{i_{w+3}}$	0/1
	\vdots	\vdots	\vdots
	$PartOfS_r$	α_{i_N}	0/1

the width type of another case, an so on); (b) a *value relation* (V_{i_s}) is defined as $(\alpha_{i_k}, \alpha_{j_l}) \in V_{i_s} \Leftrightarrow \alpha_{i_k} = f(\alpha_{j_l})$, where α_{i_k} and α_{j_l} are numeric attributes and f is a function and could have different forms depending on context (e.g. the height of a shape is two times its width; the width of a shape is three times the height of another shape, etc.). A case is considered *specific* when it does not have dependency relations and is considered *general* when it has at least one dependency relation.

Definition 4. *The set of relations between cases is represented as $RC_i = \{RC_{i_1}, RC_{i_2}, \dots, RC_{i_P}\}$, where one of the cases in each relation $RC_{i_j}, \forall j = \overline{1, P}$ is the current case (C_i).*

Two time-relations are used: (a) *Prev* relation indicates the previous case with respect to the current case: $(C_i, C_j) \in Prev$ if $t(C_j) < t(C_i)$ and (b) *Next* relation indicates the next case with respect to the current case: $(C_i, C_k) \in Next$ if $t(C_i) < t(C_k)$. Each case includes at most one of each of these two relations ($p \leq 2$).

Definition 5. *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.*

To illustrate how the knowledge representation and the way in which the identification mechanism operates, a task called “footpath”, typical in secondary school mathematics UK curriculum, is used, which requires to find the number of tiles that surround a pattern like the red one displayed in Fig. 1. There are several strategies for constructing the surrounding for that pattern; among them some are more desirable (see Fig. 2) than others, in the sense that they facilitate generalisation.

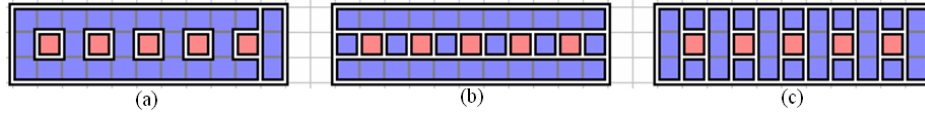


Fig. 2. Some desirable strategies for the footpath task: (a) forward C; (b) HParallel (horizontally parallel); (c) VParallel (vertically parallel).

Besides multiple possible constructions, there are several ways of reaching the same construction. A possible trajectory for the “forward C” strategy is illustrated in Fig. 3. The learner may start with the footpath (the red tiles) and then build a group of five blue tiles around the leftmost red tile having the form of a “C”; this group is iterated five times (the number of red tiles). Finally, a vertical pattern of three tiles is added at the right of the footpath. The details for most steps of this particular strategy are displayed in Table 2.

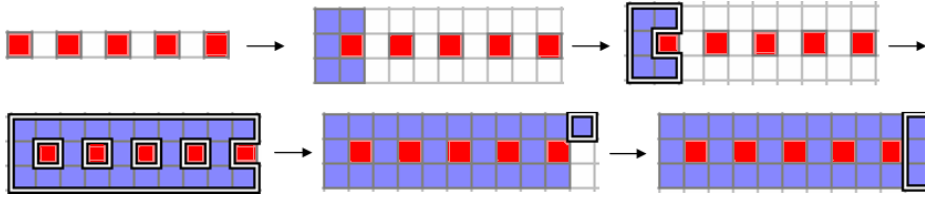


Fig. 3. Possible steps for “forward C” strategy.

Table 2. S_u definition for each step of the “Forward C” strategy.

