EXTENDED ABSTRACT: Toward Smart Mentor Dispatch: Can we predict when children need help to overcome errors or other roadblocks?

Chinmay Sheth
McMaster University, Hamilton, Ontario
shethc@mcmaster.ca

Vaitheeka Nallasamy
McMaster University, Hamilton, Ontario
nallasav@mcmaster.ca

Kruthiga Karunakaran
McMaster University, Hamilton, Ontario
kk@mcmaster.ca

Christopher Kumar Anand
McMaster University, Hamilton, Ontario
anandc@mcmaster.ca

1 Introduction

McMaster Start Coding (`http://outreach.mcmaster.ca`) has introduced 25,000 children in Grades 4 to 8 to functional programming in Elm over the last 5 years using socially constructive learning. We have both a WebIDE and an iPad app, ElmJr, for teaching. The iPad app is preferred for teaching younger children, and only allows for the creation of a limited subset of the Elm language. The WebIDE allows children to ask for help in a chat pane and mentors can edit and compile a copy of the children's code to understand the problem and double-check the proposed solution. It has been the preferred interface since the beginning of the pandemic, since all teaching has been virtual, and it allows many problems to be resolved without engaging screen sharing, which does not work well for large classes. But children do not always ask for help when they need it, and the perception of classroom teachers and mentors is that not being able to see frustration on their faces is a big disadvantage of virtual education. This raises the question: Could we identify children who need help based on their interaction with the WebIDE?

In order to be able to give mentors access to modules with pending help requests, all versions of all modules are left on the file system of the server. Each module is stored using the childrens' randomly-assigned IDs, timestamp, and slot number and type. We extracted 254,708 compilation attempts from 5330 users, from 2019 until January 2022.

We also wanted to think about the place an improved tool could have in our outreach efforts, which can be summarized by the following research questions:

RQ1 Can we identify children who need help based on the modules they attempt compile?

RQ2 Can we use the success or not of the final attempt as a marker of needing help, and predict this success from previous compiles?

RQ3 Can we identify a cluster corresponding to the 150 undergrads from their compilation history?

RQ4 Can we predict the slot type (picture, animation, etc.) based on the compilation history?

RQ5 Can we predict error types based on slot type, so that instructors can tailor examples to address the most common errors children have not previously seen in instruction?

Submitted to:
TFPIE 2022

RQ5 Can we create candidate metrics to predict classes with particular challenges (e.g., math below grade level, lack of engagement) which could be tested in future prospective studies?

The remaining sections will explain our *methods* for this design iteration, including app design, curriculum design and evaluation through observation and challenges; *results* of the challenges and analysis of the children's state diagrams; *discussion* of the results leading to proposed *future work*.

2 Background

2.1 McMaster Start Coding Program

This McMaster University Outreach Program has been operating for the past decade. A mainly volunteer group of undergraduate and graduate students develop lesson plans and deliver free workshops to schools, public libraries, and community centres in the Hamilton, Ontario, Canada area [**?**]. During the COVID-19 pandemic, the program has shifted online and has taught a record number of students. Since 2016, we have taught over 22,000 students in nearly 1,000 classrooms. The goal of the program is to foster interest and ability in STEM subjects through coding, especially for those groups who are underrepresented in STEM subjects, such as girls and underprivileged youth.

To support these workshops, we have developed several tools, including:

1. An open-source Elm graphics library, GraphicSVG.
2. An online mentorship and Elm compilation system incorporating massive collaborative programming tasks, including the Wordathon[1] and comic book storytelling[2].
3. A curriculum for introducing graphics programming designed to prepare children for algebra.
4. A type- and syntax-error-free projectional iPad Elm editor, ElmJr.

2.2 WebIDE

In our "production" IDE, we deliberately do not store any identifiable information, so this dataset includes children who programmed during one class visit, a series of class visits, one or more summer camps, training sessions for potential undergraduate mentors and undergraduate students taking "Introduction to Software Design using Web Programming" (a first-year course for Computer Science students).

Using a video-game metaphor, children have access to different coding environments, originally with 10 slots available for different modules. Different slots hide some or all components in a working program. For example in Picture slots, the main function is hidden and no interaction is possible, instead children must define a myShapes top-level definition which must be a list of shapes. In Animation slots, this definition is a function with an input which is a record with a single field, the current time in seconds since the beginning of the unix epoch. This allows us to infer a lot about the aims and/or experience of the users from the slot type they are using.

