

Case-based reasoning for procedure learning by instruction

Jim Blythe

USC Information Sciences Institute
Marina del Rey, CA, USA
blythe@isi.edu

Thomas Russ

USC Information Sciences Institute
Marina del Rey, CA, USA
tar@isi.edu

ABSTRACT

To control intelligent tools that perform a variety of complex procedures, users need to be able to both modify existing procedure descriptions and communicate new procedures. In one approach, the user describes fragments of a procedure with text, and the tool searches the space of potential procedures for a match. This approach sometimes provides too little guidance for users, yet providing templates for guidance can require an expensive knowledge engineering effort in each new domain. We investigate the use of case-based reasoning to help guide the user, treating previously-defined procedures in the domain as cases. We describe domain-independent methods to find similar procedures while the user creates or modifies a procedure, to suggest potential steps to copy and to manage mapping the variables from the existing procedure into the procedure being edited. In some cases, the mapping tool suggests auxiliary steps to copy along with the desired steps, following an approach similar to derivational analogy. We evaluate the potential of this approach with an implemented tool, CB-Tailor, in a travel domain containing a number of procedures that may be added by the user. Our experiences suggest that the tool can provide useful guidance in a realistic set of situations.

Author Keywords

knowledge acquisition, procedure learning, learning by instruction

ACM Classification Keywords

I.2.6 [Artificial Intelligence]: Learning — knowledge acquisition.

INTRODUCTION

A growing number of software agents perform procedures for their users that may combine several steps, including conditional branching and loops. For example, desktop assistant tools may organize mail and calendar items, or provide access to web-based data. These steps may often be combined in novel ways, as in mash-ups such as Yahoo pipes

[12]. Allowing end users to define and customize these procedure-based applications is an important form of knowledge capture, potentially giving users unprecedented flexibility and control of their applications.

While mash-ups bring the notion of re-combining substeps in novel ways to a wide audience, existing systems lack the flexibility and control required to support a broad range of procedures, nor do they provide the support required for flexibly defining procedures. AI-based techniques for procedure learning have the potential to provide this support through reasoning about the user's intention and through background knowledge about the domain of application. Procedure learning approaches may combine several sources of information, for example information from demonstrations or examples [8, 10, 6] or from user instruction or annotation [3, 1].

Procedure learning by instruction is an important component of such a system, since in many situations it may be difficult or tedious to provide examples, while instruction may be a more direct way to specify a numeric threshold, for example, or how to generalize an instance such as a specific person based on database queries. However, users do not find current tools for procedure editing easy to use, for a number of reasons. First, the commands to be added to a procedure may be complex, and users are still required to generate them, at least by description. Second, there may be several interpretations of a user instruction, corresponding to alternative matching commands. The user may need to know details of the underlying system to determine the correct alternative. Third, a sequence of instructions is often required to enact a common operation. For example, to filter a list by some external criterion the user might need to access a fresh data source, extract a field from the data and then use the data in a loop, or combined with another source.

One solution to these problems might be to provide higher levels of control for specific ways to combine data, for example specific templates for filtering lists or performing database joins. However, these quickly become cumbersome — many templates are needed and the user has to know or recognize the correct one. In addition, these templates must be created in advance of the tool's use, increasing programmer requirements, introducing the risk of error and reducing flexibility to changing environments.

A more flexible solution is to use previously-built procedures as 'cases' to aid in interpreting instructions and guid-

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. IUI'08, January 13-16, 2008, Maspalomas, Gran Canaria, Spain.

ing the user. These cases can help the user in several ways. Since program idioms may be reused, they can help to interpret the instruction, or they can help a user to recognize and reuse the desired operation rather than generate it by description. Also, since sequences of similar modifications are common, they can help the tool to predict what the user might want to do next, allowing it to guide the user. Cases also require less advance programming from the developers, who may be domain experts, and users do not have to choose from a large set of templates.

However there are at least two potential challenges to overcome. First, low precision in matching the appropriate existing procedures may mean that the tool creates more spurious suggestions than useful ones, reducing the user's efficiency and trust in the tool. Second, the mapping process has a cost for the user, either to manage it or to supervise and potentially correct it if it is automated. Even following a useful suggestion from a prior case will carry a cost in noticing the similarity and managing the mapping. For example, if the tool automatically remaps variables in the source procedure to appropriately typed variables in the matching procedure, the user needs to check the appropriate mapping has been chosen and correct it if necessary. If the tool continuously asks the user for mapping information rather than guessing it, the cost may prohibit its use.

