Case-Based Evaluation of Novice Programs

Hiroyoshi Watanabe, Masayuki Arai and Shigeo Takei
School of Science and Engineering, Teikyo University
1-1 Toyosatodai Utsunomiya-shi Tochigi, 320-8551 Japan
E-mail: {hiro,arai,takei}@ics.teikyo-u.ac.jp

Abstract. This paper proposes a case-based evaluation assistant system that supports teachers in their evaluation work of students' programs. The target evaluation tasks are to judge whether a student's program satisfies the requirements of the given problem and to give advice to students. This paper describes the idea of a case-based evaluation assistant and the details of constructing a case-base and reasoning processes with cases on the implemented system. The case-based evaluation assistant system compares a program submitted by a student with evaluation cases in the case-base. If some case matches the program, the system applies the judgment and advice on the case to the student's program. The target programming language of the implemented evaluation assistant system is a simple assembly language. The implemented system has a function of evaluating programs' actions in addition to case-based evaluation. The system had been utilized in actual classes for two years and the results showed that the system reduced the teachers' evaluation work drastically.

1. Introduction

In typical programming exercise classes, teachers' workloads tend to be very heavy. Teachers give students various problems in order to get the students to understand important concepts in programming. Teachers give advice to students and they have to read very many programs and/or reports to see if the students understand the concepts.

There are two approaches to support teachers. The first one is to provide students a diagnostic tool of their programs and students can learn from the output of the tool [1]-[8]. Hopefully, the tool reduces teachers' loads of advice giving. Most of the approach aims at automating the whole evaluation work of programs using a knowledge-based technique. The second approach is to support teachers in their evaluation work [9]-[12]. We took this latter approach, because supporting teachers with their evaluation work can assist them in efficiently supporting students. That is, reducing teachers' evaluation work gives teachers extra time to advise students, and teachers' advice is much better than advice generated by computer systems at the moment.

We propose a case-based evaluation assistant model of novice programs. Case-based reasoning [13][14] is one of the best approaches to implement the evaluation assistant. It requires much less heuristics or knowledge than traditional knowledge-based approaches [1]-[8]. Besides, the style of using past cases is well matched with the style of teachers' evaluation work. The disadvantage is that case-based systems cannot deal with programs of the type previously unseen. However, it is not a major problem because the program evaluation work is considered collaboration between teachers and computer systems.

We implemented a case-based evaluation assistant system for a simple assembly language CASL in order to demonstrate the effectiveness of our idea. CASL is adopted in examinations for information-technology engineers certified by the Japanese ministry of international trade and industry.

