APE: Learning User's Habits to Automate Repetitive Tasks

Jean-David Ruvini
LIRMM – University of Montpellier
161 rue Ada – 34392 Montpellier - France
+ 33 4 67 41 86 13
ruvini@lirmm.fr

Christophe Dony
LIRMM – University of Montpellier
161 rue Ada – 34392 Montpellier - France
+ 33 4 67 41 85 33
dony@lirmm.fr

ABSTRACT

The APE (Adaptive Programming Environment) project focuses on applying Machine Learning techniques to embed a software assistant into the VisualWorks Smalltalk interactive programming environment. The assistant is able to learn user's habits and to automatically suggest to perform repetitive tasks on his behalf. This paper describes our assistant and focuses more particularly on the learning issue. It explains why state-of-the-art Machine Learning algorithms fail to provide an efficient solution for learning user's habits, and shows, through experiments on real data that a new algorithm we have designed for this learning task, achieves better results than related algorithms.

Keywords

Learning Interface Agents, Programming by Demonstration, Machine Learning, Interactive Programming Environments.

1. INTRODUCTION

Users of interactive environments waste a lot of time performing similar and repetitive tasks. To decrease the burden of entering repetitive sequences of commands, interactive tools generally offer macro or script languages to allow the user to write a program that can be later on invoked to perform a sequence of commands automatically. This approach has two limitations:

1. Writing a program require programming knowledge that many users do not have.
2. Writing or invoking a program requires efforts from the user and disrupts the user's work-flow.

Two kinds of solution have been proposed to overcome these limitations. Originally inspired by macro recorders, Programming by demonstration (PbD) [3] systems let the user demonstrate what the task to automate should do and create a program (containing variables, iterative loops or conditional branches) from observing this demonstration. They offer an efficient solution to the first problem because the user does not have to write code, but they require efforts from the user to demonstrate the task to automate or to invoke the created program (second limitation).

Learning Interface Agents (LIA) [5] learn correlations between situations the user encounters and the corresponding actions he performs, and assist the user by suggesting to perform automatically some part of user's work. For instance, CAP [6], an assistant for managing meeting calendars, suggests default values regarding meeting duration, location, time and day-of-week. Maes's assistants [5] advise users for some application specific operations like managing mails, scheduling meetings or selecting articles in news. ClipBoard [8], an interface for Unix, predicts the next command the user is going to perform. WebWatcher [1], an assistant for the world wide web, suggests links of interest to the user. OpenSesame! [2] runs in background on Macintosh, and learns repetitive tasks in opening and closing files or applications, emptying trash, rebuilding desktop. Although LIA provide a solution for the second limitation (repetitive actions are detected and suggested), existing assistants do not create programs and suggest only single actions and not sequences of actions.

Eager [3] can be seen as one of the first attempts to bring together PbD systems capabilities to create programs, and LIA facilities to predict user's actions. When Eager detects two consecutive repetitions in the last user's actions, it assumes they are the first two iterations of a loop, and proposes to complete the loop (until "a condition" is satisfied or following some typical patterns like linear sequences of integers or days of the week). Eager overcomes the two limitations identified above: it is able to replay sequences of actions (loops) but does not require the user to program, and it detects and suggests repetitive tasks without any user's intervention. However, it has a limitation: it detects repetitive tasks and makes suggestions only when repetitions are consecutive.

The APE project takes on Eager idea to bring together PbD and LIA, but focuses particularly on designing an assistant able to replay repetitive sequences of actions even when they are not consecutive. To achieve this goal, APE employs Machine Learning techniques to learn user's habits. This paper reports research conducted to address a key issue of this learning task: designing an assistant which makes "the right suggestion at the right moment" and does not constantly bother the user with incorrect suggestions. APE is integrated into the VisualWorks 3.0 Smalltalk ObjectShare programming environment. It can be downloaded at `http://www.lirmm.fr/~ruvini/ape`.

In the following we present APE. We explain what makes learning user's habits difficult, why existing Machine Learning algorithms do not have the potential for this learning task. We briefly present a new algorithm we have designed for this task and we compare its results when applied to real data, with C4.5 [10]. We finally give perspectives for future research.

