Online Identification of Learner Problem Solving Strategies Using Pattern Recognition Methods

Ulrich Kiesmueller[1*], Sebastian Sossalla[1◇], Torsten Brinda[1*], Korbinian Riedhammer[2*]
[1]Computer Science Education and [2]Pattern Recognition Lab
Univ. Erlangen-Nuremberg, Martensstr. 3, Erlangen, Germany
*firstname.lastname@informatik.uni-erlangen.de,
◇Sebastian.Sossalla@mathe.stud.uni-erlangen.de

ABSTRACT

Learning and programming environments used in computer science education give feedback to the users by system messages. These are triggered by programming errors and give only "technical" hints without regard to the learners' problem solving process. To adapt the messages not only to the factual but also to the procedural knowledge of the learners, their problem solving strategies have to be identified automatically and in process. This article describes a way to achieve this with the help of pattern recognition methods. Using data from a study with 65 learners aged 12 to 13 using a learning environment for programming, a classification system based on hidden Markov models is trained and integrated in the very same environment. We discuss findings in that data and the performance of the automatic online identification, and present first results using the developed software in class.

Categories and Subject Descriptors

K.3.2 [**Computers And Education**]: Computer and Information Science Education—*computer science education*

General Terms

Algorithms, Measurement, Performance, Human Factors

Keywords

Computer Science Education, Secondary Education, Problem Solving Strategies, Algorithms, Tool-Based Analysis, Pattern Recognition

1. INTRODUCTION

Computer science tutors often employ learning and programming environments that provide visual programming such as Alice [5], Scratch [11] or Kara, the programmable ladybug [13]. In some federal states of Germany (e.g., Bavaria), the basics of algorithms are taught as early as in the 7th grade (age 12 to 13 years). During their first steps in programming using age-based learning and programming environments, learners solve small programming tasks. Although they had learned a certain approach from their teacher, many different problem solving strategies can be observed while the learners complete the tasks.

From a constructivist perspective, teachers should not ignore the learners' existing knowledge [3] but have to find out what the learners' concepts are and then guide the learner to an independent problem solution.

1.1 Problem Solving Strategies

Problem solving is finding a correct path through the problem space from an unrequested initial state to the requested goal state [12]. In the field of psychology, the problem solving process is divided in two not necessarily separated phases which are processed consecutively:

1. Construct (or evoke, if present in solvers' long time memory [12]) the problem space.
2. Search for the path through the problem space leading to the problem solution.

Real problem solving does not require the complete problem space to be built before looking for the correct path. The author described in [10] the range of problem solving strategies that are noticeably related to the learners' approach to solve the given tasks. There are two (pre-)structured problem solving methods – the *top down* and the *bottom up* strategy. For *top down*, learners search for the correct path through the problem space using a breadth first search. Vice versa, *bottom up* is to explore the problem space depth first. Learners using the *hill climbing* strategy neither structure the whole problem nor the sub problems but they have in mind how their program should work in general. The learners actively control the program execution to verify each step of their incremental solution. For that, they do not need a compulsory system error message. In contrast to the above, *trial and error* strategy means testing the whole solution approach and fixing errors in a rather random order.

1.2 Learning and Programming Environment

In the field of computer science, creativity is an important factor for arousing the learners interest, to motivate them and fostering this motivation [14]. In [15], Shneiderman describes criteria for creativity supporting software (in this case: learning and programming environments). They should provide immediate and helpful feedback to the learners, feature pain-free exploration and experimentation, use visual

ITiCSE'10, June 26–30, 2010, Bilkent, Ankara, Turkey.

