Work in Progress: Identifying and Analyzing Original Projects in an Open-Ended Blocks Programming Environment

Franklyn Turbak, Eni Mustafaraj, and Maja Svanberg
Computer Science Department, Wellesley College
Wellesley, Massachusetts, USA
Email: {fturbak,emustafa,msvanber}@wellesley.edu

Michael Dawson
Independent Researcher
Waltham MA, 02451, USA
Email: mijoda@gmail.com

***Abstract*—Tens of millions of people have used online blocks programming environments like App Inventor to learn how to program and build personally meaningful programs and apps. We want to improve blocks programming environments and pedagogies by using learning analytics to identify common problems and then address them. For most users, there is no information about which projects are *original* (built from scratch by individuals or groups based on their own ideas and current programming skills) vs. *unoriginal* (based on tutorials, class exercises, etc.). To understand what App Inventor users are learning and what misconceptions they have, we need to filter out unoriginal projects and focus on original ones.**

Here we describe two key aspects of our work in progress towards this goal. First, we have developed feature-vector representations of App Inventor projects that formalize a notion of structural similarity between them. This representation facilitates filtering out unoriginal work like tutorials and can be used within a group of learners to distinguish classroom activities from original projects. Second, we have developed a graph clustering technique based on project creation timestamps to discover groups of App Inventor users that appear to be taking a course together — essential information for distinguishing original vs. unoriginal work that is not explicitly represented in our datasets.

I. Introduction

In blocks programming environments, programs are assembled out of fragments shaped like jigsaw puzzle pieces. Because they lower barriers to programming [1], these environments, which include App Inventor, Scratch, Snap!, Blockly, Pencil Code, and Alice, have become popular ways for beginners to learn programming concepts and for casual programmers like scientists and hobbyists to write programs.

For example, MIT App Inventor democratizes mobile app creation by empowering those without previous programming or app-building experience to build their own apps [2]. In App Inventor, an Android mobile app can be created in two stages in an online browser-based visual programming environment. First, the user interface components (e.g., buttons, labels, text boxes, images, canvases) and functional components (e.g., camera, sound recorder, GPS location sensor, speech-to-text converter, speech recognizer) of the app can be configured using a drag-and-drop editor. Second, the behavior of the app is specified by connecting visual blocks that correspond to abstract syntax tree nodes in a traditional programming language. Some blocks represent events, conditions, or actions for a particular app component (e.g., the button has been pressed, take a picture with the camera) while others represent standard programming concepts (variable getters and setters, conditionals, loops, procedures, lists, etc.) Nearly 5 million registered App Inventor users have created over 19 million apps, and there are over 350 thousand active monthly users.

Since 2009, the first two authors have used App Inventor in numerous introductory courses, faculty workshops, and other activities. In our experience, App Inventor does lower barriers to making mobile apps. But users often have trouble making their apps work as desired. Some problems are rooted in computational thinking bugs. For example, the state variables of a loop are often improperly initialized in a way that allows it to behave correctly the first time it is run but not on subsequent runs. Working aspects of an app can be implemented in overly complex and roundabout ways. App Inventor programmers often make multiple copies of existing blocks and screens in situations where abstraction mechanisms like procedures and screen templates filled by data would avoid such duplication.

Our long-term goal is to use learning analytics to identify difficulties encountered by blocks programmers and to alleviate these difficulties by improving the programming environments and their associated pedagogies. Towards this goal, we are currently analyzing two datasets of App Inventor users collected in the 27 months between Dec. 2013 and Feb. 2016: all projects of 10 thousand randomly chosen users, and all projects of the 46,320 so-called prolific users, who have created 20 or more projects.

The open-ended nature of App Inventor and lack of information about its users presents many challenges for our research. App Inventor collects no demographic data on users other than what is provided in an optional survey completed by only a small percentage of users. For most users, we have no information on their gender, age, geographic location, programming background, etc. App Inventor accounts and projects normally have an email address, but these have been

DOI reference number: 10.18293/DMSVLSS2017-021

removed from our two large datasets as part of deidentifying them. Importantly, there is no information associated with users or projects that explicitly indicates whether a user took an App Inventor course or whether a project was created as part of a course or other coordinated activity.

In order to understand conceptual difficulties with App Inventor, we want to distinguish *original projects*, in which users create a project from scratch or significantly enhance an existing project based on their own ideas and current programming skills, from *unoriginal projects*, in which users create a project by following the steps of an online tutorial or a guided classroom exercise. We make this distinction for two reasons: (1) filtering out the unoriginal projects of users lets us focus on their skill progression and the misconceptions they have when building original projects; and (2) we can see whether constrained activities like tutorials and classroom exercises involving a particular concept help users with that concept in subsequent open-ended activities.