2. GENERAL PRESENTATION

APE is made of three software agents, an Observer, an Apprentice and an Assistant, working simultaneously in the background without any user intervention. Table 1 defines our terminology.

Action : intervention of the user on the environment, e.g. window management, menu item selection, button pressing, entering text, etc. An action is parameterized by, among other things, the tool (Browser, Debugger, etc.) on which it has been performed.

Trace : history of user's actions.

Task : sequence of actions of the trace.

Repetitive task : task occurring several times in the trace.

Situation : sequence of actions of the trace of a given size n, n being a parameter of the learning algorithm.

Situation pattern : regular expression matching one or more situations.

Habit : pair "situation pattern - repetitive task" such that the situation pattern matches all situations preceding the repetitive task.

Table 1 : Basic definitions

2.1 The Observer

The Observer monitors user's *actions*, reifies them into dedicated Smalltalk objects and stores them into the *trace*. The Observer also sends messages in background to the Apprentice and to the Assistant to inform them that the user has performed a new action.

2.2 The Apprentice

The Apprentice uses the trace to detect *repetitive tasks*, to learn *situation patterns* in which they are performed, and thus to build a set of *habits*. It detects repetitive tasks of the trace using the classical text searching algorithm KMR [4]. It is able to detect repetitive sequences of actions as well as correction of repetitive programming errors or writing of repetitive pieces of code. It uses a Machine Learning algorithm to learn situation patterns. Let A_1, A_2, A_3 and lowercase letters from a to d denote actions, let α and β denote action parameters and let "." be a special character (a wildcard) that matches any single action or action parameter. It is able to learn 3 kinds of situation patterns:

- *Unordered*: the order in which actions are performed does not matter; in such cases, situations such as $A_1A_3A_2$ and $A_2A_1A_3$ can be characterized by the situation pattern $\{A_1,A_2,A_3\}$.
- *With wildcards*: the situations $A_1abA_2(\alpha)$ and $A_1cdA_2(\beta)$ can be characterized by the situation pattern "$A_1..A_2(.)$". The number of wildcards is not limited.
- *Unordered with wildcards*: $A_1A_2(\alpha)A_3$ and $A_2(\beta)A_3A_1$ can be characterized by the situation pattern $\{A_1,A_2(.),A_3\}$.

2.3 The Assistant

The Assistant exploits what has been learned to automatically propose to the user, when appropriate, sequences of actions that he might want to perform again. More precisely, when the Assistant detects that the last user's actions match one or several learned situation patterns, it displays in the Assistant window the repetitive tasks of the corresponding habits (see Figure 1). The user can ignore that window or ignore these suggestions (non-obtrusive behavior) or mouse-click on one of them to automatically perform the related actions. The number of suggestions the Assistant can make is limited to four because browsing suggestions puts a workload on the user.

2.4 ILLUSTRATIVE EXAMPLE

Figure 1 shows two snapshots of VisualWorks screens, both including an Assistant window (labeled "Assistant") and the main APE window (labeled "Ape Agents"). A user is testing his application: a multi-process simulation of the classical ``n-queens'' problem. In the ``situation before firing a habit'' (top snapshot), the user selects a Smalltalk expression (arrow 1) in the "Workspace window" whose evaluation (arrow 2) raises an exception leading to the opening of an "Exception" window. Because, this situation matches a situation pattern learned by the Apprentice, the Assistant offers to perform the related repetitive task: "open, move, resize a debugger and select stack index 5". This repetitive task is exactly what the user intents to do and he mouse-clicks on the proposition (arrow 3), resulting in a debugger displaying a user's method ("situation after a habit has been fired" - bottom snapshot). In this example, the Assistant has saved five actions to the user.

3. THE LEARNING TASK

3.1 What Makes The Problem Difficult

Four requirements have conducted the choice of the algorithm used to learn situation patterns. These requirements are :

R1 Low training time : let us call *training time* the time it takes the algorithm to learn situation patterns. The training time must be very low to allow the Apprentice to learn habits related to small sections of the trace (the corresponding situation patterns have to be learned very rapidly from a few situations), as well as habits appearing on very long traces and reflecting the general user's habits.

