Identifying Informatively Easy and Informatively Hard Concepts

R. PAUL WIEGAND, Winthrop University, USA

ANTHONY BUCCI, Independent, USA

AMRUTH N. KUMAR, Ramapo College of New Jersey, USA

JENNIFER ALBERT, The Citadel, USA

ALESSIO GASPAR, University of South Florida, USA

In this article, we leverage ideas from the theory of coevolutionary computation to analyze interactions of students with problems. We introduce the idea of *informatively* easy or hard concepts. Our approach is different from more traditional analyses of problem difficulty such as item analysis in the sense that we consider Pareto dominance relationships within the multidimensional structure of student–problem performance data rather than average performance measures. This method allows us to uncover not just the problems on which students are struggling but also the variety of difficulties different students face. Our approach is to apply methods from the Dimension Extraction Coevolutionary Algorithm to analyze problem-solving logs of students generated when they use an online software tutoring suite for introductory computer programming called *problets*. The results of our analysis not only have implications for how to scale up and improve adaptive tutoring software but also have the promise of contributing to the identification of common misconceptions held by students and thus, eventually, to the construction of a concept inventory for introductory programming.

CCS Concepts: • **Social and professional topics** → **Computer science education**; • **Applied computing** → **Education**;

Additional Key Words and Phrases: Introductory programming education, dimension extraction, multiobjective optimization, performance analysis, learning analytics, problets

ACM Reference format:

R. Paul Wiegand, Anthony Bucci, Amruth N. Kumar, Jennifer Albert, and Alessio Gaspar. 2021. Identifying Informatively Easy and Informatively Hard Concepts. *ACM Trans. Comput. Educ.* 22, 1, Article 7 (October 2021), 28 pages.

<https://doi.org/10.1145/3477968>

This work is supported by the National Science Foundation, under grant #2038406, #1504634 and #1502564.

Authors' addresses: R. P. Wiegand, Winthrop University, 701 Oakland Avenue, Rock Hill, SC; email: wiegandrp@winthrop.edu; A. Bucci, 32 Speridakis Terrace, Cambridge, MA; email: abucci@bucci.onl; A. Kumar, Ramapo College of New Jersey, 505 Ramapo Valley Road, Mahwah, NJ; email: amruth@ramapo.edu; J. Albert, The Citadel, 171 Moultrie Street, Charleston, SC; email: jennifer.albert@citadel.edu; A. Gaspar, University of South Florida, 4202 E. Fowler Avenue, Tampa, FL; email: alessio@usf.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.

1946-6226/2021/10-ART7 \$15.00

<https://doi.org/10.1145/3477968>

1 INTRODUCTION

Our work can be seen as part of the ongoing efforts to leverage data-driven approaches to improve both our understanding of what makes computing difficult for students, as well as indirectly provide insights to further improve teaching techniques. In this article, we discuss a novel data-driven approach to describing problem difficulty in an introductory programming course. In addition to easy and hard concepts, as traditionally understood, we propose the notion of *informatively easy* and *informatively hard*, based on Pareto dominance. Our motivational research question is as follows: Can one leverage the multi-dimensional nature of student–problem interactions to reveal problems that are hard (or easy) that are informative in the sense that they provide distinguishing information about student performance. The overall goal of this research is to eventually contribute to the design of a concept inventory.

Definitions for these new dominance-based measures are provided in Section 3.2. To validate the potential of this new perspective, we look into how dominance-based methods can be applied by using our approach to analyze data collected by a software tutor for programming, called *problets* (problets.org). The data were collected from students drawn from four semesters and focused on three high-level subjects commonly covered in introductory programming courses. We present the results and discuss their significance.

While enrollment in **computer science (CS)** undergraduate programs is increasing, attrition in the introductory programming course is legendary [6]. Students find introductory programming courses to be hard [56, 72] or at least the way they are currently taught [55]. This presents educators with some fundamental questions. For example: What are the concepts in introductory programming that are hard/easy for students? A better understanding of difficulty could help educators better target the hard concepts, provide focused support to students, and possibly improve retention in the course.

This challenge brings into sharp focus the value of diverse, better automated, unsupervised practice spaces for a large number of students. **Intelligent Tutoring Systems (ITS)** such as *Problets* (problets.org) [52–54] provide the foundations to automate many aspects of how potentially large numbers of students may supplement classwork and further their learning during unsupervised practice sessions. However, such systems are not currently entirely autonomous. ITS still require domain experts (e.g., instructors) to hand-design the exercises, problems, and overall learning experiences. This issue is further exacerbated as the number of students needing programming practice increases. Currently, generating large numbers of exercises that are both diverse and pedagogically relevant from a limited suite of hand-crafted ones is extremely time-consuming. Furthermore, hand-labeling problems in advance, so that an ITS may assign them meaningfully to learners, may result in an overly rigid system unable to accommodate the learning differences of a diverse population of students. Therefore, a more dynamic solution is needed.

However, scaling ITS to accommodate a growing, diversifying population of users entails more than scaling up the number and diversity of exercises. Any system that can efficiently adapt problem content to students must be based on understanding of the structure of the problem and concept space. At a minimum such a system must differentiate problems that help uncover particularly challenging concepts versus problems that verify rudimentary skills. In addition to *hard* and *easy* problems, though, there is a more nuanced gray area. It is clear enough that overly hard problems, that all students fail to solve, do not encourage learning any more than overly easy problems all students deftly solve. But how do we understand the area in between these extremes?

Consider the challenge of adapting a set of practice problems to the specific pedagogical needs of a given set of students. One prerequisite of solving this problem is the ability to identify the characteristics that make a practice problem more “pedagogically relevant” than another for these students. Traditionally, the notion of difficulty has been central to such investigations. When

looking at student–problem interactions, as captured by the logs of an ITS, the concept of difficulty is expressed in a dual form: The performance of students against the practice problems revealing the difficulty level they are able to handle, and the difficulty of practice problems revealing the average performance of students. Such statistics are easily extracted from logs and facilitate a variety of analyses; however, such simplified statistics fail to look deeper at the multidimensional nature of student–problem interactions and fail to fully explore the structure and dynamics of this highly interactive space. In particular, we are interested not just in problems at varying levels of difficulty but in particular those that fundamentally differentiate student performance so that educators (or an ITS) can identify which concepts are understood and which are not.

Given the similarities between co-adapting populations of students and problems, and applications of coevolutionary algorithms to machine learning problems, it is instructive to consider how the theory of coevolutionary computation can be useful for providing a deeper, multidimensional analysis within educational scenarios. More specifically, we may leverage theoretical work, aimed at understanding the structure and dynamics of co-adaptive optimization systems, from the study of *coevolutionary computation*. Typical applications of coevolutionary algorithms involve simultaneously discovering *tests* that are increasingly informative about a *learner's* performance while finding learners that perform increasingly better on such tests—for example, learning from self-play for board games [63]. Such a scenario is sometimes referred to as the teacher–learner coevolutionary learning framework or simply coevolutionary learning [29, 42].

Using the idea of coevolutionary learning, we examine student–problem interactions from a coevolutionary perspective. The underlying working hypothesis is that the same pathological dynamics that are avoided in coevolutionary learning applications by switching to a Pareto-based view of comparisons among learners, will also address their counterparts in educational situations. For instance, the loss of gradient between the learners and tests populations [82] is analogous to the design of practice exercises that are too difficult for the current learner population.

Specifically, we have applied the *dimension extraction* mechanisms from coevolutionary theory [9] to analyze the underlying interaction space depicted by logs from the Problets intelligent programming tutor. This further extends our previous efforts to use dimension extraction methods to help inform intelligent tutoring systems [11, 84]. The analysis and discussion in this article also provide opportunities to reconsider the traditional concepts of easy versus difficult problems in a manner allowing instructors to gauge the benefits of a given problem in terms of what information it provides about students' performance.

In a sense, our method can be seen as extracting the geometric structure of a space defined by the interactions of a set of students on a set of problems. Notions such as difficulty and informativeness of problems relate to concepts embedded in the problem set itself, and our goal is to use dimension extraction to provide some insight into what students do and do not understand within that problem set.

2 BACKGROUND

One of our critical research motivations is to find ways to identify how students conceive of (and misconceive of) subject material. To this end, we begin our background discussion looking generally at how our work relates to *concept inventories*. This leads naturally to a discussion of related literature more specifically relevant to identifying *misconceptions* students have in introductory programming classes. From there, it is important to understand how people have approached such issues, so we then review learning analytics, followed by a discussion of cognitive diagnosis modeling, which is closer to the kind of multi-dimensional modeling of student performance that is at the core of our proposed approach. Finally, we review the specific literature that directly inspired our analytic approach: coevolutionary computation and dimension extraction.

2.1 Concept Inventories

Easy versus difficult problems can reveal students' conceptions that are at odds with accepted conceptions, often referred to as "misconceptions." Research has shown that students harbor a number of misconceptions across a variety of topics. Mathematics and science education researchers have been documenting misconceptions for decades [17, 70]. However, the majority of this work in computer science education [2, 47, 74], has taken place more recently. Documenting and addressing misconceptions has naturally led to concept inventories.

A concept inventory is an instrument, often a multiple-choice quiz, used to evaluate the learners' understanding of specific concepts. As such, it is considered a criterion-referenced test in so far that the outcome of the test is informative about the level of understanding of the subjects being tested.

