Control-Flow-Only Abstract Syntax Trees for Analyzing Students' Programming Progress

David Hovemeyer, York College of Pennsylvania
Arto Hellas, University of Helsinki
Andrew Petersen, University of Toronto Mississauga
Jaime Spacco, Knox College

Introduction

- (Online) programming platforms are capturing lots of data about student work on exercises and assignments
  - Submissions
  - Test results
  - Compiler errors and warnings
  - Fine-grained edits (maybe)
- What to do with this data?
  - What can it tell us about student behavior?
  - Can it help us identify students who are struggling?
- Lots of previous work
  - Jadud, ICER 2006, Methods and Tools for Exploring Novice Compilation Behaviour
  - See ITiCSE 2015 Working Group report

What can the *code* tell us?

- Much previous work has focused on artifacts derived from student code
  - Execution results (compilation errors, static analysis warnings, test results)
  - Aggregate information (LOC, edits)
- Our thought: can we find a useful way to analyze the code itself?
  - Look deeper into program structure and semantics
  - But abstract away "less interesting" details
- Focus on control flow
  - Traditional source of difficulty for students learning to program

CFASTs

- CFAST = "Control-Flow-only Abstract Syntax Tree"
  - Start with the AST for a function/method
  - Retain only intraprocedural control-flow structures (if/else/for/while/break/etc.)
- Example:

```
def insert(lst, v):
  if v > max(lst):
    lst.insert(0, v)
  else:
    lst.reverse()
    for i in range(len(lst)):
      if (v < lst[i]):
        lst.insert(i, v)
        break
```

```
FunctionDef
  If
  Else
    For
      If
        Break
```

CFASTs and correctness

- A CFAST can only be constructed from a syntactically correct program
  - So, a CFAST-based analysis won't see submissions which don't compile
- A "correct" CFAST is one which was observed in at least one completely correct program (all tests passed)
  - A program with a "correct" CFAST isn't necessarily correct!
  - But it might be on track to becoming a correct program

Research questions

1. Do CFASTs encode useful information about student programming behaviour?
2. Can CFASTs be used to identify students in difficulty?

Data sets

We analyzed data from three CS 1 courses:

1. CS 1 at University of Toronto
2. CS 1 at University of Helsinki
3. CS 1 at York College

Course	Total # activities	Concepts addressed if	loops	both
1	9	0	4	5
2	9	5	2	2
3	9	4	5	0

What is in the data?

- Code snapshots for explicit student submissions
  - Students received feedback after every submission
- Results from unit tests

- The problems are only a *subset* of the exercises presented to students
  - Problems focusing on conditionals and loops were selected
- The problems served different purposes in each course
  - Course 3 (York College): quick drill and practice targeting basic concepts
  - Courses 1 and 2 (Toronto and Helsinki): more challenging problems

Limitations

- The problems analyzed are a small subset from early in the course
  - Late course topics, which may feature heavily on exams, are not explored
- Blind to individual contexts: we can see *what* students did but not *why*
  - We assume submission behaviour is primarily influenced by a desire to solve the problem, but that may not be the case (e.g., network connectivity issues)
- Evaluation of ability is based on exam scores
  - The only common metric, but also one with different meaning at each institution

Interesting finding 1

For many exercises, most submissions are covered by a small number of CFASTs.

The exceptions are problems with (relatively) complex decision structures.