Table 1: Categorization and grouping of the learner-system-interactions (LSI).

cat.	LSI
0	STATE_ADDED STATE_REMOVED
1	TRANSITION_TO_CHANGED
2	TRANSITION_ADDED TRANSITION_REMOVED
3	TRANSITION_INPUT_CHANGED STATE_SENSORS_SET
4	TRANSITION_COMMAND_ADDED TRANSITION_COMMAND_REMOVED
5	PLAYING
6	STOPPED
7	AMBIGUOUS_TRANSITION_EXCEPTION COMMAND_EXCEPTION NO_TRANSITION_EXCEPTION NO_START_STATE_EXCEPTION

programming and allow problem solving in a step-by-step manner. All these conditions are complied in learning and programming environments like Kara in the finite state automaton based version (FSMKara [13]). An example task could be to move the ladybug Kara along a certain path, eat something and come back. This procedure has to be modeled using a finite state automata that can use Kara's sensors. FSMKara is the basic instrument for our research as the learner-system-interactions are recorded for later use.

1.3 Categorizing Learner-System-Interactions

For our experiments, only a reasonable selection of learner-system-interactions (LSI) is recorded – those which are relevant to the problem solving process. The categorization is displayed in Table 1. As explained in [10], structuring the problem as such is represented by the interactions "editing" and "changing the final state machine" (cat. 0). Fine-structuring the problem in several sub-problems is equivalent to the interactions "creating" and "editing the conditions and branches" (cat. 1, 2 and 3). Iterations, branches and sequences of command sequences are between the fine structure and the solution of a sub-problem. These are modeled with the help of editing transitions. The final solution of a sub-problem is represented by the editing of command sequences (grouped in cat. 4). Additional LSI are the program executions ("play", cat. 5) and terminations ("stop", cat. 6) in order to test the (partial) correctness of the solution. Category 7 accumulates the system error messages. The above results in a total of eight LSI categories.

This article is structured as follows. After a description of the data acquired using the programming environment FSMKara, we give a brief insight to the observed problem solving strategies. In Section 3, we describe the pattern recognition method used to automatically identify these problem solving strategies in process, i.e., while the learner actually interacts with the system. Section 4 describes a first evaluation using the proposed method in class. In Section 5, we analyze the results using the proposed method and draw possible conclusions. We end with an outlook on future work presented in Section 6.

2. DATA

2.1 Observed Learners' Strategies

The above introduced learning and programming environment FSMKara was extended by a module logging all LSI, either to a file or for direct use within the software (see Section 3.5 and Figure 4). Exploiting that, we conducted a study with 65 learners at German grammar schools, aged 12 to 13 years, as described in [10]. Each learner works on average on two tasks during the study. In this setup, about 13000 LSI were recorded in 188 sessions. Using the log files and additional data from "thinking aloud" and guided interviews, the four learning strategies as described in Section 1.1 could be identified throughout the sessions. For the FSMKara task they can be described as follows.

- **Top down** – The problem is structured as a whole. Thereby, the learners add and edit states and branches, subsequently editing Kara's sensors (fine structuring the problem). Last, they fill in commands to solve the problem.
- **Bottom up** – The learners create and edit only one branch and subsequently fill in commands to solve this sub-problem. They repeat this process until the last sub-problem is solved.
- **Hill climbing** – The learners focus on a single step of the program. With a general concept in mind, they subsequently start the program execution and actively stop the execution as soon as they realize whether the currently tackled sub-problem is either correct or incorrect. They correct their mistakes if necessary, or proceed to the next step. This procedure is repeated until the final solution is found.
- **Trial and error** – The learners focus on only one step of the program and start the program after every single step and wait for a system error message. They try to correct their mistakes and repeat this process. That way, they try to find the correct solution.