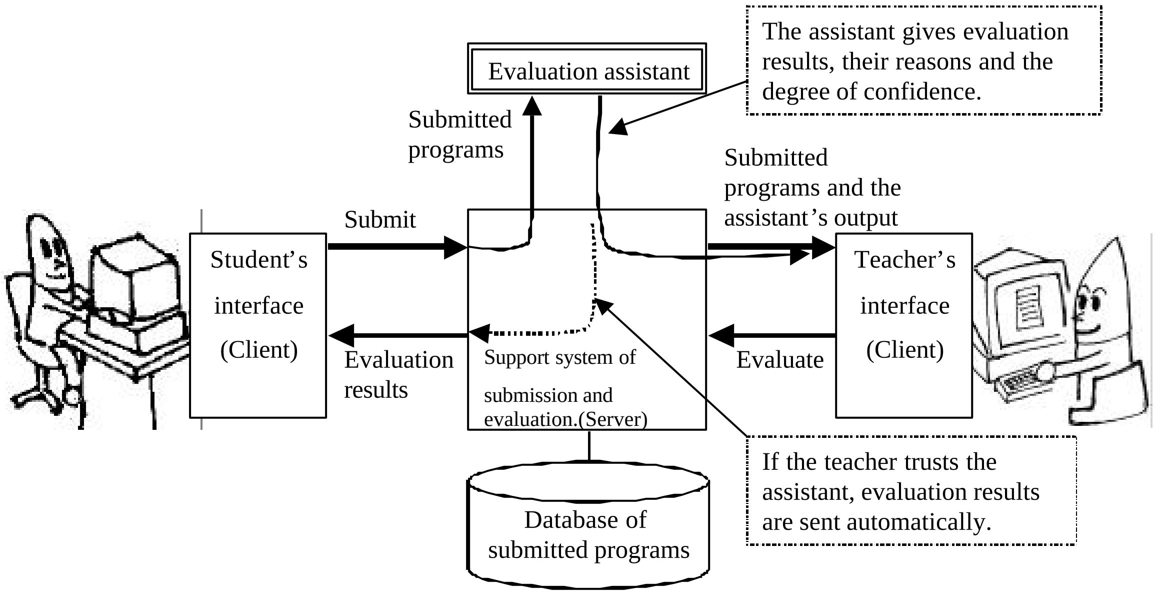

The assistant gives evaluation results, their reasons and the degree of confidence.
Evaluation assistant
Submitted programs
Submitted programs and the assistant's output
Submit
Student's interface (Client)
Teacher's interface (Client)
Evaluation results
Support system of submission and evaluation.(Server)
Evaluate
If the teacher trusts the assistant, evaluation results are sent automatically.
Database of submitted programs

Figure 1 An evaluation assistant system of programs.

2. Case-Based Evaluation assistant

2.1 Target Tasks of Program Evaluation

The target evaluation tasks of a student's program are (1) judging the acceptability of the program and (2) advising the student on the program.

The first task is judging whether a student's program satisfies requirements of the given problem. When teachers set problems, they have educational intentions about what students should learn, namely concepts, algorithms, instructions and so on. Teachers read students' programs to see whether the educational intentions are achieved. Therefore, teachers accept a student's program when the program satisfies requirements of the given problem. The first task is defined on the assumption that students have to submit their programs over and over until their programs are accepted.

When the teacher evaluates a submitted program, he or she judges whether the student's program satisfies requirements of the given problem. Usually, the teacher examines not only the action of the program but also the method of the implementations. Consider the case where the description of a problem includes a phrase such as "using a stack". In this case, the teacher intends to make students learn about a stack and he or she would reject a program implemented without a stack, even if the program's action satisfies requirements. Also, some teachers may reject too complicated programs and advise students who wrote the programs to make them simpler.

The second task is giving written advice. Teachers give advice to students whether they accept a program or not: teachers give advice about the reasons why the program is rejected, and advice about improving the program even if the program is accepted.

2.2 Evaluation assistant System

Figure 1 illustrates an evaluation assistant system in the electronic program submission environment. The evaluation assistant pre-evaluates submitted programs by case-based reasoning. Teachers can edit the results from the evaluation assistant when they evaluate the programs. If the teacher trusts the evaluation assistant, the results from the assistant can be sent to students directly. Such an evaluation assistant is expected to save a teacher a lot of time and energy.

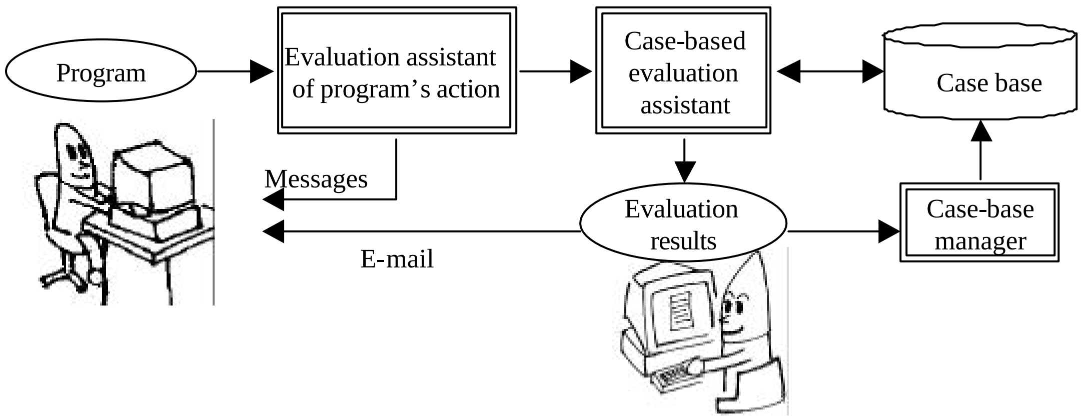

Program
Evaluation assistant of program's action
Case-based evaluation assistant
Case base
Messages
Evaluation results
Case-base manager
E-mail

Figure 2 Evaluation processes in the implemented system.

