On the Potential of Evolved Parsons Puzzles to Contribute to Concept Inventories in Computer Programming

Mr. A.T.M. Golam Bari, University of South Florida

ATM Golam Bari, student member IEEE, is a Ph.D. student in Computer Science & Engineering Department at University of South Florida, USA. He received the ME and BSc. degree in Computer Science & Engineering from Kyung-Hee University, South Korea and Dhaka University, Bangladesh, in 2013 and 2007, respectively. His main research interest involves Coevolutionary Algorithms, Dynamic Optimization, Bio-data mining.

Dr. Alessio Gaspar, University of South Florida

Dr. Alessio Gaspar is an Associate Professor with the University of South Florida's Department of Computer Science & Engineering and director of the USF Computing Education Research & Evolutionary Algorithm Laboratory. He received his Ph.D. in computer science in 2000 from the University of Nice Sophia-Antipolis (France). Before joining USF, he worked as visiting professor at the ESSI polytechnic and EIVL engineering schools (France) then as postdoctoral researcher at the University of Fribourg's Computer Science department (Switzerland). Dr. Gaspar is an ACM SIGCSE, SIGITE and SIGEVO member and regularly serves as reviewer for international journals & conferences and as panelist for various NSF programs. His research interests include Evolutionary Algorithms, Computing Education Research, and applications to Computer-Assisted Teaching & Learning. His technology interests include Linux System Administration, Programming, Web App Development, and open source technologies in general.

Dr. R. Paul Wiegand, University of Central Florida, School of Modeling, Simulation, & Training

Dmytro Vitel

I was born in Ukraine, 1988. In 2011 I finished Taras Shevchenko National University of Kyiv and obtained degree Master of Science in Applied Physics. In August 2017, I was accepted into MSIT program at University of South Florida. Eventually, program was changed to MSCS.

Mr. Kok Cheng Tan

Kok Cheng Tan is a present PhD student of Computer Science at University of South Florida. He tends to work toward data science fields such as machine learning and data mining. He has eight-year teaching experiences and interested in exploring academical present trends.

Stephen John Kozakoff, University of South Florida

On the Potential of Evolved Parsons Puzzles to Contribute to Concept Inventory Design in Introductory Programming Courses

ATM Golam Bari[1] bari@mail.usf.edu

Alessio Gaspar[1] alessio@usf.edu

R. Paul. Wiegand[2] wiegand@ist.ucf.edu

Dmytro Vitel[1] dvitel@mail.usf.edu

Kock Cheng Tan[1] kokcheng@mail.usf.edu

Stephen Kozakoff[1] kozakoff@mail.usf.edu

[1]University of South Florida,
Computer Science and Engineering, 4202 E. Fowler Avenue
Tampa, FL, 33620, USA

[2] Institute for Simulation & Training
EECS, University of Central Florida
4000 Central Florida Blvd
Orlando, Florida, 32816, USA

Abstract

Our goal is to investigate whether techniques to automatically generate practice problems have also potential to assist in constructing Concept Inventories (**CI**) for computer programming. More specifically, we focus on a specific type of practice problem, Parsons puzzles, aimed at novice programmers. In this study, we propose **EvoParsons** - an automated way of evolving Parsons puzzle for newbie computer programmers, and more importantly we establish that EvoParsons can be a stepping stone of automating the process of building CI. EvoParsons is a software tool to improve students' learning of computer programming. It is developed, maintained and distributed by our team. The state-of-the art techniques of building CI largely depends on several iterations of settings among faculties, interviews and surveys from students. This so called Delphi method largely depends on knowledge of domain experts, feedback from students, surveys etc. EvoParsons goal is to automate this process by applying Competitive Coevolutionary Algorithm (**CCoEA**) and Interactive Evolutionary Algorithm (**IEA**). In this paper, we first describe EvoParsons working mechanism, its benefits over other existing systems of generating Parsons puzzle. Second, we use EvoParsons' interaction data with actual student to describe its potential to contribute for building CI. To do so, we perform data driven analysis of EvoParson's

misconceptions that are found at the last generational puzzle of the experiment. We also analyze its interaction log to investigate pedagogical importance of misconceptions in successive generations. Experimental analysis shows that EvoParsons evolves interesting misconceptions, discards trivial ones, maintains an order of misconceptions in its subsequent generations of evolution.

1 Introduction

Parsons programming puzzles[1] are family of code completion assignments where codes along with distractors are given in random order. The task is to sort the code in correct order by selecting correct code of lines only. Since their inception, several independent studies have repeatedly illustrated the benefits of Parsons puzzle[2,3,4,5,6].

