Collaborative Example Selection in an Intelligent Example-Based Programming Environment

Peter Brusilovsky
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
plb@cs.cmu.edu

Gerhard Weber
Department of Psychology
University of Trier,
Trier, D-54286, Germany
weber@cogpsy.uni-trier.de

Abstract: This paper discusses problems related to finding relevant examples in educational example-based programming environments. We consider several approaches to organizing example selection in such environments. The paper defends the collaborative approach to example selection which implies that the user and the system collaborate in the process of example selection. The search process is driven by the user and the system supports the user with all available knowledge. To demonstrate possible ways of implementing this approach we describe how several forms of collaborative example selection were implemented in ELM-PE, an intelligent programming environment for learning LISP.

Introduction

It has often been claimed that humans use solutions to previous problems to solve new problems or planning tasks. Especially in the domain of programming, both experienced and novice programmers often use code examples they have created or learned in the past to solve new programming tasks. These experimental facts inspired the design of a number of special "example-based" programming environments for both novices and professionals (Fischer, Girgensohn, Nakakoji, & Redmiles, 1992; Guzdial, Weingard, Boyle, & Soloway, 1992; Neal, 1989; Weber, in press). Such environments support example-based programming by providing example libraries and special interface tools to facilitate the process of finding relevant examples and re-using them to solve the problem at hand.

This paper discusses problems related to finding relevant examples in educational example-based programming environments. We consider several approaches to organizing example selection in such environments. The paper defends the collaborative approach to example selection which implies that the user and the system collaborate in the process of example selection. To demonstrate possible ways of implementing this approach we describe how collaborative example selection is implemented in ELM-PE, an intelligent programming environment for learning LISP.

Example Selection in Educational Example-Based Programming Environments

In most educational example-based programming environments an interface for selecting relevant examples is quite simple: a menu-based choice from the list of available examples (Neal, 1989), or the possibility to search for an example using the patterns from the name, the problem statement, and the code of the example itself (Faries & Reiser, 1988). An experiment (Faries & Reiser, 1988) shows that even with these simple tools the users are often able to find structurally similar examples in a reasonably small example space. However, in the beginning of learning a new programming language, the students may lack the necessary knowledge and experience to find a relevant example. A more recent and comprehensive experiment with novice LISP programmers (Weber, 1995) shows that only in two thirds of all cases were the users able to find the most relevant example using simple example-selection tools. The lack of more efficient example-selection tools seems to be a bottleneck of educational example-based programming environments.

In this section we discuss three approaches which can be applied to help novices to select a relevant example in an educational context: the student-driven approach, the system-driven approach, and the collaborative approach. We argue that collaborative example selection is the most suitable and powerful approach for example selection in example-based programming environments.

Student-Driven Example Selection

One of the possibilities to support users in the process of example selection is to provide more powerful tools for student-driven example access. A promising technology to apply here is the hypermedia technology. Browsing can be an effective way for selecting relevant examples in example-based programming environments. Experience (Weber, 1995) demonstrates that in many cases users can recognize useful structurally similar examples, but can not find them themselves using simple interfaces for example access. The problem is how to structure all available examples into a hyperspace and what kind of links to provide for navigation.

A useful experience in applying hypermedia technology in the context of learning programming is reported by Linn (1992). She describes several hypermedia based systems for organizing items of programming knowledge (such as standard re-usable "patterns" of code or commented program examples) and students' own programs. Linn (1992) considers and compares two primary ways to organize a hyperspace: expert structuring and user structuring. The hyperspace of existing programming knowledge was structured by the domain experts. At the same time, the students have the possibility to introduce their own structure for existing knowledge, and to make new links between examples and from examples to basic knowledge.

The systems reported in (Linn, 1992) can serve as prototypes for constructing a hypermedia system to support example-based programming. In such a system the "existing" part of the hyperspace (programming knowledge and program examples) could be structured by the domain experts and the "added" part (programs from the students' personal problem-solving history) could be structured and connected to "existing" parts by the students themselves. Unfortunately, the experience reported in (Linn, 1992) shows that the students' structuring can not be the primary way to organize the "added" part of the hyperspace. Students do not like to spend their time for such a "side" activity as structuring the hyperspace and, if they are pushed to do that, they do it quite irregularly.