R2 Low prediction time : let us call *prediction time* the time it takes to the Assistant to inspect the set of learned habits and to make a suggestion. Obviously, it must be very low.

R3 User intelligible situation patterns : this is not critical but would allow the user to inspect or edit the learned habits.

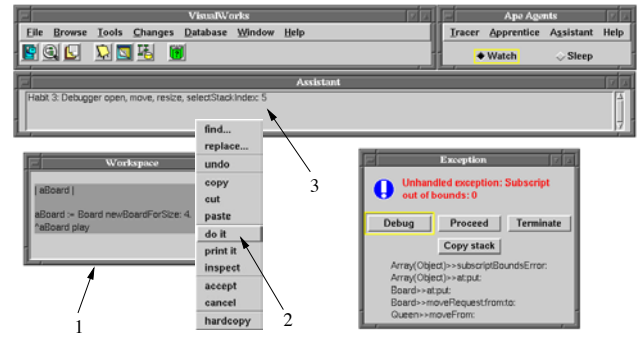
R4 Low error rate and low generalization : an assistant that continuously bothers the user with incorrect suggestions would be useless because the user would rapidly ignore it. Our Assistant has to minimize the number of incorrect suggestions. This means two things: (1) to minimize the number of incorrect suggestions when the user is about to perform a repetitive task (i.e. suggesting the wrong repetitive task) and (2) to minimize the number of suggestions when he is not about to perform a repetitive task (no suggestion should be done). To make few error in case (1) the algorithm must have a low error rate. In case (2), if some learned situation patterns match the last user's actions, it will make a suggestion. This may occur if the situation patterns are too general. As a consequence the Machine Learning algorithm employed has, on one hand, to generalize as far as needed to have a low error rate (case (1)), and on the other hand, to generalize as little as possible to avoid too general situation patterns (case (2)).

3.2 Existing Algorithms

Various kinds of algorithms have been proposed in the field of Machine Learning. Concept learning and neural networks algorithms do not meet requirement R1 (because in our application the number of possible actions and possible values for the parameters of these actions are large), instance-based algorithms do not meet requirement R2, instance-based, statistical and neural networks algorithms do not meet requirement R3. None of the existing Machine Learning algorithms have been designed to generalize as little as possible (requirement R4). However, concept learning and decision trees algorithms are more suited for learning user's habits. Eligible kinds of algorithms are decision trees because they miss only one requirement. They have notably been used in CAP [6]. The state-of-the-art decision trees learning algorithm is C4.5 [10]. Our tests have confirmed (see section 4) that C4.5 although fast (incremental versions exist) and having a low error rate, produces too general situation pattern, leading our Assistant to make too incorrect suggestions.

Situation before firing a habit

The Assistant proposes to open a debugger...



Situation after a habit has been fired

The user mouse-clicked on the proposition

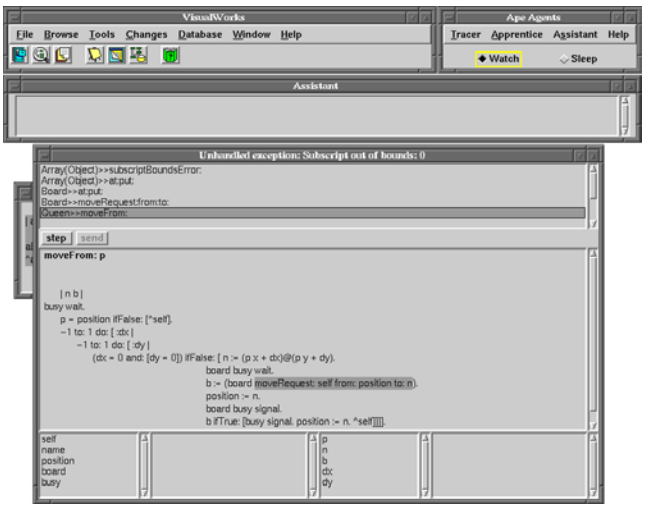


Figure 1

3.3 Overview of IDHYS

IDHYS [11], our new concept learning algorithm is inspired by the *Candidate-Elimination algorithm* (CEA) [7].