In our "preproduction" IDE, slots are replaced by activities, with named modules, publication, importing and forking, with rules on visibility and moderation to be determined. We are also using emails as credential for undergraduates, making some users identifiable. So, in the future, we will have more information about the provenance of code, a finer-grained set of "activities" to replace the slot system,

[1] http://outreach.mcmaster.ca/#wordathon2019
[2] http://outreach.mcmaster.ca/#comics2019

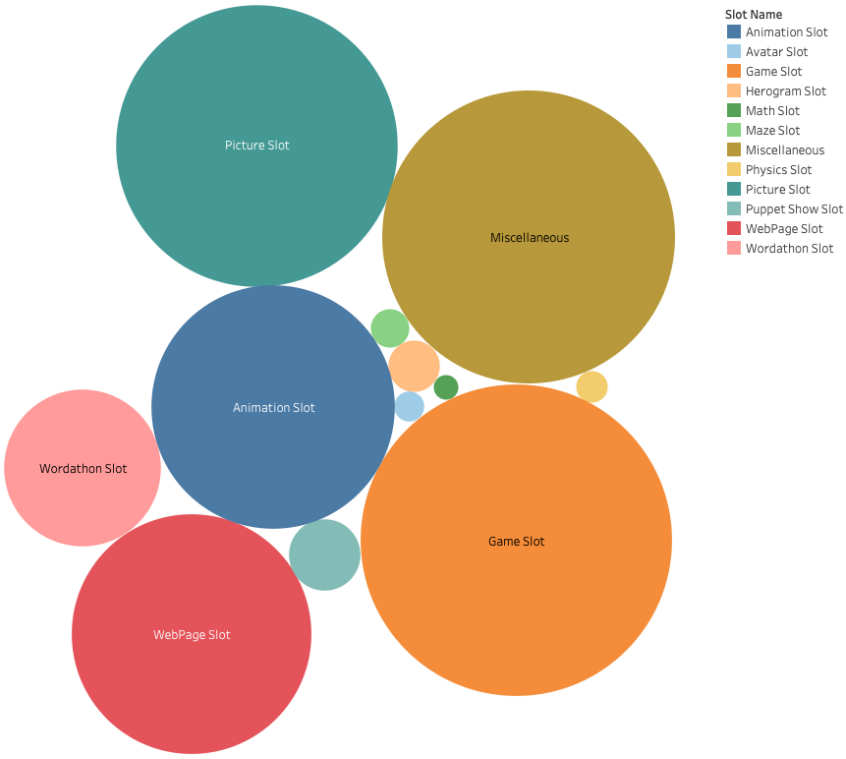

Figure 1: Slot names in a bubble chart; the size is relative to the number of errors in each slot.

and the ability to calculate and use real-time analytics. Undergraduate will also be able to do all of their work in the system, and easier sharing of code will incent them to do so.

3 Dataset

Figure 1 shows the partition of the data into different slots. Most of the data is associated with the basic slot types, (Picture, Animation and Game slots), with a significant number of compilations of WebPage slots which is unfortunately an aggregate of several different activities, including an html resume template, slots for creating word games (with access to hidden modules with word illustrations from the Wordathon slot), and slots for iteratively tracing Bezier curves or polygons, and finally slots to create word illustrations with a transparent outline indicating the viewport to which they will be clipped in the word games.

To try to answer RQ5, we explored error statistics using different visualizations in Tableau, and discovered many patterns. Figure 2 shows the numbers of errors for the most common slots, and most common errors, and in fact, there is large difference, so the first error to alert children to when they start in the Picture slot is UNFINISHED LIST. It probably makes sense for instructors to also discuss UNMATCHED COMMA which is most common in Picture slots, since they both involve lists, and Picture slots are about lists.

TYPE MISMATCH is most common in Game slots. This is partly because interaction requires the introduction to the model-view-update pattern in which types need to match. This is also the first time that records are used in our teaching. These introduce myriad type errors.