2.2 Annotation

While verifying the existence of the above problem solving strategies, the log files were annotated to mark the occurrences of the strategies throughout the sessions. As a result, instances of the specific pattern solving strategies could be represented as extracts of the log file in an XML-like manner. Figure 1 shows an example extract from the log files representing an instance of the "bottom up" pattern. For the completeness of the annotation, all interactions between instances of the defined four strategies are accumulated in instances of an "unidentified" pattern.

```
<HMMBU>
12:04:43   Kara and... (easy)   TRANS_ADDED
12:04:43   Kara and... (easy)   TRANS_INPUT_CHANGED
12:04:44   Kara and... (easy)   TRANS_INPUT_CHANGED
12:04:45   Kara and... (easy)   TRANS_INPUT_CHANGED
12:04:49   Kara and... (easy)   TRANS_COMMAND_ADD
</HMMBU>
```

Figure 1: Instance of a labeled "bottom up" pattern in an XML-like format showing the time stamps, the current task and the learners interaction.

```
2; 3; 3; 3;
3; 3; 4; 4; 4; 4; 4;
0; 2; 3; 3; 3; 4;
```

Figure 2: Example LSI sequences for the "bottom up" pattern.

In a next step, these XML-like instances were reduced to sequences of LSI and grouped by problem solving strategy in order to get an idea how the strategies are expressed in terms of sequences of LSI. This is necessary for the later automatic identification which will be introduced in the next section. Figure 2 shows three instances of the "bottom up" pattern. Note that the patterns have different lengths and have a similar but possibly different structure.

2.3 Statistical Analysis

We now observe two important things. First, a sequence of LSI can be classified as one of the four problem solving strategies or to the "undefined" pattern. Second, each learner seems to use different strategies, i.e., learners change the way how they approach the problem. This confirms findings in [7], that there are general problem solving strategies but each learner applies an individual method using these. Equivalent to this, there are different patterns identifiable in the chronology of the LSI during the learners work on a single task. The probability to which pattern the learners' interactions change is variable. Table 2 shows the probabilities for every possible transition between two patterns as counted in the acquired data. It is remarkable that learners whose chronology of LSI show a special pattern in problem solving, it is very likely that the same pattern is applied more than one time (see diagonal elements of Table 2) – with the exception that the "top down" pattern is not repeated but often changed to "bottom up" (for further discussion of this fact see Section 5). At least, when a learner starts a new solution attempt he will use the same strategy as used in the first place, which is often further repeated. This indicates an individual preferred strategy.

3. METHOD

3.1 Introduction

The manual annotation and analysis of the data confirmed the four problem solving strategies ("bottom up", "top down", "hill climbing", and "trial and error") which are made up by categorized LSI sequences coded by symbols 0-7. To allow for sequences of LSI that cannot be associated with any of these four classes, a garbage strategy "unidentified" is introduced to account for the rest. Our task is now to automatically identify the strategies used by the learner given the history of LSI.

Table 2: Transition matrix between patterns. Read "transition from row to column strategy". TD: "top down", BU: "bottom up", HC: "hill climbing", TE: "trial and error", UI: "unidentified". The UI pattern has to be considered separately as it usually appears as a transition between two other strategies.

	TD	**BU**	**HC**	**TE**	**UI**
start	0.127	0.530	0.052	0.030	0.261
TD	0.017	***0.414***	0.103	0.172	0.275
BU	0.016	**0.518**	0.069	0.153	0.216
HC	0.011	0.156	**0.218**	0.117	0.413
TE	0.003	0.117	0.075	**0.482**	0.270
UI	0.007	0.094	0.169	0.135	**0.397**

This is a fundamental pattern recognition problem which comes in many flavors, the most widely known is continuous spoken digit recognition. Here, the computer needs to recognize the digits spoken by the speaker (the learners' applied strategies), given a spoken utterance (the observed sequence of LSI). Similarly to the "unidentified" strategy, digit recognition usually makes use of a *silence* or *noise* class to account for sound between two digits.

3.2 Hidden Markov Models

The key part of a successful model is to allow for variations in the sequences, distinguishing it from strict sequence matching. This is necessary as the observed sequences may vary – for speech recognition, the speaker or the microphone may change (resulting in different sound), for problem solving strategy identification, similar LSI sequences may be associated with the same strategy.