Learning our situation patterns can be seen as a *concept learning* problem. Concept learning involves acquiring the general definition of a concept from training examples labeled either as *positive* or *negative* examples of this concept. For each repetitive task RT (concept*)*, the Apprentice has to induce situation patterns, using a set of situations including the situations preceding RT in the trace (positive examples) and the situations preceding any other repetitive task of the trace (negative examples).

The CEA learns by building two sets of generalizations of the positive examples: the most specific generalizations and the most general generalizations. IDHYS builds an approximation of the most specific generalizations only. It processes the positive examples incrementally. It starts with a very specific situation pattern (indeed the first positive example itself) and progressively generalizes this situation pattern with the subsequent positive examples. IDHYS does not build situation patterns for the negative examples which are only used to bound the generalization process. This incremental bottom-up approach makes IDHYS not sensitive to actions with large sets of possible values for their parameters. As a consequence it has a low computing time. Our test (see section 4) shows it also has a low error rate and does not over-generalize to build the situation patterns.

During the learning process, IDHYS computes a numerical evaluation of the quality of the learned situation patterns using the constant time Laplace correction [9]. This allows the Assistant to sort by quality the suggestions made to the user.

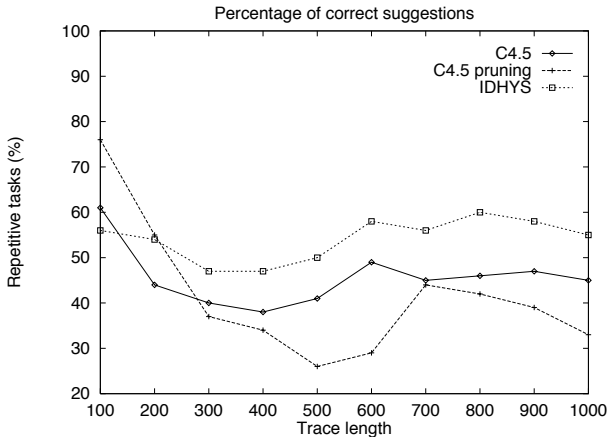


Figure 2

4. USE AND EXPERIMENTAL RESULTS

APE is implemented, operational and experimentally used by ourselves and by a pool of 50 students enrolled in a Smalltalk course at the master's level. This section describes technical experimental results and analyses the user feedback.

4.1 Experimental results

Concerning technical results, APE correctly works and makes the suggestions we expected it to. It also makes many suggestions we did not think of. We report here experiments conducted on 10 traces of 2000 actions long, collected during real usage of the software by the students.

The percentage of correct suggestions (Figure 2) is the amount of repetitive tasks the Assistant has correctly suggested. Figure 2

shows that IDHYS achieves a higher percentage than C4.5. The percentage of excessive suggestions (Figure 3) denotes the amount of user's actions for which the Assistant has made a suggestion whereas no suggestion was expected (i.e. the user did not performed a repetitive task). IDHYS achieves a lower percentage than C4.5, particularly when learning from few situations (on small sections of the trace).

For a given student trace, these percentages have been evaluated by cross-validation as follow: we have trained the Apprentice on a part of the trace and we have then tested the Assistant on another part of the trace to see how often one of its suggestions coincides with user's actions. This process has been repeated for a situation length varying from 1 to 10 and the resulting percentages averaged. IDHYS achieves best results when the Apprentice learns user's habits every 100 actions and for a situation length of 3 actions (i.e. when the Assistant uses the 3 last user's actions to predict repetitive tasks). For these values, the Assistant correctly suggests 64% of the repetitive tasks and makes undesirable suggestions for 22% of the user's actions. Concerning training time, the Apprentice learns user's habits on a 1000 action trace (corresponding to one our of user's work) in 25 seconds.