The computing education literature has formalized lists of topics taught in certain courses and areas of computing into CI. These inventories have been developed for digital logic[7], discrete mathematics[8], operating systems[9], Algorithms analysis[10] and introductory programming[11,12,12,13,14]as a few examples

While these studies demonstrate significant impact, current methods for CI creation lack consensus, have difficulties identifying appropriate distractors, and are overall resource-intensive to apply.

In this work, we first describe our in-house system of evolving Parsons puzzle; EvoParsons. Then we perform a data driven analysis of EvoParsons' interaction with actual student to conclude that EvoParsons can be a stepping stone to automate CI building process.

In short, EvoParsons is a hybrid system of CCoEA[15 16] and IEA[17]. CCoEA is a population-based search meta-heuristic. It maintains a co-adaptive interaction between participating populations. By switching such interactive blend of evolution, EvoParsons opens the door of unprecedented opportunities to go beyond just adapting to independent learning experiences.

We believe this qualitative innovation brings our algorithms closer to the level of generalization that also differentiates educational research from teaching practice. As such, evolutionary techniques may not only benefit individual students, but also unveil insights motivating further educational research. We investigate EvoParsons to answer the following two research questions;

- **Research Question 1 (RQ_1):** Does EvoParsons evolve *relevant misconceptions*[18] that classify student's errors? By relevant we mean if the interesting and harder misconceptions are getting priority over trivial misconceptions.
- **Research Question 2 (RQ_2):** Does EvoParsons consistently evolve meaningful misconceptions?

There have been limited applications of evolutionary techniques to the educational domain, in general, and to the automated generation of Parsons puzzle in particular. This work also established the foundations for the study of coevolutionary learning dynamics in populations of human learner from a game-theoretic perspective[19]. Even more recently, theoretical results explaining observed coevolutionary dynamics have been applied to gain insights about the difficulties encountered by novice programmers in an introductory programming course[20,21].

2 Related Work

The original Parsons puzzle[1] were generated using a framework named Hot Potato[22]. The framework allows drag-and-drop facility. The puzzle designer needs to provide a question header describing what the program is intended to do, followed by the drag-able items; i.e., the valid fragments of code in order, followed by distractors. The correct solutions and distractors are then shuffled before to be presented to the students.

In[23], the authors presented Parsons puzzle for CS1 examination. Each line was rearranged and presented as two slightly syntactic variations. Students' task is to select the correct line and re-arrange the lines.

ViLLE[24] is a language independent program visualization tool. It also includes Parsons puzzles created by its user. It does not support distractors. Students can step through a visualization of his/her execution and sort some of the line of codes.

js-parsons[25] introduces a new family of Parsons puzzle for the Python programming languages. It was motivated by the student feedback obtained in the original Parson's puzzle work[1]. js-parsons supports two visualization modes; basic mode in which lines of code are sorted and distractors are not allowed, and left to right mode, in which distractors are allowed.

Ericson's work on dynamically adaptive Parsons problem[26] is one of the most recent and thorough research on the topic. Adaptive Parsons puzzles are implemented as a variant of js-parsons and take into consideration learners' past and current performance in order to tune the difficulty level of the next puzzles presented to the student.

Another recent work, Epplets[27], generates Parsons puzzle as randomized instances of parameters puzzle templates. A puzzle is written in a specific template notation. The template is then used to generate an entire program, in the correct order. Distractors are created by a library of bug specifications which contains a regular expression pattern.

While providing Parsons puzzles to the students, the learning tool should not distribute random or similar puzzles more frequently. Instead, tools need to consider a balance between interesting, similar and random puzzles. This is an important policy of puzzle distribution that can improve learning gradient for the students. Otherwise, learners may lose interest to interact with the system. We did not find any available tool that give focus such disbursement policy to make the system engaging for its learners.

In addition, all the studies described so far adapt a single user to improve his/her learning skills. While this is a great benefit, we should also consider adaptability of the whole user base, not just a single user. Not any tools described above or found in the literature focus on generalization of difficulties while solving Parsons puzzles. This is because the single user adaptation in traditional

ITS can't reveal anything about the general difficulties students struggle during their interaction with the system.