This paper describes CB-Tailor, an implementation of case-based reasoning for task learning in the Tailor task learning by instruction system. We explore the tradeoffs outlined above through a set of travel-related procedures that the user is creating and/or modifying. These procedures were chosen to have a reasonable proportion of similar subtasks, *e.g.* finding distances between objects or placing icons on a map, and also to cover a range of activities a user might want to perform in building a customized travel planner using the tool. By analyzing the relative proportion of useful versus spurious recalls that the tool makes while the user performs a fixed set of procedure creations and modifications, we show that the tool can be useful in this domain.

In the next section we introduce the problem of learning procedures from a user by interpreting instructions, and describe the Tailor system that performs this task. Next we describe the CB-Tailor module, both its procedure recall and mapping algorithms and the UI choices to bring this information to the user in an understandable way. After this we describe the set of related procedures that we used to analyze the effectiveness of CB-Tailor over raw Tailor. Finally, we discuss future avenues of work using cases to aid in procedure learning.

PROCEDURE LEARNING BY INSTRUCTION IN TAILOR

Tailor [3] allows users to add or remove steps and conditions in a procedure by describing the new step or condition and its location. For example, in building a procedure to find hotels with nearby restaurants, the user can say "get the hotel distance after finding the restaurants", assuming a step to find the restaurants is already present. Tailor performs a dynamic programming search to find matching code fragments. In this case it finds a 'get distance' procedure that requires

two addresses with a street and zip code. Using the address of a referenced hotel allows the procedure to match the user's instruction. The other argument is filled in with the address of the restaurant, since it has also been referenced in the procedure. Using this approach, the user does not need to know the exact name or form of the substep or condition to add, or the arguments it takes. Each instruction is "interpreted" in terms of an existing procedure or test by searching for valid code fragments that can be added to the procedure, building on variables whose values have already been filled.

Often, more than one match is found, and a thresholded subset is presented to the user. The user must therefore either provide unambiguous descriptions from which the system can generate code or recognize which of many possible interpretations found are the ones desired. For example the user may not know that a procedure to find hotel prices requires a check-in and check-out date. Tailor will find several alternative interpretations based on forming dates in different ways and perhaps using different web services. The user can then easily be overwhelmed by the sheer number of choices.

One technique by which human programmers reduce complexity when building procedures is copy-and-edit from similar procedures. We are investigating case-based reasoning in Tailor to support this approach for end users.

EXPLOITING EXISTING PROCEDURES IN CB-TAILOR

Programmers often retrieve existing procedures that contain some of the elements required in a new procedure, and use them for guidance and even initial code. We seek to provide the same kind of help for non-programmers within Tailor. We use ideas from case-based reasoning to find "similar" previously-built procedures and provide automatic mappings when steps are copied. A procedure is viewed as a structured set of subtasks, with such structural elements as sequences, subtasks performed in parallel, conditional branching and loops. To match a procedure to similar procedures while it is being built, we flatten each procedure to consider a single ordered list of subtasks. Then we search for tasks with the largest common subsequence with the current task. This search is performed efficiently with dynamic programming, as described in the next subsection. Tailor automatically brings the best matches by this heuristic to the user's attention, and also allows the other matches to be explored. When the user selects a potential match, useful parts of the procedure that may be copied into the current procedure are highlighted.

When the user selects a subtask of the procedure to copy and a target location, the variables in the mapped procedure need to be matched with variables in the target procedure. To do this, Tailor first searches for variables whose types match the requirements for the step being copied. If there are no matches, it first checks for previous substeps in the source procedure that establish these variables and, if found, considers transferring them with the requested step, recursively checking their input variables. This approach is similar to derivational analogy [11], with some key differences that

are discussed in section . If no match is found in this way, Tailor performs a search to find plausible ways to introduce matching variables, in the same way as is done to interpret user instructions. We explain the two steps in more detail, using the following problem as an example: Find restaurants that are near a hotel for a planned trip. Suppose we have two procedures built previously, one that finds hotels in a city with vacancies on given travel dates and another that finds restaurants in a zip code. As the user creates the procedure, the first step added finds hotels in a city, using a procedure that wraps an html source, based on wrapper induction [7]. A second step finds restaurants with the same zip code as the hotel. Since this step was used in the earlier procedure to find restaurants near movie theaters, it is retrieved and shown to the user. Figure 1 shows how the similar procedure is displayed to the user, showing both the matching steps and the useful subsequent steps that could be copied into the target procedure.