Creating a concept inventory for a specific topic or course is a significant undertaking. Doing so requires researchers to both enumerate relevant concepts and assess their level of difficulty. Please note that, in this work, we define a *concept* as a mental category for facts, objects, events, people, ideas, skills, and competencies that transcends the perspectives and limits of a specific subject-area [27].

Historically, physics is often regarded as a pioneer in leveraging concept inventories in discipline-based education research [32, 39]. This work also supported the evaluation of the impact on learners of specific pedagogical interventions such as peer instruction [19, 20] in physics courses.

The impact of concept inventories on physics education inspired computing education researchers to dedicate significant effort to design concept inventories on various subjects. Targeted subjects include digital logic [34], computer architecture [35–37, 64], discrete mathematics [1], operating systems [83], algorithms and data structures [21, 45, 80], and analysis of algorithms [28]. While, in this work, we will focus on concept inventory research applied to introductory programming courses, the interested reviewer is referred to Reference [77] for an introductory overview of concept inventories in computing education.

With respect to introductory programming courses, several attempts are of particular interest. In Reference [30], the authors first identified a list of topics corresponding to hard concepts in programming, then ranked them by importance and difficulty. This work exemplifies the benefits of using a well-established methodology to reach consensus among experts. In this specific study, such consensus was reached by using a Delphi process. This initial work was later refined to apply the above-mentioned Delphi process to three different introductory computing topics [31]. It also motivated research on identifying students' misconceptions: See Reference [43] and References [12–15] for recent examples and a discussion of their relevance to concept inventory design. In the same vein, the Foundational CS1 exam, a programming language independent exam, has been proposed as a validated instrument to reliably measure students' understanding of some key CS1 high-level concepts (see Reference [78] and Reference [79]). This instrument was later re-designed to ensure replicability of the original study [62].

These efforts illustrate the difficulty and resource-intensive nature of establishing concept inventories. First, the key concepts have to be identified and iteratively revised by a panel of experts. Second, distractors have to be identified for each question in the concept inventory. This typically involves manual analysis of large quantities of open ended exam questions in which students' errors are identified and collated. Finally, students must be interviewed to validate the identified misconceptions. The reader is referred to Reference [7] for a recent analysis of the literature on programming skills assessment, including some of the above-mentioned ones.

For this reason, alternative processes are also being explored. For example, using open source team dynamics and contribution models [66]. Another approach is to survey instructors of CS1

and CS2 courses directly about how much time they spend on various topics, then correlate the time with student performance [38].

It is interesting to note that the above-discussed studies do not generally delve into low-level concepts within each of the concepts they identify. To create a concept inventory for introductory programming, assessment items must be created on a continuum of specific concepts within each topic that students do not understand. We address this issue of granularity, discussed in Reference [65], by identifying low-level concepts in most of the topics typically covered in introductory programming, particularly those concepts whose hardness can be measured using data-driven analysis.

2.2 Difficulties in Introductory Programming

Despite the amount of work dedicated to identifying misconceptions held by novice programmers (see Reference [67] for a recent overview), there is still considerable variation and disagreement among those who teach introductory programming. A recent meta-analysis [71] sought to determine the most effective instructional approaches and conditions. The study found that “teaching programming through metacognition seems effective, and the metacognitive skills acquired during instruction may ultimately impact students’ problem-solving performance” (p. 13). However, with the large variation among courses, despite recent efforts in benchmarking [73], there is still much to learn regarding the instruction of introductory programming as substantial failures are still common in introductory CS courses [8, 81]. Moreover, some students pass CS1 without the expected fluency and competence. A series of papers about learning programming [58–60] discussed threshold concepts and the need to understand where students get “stuck.” Reference [16] speculates that “a feedback mechanism [could] be incorporated to ensure better results in the code deliveries and enhance the management of cognitive processes as an effective strategy for solving problems” (p. 196). Problets aligns well with this metacognitive approach through practice problems that provide feedback to students with step-by-step explanations. A review of literature on programming barriers in Reference [46] suggested that syntax and semantics are major barriers for beginning programmers but it is possible to make programming accessible “by simplifying the mechanics of programming, by providing support for learners and by providing students with motivation to learn to program” (p. 131). Research in the areas of cognitive load theory and problem-solving show that extensive problem-solving allows “learners to automate basic procedures and devote more working memory resources to making sense of novel aspects of relatively new problem situations” (p. 177, Reference [44]) suggesting an ITS such as Problets would be useful in an introductory programming course and provide valuable data.

2.3 Learning Analytics

In the field of learning analytics, several methods are used to evaluate student learning in ITS. **Bayesian Knowledge Tracing (BKT)** is an algorithm frequently used to model student learning. Corbett and Anderson’s [18] original BKT predicted mastery learning in a programming tutor. However, problem difficulty was not taken into account. Mostafavi, Eagle, and Barnes [61] distinguished between higher and lower proficiency problems in applying BKT to the Deep Thought tutor. More recently, Martori, Cuadros, and González-Sabaté [57] used BKT with a skill difficulty index, the percentage of correct answers in question tagged with a single skill to determine the relationship between the measures. They were found to be highly correlated but most effective for easy or difficult skills. While BKT is effective for determining a student’s knowledge over a set of problems, it does so formatively and assumes change in student knowledge over the assessment [76].

Another method of determining item difficulty in learning analytics is **item response theory (IRT)**. IRT is used for assessments in which the student is assumed to have already gained a certain

level of knowledge that does not change within the assessment [24, 76]. IRT is used extensively in computerized testing, such as SAT, to select problems appropriate to the student's level [26]. Rasch models provide methods that are mathematically similar to one parameter IRT to model participant responses to items that is math, though Rasch models discard data that do not fit the model and thus do not permit abilities to be estimated in extreme cases [4, 33].

IRT does provide one kind of measure for item difficulty, the likelihood of a correct response. It also provides mechanisms to measure item discrimination in terms of the shape of the item response function. Perhaps most saliently, as a means to assess the reliability of items to provide discriminatory power, item information functions provide a measure of the precision an item manages across the levels of a latent trait [5]. This is a statistical measure of the inverse of the error in the trait level and consequently often interpreted as the “*information*” of the item [5]. However, this is a different and more aggregate sense of “*information*” than the one we are employing for our purposes. IRT information curves can determine the degree to which one item has discriminatory power over participants and whether items discriminate *differently*, but the **Dimension Extraction Coevolutionary Algorithm (DECA)** organizes students and problems hierarchically based on *how* each discriminates based on dominance.

Further, because of its statistical nature, IRT involves a host of assumptions that are not necessarily met in realistic conditions. For one, a sufficient sample to estimate parameters for item and participant models are necessary and is highly susceptible to sample error [75]. Additionally, item responses are typically assumed to be uncorrelated, a property known as “local independence” and information functions are assumed to be unimodal (a single “construct” for each discriminator) [69]. Violating this assumption can have a direct impact on the slope of the characteristic curve, which distorts item discrimination. DECA, however, makes none of these assumptions and instead recovers the true individual pre-ordering entailed by the student–problem relationships.

Traditional IRT also assumes that the data are unidimensional, though **multi-dimensional item response theory models (MIRT)** are also available. MIRT methods still rely on local independence, as well as a kind of monotonicity in the response space [68]. Additionally, MIRT is multidimensional in the sense that it models multiple latent traits simultaneously; however, this is different from the notion of multidimensionality uncovered by DECA. The geometries extracted by DECA uncovers the underlying dimensional structure of the problem (item) space. MIRT relies on problem sets being constructed with sufficient local independence between items and that number of latent traits are known, whereas DECA makes no such assumption and *learns* how many and what the underlying dimensions are.

Comparing the actual difficulty measure and informativeness results of DECA and IRT is not constructive, because they define both of these quite differently. They will undoubtedly produce different results, and both different from traditional measures of difficulty. The purpose of this article is to introduce the idea of using a dominance-based approach to extracting the geometry of the student–problem space to examine questions such as informative difficulty in a fundamentally different way. So we confine our attention to understanding the consistency and generalizability of DECA in that setting. Still, comparing how consistent and generalizable IRT versus DECA approaches to informative difficulty would be constructive, and future studies should investigate how DECA compares to IRT specifically in terms of such stability concerns.

2.4 Cognitive Diagnosis Modeling

Cognitive Diagnosis Modeling (CDM) is a close cousin to what we hope to achieve with DECA. In CDM, a subject matter expert determines a set of student skills relevant to the educational items to be presented to them. Each student can then be assigned a vector, usually binary, indicating their mastery of each skill. Most models also require a subject matter expert to create a Q-matrix

mapping items to the skills required to complete each item. Additionally, the subject matter expert must determine whether success probability for each item requires mastery of all skills indicated in the Q-matrix (conjunctive models), or only requires mastery of some subset (disjunctive models). Adapting item selection to students requires a psychometric student model such as DINA, NIDA, or the fusion model. See Reference [41] for a more thorough review.

Our aim in this article is to address the specific question of whether DECA can reasonably be used to assess problem difficulty. Additionally, we are concerned with whether the multidimensional nature of DECA's output allows for a more nuanced view of problem difficulty that emerges from real-world student outcome data instead of from a probabilistic model. Importantly, aside from the creation of the item sets, no subject matter expertise is required to apply DECA; DECA only requires each student's score on each item to proceed. More detailed information could be gleaned both about the items and about individual students in a downstream analysis of DECA output if we also had a Q-matrix and a student psychometric model.