3 System: EvoParsons

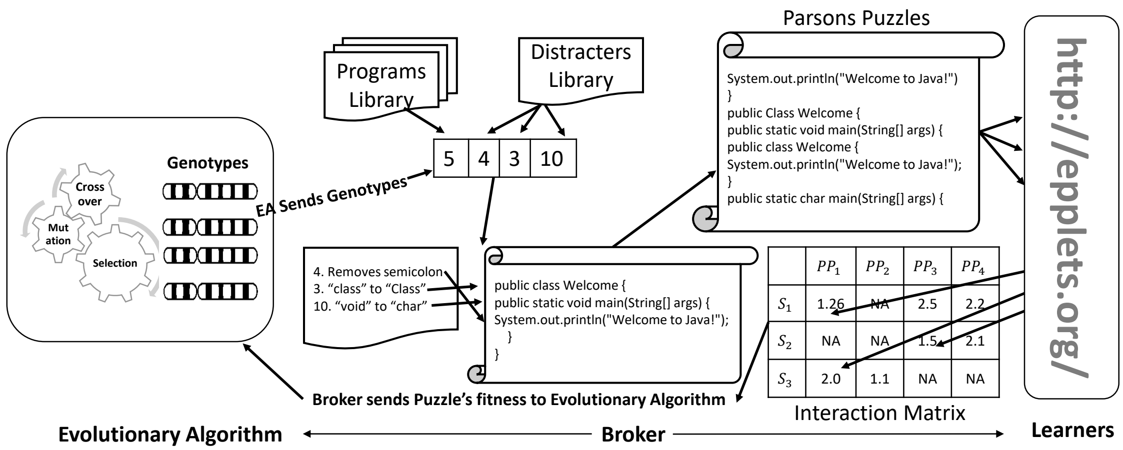

Figure 1: Overall workflow of EvoParsons. Evolutionary Algorithm (**EA**) sends genotypes to the **Broker**. The genotypes are mapped into Parsons puzzles inside the broker. Puzzles are then distributed to the students, on demand. All the student activities are sent back to the broker and then EA.

Figure 1 depicts the overall architecture of EvoParsons. It has three modules – EA, broker, and learner interface. The broker is responsible for mapping genotypes into Parsons puzzles by using both a "Program" and a "Transform" library. Program library contains several Java programs that includes concepts for the beginner programmers (e.g., variable declaration, initialization, control and loop statements). Transform library are used to match valid program lines. Then transform those valid lines to generate distractors. The puzzles are distributed on-demand one by one to the students after they log into the system. Students solve their assigned puzzle by identifying and "trash"-ing the distractors, and putting valid lines into correct order. As soon as a student submits correct ordering of the puzzle, the broker uses this information to identify the next puzzle to provide to that student and forwards this information, augmented as relevant, to the EA. The EA then uses this information to determine the fitness of the genotype that corresponds to this Parsons puzzle. In EvoParsons, the EA is a variant of Population-based Pareto Hill Climber (**PPHC**)[28].

3.1 Theoretical Basis of EvoParsons

It maintains the competition between Parsons puzzles and student population such that the puzzles evolve while students interact with EvoParsons. On the other hand, the evaluation of puzzles by student evaluators requires crafty user fatigue mitigation techniques[?] .

3.1.1 PPHC variant: EvoParsons' Core Algorithm

PPHC[28] features two populations; one for candidates and one for tests. Each individual in the candidate population interacts against each of the test individuals, and vice versa. Each test is treated as an objective in the sense of multi-objectives optimization[29]. The outcome vectors of two individuals (either for candidate or for test) can then be compared using the concept of Pareto-dominance.

In PPHC, each individual that encodes a Parsons puzzle is referred as parent. A parent is mutated in order to create a so-called child; another individual, slightly modified, that encodes a new Parsons puzzle. So, each individual has two forms - parent and child.

The outcome vector of a child ($\vec{x_c}$) Pareto-dominates that of a parent ($\vec{x_p}$), which is denoted by $\vec{x_c} \succ \vec{x_p}$, iff

- $f_i(\vec{x_c}) \geq f_i(\vec{x_p})$ for all i in $\vec{f}$
- There is at least one i such that $f_i(\vec{x_c}) > f_i(\vec{x_p})$

A child replaces its parent in the next generation if it is *strictly better*; i.e., if its outcome vector Pareto-dominates that of parent's. The candidates are thus evolved based on the concept of Pareto dominance.

In EvoParsons, we use a variant of PPHC, named PPHC-P. This variant uses Pareto dominance for both candidate and test evolution[30]. From the implementation perspective, we leveraged time-established, software components;

- Amruth Kumar's latest extension to the *Problet* tutoring system, *Epplets*, available at *http://epplets.org/*, which allows students to interact with Parsons puzzles and receive automated feedback.
- Sean Luke's *ECJ* Java framework, available at *https://cs.gmu.edu/texttildelow/projects/ecj/*, which provides implementations of many EA variants and that we extend to also implement P-PHC-C.

We extend both components so as to allow them to inter-operate via the broker, and communicate with the latter using Remote Method Invocation in Java technology.

3.1.2 Evolution of Parsons Puzzle using P-PHC variant

Let us take, as example, the parent genotype $PP_1 = [5, 4, 3, 10]$. P_1 mutates and creates its child genotype $PP_2 = [6, 3, 4, 11]$. Both PP_1 and PP_2 are mapped from genotype to Parsons puzzles.