To overcome these variations, the pattern recognition community successfully used statistical automata, i.e., automata that change states according to a certain probability. Though these exists in many variations, the most popular are so-called hidden Markov models (HMM) [4] that can model temporal statistical dependencies[1].

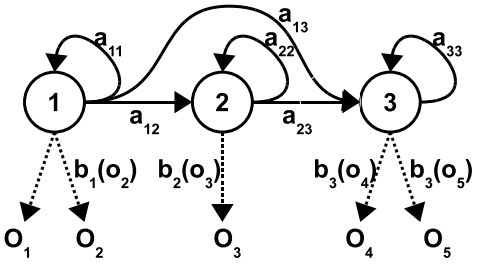

Figure 3: A left-to-right hidden Markov model with transition probabilities a and output distributions $b(\cdot)$.

So how do HMM work? In general, an HMM is an automata made up by states and transitions (similar to the one shown in Figure 3). In contrast to a regular automata that has defined transitions for input symbols, HMM transitions depend on the *transition probability* a_{ij} from state i to state j and the *output distribution* $b_j(\mathrm{O}_t)$ providing the probability that the current symbol O_t is emitted of state j. Additionally, the model may have *entrance probabilities* π instead of a defined start state. Given these parameters and a certain observation sequence, one can compute the likelihood that the model actually *generated* the observation.

Without going into details, the *training* of the models, i.e., estimating the transition and output probabilities to match a certain pattern, can be done for example using the

[1]Usually, a first order Markov assumption is made, i.e., the probability of a certain transition is dependent on the next input token and the current state

iterative Baum-Welch algorithm [2]. Given training instances, in this case LSI sequences associated with a certain problem solving strategy, initial model parameters are improved step by step to increase the likelihood that the model generated these examples. This optimization is based on the *expectation maximization* principle [6] rooted in information theory. Interestingly, though the optimization only finds a local optimum, random or uniform initializations of the parameters usually yield a good performance. These and other training algorithms can be found in the literature (e.g., [9]).

3.3 Decoding

Up to now, we trained one HMM for each strategy which can be used to determine the most likely strategy given a fixed sequence of LSI. Back to continuous digit recognition this means, we have one model for each digit. However, from Section 2 we know that the LSI sequence can contain on the one hand the "unidentified" pattern and on the other hand multiple occurrences of all strategy patterns. Recalling the digit recognition example, the audio stream can contain several digits and noise events.

The process of finding the sequence of HMM (resp. digits, strategy patterns) in the whole observation sequence (resp. audio signal, overall LSI sequence) is called *decoding*. Basically, we search the most likely *HMM sequence* (and thus the applied problem solving strategies) matching the whole LSI sequence. This search problem can be efficiently solved using the Viterbi algorithm [16] that produces the most likely *state* sequence q^* given an HMM and an observation sequence. The algorithm is based on dynamic programming and uses the stack variables $\vartheta_t(j)$ and $\psi_t(j)$ which are computed in an iterative way. The index t denotes the LSI within the sequence and j denotes the state index.

- **Initialization**

$$\forall_j \quad \vartheta_1(j) = \pi_j b_j(O_1) \;\; ; \;\; \psi_1(j) = 0$$

- **Recursion**

$$\begin{aligned} \forall_j \quad \vartheta_t(j) &= \max_i(\vartheta_{t-1}(i)a_{ij})b_j(O_t) \\ \forall_j \quad \psi_t(j) &= \text{argmax}_i \vartheta_{t-1}(i)a_{ij} \end{aligned}$$

- **Termination**

$$q_T^* = \text{argmax}_j \vartheta_T(j)$$

- **Backtracking** The optimal sequence is given by

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \;\; ; \;\; t \in \{(T-1), \ldots, 1\} \quad .$$

3.4 Putting Things Together

For our experiments, we use a limited topology allowing only left-to-right transitions (i.e., $a_{ij} = 0$ for $i > j$ within strategy models) to account for the temporal structure of the problem solving strategies. An example is shown in Figure 3.