2.5 Co-optimization and Coevolution

The challenge of defining hard and easy problems arises in the study of *co-optimization* problems and *coevolutionary* algorithms [63]. Co-optimization problems are distinguished from optimization problems by the presence of two or more types of entity (e.g., student or problem) that might vary and can interact with one another with a measurable outcome or score.

Interestingly, many early coevolution and co-optimization researchers anticipated that simple measures of performance over these outcomes would be sufficient to yield “arms race” dynamics in coevolutionary learning. When such dynamics occur, the population of learners and the population of tests challenge each other causing mutual improvements. However, while some results confirmed the potential of such dynamics [40], others provided indications that many other, less beneficial, coevolutionary dynamics often emerge instead [82]. It is only recently, by adopting more sophisticated notions, such as Pareto dominance when comparing the performance of two learners over more simple aggregated measure of the outcomes, that the coevolutionary learning community has established a successful theoretical framework. The fundamental observation by such theory is that to build successful learning gradients, one must focus on the *information* about how tests distinguish different types of learner performances.

In the present context, we have computer science students and a variety of computer programming problems we might present to those students; any student can attempt—interact with—any of the problems, and they receive a score. Therefore, the real-world domain of students solving problems resembles a co-optimization problem, and we believe the tools and techniques developed in the study of those algorithms can be fruitfully applied to this domain.

One foundational and relevant line of work in co-optimization and coevolutionary algorithms concerns *dimension extraction* methods [9, 10, 22, 23]. Abstractly, this work hypothesizes that the information gleaned from interactions, for instance the scores of students solving problems, can be decomposed into a vector-space-like *coordinate system*. Within a coordinate system, there are potentially *multiple axes*, or *dimensions*, consisting of a linearly ordered subset of the set of all entities. Entities further along a given axis are “no worse than” those preceding it. Across two different axes, entities are incomparable to one another, in the sense that an entity on one axis will be better in some ways, but worse in other ways, than an entity on another axis. This method has been used for a number of purposes, including automatically identifying key conceptual tactics in the game of Nim [23].

We will apply a minor variant of the dimension-extraction algorithm presented in Reference [22] to analyze data about populations of students and programming test problems. This analysis will help us distinguish student performance, a key insight of this article.

3 METHODS

3.1 Problots

The data for this study were obtained from a software tutoring suite for introductory programming, called problots (problots.org). A *proble*t covers a topic typically found in an introductory programming course. For example, arithmetic expressions, loops, and selection. Each proble

t addresses 9–25 learning objectives. It generates problems on these learning objectives as randomized instances of parameterized problem templates. The skills addressed by problots include evaluating expressions, predicting the output of programs, debugging programs and identifying the state of the variables in a program.

- **Pre-test problems** each focus on a specific learning objective and are used to build the student model needed for adaptation. During the pre-test, students are provided no feedback unless their answer is incorrect.
- **Practice problems** each focus only on the concepts on which the student solved the pre-test problem incorrectly. Practice problems are presented until the student masters each concept, i.e., solves a minimum threshold number of problems correctly for each concept. During this practice, step-by-step explanation of the correct answer is provided as feedback for each problem. Such explanations have been shown to help students learn [48].
- **Post-test problems** focus only on the concepts that the student mastered during the practice stage. During this stage, once again, students are not provided any feedback.

Since the purpose of this study is to identify easy and hard concepts, we used only pre-test data, which represents a snapshot of what the students knew and did not know before using the proble

t. Typically, problots are used by students as after-class assignments. So, the pre-test data represent what the student learned from classroom instruction. Since problots are deployed over the web, students can use each proble

t as often as they wish. For this study, data were used for each student from the first session in which the student had attempted all the pre-test problems. Problots are used by students from multiple institutions each semester. For this study, data from all the students of all the institutions that used the proble

t in four semesters (Fall 2012–Spring 2014) were considered.

The dataset is maintained by the maintainers of the online problots system. As we previously mentioned, there are 17 tutors in the problots suite. These are typically used in the order in which the topics are covered in class. For this study, we picked three tutors that were used the most by the students, and hence, produced a larger dataset, on arithmetic expressions, selection statements and while loops. These three tutors were set up and used identically during Fall 2012 through Spring 2014. During those four semesters, the tutors were used by students from a diverse range of learning institutions in introductory programming courses, including 14 unique high schools, 6 unique community colleges, and 36 unique baccalaureate institutions. The tutors were used for Java, C++, and C# programming languages, because these were the common languages used in introductory programming courses. Among the users, roughly 30% self-identified as female and 20% self-identified as being from an underrepresented group.

Table 1 lists the number of times each of these tutors was used during those semesters. Table 2 shows a breakdown of our student-proble

t data. The total sums for each category do not always match, since not all students reported race or gender.

Each tutor was used in a different controlled study. The proble

Table 1. Usage of All the Selected Tutors: Fall 12–Spring 14

Probletem uses	Fall 12	Spring 13	Fall 13	Spring 14
Arithmetic	573	594	856	957
if/if-else	503	651	643	763
while loop	508	644	765	409

Table 2. Detailed Breakdown of Student Probletem Data Involved in This Analysis

		Arithmetic	Selection	Loop
Language	Java	1,710	2,118	1,417
	C++	793	880	399
	C#	7	17	1
Institution	High School	190	514	265
	Community College	252	261	175
	Baccalaureate	2,082	2,240	1,385
Gender	Male	1,496	1,675	1,012
	Female	572	696	365
Race	Caucasian/Asian	1,607	1,837	1,053
	Underrepresented	390	479	279

those who were not [52]. The selection probletem was used to study the hypothesis that students of whom self-explanation was required learned better than those who were not [51]. The while loop probletem was used to study whether limiting the number of times students could revise their answers would preserve the benefits of error-flagging feedback while minimizing gaming behavior among students [50]. However, since we only considered pre-test data for this study, and the pre-tests were identical for both control and test groups, we combined data from both groups for this study.

In an earlier work [84], we presented the results of applying dimension extraction to data from a single semester with a qualitative assessment of the possible interpretation of those results. In this article, we expand upon our prior analysis to present a deeper and more quantitative analysis of three specific probletem topics from data spanning four semesters.

3.2 Assessing Difficulty with Dimension Extraction

As briefly discussed in Section 2.5, one method of approaching co-optimization problems is through the application of so-called *dimension extraction methods* such as the DECA [22]. Abstractly, these methods search through *candidate solutions* to some problem while simultaneously also finding informative *test cases* for those solutions. This is achieved by gradually building up a model of the internal geometry of the comparisons between candidate solutions and tests – that is, extracting the underlying dimensions in which tests distinguish candidate performance and vice versa.

While we are not performing co-optimization, once one notices that candidate solutions are analogous to students learning concepts in some domain, and test cases are analogous to the problems those students solve, the core dimension extraction routine can be directly translated to student–problem analysis. Even when every student attempts every problem, complex relationships among the outcomes are possible, making comparisons between students problematic. For example, one student may correctly solve all but one problem, while some other student solved this particular problem correctly but missed all others. The first student may be said to have performed better on average, it may be the single test they missed was especially difficult, so there is a sense in

which the second student performed better. Terminology from multi-objective optimization and co-optimization is useful to clarify cases like this one, specifically *Pareto Dominance*.

Definition 3.1. Given n -dimensional vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we say that $\mathbf{x}$ **Pareto dominates** $\mathbf{y}$ iff $\forall i \in \{1 \dots n\} x_i \geq y_i$ and $\exists j \in \{1 \dots n\}$ such that $x_j > y_j$.

That is, we can say that student A *dominates* student B if A gets every problem B got right, and at least one more. When a dominance relationship exists, the two students can be *compared*. If A neither dominates B nor is dominated by B , then the two students are *incomparable*. This means that there are problems that A right that B did not *and vice versa*.

Based on these observations, we can use dimension extraction to perform *problem analysis* to construct a coordinate system of problems. Each dimension of the coordinate system can be thought of as a cluster of problems that is linearly ordered. Problems further along a dimension are not passed by at least the same students as those lower on the dimension. More formally, if p_1 and p_2 are two problems on the same dimension, V_{p_1} the set of students that did not pass p_1 and V_{p_2} the set of students that did not pass p_2 , then p_2 is further along the dimension than p_1 if and only if $V_{p_1} \subset V_{p_2}$. Problems on different dimensions are those on which students performed incomparably. For two such problems neither V_{p_1} nor V_{p_2} is a subset of the other. Intuitively, each dimension corresponds to a different concept that differentiated student performance and the problems that are highest on each dimension are the problems the most students did not solve correctly.

Algorithm 1 provides pseudocode to extract the orthogonal problem dimensions from the outcomes of a set of students all taking a set of problems. The function FILTERSPANNEDPROBLEMS corresponds to the idea of removing some problems whose outcome information can be found in two or more other problems, see Reference [22] for more information.