The mapping process inside broker starts by retrieving program number 5 from the programs library and transforms numbered $4, 3, 10$ from the transforms library. Similarly, we will assume that the transforms $4, 3, 10$ respectively remove the semicolon after a statement, capitalize the "class" keyword to "Class", and replace occurrences of keyword "void" by keyword "char". For the sake of this example, we will assume that program 5 simply displays "Welcome to Java"

```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

After applying the above transforms in program 5 and shuffling the valid and invalid line of codes, we get the following Parsons puzzle.

```
  System.out.println("Welcome to Java!")
  }
 }
public Class Welcome {
  public static void main(String[] args) {
public class Welcome {
  System.out.println("Welcome to Java!");
public static char main(String[] args) {
```

Similarly, PP_2 is mapped into a different Parsons puzzle, using the same mapping process. The puzzle mapped from PP_2 may be totally different than that of PP_1. Both puzzles of the pair (i.e., PP_1 and PP_2) are evaluated by same set of students.

We use a minimum of two evaluations for each puzzle in the pair. As soon as a pair puzzle have interacted with same set of minimum number of students, the relative number of moves (ratio of of required move to solve that puzzle and total line of the puzzle) of parent and child are compared based on Pareto dominance. For example, two students S_1 and S_3 both solve PP_1 and PP_2. They take 1.26 and 2.0 relative number of moves to solve PP_1. So, PP_1's interaction outcome vector is $V_{PP_1} = (1.26, 2.0)$ (Please see Figure 1). Similarly, PP_2's outcome vector is $V_{PP_2} = (1.09, 1.11)$. As V_{PP_1} Pareto dominates V_{PP_2}, PP_1 is strictly better than PP_2. Hence, PP_1 is kept in the next generation. The same pair-wise dominance relation is computed for all the individuals.

The algorithm moves to the next generation when each (parent, child) pair have been evaluated by the minimum number of unique students. The students are anonymously uniquely identified by EvoParsons. As it is unethical to force all the students to solve all the paired puzzles, the broker needs to maintain better selection policy to expedite PPHC-P's evolution. consider that it has been properly evaluated (minimum of 2 evaluations).

4 Experiment

EvoParsons was used during Spring 2017 with Information Technology students enrolled in an online introductory programming course at the University of South Florida (COP2512 Programming Fundamentals for IT). The course is meant as a first introduction to programming for sophomores and is a state mandated prerequisite for the USF BS in Information Technology program[1]

We conducted one experiment at the beginning of the semester after students were exposed to basic Java concepts; data types, selection and iteration. In the course timeline, this means that it took place after module [203] (see previously referenced website for details). During this experiment, students were assigned to use our software and work on evolved Parsons puzzles for a minimum of 30 minutes as practice. A total of 107 students participated in this first experiment. The broker had 38 items in its transforms library and 40 Java programs in its program library. The genotypes were set to a length of 10 and the population size to 10 genotypes. The programs library covered three Java topics that had been presented early in the course; data types, selection and iteration from module [201], [202] and [203] respectively. PPHC-P ran for a total of six generations as students worked on their assignments and explored 79 unique genotypes.

5 Results

We consider the relative number of moves students took to solve each of those 10 puzzles in both experiments (Please see Table 5). We found that students took fewer moves on average in the second experiment. Two out of the 10 puzzles showed significant improvement ($p < 0.01$ for **Palindrome Detector** and $0.01 < p < 0.05$ for **Multiplication Table (variant)**). The differences for other puzzles failed to prove statistically significant.

5.1 EvoParsons's Evolution of Misconceptions (Answering RQ_1)

Identifying misconceptions is the stepping stone of building CI for any discipline. The Delphi method[11] is used to build CI for computer programming. Recently the study of semi-structured interviews[14,18] is also considered to classify students' misconceptions in introductory C programming. While Delphi and other state of the arts have identified some misconceptions in computer programming, these are based on data analysis of previous examinations, and mostly domain specific knowledge from the educators, faculties and students. While answering the above question, we investigate if EvoParsons evolve misconceptions that are validated by state of the arts.

