PROVIDING USEFUL FEEDBACK TO BEGINNER PROGRAMMERS

Peter Ward
Associate Professor James Curran & Professor Judy Kay
School of Information Technologies
Faculty of Engineering & Information Technologies

Introduction

- Learning programming is difficult, and beginners often struggle: feedback is important to ensure they don't give up.
- Students submit programs to solve a problem, instructors will have one (or more) model solutions.

```
1  num = int(raw_input('Bank balance: '))
2  if num < 0:
3      print 'In the red!'
4  else:
5      print 'In the black.'
```

Figure 1: *A model solution to a problem*

Background

- A program can be represented as an Abstract Syntax Tree (AST), for example Figure 2.

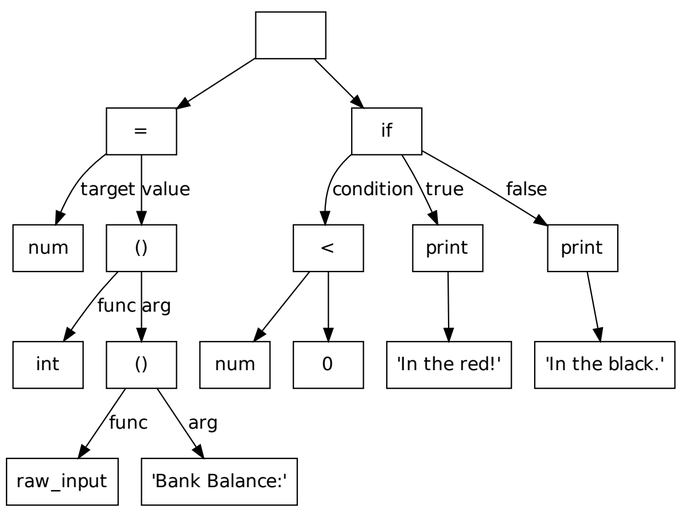

Figure 2: *The AST representation of Figure 1*

- The AST of a submission and a model solution can be compared to get a similarity score $0 \leq s \leq 1$, which correlates well with marks assigned by human markers. [1]

Generating Feedback

- A key design goal of the system was to minimise the amount of work required to be done by the instructor.
- The feedback we aim to generate will identify *where* their solution needs to be improved, and *what* needs to be done to it.
- To do this, the nodes of the student's solution's AST are matched with the nodes of the model solution's AST.
- The nodes of the model solution are annotated with *section labels*, and this is then used to calculate metrics for how well each section matches, and how much missing content there is.
- For example, consider the student's submission in Figure 3. This is a valid program, but due to the mixing of types when comparing `num` and `0` on line 2, it fails to work as expected. However, the student's actual error is failing to convert the input to an integer, not the comparison.

```
1  num = raw_input('Bank balance: ')
2  if num < 0:
3      print 'In the red!'
4  else:
5      print 'In the black.'
```

Figure 3: *A student's submission to a problem*

Matching programs

- To match the nodes of the two trees, we score each possible assignment of nodes between the two trees, and take the maximum scoring assignment.
- Assignments are limited, such that if a node in the left tree has been assigned to a node in the right tree, the descendants of the left node can only be assigned to descendants of the right node.
- This means the algorithm is polynomial, not exponential, and allows for a dynamic programming algorithm with time complexity $O(5)$.
- Since the ASTs are relatively small, and do not have a high branching factor, this is fast enough in practice (< 1 second / problem).
- The assignment of nodes is then used to identify the sections of code in the student's program. Figure 4 shows an example of the matching for the section of code which reads a number.

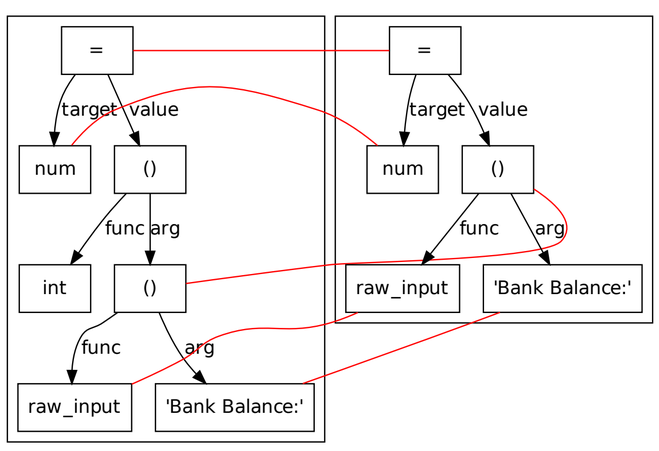

Figure 4: *Matching of the problematic section of code*

Identifying errors

- We extract several features for each section of code in the matching:
- `score` is the average score between the matched nodes.
- `unmatched left` and `unmatched right` is the percentage of nodes on the left and right trees which are not matched, respectively.
- `relsize` is the number of nodes in the student's subtree, divided by the number of nodes in the solution's subtree.
- An appropriate machine learning model can then be built to identify problematic sections of code.

Experiments

NCSS Challenge 2012

- A mockup version of the system was implemented for the 2012 NCSS Challenge, a programming competition for high school students.
- Programming problems and notes to help solve them were released weekly.
- Simple detection routines for missing code were written: missing input or output statements, conditionals, loops, and some problem-specific detectors.
- Hints were given immediately to the students, below the results of the automatic testing (indicating whether they passed or failed).
- Each hint had two buttons to mark the hint as helpful or not helpful.
- Table 1 shows the percentage of students retained between weeks (e.g., the percentage of students who submitted something in Week 1 that went on to submit something in Week 2), broken down by whether they
  1. did not get a hint in that week;
  2. got a hint, but did not mark it as helpful or not.
  3. got a hint, and marked it as helpful or not.

Hints	Week 1	Week 2	Week 3	Week 4
none given	80.6	76.9	75.8	64.7
given, not acknowledged	84.9	77.7	83.3	73.3
given, acknowledged	80.9	79.9	87.2	77.7

Table 1: *Percentage of students retained between weeks*

Matching programs

- We used data from the 2011 NCSS Challenge to evaluate our system.
- A random sample of submissions from three questions from the first two weeks were selected, and each section of code was annotated with a `good` or `bad` label, indicating if it was a problematic section of code.
- Using an equal split of `good` and `bad` sections, a C4.5 classifier achieved an accuracy of 76.7%.

Future work

- Use multiple model solutions to support a wider diversity of student submissions.
- After identifying a bad section of code, suggest an appropriate transformation to fix the code (e.g., add an `int()` call).

References

[1] Kevin A. Naudé, Jean H. Greyling, and Dieter Vogts. "Marking student programs using graph similarity". In: *Computers & Education* 54.2 (Feb. 2010), pp. 545–561.