Finding similar procedures

We use a simple algorithm to find similar procedures. To compute the similarity between two procedures we ignore any structure in the body of the procedure, for example from nested sequence, parallel or branching constructs, and consider each procedure as a list of steps. We next compute the largest common subsequence between the two procedures using a dynamic programming approach that requires steps to be in the same order, but is insensitive to intervening steps.

We show the user one of the candidates with the longest common subsequence with the procedure currently under construction. A menu allows the user to pick other procedures with the same length, or optionally shorter common subsequence. Within the procedure, steps that could be mapped into the user's target procedure are identified, and shown through highlighting. In the figure, those steps are shown in white, while gray marks steps for which CB-Tailor does not have sufficient information to create a mapping.

Mapping steps between procedures

Once the user selects a step to copy, it must be modified according to the context of the target procedure. This involves aligning the input variables from the copied step with variables or constants that are available in the target procedure. We propagate type information and use a type hierarchy to identify candidate variables or constants in the target. We prefer exact matches to those that rely on type subsumption. From the highest-ranked candidates we then arbitrarily choose one match.

The modified, mapped steps are inserted into the target procedure and CB-Tailor then performs its normal analysis, checking for type mismatches and unused variables. The editing tools can be used to correct any errors resulting from the mapping choices made during the transfer operation.

EVALUATION IN A TRAVEL DOMAIN

To evaluate the tool, we took a travel domain in which a number of procedures had already been created to access information from web-based data, for example available hotels

given a city and check-in and check-out dates, restaurants for a zip code and their descriptions, and a distance server given two addresses, based on yahoo. We created fifteen canonical tasks and simulated building pairs of these tasks. CB-Tailor was frequently able to find relevant tasks, allowing the user to copy rather than create at least one step in the second task.

RELATED WORK

One of the most closely related pieces of work is that of Huffman and Laird on Instructo-Soar and related agents [4]. These agents receive instructions from users about how to achieve a task that is currently being executed, and use situated explanation to reason about the advice in the context of this task. The most important differences are that Tailor reasons about an abstract process description, rather than situated in a particular task, and Tailor does not assume enough domain knowledge to support explanation of the advice. Work in Tailor on recognizing user intent and mapping to well-defined modifications is also novel.

Programming by demonstration (PBD) [6, 8] allows users to specify procedure knowledge by giving examples to a system that learns the procedure by induction. This has the advantage that in many cases the user does not have to reason about the abstract procedure, just its behavior in certain cases. However, the user must provide scenarios that contain most of the information that is relevant for learning the procedure. We view PBD and learning by instruction as complementary, and are actively working to integrate our tool with a PBD system using a shared UI.

The Expect system provides an explicit language for procedures and tools to help users create or modify procedural information using english-based editors and reasoning about interdependencies [2, 5]. Tailor differs from Expect in taking short sentences of instruction as the principal source for procedure modifications, and using a standard procedure language, a variant of PRS, rather than a proprietary one. Short sentences mean it provides assistance at a higher level, since the user says more about the purpose of the change and less about the details than in a system like Expect.

Myers [9] describes the Advisable Planning system that has a language for high level advice. It was initially developed for an HTN planner although it is currently also being applied to Spark. The constructs in the language constitute advice about choosing particular fillers for roles in procedures, e.g. the agent or instrument, or choosing the procedure to use to perform a task. These constructs specify values for possible choices in the procedure knowledge base rather than modifying which substeps are performed and when.

Our approach to case-based reasoning is inspired by derivational analogy [11] in which a planning case is followed within a planning system that can follow the derivation of the plan, rather than blindly copying the steps of the plan. In this way, steps that are unnecessary can be omitted from the case, and extra steps may be added as required in the new situation. In our approach, auxiliary steps may be supplied by Tailor when needed to supply the required variables

The top window shows the similar procedure that CB-Tailor finds, presented to the user with text generated from the procedure language using templates. Colors are used to show how substeps match between this procedure and the user's procedure below. In the bottom window, the user has chosen to copy one of the available steps through the menu.

Here, the user can copy the step to compute the distance between a restaurant and a movie theater. After doing this, the user modifies the calculation to be distance between a restaurant and a hotel. The final step of comparing the distance and collecting the results in a list are also shown, and can also be copied. After that there are no more steps that could be copied in this case, however, the user can still check that the step is being applied correctly by comparing it with the previous, working procedure.