[1]The course material is freely available at *http://cereal.forest.usf.edu/edu/COP2512/* so that the reader may have access to all details regarding the material to which students were exposed prior to using our software.

Table 1: Mean relative moves for the last generational puzzles evolve by EvoParsons. The moves are shown for both experiment. Please note that, Experiment #1 is evolved while Experiment #2 is non-evloved version but with same class.

Name of the puzzle	μ_{Exp1}	μ_{Exp2}	**Remarks**
Compute Circle Area, V3.0	3.5	1.9	$p > 0.05$
HexaDecimal to Decimal	3.41	2.94	$p > 0.05$
Palindrome Detector	2.62	1.86	$p < 0.01$
Do while loop with sentinel	1.6	1.15	$p > 0.05$
Greatest Common Divisor	1.22	1.30	$p > 0.05$
String Operation	1.38	1.31	$p > 0.05$
A Simple Quiz for Subtraction	1.31	1.27	$p > 0.05$
Multiplication Table	1.7	1.65	$p > 0.05$
Multiplication Table (variant)	1.66	1.35	$0.001 < p < 0.05$
A Guess Game	1.31	1.35	$p > 0.05$

5.1.1 Misconceptions Covered by EvoParsons Transform Library

The library of misconception in EvoParsons is compiled mostly for Java programs. More specifically, we design transforms that captured syntactical and semantic bugs. Table 2 categorizes some of the misconceptions that can be generated by EvoParsons. Please note that "a", "b", "x" and "xxx"' are the placeholders for the actual variables used in our program library.

5.1.2 Misconceptions Evolved by EvoParsons vs State of the Arts

EvoParsons' misconception library is mostly designed from literature of CI[18] for C programming and also from[31,32] where authors discussed errors found in C and Java Programs. We focus on the errors that are common for both C and Java Programs. For example, the "removing semicolon" error can be applied for both languages. On the other hand, some of the misconceptions in C are out of scope for Java e.g., pointers-related errors. Also, the rules we follow to design misconceptions for EvoParsons hinder implementing some of the errors like "out of scope" error.

However, we also design some misconceptions that we think may help a novice programmer to learn syntax and semantic errors. Some of them are as follows;

- Converting "%" into "/" found in any expression
- Capitalizing keywords e.g., "int" into "Int"

Adding such new misconceptions in EvoParsons transform library helps us to investigate if they are found in the last generation of P-PHC or they are discarded by EvoParsons in earlier generations. We also listed common misconceptions from CIs of computer programming. Figure 2 shows them. Please note that, EvoParson's transform library is also capable to produce these misconceptions. As mentioned earlier, this library has some other new misconceptions. We were

Table 2: Type of misconceptions and their examples that can be evolved by EvoParsons's transform library

Misconceptions	Sub Types	Original Line	Error Line
Semantic	**Off by One**	if ($a <= b$)	if ($a < b$)
		else if ($a > b$)	else if ($a >= b$)
		while ($a < b$)	while ($a <= b$)
		char a = x.charAt(0);	char a = x.charAt(1);
		a = 0;	a = 1;
	Miscellaneous	if ($a\%b$)	if (a/b)
		if (xxx) {	if (xxx)
		while (xxx) {	while (xxx)
Syntactical	**Capitalized Keywords**	main	Main
		class	Class
		int a, b;	Int a, b;
		public	Public
	Remove Semicolon	number = x;	number = x
		int a = b;	int a = b
		System.out.println(”xxx”);	System.out.println(”xxx”)
	Miscellaneous	if(xxx)	if(xxx
		while(xxx)	while(xxx
		$==$	$=$

interested to see if EvoParsons last generational puzzle have those misconceptions. To do so, we list those misconceptions in Table 3

Table 3: Misconceptions found at the last generation of EvoParsons

Original lines	Their distractors
public class TestDoWhile	Public class TestDoWhile
public class SubtractionQuiz hline if (number1 - number2 == answer)	Public class SubtractionQuiz if (number1 - number2 = answer)
public static void main(String[] args)	public static void Main(String[] args)
while ($k \leq n1 \quad \&\& \quad k \leq n2$)	while ($k \leq n1 \quad \&\& \quad k \leq n2$
if (n1 % k == 0 && n2 % k == 0)	if (n1 / k == 0 && n2 % k == 0)
guess = -1; number = (int)(Math.random() * 101);	guess = -1; number = (int)(Math.random() * 101)
int guess, number;	Int guess, number;
else if (guess $>$ number)	else if (guess $\geq$ number)

5.1.3 Discussion

We observed that the misconceptions found in the puzzles from the last generation of P-PHC include some of the misconceptions already identified by CIs in computer programming. It is also

<table>
<tr><th colspan="2">Koenig et al., 1989</th></tr>
<tr><th>Original Line</th><th>Error Line</th></tr>
<tr><td>if (a == b)</td><td>if (a =b)</td></tr>
<tr><td>a < b && c < d</td><td>a < b & c < d</td></tr>
<tr><td>a < b || c < d</td><td>a < b | c < d</td></tr>
<tr><td>if(a > big)
big =a;</td><td>if (a > big);