S_u	$N_u(C)$	$N_u(RA)$	$N_u(RC)$
Step 1	C_1	-	-
Step 2	$C_1, C_2, C_3, C_4, C_5, C_6$	-	$Prev(C_{i+1}) = C_i$ for $i = \overline{1, 5}$ $Next(C_i) = C_{i+1}$ for $i = \overline{1, 5}$
Step 3	C_1, C_2	-	$Next(C_1) = C_2$ $Prev(C_2) = C_1$
Step 4	C_1, C_2	$\alpha_{23} = \alpha_{13}$ $\alpha_{23} = DEP(\alpha_{13})$	$Next(C_1) = C_2$ $Prev(C_2) = C_1$
Step 5	C_1, C_2, C_3	$\alpha_{23} = \alpha_{13}$ $\alpha_{23} = DEP(\alpha_{13})$	$Next(C_i) = C_{i+1}$ for $i = \overline{1, 2}$ $Prev(C_{i+1}) = C_i$ for $i = \overline{1, 2}$
Step 6	C_1, C_2, C_3	$\alpha_{23} = \alpha_{13}$ $\alpha_{23} = DEP(\alpha_{13})$	$Next(C_i) = C_{i+1}$ for $i = \overline{1, 2}$ $Prev(C_{i+1}) = C_i$ for $i = \overline{1, 2}$

The first step includes only one case: the red tiles pattern. After some intermediate steps, not illustrated here, the second step includes 6 cases, i.e. the red pattern and five single blue tiles, which are in a given order as expressed by the set of $Prev$ and $Next$ relations. In the third step, the 5 blue tiles are grouped in one pattern which now becomes C_2 ; consequently, at this point there are 2 successive cases. In the fourth step, the second case, i.e the group of 5 blue tiles, is repeated 5 times (the number of red tiles), so now there is also a value and a dependency relation. In the fifth step a new blue tile is added, becoming C_3 and in the sixth step this tile is iterated 3 times; in the last two steps, the relations between attributes and between cases are the same as in Step 4.

3.2 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. Thus, to identify components of a strategy, four similarity measures are defined:

1. Numeric attributes - Euclidean distance: $D_{IR} = \sqrt{\sum_{j=v+1}^w o_j \times (\alpha_{I_j} - \alpha_{R_j})^2}$ (I and R stand for input and retrieved cases, respectively).
2. Variables: $V_{IR} = \frac{\sum_{j=1}^v g(\alpha_{I_j}, \alpha_{R_j})}{v}$, where g is defined as: $g(\alpha_{I_j}, \alpha_{R_j}) = 1$ if $\alpha_{I_j} = \alpha_{R_j}$ and $g(\alpha_{I_j}, \alpha_{R_j}) = 0$ if $\alpha_{I_j} \neq \alpha_{R_j}$.
3. Relations between attributes - Jaccard's coefficient: $A_{IR} = \frac{|RA_I \cap RA_R|}{|RA_I \cup RA_R|}$. A_{IR} is the number of relations between attributes that the input and retrieved case have in common divided by the total number of relations between attributes of the two cases.
4. Relations between cases - Jaccard's coefficient: $B_{IR} = \frac{|RC_I \cap RC_R|}{|RC_I \cup RC_R|}$, where B_{IR} is the number of relations between cases that the input and retrieved case have in common divided by the total number of relations between cases of I and R .

The similarity metrics presented above were chosen for their appropriateness to the types of data. The Euclidian distance and the matching coefficient for variables are typical metrics for numeric and string data. The Jaccard coefficient was chosen for relations matching for its suitability to compute similarities between sets [8].

To identify the closest strategy to the one followed by a learner during construction, cumulative similarity measures are used for the four similarities: (a) Numeric attributes: as this metric has a reversed meaning compared to the other ones, i.e. a smaller number denotes greater similarity, the following function is used to bring it in line with the other three similarity measures, i.e. a greater number denotes greater similarity: $F_1 = z / \sum_{i=1}^z D_{I_i R_i}$ if $\sum_{i=1}^z D_{I_i R_i} \neq 0$ and $F_1 = z$ if $\sum_{i=1}^z D_{I_i R_i} = 0$, where z represents the minimum number of cases among the two compared strategies; (b) Variables: $F_2 = (\sum_{i=1}^z V_{I_i R_i}) / z$; (c) Relations between attributes: $F_3 = (\sum_{i=1}^z A_{I_i R_i}) / y$, where y represents the number of cases in the retrieved strategy that have relations between attributes; for example, "forward C" strategy has three cases and only one that has relations between attributes, i.e. $y = 1$.; (d) Relations between cases: $F_4 = (\sum_{i=1}^z B_{I_i R_i}) / z$.