While the "trial and error" and "unidentified" models only contain three states, the other patterns are modeled by five states with regard to their variety of training samples.

From a uniform distribution and transition initialization, we used the JAHMM[2] toolkit to train the strategy models with the annotated data described in Section 2.

[2]`http://code.google.com/p/jahmm/`

For the decoding, the five models were combined to one large HMM, introducing additional transition probabilities at the beginning and end of each strategy model to allow for any sequence of strategy patterns. The entrance probabilities are constrained to only allow initial states of strategy modules. This enables the LSI sequence to start with any strategy. They were set to the probabilities observed in the data (cf. Table 2). For a given LSI sequence of arbitrary length, the Viterbi algorithm is applied using this combined HMM. The optimal state sequence within the combined HMM can be translated into the sequence of strategy models and thus into the sequence of problem solving strategies applied by the learner.

3.5 IdentiKara

The whole software architecture is depicted in Figure 4. The original FSMKara software is extended by the TrackingKara module that receives all interaction events from the main program using an observer pattern. TrackingKara can now either just record the data to log files (for later analysis), or pass them to the integrated IdentiKara module, that uses the trained HMM and the above introduced Viterbi algorithm to determine the past and current problem solving strategies. Using manually labeled log files and the JAHMM library, the HMM can be trained (or updated in case of existing models).

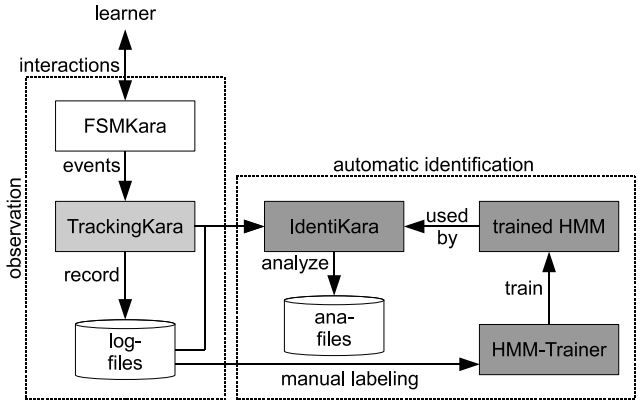

Figure 4: Software architecture including FSMKara, TrackingKara and IdentiKara

4. FIRST EVALUATION

IdentiKara was applied in a first study with a group of 15 learners at a German grammar school. They solved the same tasks as in the studies mentioned in Section 2. The resulting data written by TrackingKara and IdentiKara was saved in folders marked by the number of their computer. Provided with a seating plan showing the students and computers in the classroom, their teacher was interviewed concerning their behavior in lessons especially during problem solving. Considering all these statements, the preferred problem solving strategy of each learner was inferred by the teacher. These results were conform in 93% with the problem solving strategies identified by IdentiKara. To confirm these results, we currently run a study with a larger sample size (see also Section 6).

5. CONCLUSIONS

The applied pattern recognition methods allow for the automatic identification of learners' problem solving strategies in process. Not only the individual patterns as described in Section 2.1 but also certain sequences of patterns are identified. These sequences indicate different problem solving strategies used by the particular learner. The rows of Table 2 show the probability that learners change from one strategy pattern to another. One can notice that the top down pattern is rarely repeated (only in 1.7% of the cases). This can be explained by the comparing the structure of this pattern to the others. All other patterns describe single steps during the problem solving process. More than one of them are needed to reach a complete solution of a problem. The "top down" strategy is typically applied just once. The high probability (41.4%) of switching from "top down" to "bottom up" reflects the fact that this strategy is hardly to be maintained even by programming experts like described in [8]. Another interesting fact is that most learners at the age of 12 to 13 years prefer the bottom up strategy.