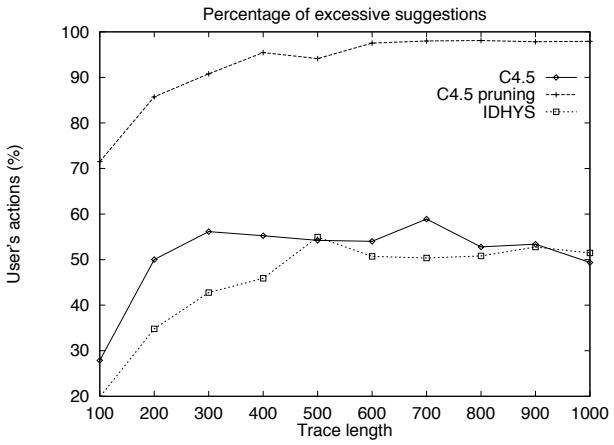


Figure 3

4.2 User Feedback

Concerning user feedback, let us recall that our users are Smalltalk beginners. About 70% of our students have considered the Assistant window during 10 minutes approximately and have forgotten it, this was quite disappointing. Fortunately, results from the remaining 30% have been very interesting. The main reasons invoked by those who have not used the suggestions are: (1) the burden of looking at the Assistant window since nothing, except a modification inside this window, indicates when a suggestion is made, (2) the difficulty to read the suggestions presented as sequences of actions. This indicates that there is a great deal of work to do in that direction, namely, how to gently alert people and how to provide a better visualization of what the Assistant suggests? The interesting point is that those who have made the effort to use the tool have rapidly learned how to use it efficiently. After a while, those users have learned which suggestions are regularly made and which ones interest them, and when the suggestions are made. In other words, they have learned to give a look at the Assistant window when they are about to perform a repetitive task and when they do know that the suggestion will be made.

5. FUTURE WORKS

As discussed in this paper, minimizing the number of incorrect suggestions is a critical issue in APE. Although it makes incorrect suggestions for only 22% of the user's actions, it would be more reliable if this amount could be reduced further. A solution to this problem we are currently investigating is to learn not only user's habits but also to predict whether or not the user is about to perform a repetitive task.

User tests reported in this paper have shown that APE would be even more attractive is the presentation of its suggestions were improved. Programming by demonstration studies have addressed the problem of creating a graphical representation of a program or a sequence of actions, and offer a direction for future research.

6. REFERENCES

[1] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In AAAI Spring Symposium on Information Gathering, 1995.

[2] A. Caglayan, M. Snorrason, J. Jacoby, J. Mazzu, and R. J. and. K. Kumar. Learn sesame: a learning agent engine. Applied Artificial Intelligence, 11:393-412, 1997.

[3] A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, (eds). Watch What I Do: Programming by Demonstration. MIT Press, Cambridge, MA, 1993.

[4] R. M. Karp, R. E. Miller, and A. L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In 4th Annual ACM Symposium on Theory of Computing, pages 125-136, May 1972.

[5] P. Maes. Agents that reduce work and information overload. Communications of the ACM, Special Issue on Intelligent Agents, 37(7):31-40, July 1994.

[6] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experience with a learning personal assistant. Communications of the ACM, Special Issue on Intelligent Agents, 37(7):81-91, July 1994.

[7] T. M. Mitchell. Version Spaces: An Approach to Concept Learning. PhD thesis, Electrical Engineering Dept., Stanford University, 1979.

[8] H. Motoda. Machine Learning Techniques to Make Computers Easier to Use. In Proceedings of IJCAI'97. Morgan Kaufmann Publishers, August 23-29, 1997.

[9] T. Niblett. Constructing decision trees in noisy domains. In I. Bratko and N. Lavraec, editors, Progress in Machine Learning, pages 67-78, Wilmslow, 1987. Sigma Press.

[10] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.

[11] J.-D. Ruvini and C. Fagot. IBHYS: a new approach to learn users habits. In Proceedings of ICTAI'98, pages 200-207. IEEE Computer Society Press, 1999