Course/ Activity	# distinct (w/ correct)	% in top 20% CFASTs
1/37	341 (101)	89.3
1/39	40 (9)	98.3
1/45	190 (43)	95.0
1/47	979 (207)	75.4
1/48	86 (14)	97.1
1/59	239 (45)	95.1
1/63	491 (143)	83.2
1/64	180 (69)	90.2
1/84	232 (97)	88.1
2/018	7 (3)	98.9
2/021	12 (5)	96.6
2/023	5 (3)	95.6
2/024	17 (8)	85.7
2/026	15 (7)	94.1
2/027	36 (7)	93.4
2/029	142 (25)	78.6
2/035	27 (6)	94.4
2/041	96 (26)	69.6
3/111222333444	39 (5)	93.9
3/bananana	9 (3)	98.6
3/checkinput	32 (8)	90.9
3/doublecoupon	9 (4)	36.0
3/keepdoubling	22 (10)	89.2
3/memberdiscount	31 (12)	84.4
3/restaurant	18 (7)	80.7
3/squares	22 (9)	87.3
3/triplecoupon	12 (7)	71.8

Interesting finding 1

For many exercises, most submissions are covered by a small number of CFASTs.

The exceptions are problems with (relatively) complex decision structures.

Course/ Activity	# distinct (w/ correct)	% in top 20% CFASTs
1/37	341 (101)	89.3
1/39	40 (9)	98.3
1/45	190 (43)	95.0
1/47	979 (207)	75.4
1/48	86 (14)	97.1
1/59	239 (45)	95.1
1/63	491 (143)	83.2
1/64	180 (69)	90.2
1/84	232 (97)	88.1
2/018	7 (3)	98.9
2/021	12 (5)	96.6
2/023	5 (3)	95.6
2/024	17 (8)	85.7
2/026	15 (7)	94.1
2/027	36 (7)	93.4
2/029	142 (25)	78.6
2/035	27 (6)	94.4
2/041	96 (26)	69.6
3/111222333444	39 (5)	93.9
3/bananana	9 (3)	98.6
3/checkinput	32 (8)	90.9
3/doublecoupon	9 (4)	36.0
3/keepdoubling	22 (10)	89.2
3/memberdiscount	31 (12)	84.4
3/restaurant	18 (7)	80.7
3/squares	22 (9)	87.3
3/triplecoupon	12 (7)	71.8

Interesting finding 2

Trial and error behaviour, as identified by long CFAST chain length, was not (necessarily) a significant predictor of exam performance.

		Correlation	
Course	Exam type	rho	p-value
1	Final written exam	-0.11	0.008
2	Final written exam	-0.17	0.34
3	*Programming, 2nd midterm*	*-0.40*	*0.003*

Since low path length may indicate both high skill and low tenacity, simple metrics, like path length are not indicative. Features of the paths may be more interesting.

Interesting finding 2

Trial and error behaviour, as identified by long CFAST chain length, was not (necessarily) a significant predictor of exam performance.

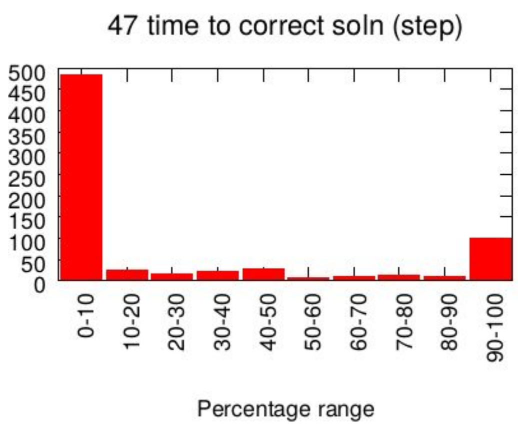
		Correlation	
Course	Exam type	rho	p-value
1	Final written exam	-0.11	0.008
2	Final written exam	-0.17	0.34
3	*Programming, 2nd midterm*	*-0.40*	*0.003*

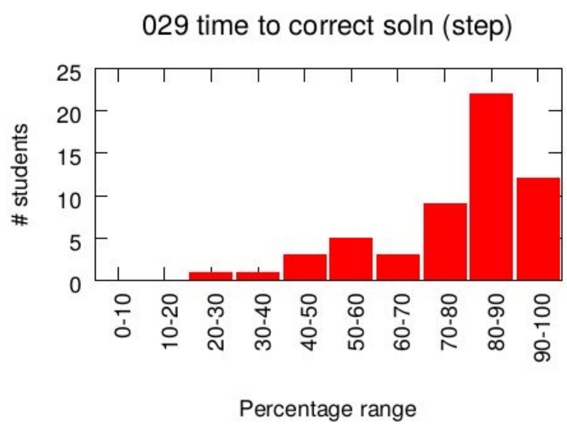
Since low path length may indicate both high skill and low tenacity, simple metrics, like path length are not indicative. Features of the paths may be more interesting.