ALGORITHM 1: Problem Dimension Extraction

Input : list of *students*, list of *problems*, matrix of *outcomes*

Output: tree of problem *dimensions*

use *outcomes* to count number of *students* who incorrectly answer each *problem*

sort *problems* by number of students that answered incorrectly

problems $\leftarrow$ FilterSpannedProblems(*problems*)

initialize *dimensions* to an empty tree

```

foreach problem  $\in$  problems do
  foreach leaf  $\in$  dimensions do
    if AreComparable(problem, leaf, outcomes) then
      | add problem as child to leaf
    end
  end
  if problem was not added to any leaf then
    | add problem as new leaf to root of dimensions
  end
end
return dimensions
  
```

However, the algorithm above can also be applied from the student perspective. A *student analysis* reverses the roles of students and problems, resulting in a coordinate system of students. Dimensions are clusters of students, and each dimension linearly orders its constituent students. Students further along a dimension pass the same problems, plus possibly more, as students lower on the

dimension. Students in different dimensions perform incomparably on the problems. Intuitively, each dimension corresponds to a “type” of learner. Students furthest along dimensions solve the most problems, and in that sense are the best performing of each type.

Application of these analyses to student–problem performance provides additional insight over traditional approaches. First, the number of dimensions extracted from problem analysis gives an implicit measure of the number of distinct *concepts* that appear in the problem set; that is, the number of different ways that problems distinguish student performance. Likewise, the number of dimensions extracted from student analysis gives an implicit measure of the number of distinct *types of learner* that appear in the student set, or in other words the number of different ways students can be grouped in terms of differences in performance. Though not the purpose of this article, it is worth noting that preliminary application of dimension extraction to concept analysis has provided some insight [11].

More salient to this article, such analyses allow us to highlight the importance of how informative different problems are. While traditionally the “difficulty” of a problem is typically determined by how many students solve or do not solve that problem, such a measure is naïve. For instance, if master Go player Lee Sedol were to play a classroom of intermediate Go students, then he would likely beat them all, and it would be impossible to determine which students were among the best in the class. Likewise, if someone who never played Go were used as the opponent, then this person would likely lose to everyone, and we would be equally uninformed about the relative performance of students. Similarly, a problem that most students miss (or get right) may not be an informative problem, because it might not provide distinctions to help educators know where students are having problems.

Instead, we want challenging problems that distinguish student performance as much as possible. As educators, we should be interested in constructing problems of varying levels of difficulty that are on the boundary of where students perform in fundamentally different ways and, consequently, where true performance can best be judged. To begin to address this shortfall, we introduce two new terms grounded in our dimension extraction analysis, *informatively hard* and *informatively easy*.

For each dimension extracted in a student analysis (i.e., distinct learner performances) we identify the *most dominant students* as the set of (equivalent) students who solved the most problems among students on that dimension. For each problem, we count the total number of most dominant students that fail that problem. Likewise, we identify the set *most dominated students* as those who (equivalently) solved the fewest problems among students on their respective dimension. For each problem, we count the total number of such students who passed that problem. A problem is *informatively hard* if it is most often solved incorrectly by the most dominant students. Similarly, a problem is *informatively easy* if it is most often solved correctly by the most dominated students. In general, we refer to these ideas *informative difficulty measures*.

By contrast, we define a problem as *traditionally hard* (or just “hard”) if it is solved incorrectly by the most students and as *traditionally easy* (“easy”) if it is solved correctly by the most students. In general, we refer to these ideas *traditional difficulty measures*.

Informative difficulty captures different relationships in the performance data than traditional difficulty. For example, it is possible for a problem to be missed by everyone *but* the strongest students and for an otherwise challenging problem to be solved by the weakest students. When incomparable relationships like this exist, it is likely that student behavior is confounding a linear expectation of performance, and is therefore interesting from an assessment point of view. Informative difficulty preserves the idea that different students understand different concepts in different ways. An informatively easy concept may be understood as the concept nearest to some other concept, whereas an informatively hard concept is farthest from any other concept.

Not only are these two notions conceptually different but, as we will show, often informative difficulty produces different results than traditional difficulty. This is consistent with our intuitive understanding in the sense that traditional difficulty is monotonic and linearly ordered, while the real performance relationships in the data are not. The more interesting, although rare, cases are when an informatively easy concept is traditionally hard and an informatively hard concept is traditionally easy. The former corresponds to when it is easy for the general population to make mistakes while solving a problem. The latter corresponds to when it is easy for the general population to guess the answer.

3.3 Consistency and Generalization

A preliminary question is the degree to which the concepts of informative difficulty remain *consistent* across a given dataset — that is, the sensitivity of the analysis to which student records were presented and in what order. To assess consistency, we performed the following experiment. For each of the 12 groups (three subjects and four semesters), we drew 500 independent samples, ran the DECA informatively hard problems analysis, and produced a ranking of the difficulty of all the included problems. Each sample was produced by randomly and independently drawing students from the dataset described in Section 3.1 (with replacement) until the sample contained the same number of students as that dataset, similar to the statistical concept of *bootstrapping* [25]. We computed the Spearman rank correlation coefficient ρ between the ranking produced by each of the 500 samples and the ranking produced by the original dataset (“*original*” as described in 3.1). The results for each of the 12 groups are shown in Figure 1 below.

For reference, we include the same results for the simple, traditional measure of difficulty (number of students that missed the problem). In general, such traditional measures are more consistent and generalize better than our approach. This is to be expected, since the measure represents straightforward counts and thus sufficiently large samples should provide good coverage of the space. The point here is that our informatively hard measure is *also* very consistent and generalizes well, while also providing more structural and discriminatory information than the simpler measure.

The results, which will be discussed in more detail in the next section, suggest that the rankings produced by our informatively hard measure are not so sensitive to data variations that the ranking reverses. This observation squares well with the preliminary consistency analysis performed in [84], where consistency was assessed in a slightly less principled way.

As a final point in this preliminary analysis, even though both the informatively hard and traditionally hard measures are consistent, they are not well correlated with one another. A DECA-based informativeness measure thereby provides a fundamentally different kind of problem analysis than traditional methods. For this we again used Spearman’s ρ , computing the rank correlation between the two for each subject and semester. No correlation for any semester and subject was larger than 0.35, and several were negative (e.g., all semesters for arithmetic expressions and two semesters for the selection topic). None of the rank correlations were statistically significant.

In Sections 4 through 6 we will test the consistency and generalization ability of the informatively hard and, as a comparison, the traditionally hard measures on arithmetic, selection, and loops datasets. To test consistency, we use the bootstrap method outlined above. In other words, for each subject we draw 500 random samples with replacement from that subject’s datasets for all four semesters tested. We applied the two difficulty measures to a given sample in each case. We then compare the two, using Spearman’s ρ , the correlation between the ranking produced by each random subset with the ranking produced by the actual dataset.

To test generalization, for each subject and each semester we compute the rank correlation, again using Spearman’s ρ , for every pair of distinct semesters. We report the Bayesian credible

Fig. 1. Spearman's ρ results for rank consistency between 500 random samples and original dataset for each subject and semester. In general, positive values near 1.0 represent strong statistical relationships (up is stronger).

interval—in fact the high density interval—of the rank correlation coefficient for both the 95% and 99% confidence levels.

4 TOPIC #1: ARITHMETIC

4.1 Arithmetic Expression Tutor

The arithmetic expressions proplet covers the following concepts/learning objectives:

- Evaluation of subtraction: e.g., $(8 - -5)$
- Evaluation of multiplication: e.g., $(5/3 + 3 * 5)$
- Evaluation of remainder on real operands: e.g., $(12\%5 + 5\%12.0)$
- Associativity of addition: e.g., $(4.3 + 5 + 3)$
- Associativity of subtraction: e.g., $(3 + 4 - 5)$
- Associativity of multiplication: e.g., $(2 * 3 * 1.5)$
- Associativity of division: e.g., $(9/3/2.0)$
- Associativity of Parenthesis: e.g., $(14/((6 - 3) + 2))$
- Precedence of addition: e.g., $(3/6 + 3\%6)$
- Precedence of subtraction: e.g., $(8/5.0 - 8/5)$
- Precedence of division: e.g., $(6 + 7/3)$
- Precedence of remainder: e.g., $(9\%5 - 5\%9)$
- Precedence of multiplication: e.g., $(8 - 2 * 5)$

Fig. 2. Snapshot of Arithmetic Expressions Tutor: Student's answer is shown in the left panel, the tutor's feedback is shown in the right panel.

- Dividing by zero: e.g., $(9/3/0)$
- Remainder when divided by zero: e.g., $(8 - 6 - 4\%0)$
- Precedence of Parenthesis: e.g., $(12/3 * (7 - 5))$

The tutor presents an expression (such as shown above), and has the student evaluate it one operator at a time. The tutor grades the order in which operators are evaluated, as well as the intermediate results entered for each sub-expression. After the student submits his/her answer, the problem displays the correct answer, once again, step-by step, as shown in Figure 2. The tutor administers 16 problems during pre-test. Each problem typically involves several concepts. For example, to evaluate the expression $3 + 4 - 5$, students must understand how to evaluate addition and subtraction, as well as the concepts of associativity of both operators. Figure 2 illustrates the software tutor's functionality for a particular problem.

As previously discussed, our analysis was conducted over four semesters in a two-year period (Fall 2012, Spring 2013, Fall 2013, and Spring 2014). A total of 1,282 students used the tutor during those semesters and solved all 16 pre-test problems.

4.2 Consistency within Each Semester

To determine whether the results of our analysis were consistent within each semester, we computed Spearman's rho (Spearman rank correlation coefficient) between each of 500 samples and the ranking produced over the original dataset. The results for each of the 12 groups is shown in Figure 3 below.

In all cases, all 500 samples are reasonably correlated with the complete dataset for both the traditional and informative difficulty measures. Both mean and median correlation values in each case are positive and generally above 0.87. Therefore, DECA appears to produce reasonably consistent results. This confirms the preliminary consistency analysis performed in Reference [84].