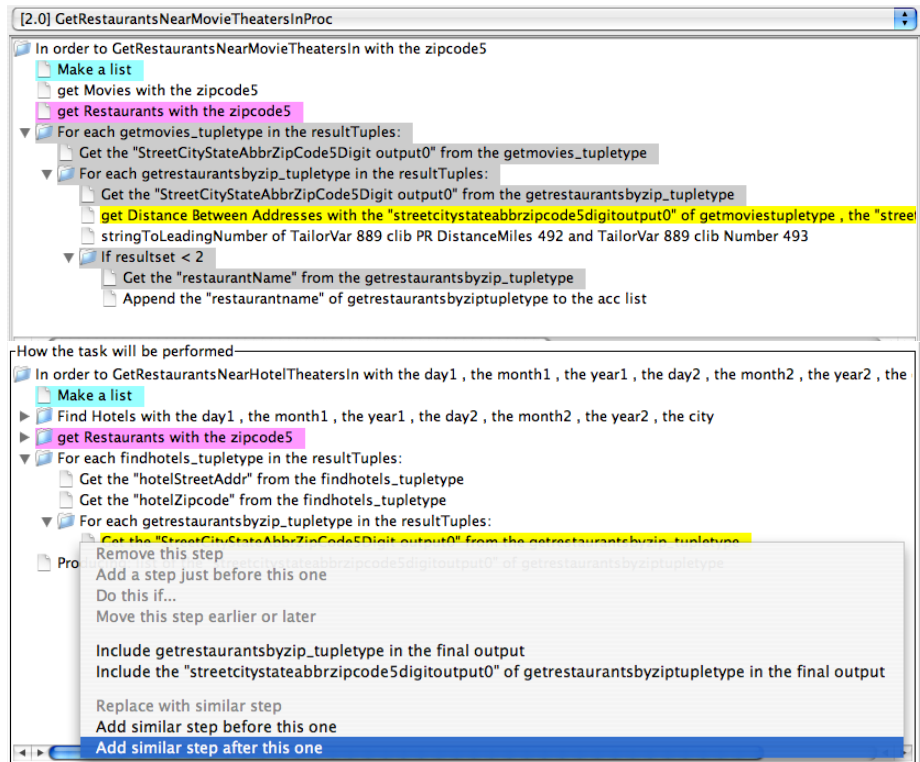


Figure 1. Copying Steps from a Similar Procedure.

for the desired steps, or they may be omitted if the new procedure has already made variables of this type available.

CONCLUSIONS

We have shown the viability of case-based reasoning to support procedure learning by instruction in an initial implementation, CB-Tailor. Using a derivational approach, the tool can flexibly include steps from the case as needed to map a desired step into the user's procedure. We are experimenting further with this approach along with other techniques to provide higher-level support to the user within the task learning tool Tailor. In this way, the user has the full flexibility of an instruction-based approach when needed, but can use a menu-based interface to add standard loops or make projections or joins on data when this is sufficient for the procedure. Our preliminary findings indicate that this approach greatly increases the usability of the tool without affecting its power for knowledgeable users.

ACKNOWLEDGMENTS

We are grateful for discussions with the ISI task learning group, including Craig Knoblock and Kristina Lerman. This research is based in part on work supported by DARPA, through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

REFERENCES

Allen, J., et al. 2007. PLOW: A Collaborative Task Learning Agent *Proceedings of AAAI 2007*.

Blythe, J. 2001. Integrating Knowledge from Different

Sources to help End Users Acquire Procedural Knowledge. *Proceedings of IJCAI 2001*.

Blythe, J. 2005. Task learning by instruction in tailor. *Proceedings of IUI 2005*.

Huffman, S. and Laird, J. 1995. Flexibly Instructable Agents *Journal of AI Research*, 3, 1995.

Kim, J. and Gil, Y. 1999. Deriving Expectations to Guide Knowledge-base Creation *Proceedings of AAAI 1999*

Lau, T.; Bergman, L.; Castelli, V.; and Oblinger, D. 2004. Sheepdog: Learning procedures for technical support. *Proceedings of IUI 2004*.

Blythe, J., Kapoor, D., Knoblock, C., Lerman, K., and Minton, S. 2007. Information Integration for the Masses: *AAAI 2007 Workshop on Information Integration*.

Lieberman, H. 2001. *Your Wish is my Command*. Morgan Kaufmann Press.

Myers, K. 2000. Planning with Conflicting Advice *Proceedings of AIPS 2000*

Oblinger, D.; Castelli, V.; and Bergman, L. 2006. Augmentation-based learning: combining observations and user edits for programming-by-demonstration. *Proceedings of IUI 2006* 202–209.

Veloso, M. 2004. Planning and Learning by Analogical Reasoning Springer Verlag, 1994

Yahoo pipes <http://pipes.yahoo.com>