big = a;</td></tr>
</table>

Error, Percentage of students done that error in Tuugalei et al., 2012
Variable not found (59%)
Mismatched brackets/parenthesis(6.4%)
; expected(1.2%) in HCS281, 1.4% in HCS286
Possible loss of precision(1.3%)
Incompatible types found (1.4%)

Errors in Caceffo et al., 2018	
Error Name	**Example**
Improper initialization of loop counter	for (int i = 1; i < 9; i++) sum = sum + i;
Wrong control flow in a loop	for (int i = 0; i <= 9; i++) sum = sum + i;

Figure 2: Some of the misconceptions aligned with EvoParsons' Program library. EvoParsons misconceptions listed in Table 2 can generate these misconceptions.

interesting that those puzzles do not have misconceptions such as float a; $->$ char a;, $\&\&->\&$, $||->|$. The analysis of log shows that these misconceptions are discarded because EvoParsons selection mechanism decided worthless to evolve.

Please note that, some trivial misconceptions such as "capitalizing keywords" are found in the last generation of P-PHC. It is interesting to investigate if they propagate to new generations because the puzzle contains some other interesting misconceptions found in the literature.

5.2 EvoParsons Consistency of Evolving Meaningful Distractors (Answering RQ_2)

We examined the log, over all the generations, of the selection decisions between parent puzzle and their respective children puzzle. This allowed us to trace the origin of the evolved puzzles in terms of successive mutations and selections steps. To do so, we took some of the puzzles found in the last generation of P-PHC and trace them from first generation and built a hierarchy of misconceptions based on EvoParson's multi-objective comparison to promote puzzles into next generation.

5.2.1 Tracing the log of EvoParsons Last Generational Puzzle

- The distractors in "Hex2Dec – Hexadecimal to Decimal Converter" are trickier than that of its pair puzzles in previous generations.

  Hex2Dec has two "Off by One" semantic misconceptions – if $a <= b ->$ if $a < b$ and char a = x.charAt(0) $->$ char a = x.charAt(1) – and another miscellaneous one; if(xxx){ $->$ if(xxx). The generational log shows that these misconceptions are better in differentiating student's performance than syntactic misconceptions e.g., (main, int) $->$ (Main, Int), removing semicolon and Off by One distractor like while(x != 0) $->$ while(x != 1).

- The misconceptions in "Palindrome detector" has two "Off by One" bugs and one syntax error; *while(xxx)* $->$ *while (xxx*. On the other hand, its other pairs in previous generations have all syntactic errors like (class, int, public) $->$ (Class, Int, Public). It seems that EvoParsons prioritizes semantic errors over syntactic ones, while promoting better puzzles in the next generation.

- The "Subtraction Quiz" suggests that the distractor — converting **==** into **=** inside a conditional expression of control statement — is harder to detect than another semantic distractor (converting **1** into **0** in a variable initialization), and also harder to detect than any other syntactical distractor considered.

- The context of "Usage of do while loop" to find out summation of all the integers provided by the user, seems harder than all the pair puzzles in previous generations. The puzzles were two different programs with different distractors at different generations; First one is "leap year determiner": determining if an input year is a leap year or not. The second one is "addition verifier": verify the addition of two already initialized integers until correct result is provided by the user. It can be inferred that the use of a **do while** loop instead of a **while** loop (in "addition verifier" puzzle), along with the use of a sentinel value in the "Usage of do while loop" program, makes it informatively harder than all of its children.

- The "GCD — Greatest Common Divisor — Calculator" is contextually harder than its parent "leap year determiner" though the latter had more syntactic distractors than the former (and same semantic distractor). However, "GCD calculator" wins from generation #4 though it looses in previous generations against "leap year determiner". This may be due to differences in the distribution of student performance between generations #3 and # 4.

Figure 3 shows the hierarchy of misconceptions by analyzing the multi-objective selection mechanism of EvoParsons. For any puzzle, the misconception at the top is interesting or contribute more to distinguish student's performance than its lower misconception. As for example, the misconception $if(a = b)$ in "Subtraction Quiz" is more important and interesting than a misconception that focus on syntax error i.e., convert main int Main.

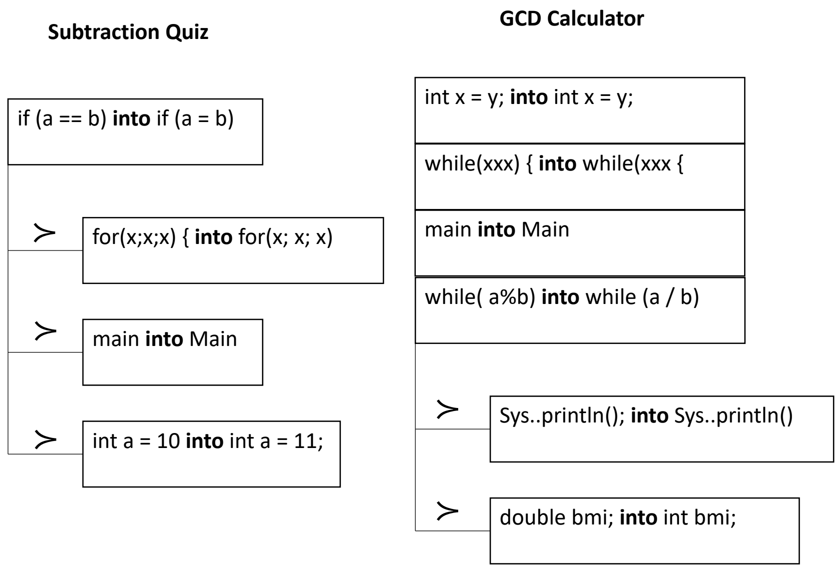

