A Data-Driven Analysis of Informatively Hard Concepts in Introductory Programming

R. Paul Wiegand

Anthony Bucci

Amruth N. Kumar

Jennifer L. Albert

Alession Gaspar

Presented by: Sterling McLeod

Introduction

- “Attrition in introductory programming courses is legendary.”

- What are the hard and easy concepts in such a class?

- Typical measures are not thorough
    - If all students miss a problem, we don’t gain much info about what students don’t understand
    - What if it is a pre-requisite concept that most students don’t understand? What if half understand the pre-requisite concepts, but not the main concept?

- This paper proposes a data-driven approach to identify easy and hard concepts

Analysis Goals

- Problem classifications:
  - **Informatively hard** – most often solved *incorrectly* by most *dominant* students
  - **Informatively easy** – most often solved *correctly* by the most *dominated* students

- Student classifications:
  - **Most dominant student** – one who solved the most problems in that concept
  - **Most dominated student** – one who solved the fewest problems in that concept

- Use Dimension Extraction Coevolutionary Algorithm (DECA) to identify structural relationships among students and problems

Co-optimization

- Co-optimization is different from typical optimization because there are multiple types of entities
  - We are interested in both students and problems

- Any student can attempt any problem(s) and receive a score
  - Many-to-many relationship between students and problems

- Decompose information (scores of students) into two coordinate systems

- “Problem Coordinate System” – each axis is a concept
- “Student Coordinate System” – each axis a subset of students, i.e. a learner type

Dimension Extraction

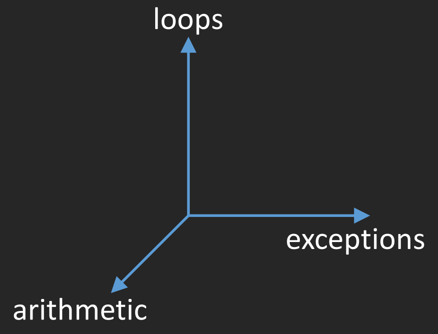
- Dimension Extraction Coevolutionary Algorithm (DECA)
- Normally: searches through *candidate solutions* and *tests*
  - But only interested in the dimension-extraction part to analyze students and problems

- *Problem Analysis*
  - Students are candidates and problems are the tests
  - Each dimension corresponds to a different concept
  - Problems further on axis are harder -> higher value = less # of students got it correct

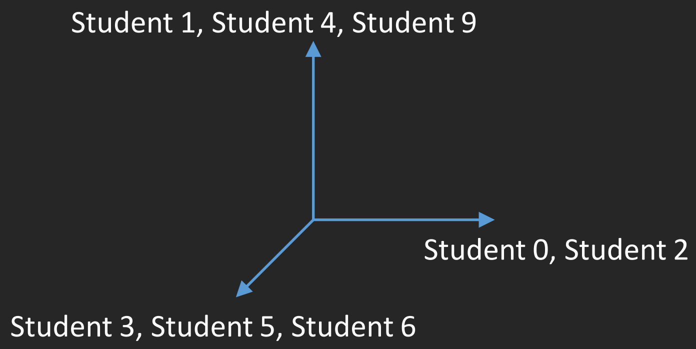
- *Student Analysis*
  - Problems are candidates and students are the tests
  - Students furthest along axis solved the most problems
  - Each dimension corresponds to a different pattern in which students performed on problems, i.e. a “type” of learner

Dimension Extraction

- *Problem Analysis*
  - # of extracted dimensions gives measure of distinct concepts in a problem set



- *Student Analysis*
  - # of ways students can be grouped based on performance



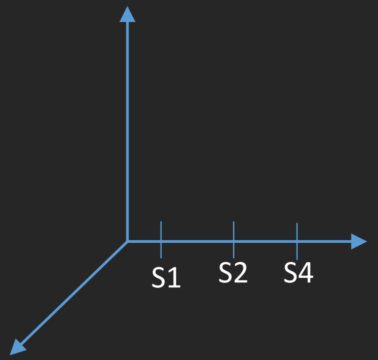
- Together, they give a notion of how *informative* problems are

