Investigating Error Resolution Processes in C Programming Exercise Courses

Yuta Taniguchi
Faculty of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan
taniguchi@ait.kyushu-u.ac.jp

Atsushi Shimada
Faculty of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan
atsushi@ait.kyushu-u.ac.jp

Shin'ichi Konomi
Faculty of Arts and Science
Kyushu University
Fukuoka, Japan
konomi@artsci.kyushu-u.ac.jp

ABSTRACT

This study investigates how we can understand students' actual status in C programming exercises from their learning activity logs. In a face-to-face course of C programming exercise, it is hard for a teacher to see who are in trouble from their apperance. It is not always true that typing something means he or she is making some progress. Therefore it is important to identify, or possibly even predict, students having difficulty from their activity patterns. Most of the prior work paid attention to only trial-and-error activities, such as compile results and execution errors. However, it tends to be overlooked that knowledge acquisition process is also worthy of attention. When a student encounters a compile error, they usually read textbooks to seek a solution. It is considered to be useful for the task whether he or she has an ability to find appropriate pages for error resolution. In this paper, we propose a method to predict whether a student can resolve errors or not. Based on students' activity logs collected from our programming environment and e-book system, we conduct experiments to show and discuss the prediction performance.

1. INTRODUCTION

The C programming language has many obstacles, such as relatively complex syntax and confusing messages of compile errors. In C programming courses, students are required to overcome those hurdles as well as to learn how to solve problems computationally. Especially, because error messages are not necessarily straightforward, students are sometimes led to irrelevant pages of textbooks and get confused. Such learning experiences perhaps lower their motivation to learn, and therefore teachers need to intervene to help students in such situations. However, not all students ask their teachers or friends when they have a tough issue, and it is hard for teachers to distinguish students in trouble from ones without problems by physical appearance. Hence data-driven approaches have been considered for identifying such *at-risk* students.

Blikstein [1] and Helminen et al. [4] performed a kind of offline analysis of students' behaviors. On the other hand, a realtime-oriented analysis was performed by Fu et al. [3]. They proposed a dashboard system for teachers to grasp the current status of all students in real-time fashion. However, none of them considered the knowledge acquisition process. Helminen et al. [4] addressed the process where students struggle to resolve errors. However, in their study, only limited activities, e.g. selecting, ordering, and indenting code fragments, are analyzed, and activities such as referring external learning materials are not considered.

For understanding students' learning processes, it is significant to know how students search learning resources for necessary information and acquire knowledges. We believe students have to learn how to resolve errors by themselves, and intervention by teachers should be controlled. Hence we have to care how much effort was made by a student to obtain necessary knowledges for resolving errors. Although we could quantize such efforts to some extent by observing how they use their textbooks during programming exercises, only a limited number of studies focused on students' trial-and-error and knowledge acquisition in learning processes of programming languages.

Toward automatic and real-time detection of students in need of help, in this paper, we analyze *error resolution processes* in which students struggle to resolve compilation errors with course materials in a programming exercise course. Furthermore, we try to predict whether they can successfully resolve errors or not in early stage of error resolution processes. Toward this end, we employ both compilation logs and page view logs of e-textbooks, and characterize compilation errors in relation to exercise questions and individual students.

2. METHODS

2.1 Error Resolution Processes

The C Programming Language belongs to the compiled languagegs, which need a compile process ahead of executing a program. In the compile process, a compiler program transform input source code into a machine code. The process involves a lot of checks and many problems are found as *er-*

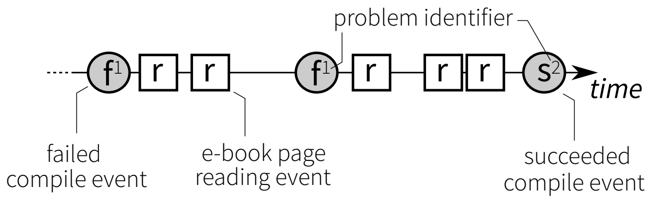

Figure 1: An example of a timeline. A timeline may consist of three types of events: failure compile event, success compile event, and e-book page reading event.

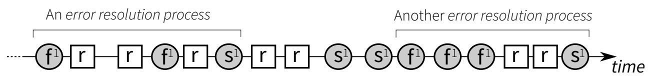

Figure 2: An example of an error resolution process found in a timeline. Every ERP starts with a failure compile event, but does not necessarily end with a success compile event.

rors before running a program. Errors found in this step could be typically divided into syntax errors and linker errors. The former type of errors requires a student to modify their source code so that it follows the syntax of C language. In the latter case, it usually occurs because of missing libraries and one need to specify required libraries on the command line (or a compiler's configurations). In this study, we call such a process as an *error resolution process (ERP)*, and analyze how students resolve such errors by themselves during exercises.

A sequence of a student's activities is called a *timeline*. A timeline consists of events, and we consider three kinds of events in this study: *failure compile event*, *success compile event*, and *e-book reading event*. The first two types occur when a student compiles his or her source code. If a compile finishes without printing any messages on a screen, we consider it is a success compile event. On the other hand, it is considered as a failure compile event if a compiler outputs some messages. An e-book reading event indicates a student started to read a page of an e-book.