Figure 3: The distractor in Subtraction Quiz helps this practice problem to be more competitive i.e., Pareto dominate than its child Practice problem. The same description applies for GCD Calculator and Guess game.

5.2.2 Discussion

Analysis of multi-objective selection mechanism of EvoParsons show that the misconceptions found in the last generational puzzles of EvoParsons promote interesting misconceptions than trivial ones. This promotion is consistent in successive generations. Please note that, evolution of such misconceptions while maintaining consistency of evolution make EvoParsons an important tool to contribute to the design of CI in introductory programming course.

6 Conclusion

We propose a new learning tool for introductory programming course that automate the process of building CI. The evolution of Parsons puzzle using our tool; EvoParsons and its contribution to automate the design of CI in introductory programming course will benefit computing education research community. It alleviates the semi-interview process of sate of the art i.e., Delphi method to build CI. EvoParsons also captures interesting and generalized misconceptions that maximize the performance of student population.

Currently this learning software can be used only for Java Programming course. However, it can be extended for other programming courses e.g., C and Python and even for different courses of computer science.

Acknowledgments

This material is based in part upon work supported by the Association for Computing Machinery's SIGCSE Special Projects 2015 award, and the National Science Foundation under awards #1504634, #1502564, and #1503834. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Our team acknowledges Dr. Amruth Kumar for allowing our prototype to connect via the *epplets.org* user interface, while working on the above-mentioned NSF proposals.

Our team also acknowledges Dr. Anthony Bucci for his valuable feedback to improve EvoParsons' coevolutionary interaction mechanism.

Besides the researchers involved in this project, many Information Technology students from the University of South Florida also significantly contributed to the project;

- Paul Burton implemented the original proof of concept software during his IT Senior Project in spring 2015, and refined it under OPS contract during summer 2015.
- Stephen Kozakoff extended the prototype and connected it to *Epplets.org* as part of his MSIT graduate practicum in fall 2015 and spring 2016.
- Himank Vats contributed to the Docker containerization of the server-side components as part of his MSIT graduate practicum in 2017.

Our team gratefully acknowledges the received funding support, as well as the participating students' dedication and enthusiasm.

References

[1] Dale Parsons and Patricia Haden. Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 157–163, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc. ISBN 1-920682-34-1. URL `http://dl.acm.org/citation.cfm?id=1151869.1151890`.

[2] Barbara J Ericson, Lauren E Margulieux, and Jochen Rick. Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research*, pages 20–29. ACM, 2017.

[3] Juha Helminen, Petri Ihantola, Ville Karavirta, and Lauri Malmi. How do students solve parsons programming problems?: An analysis of interaction traces. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 119–126, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1604-0. doi: 10.1145/2361276.2361300. URL `http://doi.acm.org/10.1145/2361276.2361300`.

[4] Ville Karavirta, Juha Helminen, and Petri Ihantola. A mobile learning application for parsons problems with automatic feedback. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, Koli Calling '12, pages 11–18, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1795-5. doi: 10.1145/2401796.2401798. URL `http://doi.acm.org/10.1145/2401796.2401798`.

[5] Barbara J. Ericson. Adaptive parsons problems with discourse rules. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, pages 259–260, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3630-7. doi: 10.1145/2787622.2787740. URL `http://doi.acm.org/10.1145/2787622.2787740`.

[6] Briana B. Morrison, Lauren E. Margulieux, Barbara Ericson, and Mark Guzdial. Subgoals help students solve parsons problems. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 42–47, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844617. URL `http://doi.acm.org/10.1145/2839509.2844617`.

[7] Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. Proof by incomplete enumeration and other logical misconceptions. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 59–70, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-216-0. doi: 10.1145/1404520.1404527. URL `http://doi.acm.org/10.1145/1404520.1404527`.