System-Driven Example Selection

In an educational context, a tutoring system usually knows the problem that the student is solving, so it can assist the user by suggesting a relevant example for this problem. The simplest approach here is to store a relevant pre-selected *course example* (i.e., a solution to a programming problem that was explained to the student earlier in the course) for each problem in the system problem set (Ramadhan & du Boulay, 1993). However, this way can not take into account students' own solutions to previous programming problems (we call it *remindings*) which are often better examples than pre-selected examples from the course (Weber, 1995). A more powerful approach can be implemented in an ITS which has knowledge about the user and the subject. Knowing the problem at hand, an ITS can apply its knowledge to select the most relevant course example or reminding for that problem.

An example of an ITS which is able to offer the student individually selected course examples and remindings (in addition to a simple student-driven tool for example selection) is ELM-PE (Weber, in press). ELM-PE stores all problem solutions of each student in an individual Episodic Learner Model (ELM) which is a case-based student model (Weber, Bögelsack, & Wender, 1993). Searching for a best analog to a given new programming problem works as follows. First, the diagnostic component is used to generate an expected solution to a new programming task. Second, the result of the diagnosis to this expected solution is stored temporarily in the ELM. Third, from computing the organizational similarity to other episodes (examples and remindings) stored with the user model, a best match can be found and offered to the learner as an example solution to a similar programming problem.

The system-driven example selection in ELM-PE works very efficiently. An experiment (Weber, in press) shows that in 96% of all cases ELM-PE selects the best possible example (all examples were ranked by experts). In about 10% of all cases the example selected by the system was better than the pre-selected course example for the same problem (being equal to it in the rest of the cases). However, ELM-PE shares a common problem of traditional ITS which try to make the best possible decision for the students and give them no way to override this decision. It is not very "user-friendly" because in some cases the system's "best" decision is not really the best one from an expert's point of view (in ELM-PE it happens only in 3.5% of all cases) and because advanced students can prefer another decision which the system does not considers as "the best" one.

Collaborative Example Selection

As we have seen, both "single agent-based" approaches (i.e., both student-driven and system-driven approaches) to example selection have problems. With the "pure" student-driven approach it is impossible to organize a powerful interface for example selection and with the "pure" system-driven approach students have no possibilities to override the suggestion of the system. The alternative to these "single agent-based" approaches is to use the intelligence of both involved agents: i.e., to let the system and the student collaborate in selecting the best example. With collaborative example selection the role of the student is to select the example and the role of the system is to support the student in selecting the best example using all available knowledge. A good example here is CatalogExplorer (Fischer et al., 1992), a utility which helps the professional program designer to find a code example to re-use (what is very close to example-based programming). In CatalogExplorer, the collaborative example selection is started by the user who specifies the requirements to the desired code. The system then uses the specification and the knowledge base (specification-linking rules) to find all relevant programs in the catalog base and to assign appropriateness values to them. The list of relevant examples ordered according to the appropriateness is presented to the student who makes the final choice. This example demonstrates that collaborative example selection combines positive features of user-driven and system-driven approaches. The knowledge available in an intelligent system supports the user in making a decision, but it is the user who makes the final choice.

Presenting an ordered list of suggestions is the simplest form of collaborative example selection. We have been investigating several possible forms of collaborative example selection in the context of ELM-PE system mentioned previously. In the following parts of the paper we describe how several forms of collaborative example selection were implemented in the new version of ELM-PE.