The output of the evaluation assistant consists of evaluation results, their reasons and the degree of confidence. The evaluation results include the judgment of acceptability (accept or reject) and written advice. The reasons for the evaluation results are based on applied cases. The degree of confidence is one of *surely*, *probably* or *unknown*. The confidence level is decided based on the similarity between a student's program and an applied case. When no case matches a student's program, the degree of confidence is assigned *unknown* and evaluation results and their reasons are not given.

The evaluation assistant system learns from final evaluation results by teachers. Learning is done by adding new cases and over-writing some information in stored cases.

2.3 Outline of Implemented System

The implemented system consists of a server, clients for students and clients for teachers. We implemented an evaluation assistant of program's actions in addition to the case-based evaluation assistant.

The server retains data of the problem information and cases for the evaluation. In addition, the server logs operations on the clients and students' states of submission. The case-based evaluation assistant is implemented on the server.

The client system for students has functions of editing, assembling and simulating programs in addition to submitting programs. The evaluation assistant of program's action is implemented in the client to distribute and reduce the loads of the server computer. The client system is implemented as a Windows^{TM} application.

Teachers use web browsers as their clients because the functions for teachers are implemented as CGI programs. The CGI programs provide two primary functions: the first function is browsing lists of students with information about their states of submission and students' programs, and the second function is referring to the system's evaluation results and editing them to complete evaluation results.

Figure 2 illustrates evaluation processes in the implemented system. At first, the actions of a student's program are tested using prepared sample data. Programs that do not run correctly are rejected by the evaluation assistant of a program's action. Only programs executed correctly are evaluated by the case-based assistant. The teacher edits the output of the case-based assistant and final evaluation results are sent to students by e-mail. The final evaluation results are also used by the case-base manager to add or over-write cases. When a case matched with a student's program with *surely* confidence and the teacher changed the judgment of acceptability or advice sentences, the case-base manager over-writes the case.

There are two modes of sending evaluation results to students. In mode-1, all of the evaluation results generated by the assistant system are sent to students after a teacher makes the final evaluation, even in the cases of a perfect match. In mode-2, the evaluation results generated by the assistant system with *surely* confidence, i.e. in the case of a perfect match, are sent to students directly. The mode of sending evaluation results is selected for each problem. Mode-2 should be used when the assistant system can evaluate most of the submitted programs with *surely* confidence, such as simple problems or problems with enough case-bases, because there are quite long time lags between the system's evaluation and the teacher's evaluation in mode-2. Unfortunately, this can result in the situation where students wrongly assume that programs with early responses are better than ones with late responses.

3. Representing and Indexing Cases

3.1 Case Representation

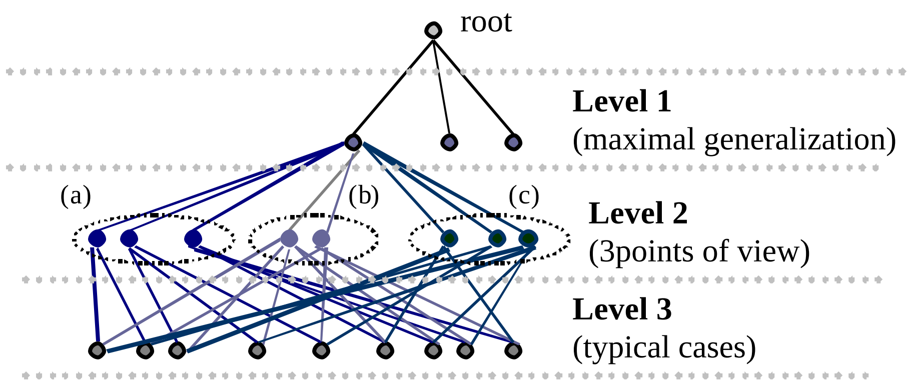
A case for the case-based evaluation assistant consists of a problem description, a solution description, retrieval information and maintenance information. In the domain of a program evaluation task, the problem description is a program list and the solution description is evaluation results, i.e., the judgment of acceptability and written advice. The retrieval information includes problem identification that the program is written for and features of the program, namely, numbers of every opcode (operation code) and redundant instructions in the program list. The maintenance information includes a teacher's name and the date of adding or updating the case.

Generally speaking, a program list as a problem description of the case should be represented in intact target programming language. In our first version of the system [11][12], the program lists were represented in the generalized form, in order to enable cases to cover several variations of program lists for the same implementation. However, we found out that teachers (users of the system) had to know about the generalized form, because they saw the program lists when the system presented applied cases as the reasons of evaluation results. Therefore, program lists in cases should be intact and indexes to cases should be constructed by using information of generalized program lists in order to expand variations of program lists covered by one case.