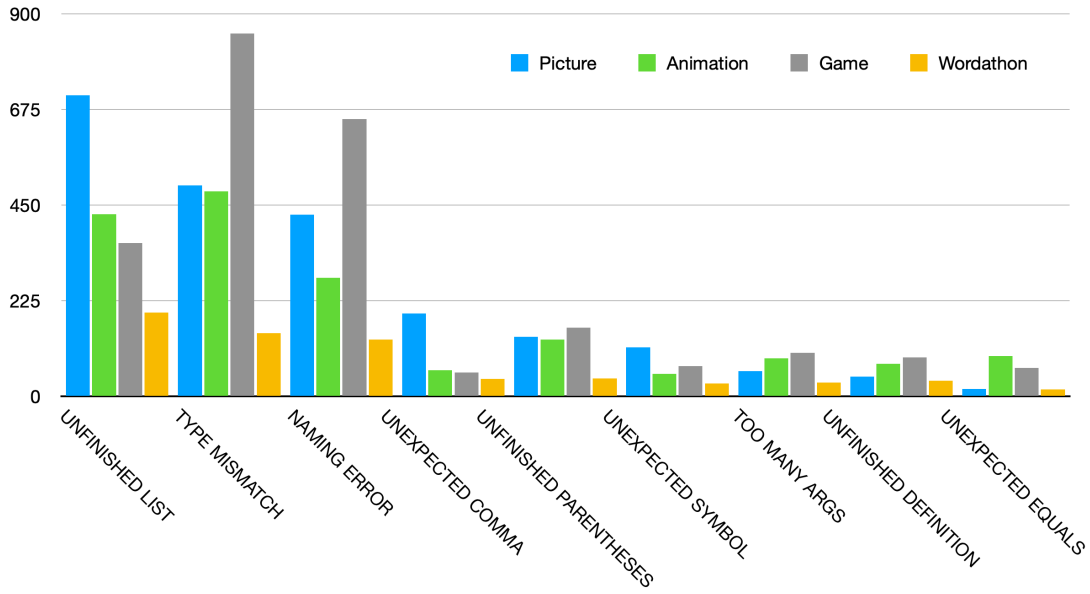

Figure 2: Most common error types for most frequently used slots.

Unfortunately, after covering these errors, the remaining errors are almost equally likely.

4 Predicting the Time to Resolve an Error

Our first attempt to predict when students might need help was to model the time between an error occurring and the error getting resolved. Figure 3 shows that the time to resolve a compiler error approximates a power law distribution. Even though most of the errors in the distribution took less than a minute to resolve, there are many errors which took considerable amount of time to resolve. Knowing this distribution, we can predict the time it would take a student to resolve an error and the chance they will resolve it on their own in a given amount of time. This distribution was determined from pre-2019 compilation data, but we believe the pattern will hold in the present dataset. If we assume that a power law is always a good approximation for time to correct an error, we can re-estimate model parameters for subsets of users or subsets of sessions, which leads to the next question: which subsets to use?

5 Clustering Approaches

We tried a number of clustering approaches, but only dimensional reduction by t-distributed stochastic neighbor embedidng (t-SNE) followed by KMeans clustering produced good results.

The data was preprocessed by computing the average for the number of functions, number of types, and number of comments for each user in the first Animation slot. Furthermore, each error type was pivoted and averaged in order to develop a sparse matrix of features for each user. Each column (i.e., feature) was affine transformed so that each column would have zero mean and unit variance, and then limited to the range $[0, 1]$ using using min-max normalization.

The t-SNE algorithm was used to non-linearly reduce the dataset to three components, followed by the KMeans clustering algorithm to identify clusters in the t-SNE features. Since t-SNE is a non-convex

