EDGEWORTH: Authoring Diagrammatic Math Problems using Program Mutation

Hwei-Shin Harriman

hwei.shin.harriman@gmail.com

Olin College of Engineering

Carnegie Mellon University

USA

Figure 1. Left: Given declarative mathematical statements representing a diagram in its standard position, EDGEWORTH automatically generates diagrams with distinct visual semantics by mutating the prompt statements. Right: EDGEWORTH makes it easy to generate diagrammatic multiple choice questions.

Abstract

Building connections between mathematical expressions and their visual representations increases conceptual understanding and flexibility. However, students rarely practice visualizing abstract mathematical relationships because developing diagrammatic problems is challenging, especially at scale. To address this issue, we introduce EDGEWORTH, a system that automatically generates correct and incorrect diagrams for a given question prompt. It does so by mutating declarative mathematical statements with visual semantics. We evaluated the system by recreating diagrammatic problems in a widely used geometry textbook.

CCS Concepts: • Applied computing → Computer-assisted instruction.

Keywords: program mutation, programming languages, computer graphics, math education

ACM Reference Format:

Hwei-Shin Harriman. 2021. EDGEWORTH: Authoring Diagrammatic Math Problems using Program Mutation. In *Companion Proceedings of the 2021 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '21)*, October 17–22, 2021, Chicago, IL, USA. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/3484271.3484978>

1 Introduction

Solving diagrammatic problems builds connections between mathematical expressions and visual representations, which is known to increase conceptual understanding and flexibility [3]. However, authoring such problems remains a tedious, manual process. Even creating simple shapes can be challenging, so problem authors often resort to recycling course content, providing multiple similar examples, or re-purposing diagrams found online [5]. We interviewed content authors and found that the challenges compound when considering

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 ACM 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.

SPLASH Companion '21, October 17–22, 2021, Chicago, IL, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-9088-0/21/10...\$15.00

<https://doi.org/10.1145/3484271.3484978>

the issue of scale, as many existing tools lack support for high-level tweaking of diagrams or are difficult to use for non-programmers. As a result, even making minor semantic edits to batches of diagrams can quickly become a large undertaking.

To address these issues, in preliminary work we have developed a system called PENROSE [10], which allows users to generate diagrams by writing mathematical statements in small programs called SUBSTANCE files. The SUBSTANCE file, which defines the content of a given diagram, is bounded by a DOMAIN schema, and rendered by a STYLE program that defines and solves an optimizable specification.

EDGEWORTH makes use of program mutation techniques to mutate the SUBSTANCE files. The concept of program mutation has long been used as a method to reveal unforeseen characteristics in software programs. It is commonly used in mutation testing [9][4][8][7] and program repair [2][6], to create new software tests or increase coverage. These tools aim to explore all possible pathways through a program to uncover hard-to-find bugs. Therefore, these tools seek out big differences in programs, while our approach intentionally looks for mutants that are closely related, as the difference between a correct and a distractor diagram is often minimal, as seen in Fig. 1.

2 System Design

EDGEWORTH takes the prompt program and synthesizes numerous alternate SUBSTANCE files, thereby altering the meaning of the resulting diagrams. The prompt undergoes a number of mutations which are randomly selected from our basis set of seven mutation operators. Fundamentally, there are 3 types of operators: adds, deletes, and updates. We uncovered four update operations for SUBSTANCE programs through our exploration of geometry problems. The full set of seven operators are as follows:

- **Add** Appends a statement to the SUBSTANCE program.
- **Delete** Removes a random statement from the SUBSTANCE program.
- **Cascading Delete** Removes a random statement and all other references to that statement.
- **Swap Arguments** Reorders the arguments passed into a statement. e.g., if A and B are Triangles:
 $\text{Similar}(A, B) \rightarrow \text{Similar}(B, A)$
- **Swap-In Arguments** Replaces the arguments passed into a statement with other arguments defined in scope. e.g., if A, B, C, D are Points:
 $s := \text{MkSegment}(A, B) \rightarrow s := \text{MkSegment}(C, D)$
- **Replace Statement Name** Replaces a statement with a different statement that takes the same type of arguments and has the same return type. e.g., given that T is a Triangle:
 $\text{Right}(T) \rightarrow \text{Obtuse}(T)$

- **Type Change** Replaces a statement with a new one that takes the same number and type of arguments, but does not necessarily return a value of the same type. e.g., if E is an Angle:

$\text{Segment } s := \text{Bisector}(E) \rightarrow \text{Right}(E)$

The number of mutations per generated program, as well as the type of statements that can be added, deleted, or replaced are configuration parameters. Thus, we are able to quickly generate diverse variations of the source diagram from a very small input.