3.2 Indexing

Shown in Figure 3, three level indexes of cases are constructed using information of generalized program lists. A generalized program list covers several variations of program lists. The indexes enable pruning of cases that have no possibility of matching the given program. More precisely, if the given program does not match some nodes in level one, then descendants of the nodes are pruned. Also, the indexes enable use of a similar case, which is not completely the same. For example, if a given program does not match any cases (nodes in level 3), but matches some node in level 2, then the node's children are available.

Level 1. Nodes in level 1 represent generalized program lists derived by applying all the generalization rules we have. Hence, the nodes are called maximal generalization. The nodes have information about maximal and minimum numbers of every opcode in the program lists, which are covered by the nodes. The information is used for estimating the possibility of matching with the nodes.



(a) Variations of used instructions.
(b) Variations of the instructions' order.
(c) Variations of redundant instructions.

Figure 3 Indexes of cases

Level 2. There are parallel links between level 1 and level 3 taking different routes, (a), (b) or (c) as they go through level2 based on three points of view, so that the priority of the links is defined in case retrieval algorithms. In fact, nodes in level 2 are classified into three groups, because the following three points of view are important when a program list is generalized:

(a) Variations of used instructions. A node in group (a) represents a generalized program list, which covers some programs of the same implementation. There are operations implemented in more than one way. For example, loading a certain value into a general register can be implemented by an instruction LEA (Load Effective Address) or LD (LoaD). To deal with such situations, we define generalized forms of instructions and generalization rules to transform them. While nodes in level 1 are derived by applying all generalization rules, nodes in group (a) of level 2 are derived by applying a subset of the generalization rules we have. We defined the subset based on our experience and teachers can customize it.

(b) Variations of instructions' order. A node in group (b) covers the same order of instructions.

(c) Variations of redundant instructions. A redundant instruction means a removable one. A node in group (c) covers some cases with the same situations about redundant instructions, that is, the same kind of redundant instructions or no redundant instructions.

Level 3. Nodes in level 3 are typical cases. Programs consisting of the same instructions in the same order are regarded as the same, even if names of labels or numbers of registers are different.

4. Evaluating with Cases

4.1 Retrieving Cases

The most similar case to a given student's program is retrieved from the case-base. Retrieval processes consist of (1) generating program features, (2) selecting a node in level 1, (3) selecting the most similar case and (4) generating information about correspondences between the given program and the selected case.

(1) Generating program features, namely, redundant instructions and numbers of opcodes. The system removes instructions one by one and checks out the execution results to detect redundant instructions. An instruction is regarded as redundant, when the execution results are the same regardless of removing the instruction or not. The system also counts numbers of each opcode in the given program, that is, numbers of LD, LEA, ADD, etc.

(2) Selecting a node in level 1. First, the system retrieves nodes, which meet conditions of opcode numbers generated in the first process. Second, the system compares program lists of the retrieved nodes with the given program. This task is called "program matching". Finally, the system selects a node that matches the given program. If no node matches the given program, the case-based evaluation can not produce a decisive result and *unknown* is assigned as the degree of confidence.

(3) Selecting the most similar case. The system investigates nodes in level 2, which are children of the node selected in the second process, and retrieve nodes that match the given program. If there is a case whose parents in all three groups, i.e., group (a), (b) and (C) match the given program, then the case is selected. This situation is called "perfect match" and *surely* is assigned as the degree of confidence. When there is not such a case, *probably* is assigned and the most similar case is selected based on the following criteria.

- A case that has two matching parents is given priority over one that has one matching parent.
- Groups, which case's parents belong to, are given priority in the order of (a) (b) (c).

(4) Generating information about correspondences. The system performs program matching between the given program and the program list in the selected case. If the program matching succeeds, the correspondence information on instructions, labels and registers between the given program and the case is generated. If the program matching does not succeed, the correspondence information is generated by matching the given program against the case's parent node in group (a).

4.2 Matching Programs

The purpose of program matching is to investigate whether the given student's program is the same implementation as the target program lists in cases or index nodes. The system tries to make consistent correspondences of instructions, labels and registers between the given program and the target program list. If the following condition1 is met, the target program matches the given program.