Fig. 3. Spearman's rho results for rank consistency between 500 random samples and original dataset for each semester for the arithmetic topic. In general, positive values near 1.0 represent strong statistical relationships (up is stronger).

Fig. 4. Spearman's rho results for rank generalizability between semesters for the arithmetic topic. The wider dark blue rectangle represents the 95% Bayesian credible intervals, while the narrow rectangle represents the 99% interval.

4.3 Generalization between Semesters

Recall that for assessing generalizability between semesters, we compute the Spearman's rho of the informatively hard rankings between the four semesters. That is, we compare how the ranking of difficulty produced for each semester correlate with one another using the Spearman's rho statistic. Figure 4 below illustrates Bayesian credible intervals for these results. These are estimated ranges for correlations between any pair of semesters with 95% and 99% probability, respectively.

The results of this analysis show strong statistical evidence that rankings produced by these metrics in one semester will be well correlated with another semester. This provides compelling evidence that results of our problem analysis in a given semester will generalize to other semesters.

4.4 Difference from Traditionally Easy/Hard Concepts

Traditionally Easy: By traditional incorrect counts, problems such as $8 - -5$ were considered the easiest ones. Such problems involve subtracting a negative number and thus correct evaluation of subtraction is the only concept covered. Fewer than 3% of students missed this problem, thus suggesting that the underlying concepts are not difficult for most students.

In comparison with our preliminary analysis [84], using four semesters worth of data confirmed this type of problem to be traditionally easy. With fewer data, a second problem was identified to be also traditionally easy, but this was not confirmed with additional data, thus suggesting that it was not as relevant.

Informatively Easy: Our DECA-based analysis found problems such as $9/3/0$ to be the most informatively easy. While about only about 13% of the students solved this problem incorrectly overall, 60% of the most dominated (weaker) students were able to solve it correctly. This finding is identical to that presented in our preliminary, single semester analysis [84].

The result means that, while this problem was easy for most students, it also had more discriminating power than the traditionally easy problem we just discussed. A simple explanation might be that this problem involves more concepts, especially divide by zero. While these concepts may not be separately difficult to most students, presenting more than one concept at a time in a single problem may permit more efficient differentiation of students in terms of what they understand and what they do not understand.

Traditionally Hard: By traditional incorrect counts, problems such as $12\%5 + 5\%12.0$ were found to be the hardest. Nearly 39% of all students solved this problem incorrectly. This finding is identical to that presented in our preliminary, single semester analysis [84].

These problems contain several concepts, including the remainder operator as well as real operands. These are both novel concepts to students who generally encounter them for the first time in an introductory programming course. As such, it is expected that most students will experience higher levels of difficulty when learning these for the first time.

Moreover, while fully parenthesized expressions can be evaluated without knowledge of precedence and associativity rules, students must know these rules when parentheses are omitted. Problems following the template that was identified as traditionally hard do not feature parentheses. As such, they require students to understand and apply the specific associativity and precedence rules of the programming language they are studying, which results in additional opportunities for misunderstandings.

Informatively Hard: Our DECA-based analysis found problems such as $5/3 + 3 * 5$ to be the most informatively hard. Integer division is also a novel concept, specific to programming. Only about 13% of students missed this problem overall, which makes this problem appear relatively easy by traditional measures. Indeed, it was only the eight most difficult problem (roughly the middle of the ranking). Though only about 16% of the most dominant (best) students missed this problem, that was the *most* missed problem by those students. In fact, nearly two-thirds of those who missed it were otherwise dominating students in the dimensions of a student analysis. By contrast, the most difficult problem as identified by traditions methods was fourth-to-last in the informatively difficult ranking—meaning, it was not a particularly informative result compared to other problems. This finding is also identical to that presented in our preliminary, single semester analysis [84].

As with the type of problem identified as informatively easy, this means that such problems have a higher discriminating power among students.

Discussion: These findings suggest the following observations. Arithmetic is a topic with which we can expect college students to be already familiar. Despite this familiarity, there are specific concepts that are novel when learning programming (e.g., typed operands and coercion, modulo operator). While it might seem natural to expect students to experience some difficulties on arithmetic problems, our analysis uncovers a deeper layer: that there is a tradeoff between the problems most

people miss (traditionally hard) and the problems that provide more ability to differentiate student performance (informatively hard). This distinction allows one to anticipate specific difficulties that will be encountered by different categories of students. This, in turn, offers new opportunities to adapt instruction to learners, based on their overall performance. For example, informatively hard problems might be worth testing further even with students featuring high scores. This also has implications for intelligent tutoring systems that have to establish which problems to have students practice with, based on a learner model often mostly characterized by their performance on previous problems.

5 TOPIC #2: SELECTION

5.1 Selection Tutor

The selection proplet covers the following concepts:

- Condition of if-else is a relational expression, evaluates to true/false
- Singly nested if-else statements (nesting in only one clause)
- Dually nested if-else (nesting in both clauses)
- Multiple if-else statements
- Condition of if statement is a relational expression, evaluates to true/false
- Condition of if statement is a declaration, evaluates to true (C++ only)
- Condition of if statement is an arithmetic expression, evaluates to true/false (C++ only)
- Nested if statements
- Multiple if statements

The tutor presents a complete program containing one or more if/if-else statements and has the student enter the output of the program one output at a time. If the student's answer is incorrect, then it explains the correct execution of the program line by line. Figure 5 shows a snapshot of the tutor, with the program in the left panel, and the student's answer in progress in the bottom right panel.

Our analysis was conducted over four semesters in a two-year period (Fall 2012, Spring 2013, Fall 2013, and Spring 2014). We considered 803 students who had solved 9 pre-test problems.

5.2 Consistency within Each Semester

Again, we computed Spearman's rank correlation coefficients between each of the 500 samples and the ranking produced over the original dataset (as described in Section 3.1) for the selection topic. The results for each of the 12 groups is shown in Figure 6.

In all cases, all 500 samples are reasonably correlated with the complete dataset for both the traditional and informatively difficult measures. Both mean and median correlation values in each case are positive and generally above 0.77. DECA again appears to produce reasonably consistent results, confirming preliminary consistency analysis performed with one semester's data [84].

5.3 Generalization between Semesters

For generalizability, we again compute the Spearman's rho of the informatively hard rankings between the four semesters. Figure 7 below illustrates Bayesian credible intervals for these results. These are estimated ranges for correlations between any pair of semesters with 95% and 99% probability, respectively.

Though not as strong as the results for arithmetic tutor, these results still show strong statistical evidence that rankings produced by these metrics, in any one semester, will be well correlated with those of another semester.

Fig. 5. Snapshot of Selection Tutor: The problem/program is shown in the left panel, the student's answer is shown in the bottom right panel.

Fig. 6. Spearman's rho results for rank consistency between 500 random samples and original dataset for each semester for the selection topic. In general, positive values near 1.0 represent strong statistical relationships (up is stronger).

5.4 Difference from Traditionally Easy/Hard Concepts

Traditionally Easy: The traditional measure identified problems involving nested if statements as the easiest problem template. Fewer than 3% of the students got such problems wrong. This finding is identical to that presented in our preliminary, single semester analysis [84].

Fig. 7. Spearman's rho results for rank generalizability between semesters for the selection topic. The wider dark blue rectangle represents the 95% Bayesian credible intervals, while the narrow rectangle represents the 99% interval.

Informatively Easy: Our DECA-based analysis determined that multiple back-to-back if statements presented the most informatively easy problem to the students. This finding is identical to that presented in our preliminary, single semester analysis [84].

Though slightly more than twice as many students miss this problem as they do nested if statements, multiple back-to-back if statements distinguish student performance better than nested if statements, since the students who miss this problem range in capabilities—indeed, 86% of the most dominated (weaker) students were able to solve this problem correctly. The fact that dominated students can perform well on this is not surprising as multiple if statements do not present additional difficulties to students. Those who are able to trace a single if statement will most likely be able to do so repeatedly, while those who did not understand its semantic will consistently fail at tracing multiple ones.

Traditionally Hard: From traditional incorrect counts, our analysis identified if-else statements involving relational expressions with a condition that evaluates to true to be the hardest of the problems. This finding is identical to that presented in our preliminary, single semester analysis [84]. In the four-semester data, roughly a third of all students missed this problem. More pointedly, though 803 students took these problems, only 405 of them missed *any* problems, and more than two-thirds of those missed this particular one.

Interestingly, this problem was not designed to be particularly challenging to students. For this reason, it is particularly difficult to interpret this finding. We have hypothesized previously that the lack of familiarity of the students with the tutor's user interface might be to blame, since this is the first problem students solve with this particular user interface. This hypothesis still remains to be validated.

Informatively Hard: Our informative hardness measure suggests that tracing the output of an if statement involving a relational expression whose condition evaluates to true is the most informatively hard problem. Please note that such problems differ from those we previously identified as traditionally hard. The latter were based on if-else statements while our informatively hard problems are only based on if statements.

Specifically, 25% of those who missed this problem were the most dominant (strongest) students. By traditional measures, this problem was the second easiest. Conversely, the traditionally hardest problem was the second most informatively difficult problem.

This result clarifies findings from our preliminary study in which we had three different problem templates competing for the title of informatively hard. With the additional data, we were able to

narrow this list down to the single problem discussed in this article. Still, this finding is also rather counter-intuitive and will require further investigation—see the discussion below.