These concepts are illustrated in Figure 1. The figure shows a sequence of eight events from left to right. Every complie events are associated with a problem that a student try to solve. Identifiers of problems are shown at the top right of each event. For this example, one of the possible interpretations of the timeline is as follows; a student were working on the problem 1 and compiled their program for it; however, they got an error from the compiler, and then they rewrite a program according to textbooks and tried again; unfortunately, it did not work well, and he or she abandoned the problem and began the problem 2; they compiled another program, and it successfully finished.

We more precisely describe an ERP based on a timeline. An ERP is a contiguous subsequence of a timeline; an ERP always starts with an failure compile event, and it can include only compile events associated with the same exercise problem. For example, Figure 2 shows two ERPs within a timeline. Since the timeline does not involve another problem in

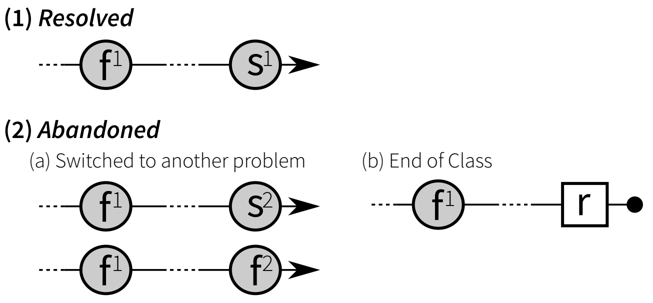

Figure 3: Possible cases of ERPs whose result could be either *resolved* or *abandoned*.

this case, we can simply identify ERPs as longest contiguous subsequences starting with a failure compile event and containing at most a single success compile event at the end of the subsequence. Another example is shown in Figure 1 which consists of all except the last event.

While several scenario could be considered for ERPs, we simply consider two kinds of outcome from an ERP: *resolved* and *abandoned*. Figure 3 shows examples for both cases. A resolved ERP ends with a success compile event associated with the same problem as the ERP's initiating failure compile event. An abandoned ERP could finish with either an e-book event or a failure compile event. The latter type of ERPs appears when a student switch a current exercise problem to another or a class is over.

2.2 Predicting Outcomes from ERPs

It might be an important information for teachers whether a student's ERP will end up with *resolved* or *abandoned* state. We consider the prediction of the outcome from an ERP in early stage of the process. To characterize EPRs, we employ n-gram features. Firstly, we obtain a textual representation of an ERP. In the representation, every event is notated as a single letter. In this paper, we use `s` for a success compile event, `f` for a failure compile event, `c` for an event of reading e-textbooks especially for the programming course, `o` for an reading event but for other course materials. Furthermore, we also encode gaps between events with `-` which denotes a gap of just 10 seconds. Any gaps less than 10 seconds are ignored. For example, a gap of 24 seconds is notated as `--`.

N-gram features are then extracted from the text representation of a timeline. As there are five letters in the representation, an n-gram feature vector will be a 5^n-dimensional vector. For example, if we use 3-gram-based features, a student is represented as a 125-dimensional vector. In this study, we combine 1- to 4-gram feature vectors into a single feature vector, i.e. $5^1 + 5^2 + 5^3 + 5^4 = 780$-dimensional vector, and use it for making prediction.

Students' first actions are considered important; for example, an expert will read error messages carefully, fix problems, and then recompile the program while a beginner will just repeat the compile or read many pages of textbooks seeking for a solution. We expect such differences are captured by n-gram representations of the first t minutes.

In our experiments, we targeted only ERPs longer than

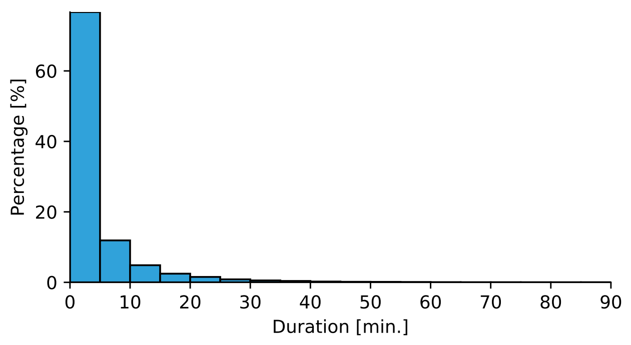

Figure 4: The distribution of durations of error resolution processes.

Table 1: Hyperparameters adjusted.

Hyperparameter	Candidates
criterion	gini, entropy
max depth	1, 2, 3, 4, 5, 6, 7, 8
max features	sqrt, log2, None

$2t$ minutes. If we use the first t minutes for predicting the outcomes of ERPS whose durations are less than $2t$ minutes, it means that we use more than a half of information to predict the final status of ERPs. That is why we exclude ERPs shorter than $2t$ minutes in this study.

We set $t = 5$ in our experiments. There are 29,216 ERPs in our dataset. Figure 4 shows the distribution of the durations ERPs took. The number of ERPs which only took less than or equal to 10 minutes was 25,886 (88.6%), and the number of our target ERPs was 3,330 (11.4%).