II. Feature Vectors for App Inventor Projects

When applying a learning analytics lens to how users learn and use App Inventor, it is helpful to formalize a notion of structural similarity between their mobile app projects. This notion facilitates filtering out unoriginal work like tutorials when analyzing projects for computational thinking and promises to be more effective than attempts (e.g., in [3]) based on project names.

One way to determine structural similarity between two App Inventor programs is to focus on their abstract syntax trees, and measure (1) which nodes appear in both trees and (2) which parent-child relationships appear in both trees. This is the basis of the *particle analysis* method developed by Sherman for determining how far away a student's project is from a known desired solution [4]. However, comparing parent-child relationships between two trees is expensive, leading to structural comparisons that are likely to be too slow for analyzing datasets involving millions of programs.

Instead, we represent App Inventor projects as feature vectors, where features include the types of components and blocks used in the program. (App Inventor has dozens of components and over a thousand types of blocks, though a typical program uses only a small subset of these.) Using feature vectors has the advantage that we can leverage standard Python data analysis libraries to determine similarity between projects by calculating distances between vectors. We are still experimenting with various dimensions of this feature vector representation to find the one that best suits our needs. For example: should the features include both component and block types or just block types (since the blocks themselves sometimes contain component information)? Do we simply care whether a feature is present or not, or do we want frequency counts for each feature? Should we give less weight to more common features (known as *term frequency–inverse document frequency (TF-IDF)* in the information retrieval literature)? What is the best way to measure distances between feature vectors in n-dimensional space? So far our experiments indicate that the generalized Jaccard metric (which divides the intersection of feature frequencies by their union) is better than the Euclidean and Manhattan distance metrics.

We have used the feature vector representation of App Inventor projects as the basis for hierarchically clustering the 902 projects created by 16 students in an App Inventor CS0 course at our institution [5]. These clusters provide a way to automatically distinguish original from unoriginal projects in a class setting. When projects done by many students are clustered closely together, we consider these projects to be unoriginal classroom activities. Projects dissimilar to other projects are considered original, as are projects of a single student or pairs of students that are clustered closely together (because they are likely to be different versions of original individual or pair projects). The automatic original vs. unoriginal categorization of projects by our algorithm closely matched the manual labelings we had given them.

III. Discovering Courses of App Inventor Users

The hierarchical clustering technique described above for distinguishing unoriginal classroom activities from original projects requires knowing which users were taking a course together. But in our App Inventor user datasets, there is no explicit information about which users might be associated with a course. Nevertheless, projects do carry a timestamp indicating when they were created. We have developed a graph clustering technique for co-occurring temporal events that leverages these timestamps to discover groups of users who appear to be taking the same course.

The key assumption underlying our technique is that students in a course are physically co-located in a classroom, receiving instruction from an instructor, who often guides students in creating a project within a relatively short time interval. If two users create a project around the same time, this is evidence that they might be in the same course, but it is not conclusive since this can happen by chance. But because courses are typically taught over many weeks or months, there are many opportunities for classmates to create programs around the same time, and two users who share many project creation times are likely to be taking a course together.

In one test of our technique, we selected the subset of the prolific users who created their first project between Aug 15 and Sep 15, 2015. This Fall15 group contained 6012 users. To find co-occurring events, pairs of project timestamps and users were created and stored into a single list that was sorted by the temporal information. Then time windows of plus/minus five minutes were pivoted on every list item to find the co-occurring projects in the interval. This information was used to create graphs in which the nodes are users, edges indicate that two users created at least one co-occurrent project, and edge weights represent the frequency of co-occurrences. After experimenting with raw frequencies, we decided to use proportional frequencies that take into account the total number of projects created by a user. The Fall15 graph was large (2,005,796 edges) even though we left out users who were using the system but were not part of the

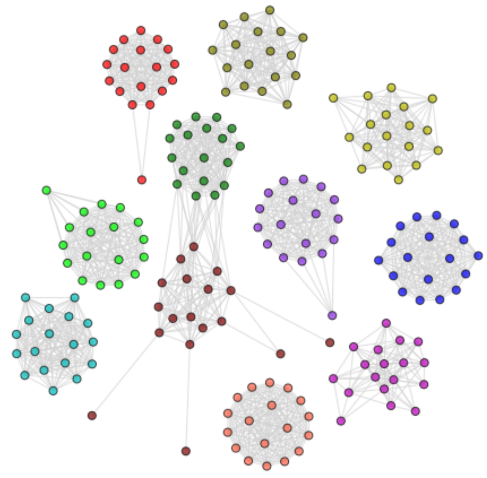

Fig. 1. Eleven clusters, all with 18 members, discovered for the Fall15 data. The cluster colored in red corresponds to the user accounts for all students in a course taught by the first author.