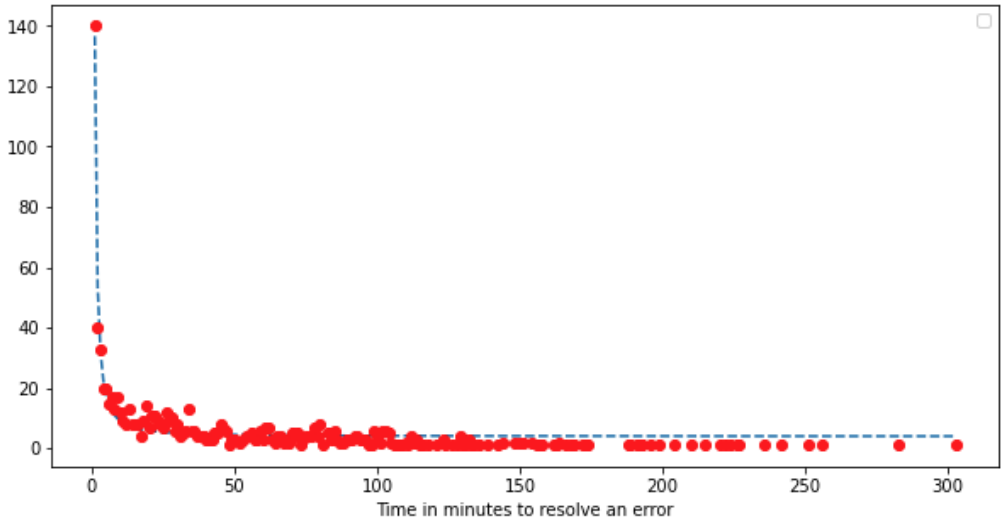

Figure 3: Fitting the data in Power Law Distribution

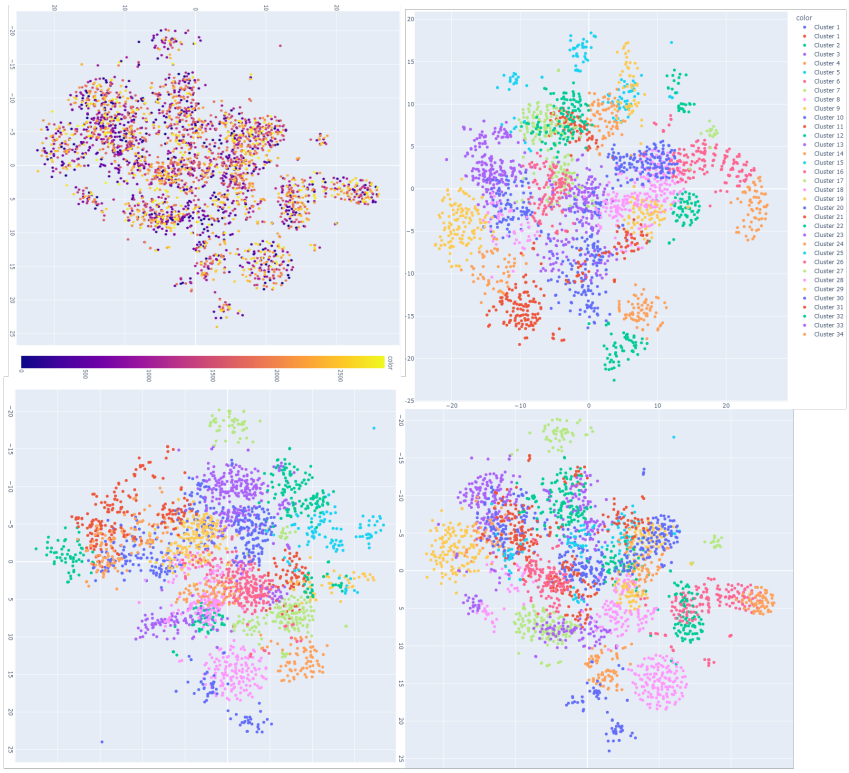

Figure 4: Compilation data reduced using t-SNE to three components, displayed pairwise (i.e., showing projections along each axis) coloured by day of last compilation (top left) and by KMeans cluster number (otherwise).

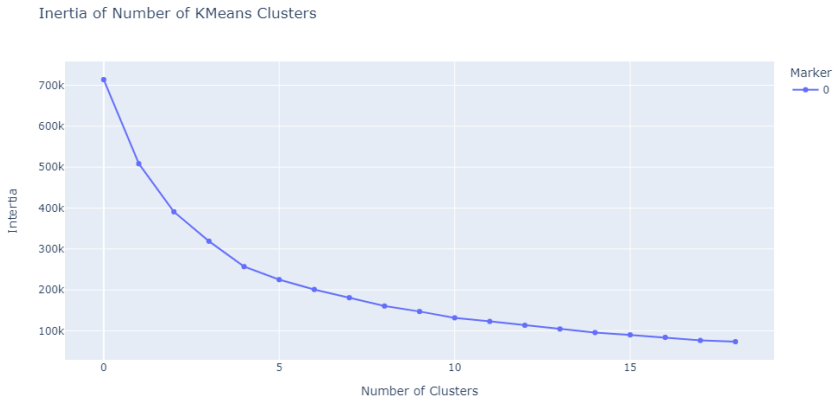

Figure 5: KMeans Inertia does not provide strong guidance as to the number of clusters in the data, because there is no "elbow".

optimization function, different initial parameters may lead to different results. We observed a lot of variability with our data. Perplexity of 100 gave good results, and KMeans Inertia (Figure 5) did not suggest a number of clusters, so we tried a "large" number, 34. Figure 4 shows the results. The top left subfigure is coloured by day of final compilation, and shows that the clusters cannot be explained by classroom, by school or by mentor, since we would see clusters with the same colouring, and there is no discernable pattern to the colouration. The remaining projections show that there are many easily identified clusters. Our hope is that we can assign meaning to these clusters. Our first approach will be to add additional information, such as time to resolve errors, information from other slots, from the total number of slots they use and the pattern of use of those slots over time. Our second approach will be to look at the code, for some of the larger clusters to try to find a pattern.

The features we have found implicitly encode patterns in beginners' functional programs. Graphical programs in Elm using our library are very different from other beginner programs because the library functions are composable resulting in very few parameters to memorize. It would be easy to isolate these compositions (using `|>` and `<|`) and add summary information to our features. It would also be easy to measure the depth of nesting of parentheses. It would be much harder to mine features of the abstract syntax tree, but this is something we would eventually like to do.

We are currently thinking about features unique to functional programs or unique to our graphics library which would make good features. We hope that discussion at the conference would help us uncover those.

Acknowledgements

We acknowledge financial support from the Faculty of Engineering. We also appreciate input from teachers, parents, and all the enthusiasm and inspiration from all the future coders we visit.