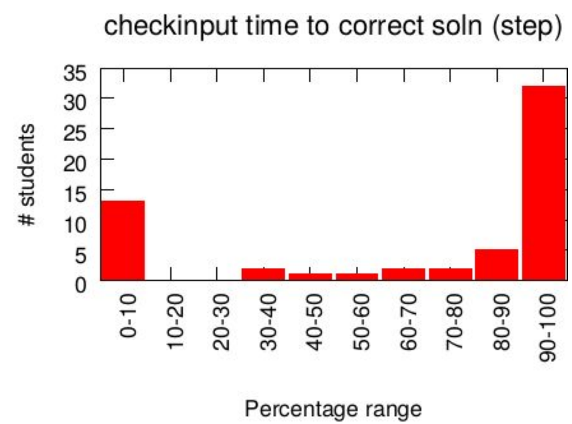
Path lengths may be significant for simpler exercises?

Interesting finding 3

For the most part, students arrive at the CFAST they submit fairly early.

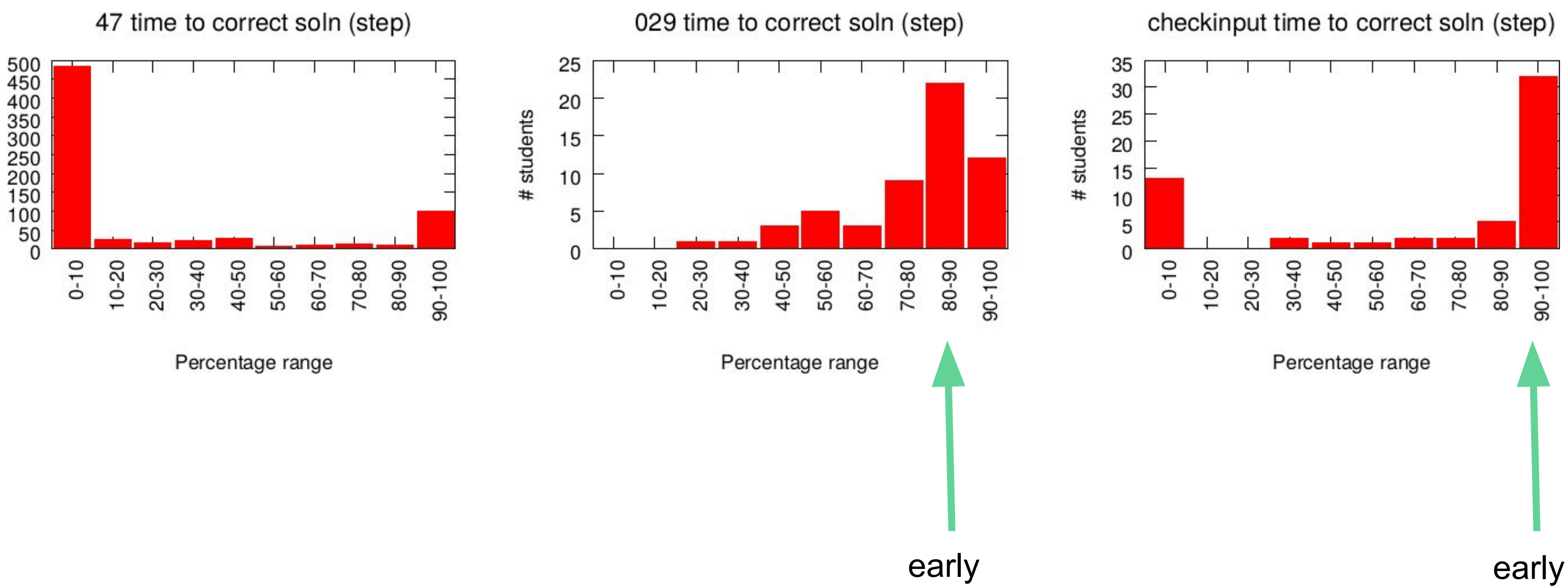






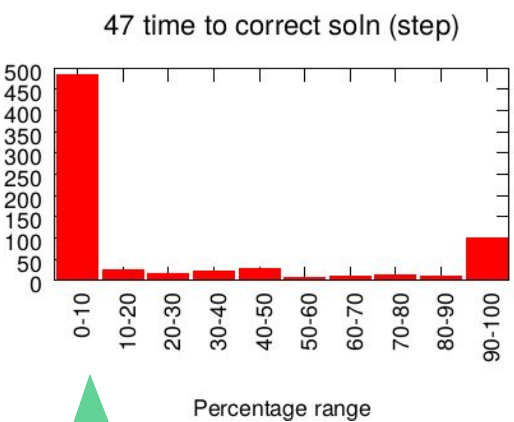
Interesting finding 3

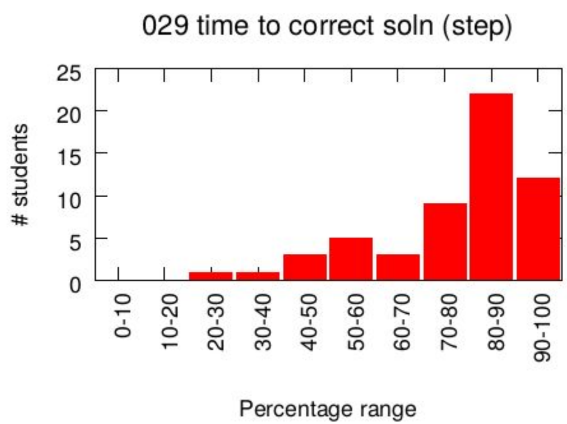
For the most part, students arrive at the CFAST they submit fairly early.

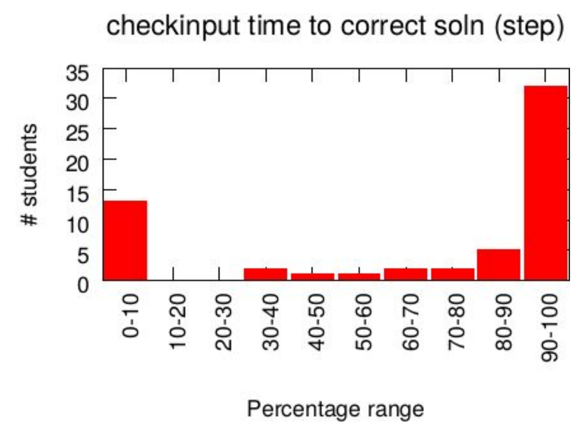


Interesting finding 3

For the most part, students arrive at the CFAST they submit fairly early.







late?

Interesting finding 3 (continued)

These largely represent two cases: a student submitting the correct CFAST in a first attempt (“late” in the chain), and a student submitting the correct CFAST early and then tinkering to get it correct: this suggests that control structures are set *early* in the process of solving the exercises.

Course 1 does not follow this trend. Students tend to change control structure more frequently in this course.

Conclusions

- Our goal was to explore whether attributes of the code, rather than results from compiling or executing the code, are useful for understanding student behaviour.

- We chose to explore the control flow embedded in the code.
- We also looked at sequences of submissions.

- CFASTs provide interesting insights into student behaviour.

Future work

- Include more information in CFASTs (e.g., loop bounds)
- Look at how is control flow added (top-down? bottom-up?)
- Use CFASTs to find characteristic solutions
- Applications?