Collaborative Example Selection in ELM-PE

The knowledge-based programming environment ELM-PE was designed to support novices who learn the programming language LISP. ELM-PE directly supports example-based problem solving. The student who encounter a problem or a lack of ideas when solving a LISP-problem can load code of any previously solved problem or previously explained example into a special *example window*. A syntax-driven *structure editor* makes it very easy for the student to copy any LISP expressions from example window to solution window. The original version of ELM-PE (Weber, 1995) supports student-driven and system-driven example selection, i.e., the student can either select a helpful example himself or herself or ask the system about the "best possible" example. Our goal was to extend ELM-PE with collaborative example selection functionality. More exactly we have tried to provide the user with several forms of knowledge-based support in the context of hypermedia-based example selection applying the knowledge currently represented in the system. In the new version of ELM-PE this knowledge is used to provide the simplest CatalogExplorer-like form of collaborative example selection, to structure the hyperspace of examples (mainly its "added" part which can not be pre-structured by the domain experts), and to provide navigation support for the user browsing this hyperspace.

Knowledge Representation in ELM-PE

The system's knowledge consists of both the common LISP domain knowledge and the episodic knowledge about a particular learner. Both types of knowledge are highly interrelated. That is, on the one hand, the system is able to consider individual, episodic information for diagnosing code and for explaining errors in addition to using the common domain knowledge. On the other hand, when explaining individual errors and examples from the learner's individual learning history, the system can combine episodic information with information from the domain knowledge.

The domain knowledge of ELM-PE is represented in a heterarchy of concepts and rules (Weber et al., 1993). Concepts comprise knowledge about the programming language LISP (concrete LISP-procedures as well as superordinate semantic concepts) and schemata of common algorithmic and problem solving knowledge (e.g., recursion schemata). These concept frames contain information about plan transformations leading to semantically equivalent solutions and about rules describing different ways to solve the goal stated by this concept. Additionally, there are bug rules describing errors observed by other students or buggy derivations of LISP-concepts which, e.g., those which may result from confusion between semantically similar concepts.

The individual learner model consists of a collection of episodes that are descriptions of how problems have been solved by a particular student. These descriptions are explanation structures (in the sense of explanation-based generalization, Mitchell, Keller, & Kedar-Cabelli, 1986) of how a programming tasks has been solved by the student. That is, stored episodes contain all the information about which concepts and rules were needed to

produce the program code the students offered as solutions to programming tasks. Episodes are not stored as a whole. They are distributed into snippets (Kolodner, 1993) with each snippet describing a concept and a rule that was used to solve a plan or subplan to solve the programming tasks. These snippets are stored as episodic instances with respect to the concepts of the domain knowledge. In this way, the individual episodic learner model is interrelated with the common domain knowledge.

Computing an Ordered List of Relevant Examples

To support the student selecting the proper example, the current version of ELM-PE is able to generate an ordered list of relevant examples. This list is generated by the *Explanation-based Retrieval* (EBR) algorithm which is described in detail in (Weber, in press). At the first step, the system uses its knowledge about the problem and the individual ELM to generate the most probable solution for the given problem. On the basis of concepts and rules that will be used to generate the new solution, the case memory can be probed for cases most similar to this solution. Concepts from the resulting explanation structure are inserted temporarily into the existing concept hierarchy of the episodic learner model. All episodic frames that are neighbors to the temporarily inserted frames contribute to computing weights for similar episodes. Competition among episodes is introduced by normalizing episodic weights with respect to the sum of weights of all organizationally similar neighbor frames. For each episode, the system computes the similarity value by summing up the resulting weights for all of its constituting episodic frames.

Finally, the system selects the episodes with the highest similarity values and presents them to the student in the form of a hypertext list of links to examples and remindings ordered according to their similarity values. The most relevant example (the only one which was presented by the earlier version of ELM-PE) stays always first in the list so the students who rely on the system's choice can easily use it. However, with this interface they have also a possibility to try the "second best" and other examples in the list. To give the students more information for selecting an example, the system shows the similarity value for each example in the list. As we noticed, an adaptively ordered list of choices is a simplest form of collaborative decision making.

Navigating Between Examples and Concepts

In ELM-PE, the conceptual domain knowledge is used to organize a small LISP reference manual with a hypermedia interface. The current version of the manual contains two kinds of nodes: LISP functions and LISP data types. Each node of the hyperspace represents a concept from the domain knowledge heterarchy and links between nodes correspond to semantic relations between concepts (only a part of concept frames are reflected in the manual, for example, the concepts representing programming goals are not browseable). The content of the hypermedia "page" for a particular node is not stored in a presentation format but generated from the corresponding concept frame. The user has several ways to enter the manual, including context-sensitive help in a structure editor.