Discussion: It is surprising that students found nested `if` statements to be easy and `if-else` statements difficult, particularly, since the latter were designed to be relatively simple problems. Our attempts at interpreting this finding led us to consider three distinct hypotheses:

- (1) The lack of familiarity with the user interface may explain why students solved it incorrectly, in larger numbers. The selection problem was among the first tutors students would generally work with in a given semester. This is due to the fact that the topic of selection is generally among the first ones covered in any introductory programming course. Similarly, the particular problems where we found this unexpected difficulty level were the first problems in the selection tutor.
- (2) These analyses also suggest that while the bifurcation of control flow may be what makes selection statements challenging rather than conditional execution itself, conditional execution may provide more information in terms of what different types of students understand.
- (3) Assuming that our expectations for a single `if` or `if-else` statement to be trivial for most students to trace is valid, the observed difficulties might be solely due to the difficulty of the relational expressions used in both templates. Our previous study revealed that the relational expressions problem also features a wide range of difficulties. After students have practice with this relational expressions problem, they might still have issues with the most traditionally difficult or informatively difficult relational expressions. When using these with both the `if`- and `if-else`-based templates, we should expect to see students struggle equally regardless of the simple control flow structure.

These hypotheses will have to be further evaluated via an appropriate experimental design to establish their validity. The data we have collected so far, along with the analysis method on which this article is focused, do not allow us to definitively establish which interpretation is valid; however, our analysis has uncovered these complex relationships and provided a road for a deeper investigation that did not previously exist.

6 TOPIC #3: WHILE LOOPS

6.1 `while` Loop Tutor

The `while` loop problem covers the following concepts:

- Condition is a relational expression
- Condition where sentinel value also changes on each iteration
- Variable is updated before being printed in the body of the loop
- Code executed after the loop exits
- Loop is never iterated
- Loop iterates only once
- Nested independent loops
- Nested dependent loops
- Multiple dependent loops

The tutor presents a complete program containing one or more `while` loops and has the student enter the output of the program, one output at a time. If the student's answer is incorrect, then it explains the correct execution of the program line by line. Figure 8 shows a snapshot of the tutor, with the program in the left panel, and the student's answer in progress in the bottom right panel.

Fig. 8. Snapshot of Selection Tutor: The problem/program is shown in the left panel, the student's answer is shown in the bottom right panel.

Five hundred forty-nine students solved all nine pre-test problems during the four semesters considered (Fall 2012, Spring 2013, Fall 2013, and Spring 2014).

6.2 Consistency within Each Semester

Again, we computed Spearman's rank correlation coefficients between each of the 500 samples and the ranking produced over the original dataset (as described in Section 3.1) for the while loops topic. The results for each of the 12 groups is shown in Figure 9 below.

As with the last two topics, all 500 samples are reasonably correlated with the complete dataset for both the traditional and informatively difficult measures in all cases. Both mean and median correlation values in each case are positive and generally above 0.67.

6.3 Generalization between Semesters

For generalizability, we again compute the Spearman's rho of the informatively hard rankings between the four semesters. Figure 10 below illustrates Bayesian credible intervals for these results. These are estimated ranges for correlations between any pair of semesters with 95% and 99% probability, respectively.

These results are similar to our results for selection, showing strong statistical evidence that rankings produced by these metrics in one semester will be well correlated with another semester.

6.4 Difference from Traditionally Easy/Hard Concepts

Traditionally Easy: By the traditional easy metric, predicting the output of a loop that iterates only once is the easiest problem. About 2.3% of students missed such problems. It is interesting to

Fig. 9. Spearman's rho results for rank consistency between 500 random samples and original dataset for each semester for the loops topic. In general, positive values near 1.0 represent strong statistical relationships (up is stronger).

Fig. 10. Spearman's rho results for rank generalizability between semesters for the loops topic. The wider dark blue rectangle represents the 95% Bayesian credible intervals, while the narrow rectangle represents the 99% interval.

note that these kinds of problems, by their nature, revert back to tracing sequential code. If a loop executes only once, then students may be able to trace through its body as they would through a few lines of sequential code, while ignoring anything loop specific. This would explain the lack of difficulty presented to most students by this kind of problem.

The same problem template was found to be traditionally easy with four semesters of data as our preliminary analysis with one semester of data [84]. However, as with the arithmetic tutor's traditionally easy case, our analysis of four semesters worth of data ruled out a second problem template that came close in our preliminary analysis.

Informatively Easy: Our informatively easy measures identified the problem involving nested independent loops. However, about 40% of the students missed it overall, thus making it, by traditional metrics, the second *traditionally hardest* problem. Roughly 76% of the most dominated students got this problem correct.

This finding is identical to that presented in our preliminary, single semester analysis [84].

Traditionally Hard: The traditionally hard metric reveals a problem that one might expect to be challenging: identifying the correct output when a variable is updated before being printed in the body of the loop. Roughly 48% of the students missed this question.

This finding is, so far, the only one that differs substantively from the ones presented in our preliminary study. In Reference [84], we identified the following problem template instead as traditionally hard: “Identifying the output of multiple back-to-back loops, wherein the behavior of the second loop depends on the execution of the first loop.” The difference between these likely stems from the increase in sample size, and therefore the results reported here are the more salient. The result also makes a kind of intuitive sense: students find it difficult when there are exceptions to established patterns.

Informatively Hard: Our DECA-based analysis suggests that the most informatively hard problem is one in which students must notice when a loop never iterates, because its condition fails immediately. Almost 20% of the most dominant students missed this problem, while strangely only 7 of the most dominated students missed it. The traditionally hardest problem was the second most informatively easy problem, while the informatively hard problem ranked in the middle by traditional measures (4 of 9), again showing that these measures can be quite different from one another.

The “gotcha” nature of this problem matches our expectations for what an informatively hard problem should be. While most students might be able to trace through a loop, they might learn to do so quite “mechanically” (i.e., making some assumptions on what the code they will be required to trace will do). An unexpected feature, such as the loop being written so that it will never iterate, might therefore throw off students regardless of their understanding of loops. As such, it can affect both students who are unable to trace loops at all, and those who are otherwise able to do so but simply overlooked the possibility of such an unusual scenario.

As with the informatively hard finding for the selection tutor, we were able to essentially clarify results from our preliminary study in which we had five different problem templates competing for the title of informatively hard. With the additional data, we were able to narrow this list down to the single problem template discussed in this article.

Discussion: These findings are a good example of how informativeness measures can provide a very different perspective about what is happening when compared to more traditional approaches. They suggest that the approach that many students take to tracing loop output may involve over-complicating the solution, making assumptions that are being incorrectly transferred from one problem to another, or mechanically tracing output without spotting higher-level patterns. In any event, dimension extraction, again, points out a new, interesting avenue of investigation about this subject.

7 DISCUSSION

7.1 Summary

In this article, we have demonstrated a data-driven approach for identifying easy and hard programming problems that are also *informative* in the sense that they create *distinctions* between students, based on their respective performance. This work extends and deepens preliminary results from Reference [84]. The method we used, dimension extraction, was based on the theory of coevolutionary computation applications to co-optimization problems. This is a natural analogy, since both coevolution and student–problem behaviors are systems in which one must develop a broader understanding from relatively simple interactions. Our results suggest that dimension extraction is consistent within a semester—meaning that informative difficulty analysis based on dimension extraction is not particularly sensitive to variations in the data. Our results also suggest that the analysis generalizes well from one semester to another.

Additionally, not only was it the case that informative difficulty gave a different result from simply counting the number of problems students got wrong, but it sometimes confounded

expectations drawn from such naïve methods: Sometimes problems that were considered to be relatively hard by traditional methods were found to be informatively easy. That is, such problems may have been missed by a number of students, but the students who missed it also got different (perhaps harder) problems right more often than other students. This helps justify the idea that the relationships between student abilities and problem concepts is often a complex and non-linear one. Clearly, information-based, data-driven approaches can be of value.

7.2 Implications for Researchers and Practitioners

The informative difficulty measure is useful, because it allows educators to understand not just whether a problem is easy or difficult, but whether that problem is elucidating what concepts students do or do not understand. The consistency of our results suggest that the informatively hard measure described here may be useful to educators using student outcomes data to develop problems that provide an informative scaffolding.

We expect the easy, hard, informatively easy, and informatively hard concepts enumerated in the previous sections to also be relevant to the design of a concept inventory for introductory programming. Since these concepts are supported by data-driven analysis, their inclusion in the concept inventory will better reflect the needs of students. As discussed in our literature review of concept inventories in the computing disciplines, such alternatives to identifying both topics on which students struggle and distractors representing common misconceptions, are worth exploring when considering the resource-intensive nature of the processes currently leveraged to establish concept inventories (e.g., Delphi process).

Dimension extraction is useful in a number of other ways, as well. For example, though beyond the scope of this article, such methods also reveal how distinct concepts are embedded in the problem set (the extracted dimensions) and how redundant or diverse coverage of these concepts are (the number of dimensions relative to the number of problems).

7.3 Limitations and Future Directions

To obtain our results, we focused on three common introductory languages, C++, Java, and C#. These were largely selected based on convenience of what was taught at the 56 institutions we considered. The concepts we considered relate to code analysis (program tracing, debugging and expression evaluation), but not synthesis (code design, writing) or evaluation (e.g., refactoring) skills as defined in the Bloom's taxonomy of the cognitive domain [3]. Our next efforts will focus on mapping these concepts to topics and distractors identified by current studies on introductory programming concept inventories [15, 31].