Fall15 group. After several experiments, we decided to filter out edges with five or less co-occurrences, given that students in our courses had 15 to 25 co-occurrences in a semester.

After the filtering process, we use the MCL graph clustering algorithm [6] on the Fall15 graph. This found 462 clusters, which varied in size from 1 to 84 nodes. Figure 1 depicts eleven of these clusters, all of size 18. We show these particular clusters, because one of them belongs to a course taught by the first author. This cluster (in red, top most-left) correctly contains all 16 registered students in the class, the one auditor sitting in on the class, and a previously unknown-to-us private account used by one of the students (the lonely node trailing the cluster).

IV. Current Status and Future Work

We are fine-tuning the choices for project feature vectors and similarity metric in the context of identifying projects in our datasets that appear to be created by following online tutorials. We plan to manually label several hundred projects by their tutorial status, and determine choices that maximize the correct identification. We will then identify tutorials in both our random and prolific datasets and compare some basic statistics between them. E.g., does one group have a higher percentage of tutorial projects than another? Are some tutorials more popular in one dataset than another?

We are also investigating extending our Fall15 course discovery experiment to our entire prolific dataset, and developing ways to verify that the clusters correspond to users taking a course together. A preliminary analysis shows that co-occurring projects within a cluster tend to occur at regular weekly times, bolstering the conclusion that they were created in a course held at these times. We also plan to measure the similarity of the co-occurring projects.

Once we have discovered courses for our prolific dataset, we will use hierarchical clustering to categorize projects that appear to be classroom activities. After filtering out tutorials and classroom exercises, we will be left with original projects that will form the basis of our learning analytics work. One form of unoriginality we do not know how to handle is determining whether any user projects are based on projects from the App Inventor Gallery, a collection of projects shared by members of the user community.

In our analysis of original projects, we plan to build upon the work of Xie and Abelson [7] to study the skill progression of users in their App Inventor projects. We also plan to investigate misconceptions and poor programming style, and see whether these improve over time or are affected by previously completed tutorials and classroom activities. In particular, we will study the usage of App Inventor's abstraction mechanisms (procedures, lists, loops, and generic components), which preliminary observations indicate are used surprisingly infrequently, even among prolific users.

As part of our work, we will develop algorithms to detect common bugs and lack of abstraction in App Inventor programs. We later plan to integrate such detection algorithms within the App Inventor environment itself to provide feedback directly to users about ways to improve their programs. Such detection algorithms could also be used within a teacher dashboard for App Inventor (such as the one sketched in [4]) to highlight students who need help with their code.

We have seen that prolific users may have several similar original projects that appear to be different versions of the same project. We suspect these projects are manual backups of a single project made by users who fear losing their work. We can use hierarchical clustering on a user's original projects to find such versions and see how common this "manual versioning" is in practice. Currently, no history is recorded for App Inventor projects indicating how they evolve over time (though Sherman has developed a fine-grained recording mechanism that may be integrated into App Inventor in the future [4]). So a sequence of versions could provide a useful coarse-grained window on the history of certain projects.

Acknowledgments

This work was supported by Wellesley College Faculty Grants and by the National Science Foundation under grant DUE-1226216.

References

[1] D. Bau, J. Gray, C. Kelleher, J. S. Sheldon, and F. Turbak, "Learnable programming: Blocks and beyond," *Communications of the ACM*, 2017, to appear.

[2] D. Wolber, H. Abelson, and M. Friedman, "Democratizing computing with App Inventor," *GetMobile: Mobile Computing and Communications*, vol. 18, no. 4, pp. 53–58, Jan. 2015.

[3] B. Xie, I. Shabir, and H. Abelson, "Measuring the usability and capability of App Inventor to create mobile applications," in *3rd International Workshop on Programming for Mobile and Touch*, 2015, pp. 1–8.

[4] M. Sherman, "Detecting student progress during program activities by analyzing edit operations on their blocks-based programs," Ph.D. dissertation, University of Massachusetts Lowell, Apr. 2017.

[5] E. Mustafaraj, F. Turbak, and M. Svanberg, "Identifying original projects in App Inventor," in *Proceedings of the 30th International FLAIRS Conference*, May 2017.

[6] S. van Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, University of Utrecht, May 2000.

[7] B. Xie and H. Abelson, "Skill progression in MIT App Inventor," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2016, pp. 213–217.