The hypermedia manual is a good starting point for selecting a proper example if a student is interested to find an example related with a particular LISP-function. To support this mode of work the system dynamically includes all examples and remindings (represented as episodes in an individual ELM) as the third kind of nodes into the hyperspace and connects them to the related conceptual nodes. It means that the student can navigate from a function to any existing example which uses this function and backwards, from an example to all functions used in it. These links can be generated easily, but the links themselves are not enough to help the student to select a proper example containing a particular function. The problem is that the same function can be used in different examples to implement different problem-solving goals while the student is usually interested to find an example which shows how to implement a particular goal. To help the student the system provides additional navigation support. On request, for each example related to the given function the system generates an explanation that lists all the goals implemented by the given function in the given example. To generate it, the system retrieves all the snippets for the stored example episode which are linked to the given function and reconstructs the goals of these snippets.

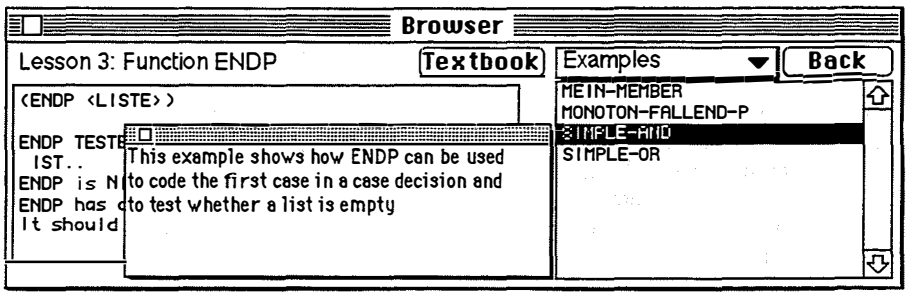

Figure 1: Navigating from a concept to related examples with explanation-based navigation support.

Similarity-Based Navigation Between Examples

To support navigation between examples incrementally added to the hyperspace, the system generates similarity links between examples. Similarity links are the most popular kind of system-generated links. Many hypermedia-based information retrieval systems are able to generate similarity links between stored items of information (Tudhope, Taylor, & Benyon-Davies, 1995). Experience with these systems shows that similarity-based navigation is a very powerful tool for searching in the hyperspace. In ELM-PE, the similarity links between examples are computed using a variant of the EBR-algorithm. As it was shown in section 3.2, for each example episode this algorithm can generate the list of structurally similar examples episodes and compute the similarity value for each of them. This list is presented to the student as a list of links from the given example to related examples. For each related example, the similarity value is shown and the list is ordered according to these values. This interface provides the student with navigation support in similarity-based navigation. It is worth mentioning that the navigation support in ELM-PE is adaptive because the similarity values are computed using the individual ELM.

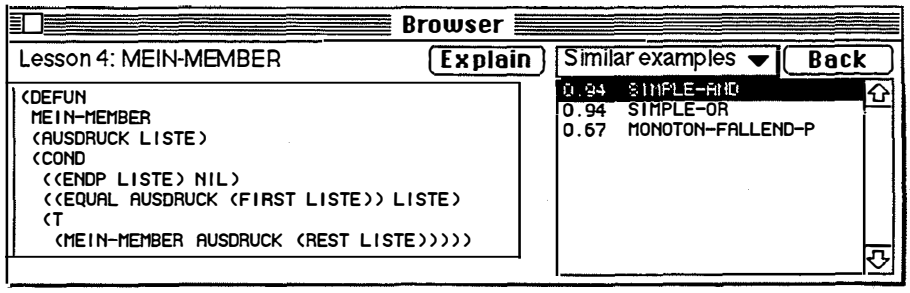

Figure 2: Similarity-based navigation between examples with adaptive navigation support

Summary and Future work

In this paper we defend a collaborative approach to example selection in intelligent example-based programming environments. With this approach, the system and the user collaborate in finding the most relevant example to use in problem solving. The search process is driven by the user (i.e., the student) and the system supports the user with all available knowledge. We have described how several forms of collaborative example selection were implemented in ELM-PE using the combination of conceptual and case-based knowledge representation. In particular, we have described example selection based on the navigation in the system-structured hyperspace.