This article helps justify the utility of metrics based on dimension extraction and lays the groundwork for the idea of using student performance data to improve problems by making them more informative, we do not close the loop in this article. However, we do ultimately intend to incorporate dimension extraction to improve the adaptive nature of the problems software tutor.

Our method is quite different from existing data-driven metrics for understanding student-problem performance and provides quite different kinds of overall results. Still, we are interested in comparing these DECA-based informative difficulty measures to multi-dimensional item response theory models in terms of consistency and generalizability.

Further, we are extending dimension extraction to be able to address more complex interactions. Currently, our analysis assumes a binary result for each outcomes (students either solve a given problem correctly or incorrectly). However, not only is there a need to address uncertainty in an outcome, but some software tutors provide more complex feedback. In the future, we will expand dimension extraction informative difficulty measures to account for these and provide the mathematics that serves as a basis for these extensions. Related to this idea is the notion that student

responses to problems is not deterministic and may include some error. There are ways to address this within the existing DECA; however, it is perhaps best addressed by expanding DECA to permit uncertainty as a student–problem outcome directly.

REFERENCES

- [1] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. 2006. Concept inventories in computer science for the topic discrete mathematics. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR'06)*. ACM, New York, NY, 132–145. <http://dx.doi.org/10.1145/1189215.1189182>
- [2] Amjad Altadmri and Neil C. C. Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE'15)*. Association for Computing Machinery, New York, NY, 522–527.
- [3] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Rath, and M. C. Wittrock. 2001. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom' Taxonomy of Educational Objectives*. Longman, New York.
- [4] D. Andrich. 2004. Controversy and the rasch model: A characteristic of incompatible paradigms? *Med. Care* 42, 1 (2004), 1–7.
- [5] F. Baker. 2001. *The Basics of Item Response Theory*. ERIC, Madison, WI.
- [6] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. 2009. Exploring factors that influence computer science introductory course students to persist in the major. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE'09)*. ACM, New York, NY, 153–157. <http://dx.doi.org/10.1145/1508865.1508923>
- [7] Mike Barkmin and Torsten Brinda. 2020. Analysis of programming assessments — building an open repository for measuring competencies. In *Proceedings of the 20th Koli Calling International Conference on Computing Education Research (Koli Calling'20)*. Association for Computing Machinery, New York, NY. <http://dx.doi.org/10.1145/3428029.3428039>
- [8] Jens Bennesen and Michael E. Caspersen. 2007. Failure rates in introductory programming. *ACM SIGCSE Bull.* 39, 2 (2007), 32–36.
- [9] A. Bucci. 2007. *Emergent Geometric Organization and Informative Dimensions in Coevolutionary Algorithms*. Ph.D. Dissertation. Brandeis University, Boston, MA.
- [10] A. Bucci, J. B. Pollack, and E. D. de Jong. 2004. Automated extraction of problem structure. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'04)*. ACM, 501–512.
- [11] Anthony Bucci, R. Paul Wiegand, Amruth N. Kumar, Jennifer L. Albert, and Alessio Gaspar. 2016. Dimension extraction analysis of student performance on problems. In *Proceedings of the 29th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS'16)*. AAAI Press, 6.
- [12] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo. 2019a. Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory. In *Proceedings of the Innovation and Technology in Computer Science Education (ITiCSE'19)*. Association for Computing Machinery, New York, NY, USA, 23–29. <https://doi.org/10.1145/3304221.3319771>
- [13] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo. 2019b. *Identifying and Validating Java Misconceptions—Complementary Material*. Technical Report 19-05. Institute of Computing, University of Campinas, SP, Brasil.
- [14] Ricardo Caceffo, Raysa Benatti Guilherme Gama, and Rodolfo Azevedo Tales Aparecida, Tania Caldas. 2018. A concept inventory for CS1 introductory programming courses in C. In Technical Report 18-06, Institute of Computing, University of Campinas, SP, Brasil.
- [15] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. 2016. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*. ACM, New York, NY, 364–369. <http://dx.doi.org/10.1145/2839509.2844559>
- [16] Enríquez Ramírez Carlos, Raluy Herrero Mariza, and Luz María Vega Sosa. 2020. Proposed framework for employing cognitive process for resolving programming problems for novice programmers. In *Proceedings of the 8th International Conference in Software Engineering Research and Innovation (CONISOFT'20)*. IEEE, 193–197.
- [17] John Clement. 1993. Using bridging analogies and anchoring intuitions to deal with students' preconceptions in physics. *J. Res. Sci. Teach.* 30, 10 (1993), 1241–1257. <http://dx.doi.org/https://doi.org/10.1002/tea.3660301007>
- [18] Albert T. Corbett and John R. Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Model. User-adapt. Interact.* 4, 4 (1994), 253–278.
- [19] Catherine H. Crouch and Eric Mazur. 2001. Peer Instruction: Ten years of experience and results. *Am. J. Phys.* 69, 9 (2001), 970–977.
- [20] Catherine H. Crouch, Jessica Watkins, Adam P. Fagen, and Eric Mazur. 2007. Peer instruction: Engaging students one-on-one, all at once. *Res.-bas. Reform Univ. Phys.* 1, 1 (2007), 40–95.

- [21] Holger Danielsiek, Wolfgang Paul, and Jan Vahrenhold. 2012. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*. ACM, New York, NY, 21–26. <http://dx.doi.org/10.1145/2157136.2157148>
- [22] E. D. de Jong and A. Bucci. 2006. DECA: Dimension extracting coevolutionary algorithm. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*. ACM, New York, NY, 313–320.
- [23] E. D. de Jong and A. Bucci. 2008. Objective set compression: Test-based problems and multi-objective optimization. In *Multi-Objective Problem Solving from Nature: From Concepts to Applications*. Springer, New York.
- [24] Christinr being MIA e DeMars. 2010. *Item Response Theory*. Oxford University Press, New York, NY.
- [25] B. Efron and R. J. Tibshirani. 1993. *An Introduction to the Bootstrap*. Chapman & Hall, New York.
- [26] Susan E. Embretson and Steven P. Reise. 2013. *Item Response Theory*. Psychology Press, Rotterdam, The Netherlands.
- [27] H. L. Erickson. 2002. *Concept-Based Curriculum and Instruction: Teaching Beyond the Facts*. SAGE Publications.
- [28] Mohammed F. Farghally, Kyu Han Koh, Jeremy V. Ernst, and Clifford A. Shaffer. 2017. Towards a concept inventory for algorithm analysis topics. In *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE'17)*. ACM, New York, NY, 207–212. <http://dx.doi.org/10.1145/3017680.3017756>
- [29] Sevan G. Ficici and Jordan B. Pollack. 1998. Challenges in coevolutionary learning: Arms–race dynamics, open-endedness, and mediocre stable states. In *Proceedings of the sth International Conference on Artificial Life*, Adami et al. (Ed.). MIT Press, Cambridge, MA, 238–247.
- [30] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. 2008. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bull.* 40, 1 (2008), 256–260.
- [31] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2010. Setting the scope of concept inventories for introductory computing subjects. *Trans. Comput. Educ.* 10, 1, Article 5 (June 2010), 29 pages. <http://dx.doi.org/10.1145/1789934.1789935>
- [32] I. A. Halloun and D. Hestenes. 1985. Common sense concepts about motion. *Am. J. Phys.* 53, 11 (1985), 1043–1055.
- [33] D. Harris. 1989. Comparison of 1-, 2-, and 3-Parameter IRT models. *Educ. Meas.: Issues Pract.* 8 (1989), 35–41.
- [34] Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Proof by incomplete enumeration and other logical misconceptions. In *Proceedings of the 4th International Workshop on Computing Education Research (ICER'08)*. ACM, New York, NY, 59–70. <http://dx.doi.org/10.1145/1404520.1404527>
- [35] Geoffrey L. Herman, Michael C. Loui, Lisa Kaczmarczyk, and Craig Zilles. 2012. Describing the what and why of students' difficulties in boolean logic. *ACM Trans. Comput. Educ.* 12, 1, Article 3 (Mar. 2012), 28 pages. <http://dx.doi.org/10.1145/2133797.2133800>
- [36] Geoffrey L. Herman, Michael C. Loui, and Craig Zilles. 2010. Creating the digital logic concept inventory. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10)*. ACM, New York, NY, 102–106. <http://dx.doi.org/10.1145/1734263.1734298>
- [37] Geoffrey L. Herman, Craig Zilles, and Michael C. Loui. 2014. A psychometric evaluation of the digital logic concept inventory. *Comput. Sci. Educ.* 24, 4 (2014), 277–303.
- [38] M. Hertz and S. M. Ford. 2013. Investigating factors of student learning in introductory courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE'13)*. ACM, New York, NY, 195–200. <http://dx.doi.org/10.1145/2445196.2445254>
- [39] D. Hestenes, M. Wells, and G. Swackhamer. 1992. Force concept inventory. *Phys. Teach.* 30 (1992), 141–158.
- [40] D. Hillis. 1991. Co-evolving parasites improve simulated evolution as an optimization procedure. *Artif. Life II* 10, (1991), 313–324.
- [41] Alan Huebner. 2010. An overview of recent developments in cognitive diagnostic computer adaptive assessments. *Pract. Assess. Res. Eval.* 15, 3 (2010).
- [42] H. Juillé and Jordan B. Pollack. 1998. Coevolving the “ideal” trainer: Application to the discovery of cellular automata rules. In *Proceedings of the 3rd Annual Genetic Programming Conference*. IEEE Press.
- [43] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE'10)*. ACM, New York, NY, 107–111. <http://dx.doi.org/10.1145/1734263.1734299>
- [44] Slava Kalyuga, Alexander Renkl, and Fred Paas. 2010. Facilitating flexible problem solving: A cognitive load perspective. *Educ. Psychol. Rev.* 22, 2 (2010), 175–186.
- [45] Kuba Karpietz and Steven A. Wolfman. 2014. Misconceptions and concept inventory questions for binary search trees and hash tables. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14)*. ACM, New York, NY, 109–114. <http://dx.doi.org/10.1145/2538862.2538902>
- [46] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (2005), 83–137.
- [47] A. Korhonen, O. Seppälä, and J. Sorva. 2015. Automatic recognition of misconceptions in visual algorithm simulation exercises. In *Proceedings of the IEEE Frontiers in Education Conference (FIE'15)*. IEEE, 1–5. <http://dx.doi.org/10.1109/FIE.2015.7344046>