The similarity between the current strategy and a stored strategy is defined as the sum of these four measures after they have been normalised as explained below. As the four similarity metrics have different ranges, normalisation is applied to have a common measurement scale, namely $[0, 1]$. This is done using linear scaling to unit range [9] by applying the following function: $\bar{x} = \frac{x-l}{u-l}$, where x is the value to be normalised, l is the lower bound and u is the upper bound for that particular value: (a) Numeric attributes: the range of F_1 is $[0, z]$; therefore the normalisation function is: $\bar{F}_1 = F_1 / z$; (b) Variables - the range of F_2 is $[0, 1]$, so no normalisation is needed; (c) Relations between attributes:

the range of F_3 is $[0, 0.5]$; therefore the normalisation function is: $\overline{F_3} = 2F_3$; (d) Relations between cases - same as the previous: $\overline{F_4} = 2F_4$.

After normalisation, weights are applied to the four measures to reflect the most important aspect of the construction: the structure. This is reflected most by the $\overline{F_1}$ metric; the $\overline{F_3}$ metric is also important as the structure of a construction is also reflected by the relations between attributes of component cases. The other two metrics ($\overline{F_2}$ and $\overline{F_4}$), although important for the generality of construction and the order of cases, respectively, have less impact on the structure. Considering these aspects, the aggregated similarity is computed using the following function: $Sim = 6 * \overline{F_1} + \overline{F_2} + 2 * \overline{F_3} + \overline{F_4}$.

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 [3]).

4 Case-base Maintenance

Our proposed approach for *eXpresser* case-base maintenance includes: acquiring inefficient simple cases, acquiring new strategies and deleting redundant ones. The focus is on the first two, but some discussion on the third is included. Fig. 4 includes examples from the footpath task introduced previously for the first two aspects (the constructions are expanded for ease of visualisation). These examples, with the maintenance rationale and mechanism, are discussed below.

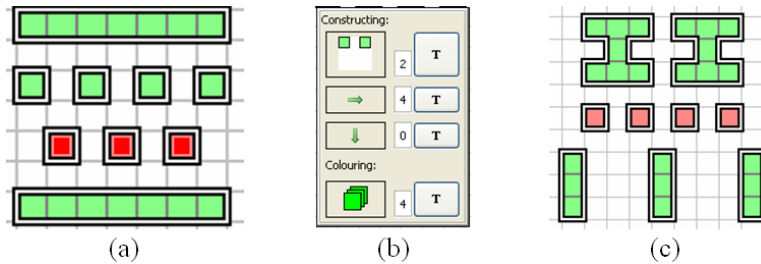


Fig. 4. (a) HParallel strategy with one inefficient component (green middle row) ; (b) property list of the inefficient component; (c) a new strategy

Acquiring inefficient simple cases. This could have several applications: (a) enabling targeted feedback about the inefficiency of certain parts of construction and (b) having a library of inefficient constructions produced by students that could be analysed further by a researcher or teacher. These findings 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, or even discussed further in class (e.g in the case of an inefficient construction that is frequently chosen by the pupils of that class).

The goal is to identify parts of strategies constructed in inefficient ways and store them in a set or library of ‘inefficient constructions’, i.e. constructions that pose difficulties to the process of generalisation. The construction in Fig. 4a illustrates an inefficient case within the “HParallel” strategy of the footpath

task: the middle bar of green 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. 4b. 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. These two ways of constructing the middle row of green 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 green tiles (C_i) to the ones used in the middle row of red tiles (C_j). If the learner would relate the value 2 of iterations of C_i to the value 3 of iterations of C_j , i.e. the value 2 is obtained by using the number of red tiles (3) minus 1, this would work only for a footpath of 3 tiles.

Algorithms 1, 2 and 3 illustrate how the 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, then they are stored (Alg. 3).