- **Condition 1**: All instructions of the target program correspond to instructions of the given student's program, and all such instructions of the submitted program which do not find their counterparts in the program of the case are redundant instructions and do not affect the program's action.

In case of matching between index-nodes and the given student's program, the following condition 2 needs to be met.

- **Condition 2**: Instructions of the target program and the given student's program correspond perfectly one-to-one.

In case of a perfect match, the following condition 3 is met, in addition to conditions 1 and 2.

- **Condition 3:** The differences of the order of corresponding instructions are trivial.

The difference of the order is regarded as trivial if the difference does not affect the judgment or the advice. At the moment, we have adopted the following two rules:

- If all instructions, despite having a different order of correspondence, have the same opcode, the difference is trivial.
- If all instructions having a different order of correspondence are instructions, which set a value to registers or memory with no indexing, then the difference is trivial.

4.3 Applying Cases

If there is a case that matches the given program, the judgment of acceptability on the case is applied to the given program. In addition, advice sentences on the case are used for the given program, although the sentences should be adapted for the given program. In other words, the sentences should be modified, if needed. The modifications of advice sentences are limited to only simple ones. That is, label names, register numbers and line numbers in advice sentences on the case are replaced with corresponding ones in the given program. Such modifications are performed using correspondence information generated in the fourth process described in Section 4.1.

If no case matches the given program, the judgment and advice is not generated and *unknown* is assigned as the degree of confidence.

5. Experiments and Discussion

5.1 Experiments

The first version of the assistant system was utilized for actual classes of the CPU and assembly language course at Teikyo University in 1999. The second version was utilized for the same course in 2000. The basic idea is the same between the systems except for indexing and retrieving cases described in Section 3.2 and 4.1. We used the assistant systems in mode-1, that is, all evaluation results were sent to students after teachers made final evaluation, because it is easy to compare evaluation results between the assistant systems and teachers.

The followings are representative problems from twelve problems in 2000:

- P1: Select the bigger of the two given integers.
- P2: Sum the given N integers.
- P3: Count the number of times a certain character appears in the given character string.
- P4: Pick up capital letters from the given character string and reverse them using a stack.
- P5: Reverse the given character string using a stack, divide the reversed string into sets of two characters and pack every pair of characters into a word.

Table 1 shows the size and complexity of these five problems and case-bases. P1 and P2 are the easiest types, while P4 and P5 are the most difficult types in the course. P1 and P2 were also presented in 1999, so that the case-bases are relatively large for such simple problems. The numbers of 'cases' of P1 and P2 were 15 and 12, respectively, in 1999. The reason why the number in 'advice giving' with regard to P5 was zero is that we did not give advice on correct programs. There was little time to advise the students who finished the exercise, because we wasted too much time to lead students toward the correct programs in that class. The problem seemed too difficult for very many students, since the program should correctly run even for the null character string. The large number of submitted programs for P5 shows the difficulty for the students.

Figure 4 shows a very simple but real example of a submitted program and a case matched to the program for the problem P2. In this example, the student's program and a program in the case are almost the same except for the order of instructions. The assistant system regarded the difference of the order in lines 2 and 3 as trivial, but not the difference of the order in lines 6 and 7; so that the assistant system generated the judgment of *probably* accept. In addition, the assistant system modified line numbers in the advice sentence from 7 to 6.

Table 1 Size and complexity of programs and case-bases for five representative problems.

Problem	P1	P2	P3	P4	P5
The minimum number of lines in a program	7	8	11	17	24
The maximum number of lines of a program	12	11	17	24	33
Number of submitted programs	107	123	193	184	476
Number of programs that run correctly	79	79	123	120	88
Number of nodes in level 1	21	16	33	39	89
Number of cases (nodes in level 3)	47	38	64	64	64
Number of programs with advice given	34	10	66	42	0
Variations of advice given	9	3	9	12	0

Case

```
1:SUM     START
2:        LD      GR1,N
3:        LEA     GR0,0
4:        LEA     GR1,-1,GR1
5:LOOP    ADD     GR0,DATA,GR1
6:        LEA     GR1,-1,GR1
7:        CPA     GR1,M
8:        JPZ     LOOP
9:        ST      GR0,ANS
10:       EXIT
```

Evaluation Result: Accept
Advice:
The instruction CPA at line 7 is unnecessary, because the flag register is set by LEA.