- [48] A. N. Kumar. 2006a. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. *Technol. Instr. Cogn. Learn.* 3, 4, 1 (2006), 256–260.
- [49] A. N. Kumar. 2006b. A scalable solution for adaptive problem sequencing and its evaluation. In *Proceedings of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH'06)*. Springer, 161–171.
- [50] A. N. Kumar. 2012. Limiting the number of revisions while providing error-flagging support during tests. In *Proceedings of the Annual Conference on Intelligent Tutoring Systems (ITS'12), Lecture Notes in Computer Science*, Vol. 7315. Springer, Chania, Crete, 524–530.
- [51] A. N. Kumar. 2014a. An evaluation of self-explanation in a programming tutor. In *Proceedings of the Annual Conference on Intelligent Tutoring Systems (ITS'14), Lecture Notes in Computer Science*, Vol. 8474. Springer, 248–253.
- [52] A. N. Kumar. 2015a. The effectiveness of visualization for learning expression evaluation. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE'15)*. ACM, New York, NY, 362–367.
- [53] A. N. Kumar. 2015b. Solving code-tracing problems and its effect on code-writing skills pertaining to program semantics. In *Proceedings of the 20th Annual Conference on Innovation & Technology in Computer Science Education*. ACM Press, 314–319.
- [54] A. N. Kumar. 2014b. A model for deploying software tutors. In *Proceedings of the 6th International Conference on Technology for Education (T4E'14)*. IEEE Press, 3–9.
- [55] Andrew Luxton-Reilly. 2016. Learning to program is easy. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'16)*. Association for Computing Machinery, New York, NY, 284–289. <http://dx.doi.org/10.1145/2899415.2899432>
- [56] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18 Companion)*. Association for Computing Machinery, New York, NY, 55–106. <http://dx.doi.org/10.1145/3293881.3295779>
- [57] Francesc Martori, Jordi Cuadros, and Lucinio González-Sabaté. 2016. Studying the relationship between BKT fitting error and the skill difficulty index. In *Proceedings of the 6th International Conference on Learning Analytics & Knowledge*. ACM Press, 364–368.
- [58] Jan H. F. Meyer and Ray Land. 2003. Threshold concepts and troublesome knowledge: linkages to ways of thinking and practising. In *Improving Student Learning—Theory and Practice Ten Years on Oxford: Oxford Centre for Staff and Learning Development*. Oxford Press, Brussels, Belgium, 412–424.
- [59] Jan H. F. Meyer and Ray Land. 2005a. Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *High. Educ.* 49, 3 (2005), 373–388.
- [60] Jan H. F. Meyer and Ray Land. 2005b. Threshold concepts and troublesome knowledge: linkages to ways of thinking and practising. In *Improving Student Learning—Diversity and Inclusivity: Proceedings of the 12th Improving Student Learning Conference*. Oxford Press, Birmingham, UK, 53–64.
- [61] Behrooz Mostafavi, Michael Eagle, and Tiffany Barnes. 2015. Towards data-driven mastery learning. In *Proceedings of the 5th International Conference on Learning Analytics And Knowledge*. ACM Press, 270–274.
- [62] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, validation, and use of a language independent CS1 knowledge assessment. In *Proceedings of the ACM Conference on International Computing Education Research (ICER'16)*. ACM, New York, NY, 93–101. <http://dx.doi.org/10.1145/2960310.2960316>
- [63] E. Popovici, A. Bucci, R. P. Wiegand, and E. D. de Jong. 2012. Coevolutionary principles. In *Handbook of Natural Computing*. Springer, 987–1033.
- [64] Leo Porter, Saturnino Garcia, Hung-Wei Tseng, and Daniel Zingaro. 2013. Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'13)*. ACM, New York, NY, 279–284. <http://dx.doi.org/10.1145/2462476.2462490>
- [65] L. Porter, C. Taylor, and K. C. Webb. 2014a. Leveraging open source principles for flexible concept inventory development. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. ACM Press, 243–248.
- [66] Leo Porter, Cynthia Taylor, and Kevin C. Webb. 2014b. Leveraging open source principles for flexible concept inventory development. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE'14)*. ACM, New York, NY, 243–248. <http://dx.doi.org/10.1145/2591708.2591722>
- [67] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. <http://dx.doi.org/10.1145/3077618>
- [68] M. Reckase. 2009. *Multidimensional Item Response Theory: Statistics for Social and Behavioral Sciences*. Springer.
- [69] P. Rosenbaum. 1984. *Testing the Local Independence Assumption in Item Response Theory*. Technical Report. ETS.
- [70] Michael J. Sanger and Thomas J. Greenbowe. 1997. Common student misconceptions in electrochemistry: Galvanic, electrolytic, and concentration cells. *J. Res. Sci. Teach.* 34, 4 (1997), 377–398.

- [71] Ronny Scherer, Fazilat Siddiq, and Bárbara Sánchez Viveros. 2020. A meta-analysis of teaching and learning computer programming: Effective instructional approaches and conditions. *Comput. Hum. Behav.* 109, (2020), 106349.
- [72] Simon, Andrew Luxton-Reilly, Vangel V. Ajanovski, Eric Fouh, Christabel Gonsalvez, Juho Leinonen, Jack Parkinson, Matthew Poole, and Neena Thota. 2019. Pass rates in introductory programming and in other stem disciplines. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (ITiCSE-WGR'19)*. Association for Computing Machinery, New York, NY, 53–71. <http://dx.doi.org/10.1145/3344429.3372502>
- [73] Simon, Judy Sheard, Daryl D'Souza, Peter Klemperer, Leo Porter, Juha Sorva, Martijn Stegeman, and Daniel Zingaro. 2016. Benchmarking introductory programming exams: Some preliminary results. In *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER'16)*. ACM, New York, NY, 103–111. <http://dx.doi.org/10.1145/2960310.2960337>
- [74] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A review of generic program visualization systems for introductory programming education. *ACM Trans. Comput. Educ.* 13, 4, Article 15 (Nov. 2013), 64 pages.
- [75] M. Stocking. 1989. Empirical estimation errors in item response theory as a function of test properties. *ETS Res. Rep. Ser.* 1989, 1 (1989), 1–76.
- [76] Steven Tang, Elizabeth McBride, Hannah Gogel, and Zachary A. Pardos. 2015. Item ordering effects with qualitative explanations using online adaptive tutoring data. In *Proceedings of the 2nd ACM Conference on Learning@Scale*. ACM Press, 313–316.
- [77] C. Taylor, D. Zingaro, L. Porter, K. C. Webb, C. B. Lee, and M. Clancy. 2014. Computer science concept inventories: Past and future. *Comput. Sci. Educ.* 24, 4 (2014), 253–276.
- [78] A. E. Tew and M. Guzdial. 2010. Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. ACM Press, 97–101.
- [79] Allison Elliott Tew and Mark Guzdial. 2011. The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education (SIGCSE'11)*. ACM, New York, NY, 111–116. <http://dx.doi.org/10.1145/1953163.1953200>
- [80] Jan Vahrenhold and Paul Wolfgang. 2014. Developing and validating test items for first-year computer science courses. *Comput. Sci. Educ.* 24, 4 (2014), 304–333.
- [81] Christopher Watson and Frederick W. B. Li. 2014. Failure rates in introductory programming revisited. In *Proceedings of the Conference on Innovation & Technology in Computer Science Education*. ACM Press, 39–44.
- [82] R. Watson and Jordan B. Pollack. 2001. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, L. Spector et al. (Eds.). Morgan Kaufmann, San Francisco, CA, 702–709.
- [83] Kevin C. Webb and Cynthia Taylor. 2014. Developing a pre- and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE'14)*. ACM, New York, NY, 103–108. <http://dx.doi.org/10.1145/2538862.2538886>
- [84] R. Paul Wiegand, Anthony Bucci, Amruth N. Kumar, Jennifer L. Albert, and Alessio Gaspar. 2016. A data-driven analysis of informatively hard concepts in introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE'16)*. ACM, New York, NY, 370–375. <http://dx.doi.org/10.1145/2839509.2844629>

Received January 2021; revised June 2021; accepted June 2021