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

```

Find most similar strategy to InputStrategy from StrategiesCaseBase
StoredStrategy  $\leftarrow$  most similar strategy;
if similarity >  $\theta$  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  $\leftarrow$  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  $\leftarrow$  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

```

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 ← 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 and dependency relations
  replace relations of NewCase with the ones of InputCase
end for
add NewCase to the set of inefficient cases

```

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 has been built and its non-generalisable relations, and, on the other hand, it is important to preserve the values of *PartOfS* attributes, so the researcher or 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 (as this is computationally cheaper).

So far, we have identified two possibilities for setting the threshold θ in Algorithm 1: the minimum *overall similarity* encountered so far, plus an error margin ($4.56 + 0.5$) or value 1 for the *numerical similarity*. This small-scale numerical validation has been done using data from classroom sessions with *eXpresser*.

New case acquisition. The goal of this type of case-base maintenance is to identify completely 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.

Fig. 4c illustrates the so-called "I strategy" (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, as some parts correspond to it. However, the similarity is low, suggesting it may be a new strategy. Without the maintenance mechanism, the learner modelling module will consider that the learner is still far from a valid solution and the feedback module will guide the learner towards the most similar 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.

Algorithms 4, 5 and 6 illustrate the process by which an input strategy could be 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 decide to go ahead with storing the new strategy, even if it is very similar to an existing one in the database.

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

```

Find most similar strategy to InputStrategy from the StrategiesCaseBase
if similarity <  $\theta_1$  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 <  $\theta_2$  then
    if InputStrategy has relations between attributes then
      RelationVerification(InputStrategy) // verifies that the numeric relation cor-
responds 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 verifying 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. 4c, 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.

So far we identified two possibilities for the value of θ_1 : (a) the maximum overall similarity (3.26) plus an error margin (0.5) or (b) value 1 for the numeric similarity.

Deleting redundant strategies. Our envisaged approach for deleting redundant strategies is to regularly calculate statistics on the usage of the stored strategies and propose to the teachers to delete strategies that are not frequently used. However, this raises some problems depending on the existence of a central or

a local case-base. If the case-base is centralised and different teachers do not agree on the deletion of strategies, a mechanism for solving this conflict may be needed. One way to proceed would be to perform the deletion without asking the teachers. However, this may affect pupils that use those strategies and their teachers as no feedback would be given by the system, leading to an workload increase for the teachers. If the case-bases are local, the acquisition of a new case and its feedback will benefit only local pupils. Also, teachers at different locations may not benefit from the authoring of colleagues from other locations.

5 Conclusions

In this paper we presented a CBR knowledge representation approach for learning modelling together with the similarity metrics employed, and proposed an approach for case-base maintenance with the purpose of extending the case-base of *eXpresser*. Preliminary testing showed accurate identification of inefficient cases (i.e. cases that pose additional difficulty to the learning process) and of new strategies. Further work includes a wider testing of the proposed approach and the integration of the CBR component with the feedback module.

Acknowledgements

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. Piaget, J.: *The Psychology of Intelligence*. New York: Routledge (1950).
2. 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. *Educational Psychologist*, 41(2), 75–86 (2006).
3. 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).
4. Pearce D., Geraniou, E., Mavrikis, M., Gutierrez-Santos, S., Kahn, K.: 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).
5. Mason, J.: Generalisation and Algebra: Exploiting Children's Powers. In L.Haggarty (ed.), *Aspects of Teaching Secondary Mathematics: Perspectives on Practice*, Routledge Falmer and the Open University, 105-120 (2002).
6. Kaput, J.: Technology and Mathematics education. In D. Grouws (ed.) *Handbook of Research on Mathematics Teaching and Learning*, NY: Macmillan, 515-556 (1992).
7. Kolodner, J.L.: *Case-Based Reasoning*, Morgan Kaufmann Publishers, Inc.(1993).
8. Yin, Y. Yasuda, K.: Similarity coefficient methods applied to the cell formation problem: a comparative investigation. *Computers & Industrial Engineering* 48(3), 471-489 (2005)
9. Aksoy, S., Haralick, R.M.: Feature Normalisation and Likelihood-based Similarity Measures for Image Retrieval. *Pattern Recognition Letters* 22, 563-582 (2001).