Our next step on this way is to provide the student with some more ways to reach the relevant example. As an additional starting point we consider re-usable program fragments - code chunks and code patterns. The arguments for the use of chunks and patterns in learning programming are provided in (Linn, Katz, Clancy, & Recker, 1992). Currently, the system supports a small set of useful fragments which can be re-used by the student using the code buffer - a special feature of ELM-PE's structure editor. These fragments can be used as the new kind of nodes in the hyperspace to let the student navigate from a fragment to any example which uses it. Examples and fragments will form a personal part of the overall hyperspace. The next step is to let a student structure this part according the original ideas of Linn (1992). In line with the spirit of collaborative example selection, the system can use represented knowledge to support the process of structuring by making relevant suggestions while leaving the decision to the student. Our hypothesis is that the collaborative style of work can solve the problems of student-driven hyperspace structuring reported in (Linn, 1992) .

References

Faries, J. M. & Reiser, B. J. (1988). Access and use of previous solutions in a problem solving situation. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 433-439). Hillsdale, NJ: Lawrence Erlbaum Associates.

Fischer, G., Girgensohn, A., Nakakoji, K., & Redmiles, D. (1992). Supporting software designers with integrated domain-oriented design environments. *IEEE Transactions on Software Engeneering, SE-18*, 511-522.

Guzdial, M., Weingard, P., Boyle, R., & Soloway, E. (1992). Design support environments for end users. In B. A. Myers (Ed.), *Languages for developing user interfaces* (pp. 57-78). Boston, MA: Jones and Barlett.

Kolodner, J. L. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.

Linn, M. C. (1992). How can hypermedia tools help teaching programming. *Learning and Instruction, 2*, 119-139.

Linn, M. C., Katz, M., Clancy, M. J., & Recker, M. (1992). How do Lisp programmers draw on previous experience to solve novel problems? In E. De Corte, M. C. Linn, H. Mandl, & L. Verschaffel (Eds.), *Computer-based learning environments and problem solving* (pp. 67-101). Berlin: Springer-Verlag.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: a unifying view. *Machine Learning, 1*, 47-80.

Neal, L. R. (1989). A system for example-based programming. In K. Bice & C. Lewis (Eds.), *Proceedings of Human Factors in Computing Systems, CHI'89* (pp. 63-68). Reading, MA: Addison-Wesley.

Ramadhan, H. & du Boulay, B. (1993). Programming environments for novices. In E. Lemut, B. du Boulay, & G. Dettori (Ed.), *Cognitive models and intelligent environments for learning programming* (pp. 125-134). Berlin: Springer-Verlag.

Tudhope, D., Taylor, C., & Benyon-Davies, P. (1995). Navigation via similarity in hypermedia and information retrieval. In R. Kuhlen & M. Ritterberg (Eds.), *Proceedings of HIM'95* (pp. 203-218). Konstanz: Universitätverlag Konstanz.

Weber, G. (1995). Providing examples and individual remindings in an intelligent programming environment. In J. Greer (Ed.), *Proceedings of AI-ED 95 - 7th World Conference on Artificial Intelligence in Education* (pp. 477-484). Charlottesville: AACE.

Weber, G. (in press). Individual selection of examples in an intelligent programming environment. *Journal of AI and Education.*

Weber, G., Bögelsack, A., & Wender, K. F. (1993). When can individual student models be useful? In G. Strube & K. F. Wender (Eds.), *The cognitive psychology of knowledge. The German Wissenspsychologie project* (pp. 263-284). Amsterdam: Elsevier (North-Holland).

Acknowledgments

Part of this work is supported by a Grant from "Alexander von Humboldt-Stiftung" to the first author and by a Grant from "Stiftung Rheinland-Pfalz für Innovation" to the second author.