[8] Vicki L. Almstrum, Peter B. Henderson, Valerie Harvey, Cinda Heeren, William Marion, Charles Riedesel, Leen-Kiat Soh, and Allison Elliott Tew. 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, pages 132–145, New York, NY, USA, 2006. ACM. ISBN 1-59593-603-3. doi: 10.1145/1189215.1189182. URL `http://doi.acm.org/10.1145/1189215.1189182`.

[9] Kevin C. Webb and Cynthia Taylor. 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, pages 103–108, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2605-6. doi: 10.1145/2538862.2538886. URL `http://doi.acm.org/10.1145/2538862.2538886`.

[10] Mohammed F. Farghally, Kyu Han Koh, Jeremy V. Ernst, and Clifford A. Shaffer. Towards a concept inventory for algorithm analysis topics. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 207–212, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4698-6. doi: 10.1145/3017680.3017756. URL `http://doi.acm.org/10.1145/3017680.3017756`.

[11] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M.C. Loui, and C. Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin*, 40(1): 256–260, 2008.

[12] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. Setting the scope of concept inventories for introductory computing subjects. *Trans. Comput. Educ.*, 10 (2):5:1–5:29, June 2010. ISSN 1946-6226. doi: 10.1145/1789934.1789935. URL `http://doi.acm.org/10.1145/1789934.1789935`.

[13] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, SIGCSE '10, pages 107–111, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0006-3. doi: 10.1145/1734263.1734299. URL `http://doi.acm.org/10.1145/1734263.1734299`.

[14] Ricardo Caceffo, Steve Wolfman, Kellogg S. Booth, and Rodolfo Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, SIGCSE '16, pages 364–369, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3685-7. doi: 10.1145/2839509.2844559. URL `http://doi.acm.org/10.1145/2839509.2844559`.

[15] D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Artificial Life II*, 10:313–324, 1991.

[16] Richard A Watson and Jordan B Pollack. Coevolutionary dynamics in a minimal substrate. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 702–709. Morgan Kaufmann Publishers Inc., 2001.

[17] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001.

[18] Ricardo Caceffo, Raysa Benatti Guilherme Gama, and Rodolfo Azevedo Tales Aparecida, Tania Caldas. A concept inventory for cs1 introductory programming courses in c. In *In Technical Report 18-06, Institute of Computing, University of Campinas, SP, Brasil*, Brasil, 2018.

[19] A. Bader-Natal. *The Teacher's Dilemma: A game-based approach for motivating appropriate challenge among peers*. PhD thesis, Michtom School of Computer Science, Brandeis University, 2008.

[20] R. Paul Wiegand, Anthony Bucci, Amruth N. Kumar, Jennifer L. Albert, and Alessio Gaspar. 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, pages 370–375, 2016. ISBN 978-1-4503-3685-7.

[21] Anthony Bucci, R. Paul Wiegand, Amruth N. Kumar, Jennifer L. Albert, and Alessio Gaspar. Dimension extraction analysis of student performance on problems. In *Proceedings of the 29^{th} International Conference of the Florida Artificial Intelligence Research Society*, FLAIRS'16, 2016.

[22] Petri Ihantola and Ville Karavirta. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education*, 10:119–132, 2011.

[23] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 113–124, 2008. ISBN 978-1-60558-216-0. URL `http://doi.acm.org/10.1145/1404520.1404532`.

[24] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Ville: a language-independent program visualization tool. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research-Volume 88*, pages 151–159. Australian Computer Society, Inc., 2007.

[25] Petri Ihantola and Ville Karavirta. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education*, 10(2):119–132, 2011.

[26] Barbara Jane Ericson. *evaluating the effectiveness and efficiency of parsons problems and dynamically adaptive parsons problems as a type of low cognitive load practice problem*. PhD thesis, Georgia Institute of Technology, 2018.

[27] Amruth N Kumar. Epplets: A tool for solving parsons puzzles. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 527–532. ACM, 2018.

[28] Anthony Bucci and Jordan B Pollack. Focusing versus intransitivity geometrical aspects of co-evolution. In *Genetic and Evolutionary Computation Conference*, pages 250–261. Springer, 2003.

[29] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[30] Alessio Gaspar, A.T.M. Golam Bari, Amruth N. Kumar, R. Paul Wiegand, Anthony Bucci, and Jennifer L. Albert. Evolutionary practice problems generation: Design guidelines. In *28th IEEE International Conference on Tools with Artificial Intelligence*, ICTAI'16, 2016.

[31] IT Chan Mow. Analyses of student programming errors in java programming courses. *Journal of Emerging Trends in Computing and Information Sciences*, 3(5):739–749, 2012.

[32] Andrew Koenig. *C traps and pitfalls*. Pearson Education India, 1989.