Student Analysis

- Student analysis axes are groups of students that perform comparably

Problems: P1 P2 P3 P4 P5 P6

Students: S1 S2 S3 S4 S5 S6

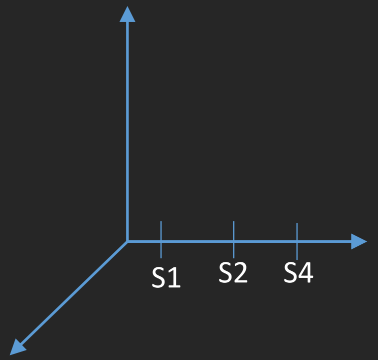


Student Analysis

- Student analysis axes are groups of students that perform comparably

Problems: P1 P2 P3 P4 P5 P6

Students: S1 S2 S3 S4 S5 S6



<u>Correct Problems</u>

S1: P1, P2

S2: P1, P2, P3

S4: P1, P2, P3, P4

Easy to compare performance

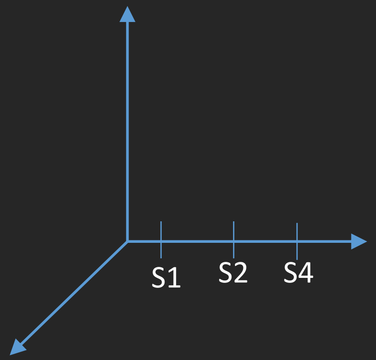
Each student along axis got the **same problems** right as previous students

Student Analysis

- Student analysis axes are groups of students that perform comparably

Problems: P1 P2 P3 P4 P5 P6

Students: S1 S2 S3 S4 S5 S6



<u>Correct Problems</u>
S1: P1, P2
S2: P1, P2, P3
S4: P1, P2, P3, P4

Easy to compare performance

Each student along axis got the **same problems** right as previous students

S3: P1, P5

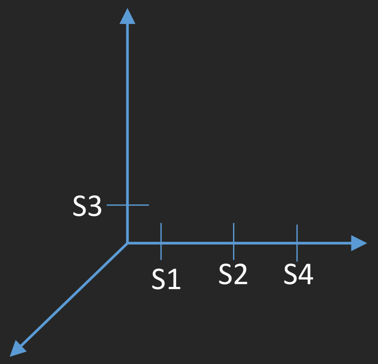
How do we compare to S1, S2, S4?

Student Analysis

- Student analysis axes are groups of students that perform comparably

Problems: P1 P2 P3 P4 P5 P6

Students: S1 S2 S3 S4 S5 S6



<u>Correct Problems</u>
S1: P1, P2
S2: P1, P2, P3
S4: P1, P2, P3, P4

Easy to compare performance

Each student along axis got the **same problems** right as previous students

S3: P1, P5

How do we compare to S1, S2, S4?

Each axis is a group of students that seem to understand the material in a non-contradictory way

Problets

- http://problets.org/
- Software assistants for programming problems
- Generate a problem, grade student solution, explain the correct solution, and log data
- Each topic has repository of 200+ problems categorized into concepts

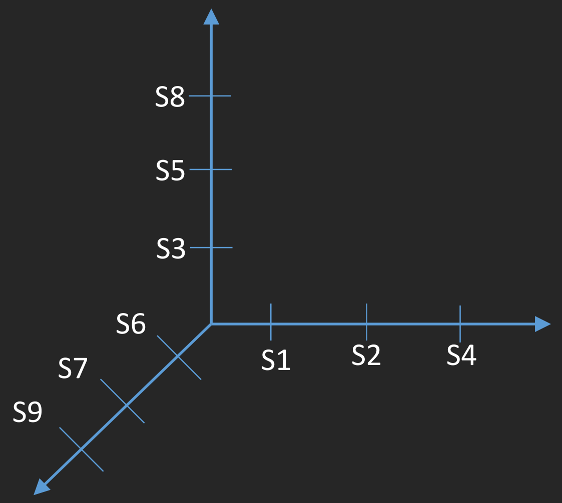
Topic	Concepts	Schools	Students
Arithmetic	25	29	982
Relational	24	20	582
Logical	21	16	572
Assignment	19	15	599
Selection	12	21	774
Switch	12	12	221
while	9	18	523
for	10	18	700
do-while	15	13	175
Advanced Loops	13	8	161
Functions/Bugs	9	7	282
Functions/Tracing	10	8	449
Array	14	14	221
Class	18	4	125

Table 1: Concepts per topic and number of schools/students who used each problet in Spring 2014