Student's program

```
1:PRG     START
2:        LEA     GR0,0
3:        LD      GR1,N
4:        LEA     GR1,-1,GR1
5:LOOP    ADD     GR0,DATA,GR1
6:        CPA     GR1,ZERO
7:        LEA     GR1,-1,GR1
8:        JPZ     LOOP
9:        ST      GR0,ZZ
10:       EXIT
```

Evaluation result generated by the system

Evaluation Result: Probably Accept
Advice:
The instruction CPA at line 6 is unnecessary, because the flag register is set by LEA.

Figure 4 An example of a submitted student's program, a case matched and evaluation results generated by the assistant system.

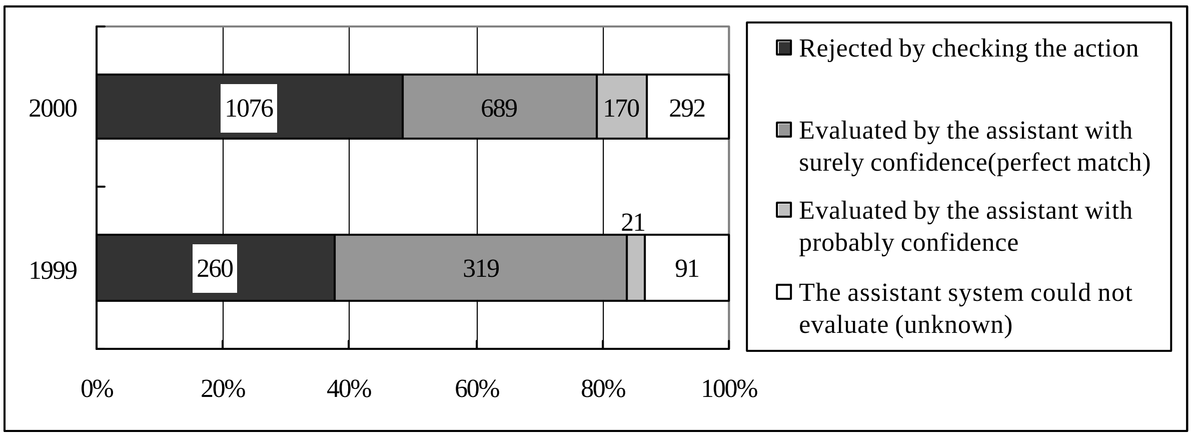


Figure 5 Results of using the implemented systems in 1999 and 2000. Numbers in the graph are numbers of programs.

During the utilization of the systems, 691 programs were submitted for five problems in 1999 and 2227 programs were submitted for twelve problems in 2000. Figure 5 summarizes how submitted programs were evaluated. Figure 5 shows checking programs' actions reduces target programs for evaluation by about 40 to 50%. Furthermore, the case-based evaluation can reduce the programs by about another 30 to 40%. As a result, target programs for evaluating by teachers can be reduced to about 20%.

The judgment accuracy is defined as the percentage of the same judgment between the case-based system and the teachers for programs evaluated by the case-based assistant, that is, "*surely*" and "*probably*" in Figure 5. The accuracy was 99.4% in 1999, and 97.6% in 2000. The judgment for five programs was different, when the assistant system evaluated programs with *surely* confidence. The cause of the difference in these five cases with *surely* confidence was just a teacher's careless mistake.

5.2 Discussion

(1) Effectiveness of the implemented system.

Figure 5 shows that the implemented system reduces teachers evaluation work drastically, even if we take account of the possibility that students submit their programs to the system more easily than to the teacher directly. That is, programs rejected by checking their actions may be fewer when students submit programs to the teacher directly. However, if the number of programs rejected by checking their actions was half of the number in Figure 5, target programs evaluated by teachers were still less than 30% of total submitted programs.

The judgment accuracy of the case-based system is sufficiently high. Teachers can trust the case-based system when it evaluates with *surely* confidence, because the system's performance on judgment is almost the same as the teachers. Our experience shows the system does not make misjudgments unless teachers make mistakes when judging *unknown* cases. The recovery of teachers' careless mistake will be our future work.

In addition to these two points, from our experience of using the system, we can say that the proposed evaluation assistant model worked well and suited activities of both teachers and students in the classes.

(2) Generality of the proposed idea.

The idea of the proposed case-based evaluation assistant system is generally available for many programming languages. However, program matching processes and indexing to cases have to be designed for target programming languages. Program understanding techniques for high-level languages have been investigated and they have produced excellent results [1]-[10]. Evaluation assistant systems for high-level languages will be implemented by adapting such program matching techniques to the proposed framework.