For the prediction, we employ random forest classifiers as a classifier. Random forests are one of the ensemble learning algorithm based on bagging technique and random selection of features [2]. This is one of the most popular classifiers not only in educational datamining community but also in wider data science community. It is known that the classifier is robust even when each dimension of feature vectors has different scales. Therefore, we simply apply random forests to n-gram feature vectors without normalizing them.

We use the implementation of random forest classifier included in scikit-learn package [6] in this study. Basically, we use the default values of the library for the classifier's hyperparameters except for ones shown in Table 1 and the number of estimators which was set to 100. We optimize these parameters by a grid search algorithm, whose implementation is also provided by scikit-learn. The candidate values for these parameters are shown in the table.

2.3 Data Collection

The course we target is the introductory courses for C programming language in our university. Primarily, all freshmen students takes the course. The class is composed of lectures and coding exercises. There are about 20 classes for the course for each year, and almost all of the courses are taught by different teachers. Although it is not enforced, we

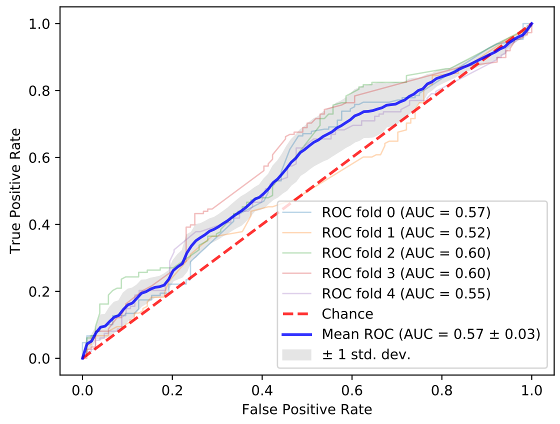

Figure 5: The ROC curve obtained from 5-fold cross validation.

have a set of standard course materials, and they are used in almost all classes.

In exercise, we use the compiler program "gcc", from the GNU compiler collection. The compiler program is modified from the original version so that it can record students' learning logs. More precisely, when it is executed, it saves given commandline arguments, the contents of given source files, and the output of the compiler as well as the time and student IDs. Since a commandline and source code are available as logs, we can reproduce what a student tried and what he or she obtained as a result. From those learning logs, we reconstruct compile events in timelines.

We also utilize students' learning logs on our own web-based e-book system BookRoll [5]. All the course materials are available on the system, and therefore we can collect students' learning logs about how student utilized those textbooks during exercises. Students' actions, such as flipping pages and adding highlights, on the system are collected immediately as events occur. In this study, we only focus on page-flipping events collected for our analysis.

3. RESULTS & DISCUSSION

We evaluated our prediction performance using AUC metric, which stands for area under the ROC curve, combining with 5-fold cross validation. The hyperparameter values obtained by the grid search algorithm were `criterion=gini`, `max_depth=6`, `max_features=sqrt`, `n_estimators=10`. Figure 5 shows the five ROC curves obtained during cross validation and the mean ROC curve computed from them.

Table 2 shows the average performance scores with four evaluation metrics. According to the table, we cannot say the perfromance is good. Althogh the recall value is relatively high, precision value (especially in the test phase) is low. AUC value for the test phase is better than the chance rate, it is not good enough for practical use.

Table 2: Cross validation result. Values are averaged over five folds.

Metric	Value
AUC (test)	0.566
F1 (test)	0.698
Precision (test)	0.598
Recall (test)	0.844
AUC (train)	0.684
F1 (train)	0.740
Precision (train)	0.640
Recall (train)	0.882

4. CONCLUSION

Focusing on students' error resolution processes, we proposed a predction method for outcomes from those process. We encode event sequences into texts, and characterize them using n-gram features. From our preliminary analysis, the proposed method could not show a good performance while it has a little bit better performance than the chance rate.

Future work include improvement of the feature representation of timelines and further analysis on the characteristics of student's activities and abilities.

5. ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP17K12804.

6. REFERENCES

[1] P. Blikstein. Using learning analytics to assess students' behavior in open-ended programming tasks. In *Proceedings of the 1st International Conference on Learning Analytics and Knowledge*, pages 110–116. ACM, 2011.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[3] X. Fu, A. Shimada, H. Ogata, Y. Taniguchi, and D. Suehiro. Real-time learning analytics for c programming language courses. In *Proceedings of the Seventh International Conference on Learning Analytics & Knowledge*, LAK '17, pages 280–288, New York, NY, USA, 2017. ACM.

[4] J. Helminen, P. Ihantola, V. Karavirta, and L. Malmi. How do students solve parsons programming problems?: An analysis of interaction traces. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 119–126, New York, NY, USA, 2012. ACM.

[5] H. Ogata, Y. Taniguchi, D. Suehiro, A. Shimada, M. Oi, F. Okubo, M. Yamada, and K. Kojima. M2b system: A digital learning platform for traditional classrooms in university. *Practitioner Track Proceedings of the Seventh International Conference on Learning Analytics & Knowledge*, pages 155–162, 2017.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.