Informatively Hard/Easy

- **Informatively hard** – most often solved *incorrectly* by most *dominant* students
- **Informatively easy** – most often solved *correctly* by the most *dominated* students

- For each dimension, the ***most dominant student*** is the student **furthest** along the corresponding axis
  - S4, S8, S9
- For each dimension, the ***most dominated student*** is the **first** student along the corresponding axis
  - S1, S3, S6



Informatively Hard/Easy

- These two measures do NOT coincide with their traditional counterparts
  - Easy: Problem that the most students got correct
  - Hard: Problem that the most students got incorrect

- It is possible for a problem to be missed by everyone but the strongest students OR for a challenging problem to be solved by the weakest student

- Student behavior is not linear expectation

- Informatively hard/easy highlight and preserve the idea that different **students understand different concepts in different ways**

Results

- Study done in Spring 2014
- 14 topics
- C++, Java, and C#
- Hard/Easy – most number of incorrect/correct responses
- Informatively Hard/Easy – Collect *<u>all</u>* most dominant/dominated students, find the problems that *<u>all those students</u>* fail/pass the most

Topic	Concepts	Schools	Students
Arithmetic	25	29	982
Relational	24	20	582
Logical	21	16	572
Assignment	19	15	599
Selection	12	21	774
Switch	12	12	221
while	9	18	523
for	10	18	700
do-while	15	13	175
Advanced Loops	13	8	161
Functions/Bugs	9	7	282
Functions/Tracing	10	8	449
Array	14	14	221
Class	18	4	125

Table 1: Concepts per topic and number of schools/students who used each problet in Spring 2014

Results

- **Arithmetic Expressions**
  - **Easy**
    - Evaluating expressions such as 10 - - 5
    - Fully parenthesized expressions such as (14/((2 – 1) + 4)
  - **Hard**
    - Modulus operator embedded in expression, such as 12%5+5%12.3
      - Unique to programming
  - **Informatively Easy**
    - Divide-by-zero expressions, 9/3/0
  - **Informatively Hard**
    - Integer division, 5/3+3*5
      - Unique to programming

Results

- **Logical Expressions**
  - **Easy**
    - Evaluating fully parenthesized expressions, (true && (true && (true || false))) that do not require precedence rules
  - **Hard**
    - Evaluating C++ expressions wherein numerical values are used as Boolean operands, 3*0&&3+0
  - **Informatively Easy**
    - Evaluating fully parenthesized expressions that do not require precedence rules
  - **Informative Hard**
    - Evaluating fully parenthesized expressions that do not require precedence rules

- Even though fully parenthesized expressions are **easy**, the best students still make mistakes when evaluating them

Results

- **While loops**
  - **Easy**
    - Predicting the output of a loop that iterates only once
    - Loop that never iterates because condition is false on first try
  - **Hard**
    - Identifying the output of multiple back-to-back loops where behavior depends on previous iteration
  - **Informatively Easy**
    - Predicting output of nested loops
  - **Informatively Hard**
    - Predicting the output of nested loops when inner loop behavior depends on the outer loop
    - Loop that never iterates because condition is false on first try

Results

- **Functions**
  - **Easy**
    - Tracing output when function is part of an expression
  - **Hard**
    - Predicting output when variable is passed-by-value to a function (C++ only)
    - Identifying bug when a return statement is missing
  - **Informatively Easy**
    - Tracing behavior of a function that has multiple conditional return statements
  - **Informatively Hard**
    - Not identifying a bug when two variables have the same name appear in two different functions
    - Identifying the C++ bug where a function is called before it is defined or prototyped

Results

- **Arrays**
  - **Easy**
    - Specifying the contents of a fully initialized array
  - **Hard**
    - Identifying the contents of an array declared with incomplete initialization
  - **Informatively Easy**
    - Predicting behavior when an element of an array is referenced before it is initialized
  - **Informatively Hard**
    - Identifying type mismatch when an array is passed as parameter to a function
- Although students are familiar with parameter passing and type mismatch by now, applying those concepts to arrays is still challenging

Thoughts

- Informatively easy/hard concepts *rarely overlap* with traditionally easy/hard concepts
- Some concepts intuitively thought to be hard turned out to be informatively easy (e.g. independently nested while loops)
- Using DECA to identify informatively easy/hard concepts can bring many new insights to teaching introductory programming courses
- This study can help lead to a concept inventory for introductory programming

Questions?