An important point regarding constructing program-matching functions is that the sameness of programs depends on the purpose of the matching task. For example, Ueno adopted a rule of replacing record-type variables with simple data type variables, as one of normalization rules for PASCAL programs [6]. The rule plays an important role in the purpose of their system ALPUS, that is, removing program variations for the purpose of specifying bugs and understanding buggy programs. However, in our evaluation tasks, programs with the record-type data should be distinguished from those with non-record-type data, because a teacher may intend to make students learn about a structural data type.

(3) Case-based approach to advise students on their incorrect programs.

The case-based approach is also available for giving advice to students who wrote buggy programs, although there are two major problems. The first problem is that enough cases in the case-base to cover many variations of buggy programs will not be assembled because there are many more variations than correct programs. Hence, we should investigate on using cases of programs that run properly, cases of other similar assignments and other resources. The second problem is that there is a question of how much advice teachers should give to students. It would not be good idea to give advice in detail for the first time. The advice system should give small hints at first and more advice later. We plan to tackle these problems.

6. Conclusions

We have proposed a framework of a case-based evaluation assistant of novice programs. Based on the idea, we implemented a case-based assistant system for a simple assembly language CASL and used it in actual classes; as a result, the effectiveness of the system to reduce teachers' evaluation work was demonstrated. Systems implemented based on the idea can be powerful tools for not only usual classes but also distance education. In the future, we plan to investigate a case-based approach to support students whose programs were rejected.

This research was supported in part by the Japanese Ministry of Education Grant No.11680400 and No.12780293.

References

[1] A. Adam and J. P. Laurent, LAURA, A System to Debug Student Programs, *Artificial Intelligence*, **15** (1980) pp.75-122.

[2] W. L. Johnson, Understanding and Debugging Novice Programs, *Artificial Intelligence*, **41** (1990) pp.51-97.

[3] W. R. Murray, Automatic Program Debugging for Intelligent Tutoring Systems, *Computational Intelligence*, **3** (1987) pp.1-16.

[4] Schorsch,T. , CAP: An Automated Self-Assessment Tool to Check Pascal Programs for Syntax, Logic and Style Errors, *Proc. of SIGCSE95*, (1995) pp.168 -172.

[5] H. Ueno, Concepts and Methodologies for Knowledge-Based Program Understanding - The ALPUS's Approach, *IEICE Trans. Inf. & Syst.*, **E78-D**, no.2 (1995) pp.1108-1117.

[6] H. Ueno, A Program Normalization to Improve Flexibility of Knowledge-based Program Understander, *IEICE Trans. Inf. & Syst.*, **E81-D**, no.12 (1999) pp.1323-1329.

[7] S. Kim and J. H. Kim, Algorithm Recognition for Programming Tutoring Based on Flow Graph Parsing, *Applied Intelligence*, **6**, iss.2 (1996) pp.153-164.

[8] S. Xu and Y. S. Chee, Automatic Diagnosis of Student Programs in Programming Learning Environments, *Proc. of IJCAI99*, Sweden, (1999) pp.1102-1107.

[9] T. Konishi, A. Suyama and Y. Itoh, Evaluation of Novice Programs Based on Teacher's Intention, *Proc. of ICCE95*, Singapore, (1995) pp.557-566.

[10] H. Suzuki, T. Konishi and Y. Itoh,, Applicability of an Educational System Assisting Teachers of Novice Programming to Actual Education, *Proc. of ICCE2000*, Taipei, **1** (2000) pp.128-132.

[11] H. Watanabe, M. Arai, and S. Takei, Automated Evaluation of Novice Programs Written in Assembly Language, *Proc. of ICCE99*, Chiba, **2** (1999) pp. 165-168.

[12] H.Watanabe, M.Arai, and S.Takei, Case-Based Evaluating Assistant of Novice Programs, *Proc. of ICCE2000*, Taipei, **1** (2000) pp.133 - 137.

[13] J.Kolodner, Case-Based Reasoning, Morgan Kaufmann Publishers,Inc. (1993).

[14] D. Leake ed., Case-Based Reasoning: Experiences, Lessons and Future Directions, AAAI Press / MIT Press, (1996).

Attending AI-ED2001 for presentation was supported by The International Information Science Foundation (IISF) in Japan.