It is up to the problem author to select the final subset of diagrams that they wish to include in their question. To facilitate this, we implemented a web app interface that allows the author to view the generated diagrams as well as their corresponding SUBSTANCE program, and select the final subset that they wish to use.

3 Evaluation

We evaluated the system by recreating problems in a widely used geometry textbook [1]. We examined all 53 diagrammatic problems in the chapter review sections to determine our mutation set, then picked a representative subset of 24 problems and implemented them in EDGEWORTH. The remaining 29 problems involved the same mutations or repeated conceptual content.

For each textbook problem, we reframe the question so that the diagram accompanying the original problem can be considered a correct answer. Then, we consider possible answers to the posed question, including “distractors”—answers that are designed to tempt students with limited conceptual understanding—as well as correct “special cases”. We then describe the original diagram as a prompt SUBSTANCE program and pass it to EDGEWORTH, which generates numerous right and wrong answers to the original problem.

In preliminary analysis we examined sets of 20 diagrams generated by EDGEWORTH based on various prompt programs. Each program was mutated 1-3 times and the correctness of each diagram was determined manually. We found that while EDGEWORTH easily generates a variety of correct and incorrect diagrams, careful selection of configuration parameters was often required to get more “interesting” diagrams (correct special cases, distractors).

4 Future Work

There are a couple of avenues for future work: AST search analysis to enable EDGEWORTH to infer a diagram’s correctness, and using mutation subtrees from prior problems as templates to apply to new problems. While our approach is effective when generating dozens of related diagrams, it is not optimal if a problem author has only a few specific cases in mind. Intelligent selection of mutations may improve EDGEWORTH’s effectiveness in the latter case.

References

- [1] Edward B. Burger, David J. Chard, Earlene J. Hall, Paul A. Kennedy, Steven J. Leinwand, Freddie L. Renfro, Dale G. Seymour, and Bert K. Wattis. 2007. *Holt geometry*. Holt, Rinehart and Winston.
- [2] Ali Ghanbari, Samuel Benton, and Lingming Zhang. 2019. Practical Program Repair via Bytecode Mutation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) (*ISSTA 2019*). 12 pages. <https://doi.org/10.1145/3293882.3330559>
- [3] DF Halpern, A Graesser, and M Hakel. 2007. 25 Learning principles to guide pedagogy and the design of learning environments. In *Association of Psychological Science Taskforce on Lifelong Learning at Work and at Home*.
- [4] Yu-Seung Ma, Jeff Offutt, and Yong-Rae Kwon. 2006. MuJava: a mutation system for Java. In *Proceedings of the 28th international conference on Software engineering*. <https://doi.org/10.1145/1134285.1134425>
- [5] Dor Ma'ayan, Wode Ni, Katherine Ye, Chinmay Kulkarni, and Joshua Sunshine. 2020. How Domain Experts Create Conceptual Diagrams and Implications for Tool Design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI)*. <https://doi.org/10.1145/3313831.3376253>
- [6] Matias Martinez and Martin Monperrus. 2016. ASTOR: A Program Repair Library for Java (Demo). In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (Saarbrücken, Germany) (*ISSTA 2016*). 4 pages. <https://doi.org/10.1145/2931037.2948705>
- [7] Shabnam Mirshokraie, Ali Mesbah, and Karthik Pattabiraman. 2015. Guided Mutation Testing for JavaScript Web Applications. *IEEE Transactions on Software Engineering* 41, 5 (2015). <https://doi.org/10.1109/TSE.2014.2371458>
- [8] David Schuler and Andreas Zeller. 2009. Javalanche: Efficient Mutation Testing for Java. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering* (Amsterdam, The Netherlands) (*ESEC/FSE '09*). 2 pages. <https://doi.org/10.1145/1595696.1595750>
- [9] Allison Sullivan, Kaiyuan Wang, Razieh Nokhbeh Zaeem, and Sarfraz Khurshid. 2017. Automated Test Generation and Mutation Testing for Alloy. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. <https://doi.org/10.1109/ICST.2017.31>
- [10] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: from mathematical notation to beautiful diagrams. *ACM Transactions on Graphics* 39, 4 (July 2020). <https://doi.org/10.1145/3386569.3392375>