5.1 Usage For Individual System Feedback

An evaluation module embedded in the research software provides the automated assessment oft the learners' solution attempts. Concerning the quality of the learners' solution attempts the automatically identified problem solving strategy can be used to create individual system messages for every combination of quality and strategy. This feedback is adapted to the learners' problem solving strategy and encourages them finding the solution by themselves without additional help. Learners will get these messages not only in case of programming errors but also on demand.

5.2 Transfer to Other Learning Environments

For the application with learning and programming environments which are based on other programming paradigms the reasonable selection of learner-system-interactions mentioned in Section 1.3 have to be adapted. A promising option is to use the control structures of algorithms (sequence, iteration, conditional branch). Additionally starting/stopping program runs for testing the solution attempt and the system (error-)messages during run-time of tests have to be recorded. The categories of interactions furthermore will be "structuring on the whole", "fine structuring", "solving a subproblem", "testing the program", "getting feedback". The HMM and the pattern recognition algorithms will be the same independent of the choice of the learning and programming environment. Only the models have to be trained one time like described in Section 3. To realize the automated assessment the test cases have to be adapted additionally.

6. OUTLOOK

The attribution of the patterns' sequences to the individual problem solving strategies was generated and discussed by experts (teachers of computer science). An additional validation requires a different way of obtaining information about the learners' problem solving strategies than observing the problem solving process step by step as it is done by the research software described in this article. A good idea is to get information about the learners' problem solving strategy with the help of questionnaires which the learners have to fill in is described in [10]. The questionnaires are based on the fundamental concepts of Ajzen's and Fishbeins's Theory of Reasoned Action and Theory of Planned Behavior [1]. In this way information about the learners' problem solving strategies can be used without considering certain steps during the problem solving process.

7. REFERENCES

[1] I. Ajzen and M. Fishbein. *Belief, attitude, intention, and behavior: An introduction to theory and research*. Addison-Wesley, Reading, MA, 1975.

[2] L. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist.*, 41:164–171, 1970.

[3] M. Ben-Ari. Constructivism in computer science education. *SIGCSE Bull.*, 30(1):257–261, 1998.

[4] K. Chung. *Markov Chains with Stationary Transition Probabilities*. Springer, 1967.

[5] M. J. Conway. *Alice: Interactive 3D Scripting for Novices*. PhD thesis, University of Virginia, Charlottesville, VA, May 1998.

[6] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[7] R. M. Felder and L. K. Silverman. Learning styles and teaching styles in engineering education: Author's Preface – June 2002. *Engineering Education*, 78(7):674–681, 1988.

[8] J.-M. Hoc. *Psychology of programming*. Computers and people series. Academic, London, 1990.

[9] B. Juang and L. Rabiner. Hidden Markov Models for Speech Recognition. *Technometrics*, 33(3):251–272, 1991.

[10] U. Kiesmüller. Diagnosing Learners' Problem Solving Strategies Using Learning Environments with Algorithmic Problems in Secondary Education. *Trans. Comput. Educ.*, 9(3):1–26, 2009.

[11] J. Maloney, L. Burd, Y. Kafai, N. Rusk, B. Silverman, and M. Resnick. Scratch: A Sneak Preview. In *Proc. IEEE Int'l Conference on Creating, Connecting and Collaborating through Computing (C5)*, pages 104–109, 2004.

[12] A. Newell and H. A. Simon. *Human problem solving*. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[13] R. Reichert. *Theory of computation as a vehicle for teaching fundamental concepts of computer science*. PhD thesis, ETH Zürich, 2003.

[14] R. Romeike. Three Drivers for Creativity in Computer Science Education. In D. Benzie and M. Iding, editors, *Proc. of the IFIP-Conference on Informatics, Mathematics and ICT: A Golden Triangle*, 2007.

[15] B. Shneiderman. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM*, 50(12):20–32, 2007.

[16] A. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Trans. on Information Theory*, 13:260–269, 1967.