Programming Pathway Clustering Using Tree Edit Distance

Bo Jiang, Zhixuan Li
School of Educational Science and Technology
Zhejiang University of Technology
Hangzhou, China
bjiang@zjut.edu.cn

John Stamper
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA
jstamper@cs.cmu.edu

ABSTRACT

In programming task, problem solving procedure contains rich information about how students use conceptual and procedural knowledge to solve programming task. In this paper, we propose a sequence clustering method over the code snapshots to model students' programming path. For a given programming task, the difference between code snapshots to a perfect solution is first computed using tree edit distance. Then, this distance value is used to quantify each code snapshot and generate the distance sequence. Finally, we use dynamic time warping and agglomerative clustering algorithm to compute the sequence distance and cluster the sequences. The clustering results show that there are five different programming path clusters among students.

Keywords

block-based programming; programming pathway; sequence clustering; tree edit distance; dynamic time warping

1. INTRODUCTION

A programming task is genrally a multi-step task where students often start with an initial state and change their state by inserting or deleting code to reach the final state. The state transition sequence constitutes a programming path from initial state to the final state. Programming pathways contain rich information that can be used widely in pattern detection [3], hint generation [2, 4] and knowledge tracing [6] .

A fundamental problem with programming pathways is how to represent and quantify the trajectories, which has seen great interests from literature. Piech *et al.* [3] modeled students' programming process using Hidden Markov Model (HMM). Rivers and Koedinger [4] constructed an optimal programming path from a partial solution to a correct solution and used this path to generate hints. All intermediate states in the path are evaluated by a desirability metric that is weighted sum of their popularity, performance, closeness to a correct solution. Piech *et al.*[2] assumed the amount of time required for an average student to generate a partial solution is a Poisson process, the path from partial solution to a perfect solution is defined as the smallest expected time to be generated by the average successful student. In recent work, Wang *et al.* [6] used the represented learning technology to create program embedding, which was fed into a recurrent neural network model to perform prediction.

In this work, we build a simple yet efficient trajectory quantification method based on Tree Edit Distance (TED) and Dynamic Time Warping (DTW), and then apply a hierarchical clustering algorithm to find different programming path patterns. This study is inspired by the Possion path proposed by Piech *et al.*[2] , but differs in that we build the metrics directly over the raw abstract syntax tree (AST) and AST sequences, instead of programming time.

2. METHOD

Step 1 : Compute the distance from partial solution to the given final solution. To understand how students program and make progress step by step, we first need to distinguish students' different programming states (code snapshots). In this work, every student's code snapshots were captured and stored as AST format. When dealing with tree data, the evaluation of tree similarities is of great interest. A standard measure for the tree similarity, successfully used in numerous applications, is the TED. TED is defined as the minimum cost sequence of node edit operations that transform one tree into another [5]. In this work, we used a very popular TED metrics, Zhang Shasha tree edit distance [7] to compute the difference between the intermediate snapshots to the given perfect solution.

Step 2 : Compute the distance between different paths. The TED between any partial solution and the best solution demonstrates the minimum numbers of inserting and deleting that need to be applied to the AST of current solution to transform it into the AST of the best solution. In other words, this distance is a indicator how close each intermediate solution approaches the best solution, we call it **approaching index**. With approaching index, we can quantify each student's programming path as a sequence of approaching indexes from his/her initial solution to final solution. To find the trend of different programming path, a similarity metrics is needed to distinguish different programming path. Note that different students may have programming paths of different lengths because they may take the different number of attempts to reach the best solution. So, the key problem is to how to compute the distance between sequences with unequal length. We use a well-known technology Dynamic Time Warping (DTW) [1] to compute the similarity between different paths, generating the distance matrix of programming path.

Step 3: Sequence clustering using hierarchical clustering. After the similarity matrix has been established, we feed it into hierarchical clustering algorithm to find the path clusters. In the following experiment, we tested the hierarchical agglomerative clustering with four different cluster similarity measures, i.e., Ward, single,average and completed linkage, and found the Ward method provided the best clustering results.

3. DATASET AND EVALUATION

We implemented our idea on a block-based programming dataset that collected by *Hour of Code*. The dataset[1] covers three basic computational thinking concept concepts loops, if-else statements and nested statements. The dataset contains 79,553 unique code submissions, and 83,955 unique programming trajectories made by 263,569 students. For computational ease, we randomly sampled 20,000 programming trajectories from the dataset to conduct clustering. Fig. 1 shows the five programming path clusters found by the proposed method.

As illustrated in Fig. 1 , the students in Cluster 1 and Cluster 2 have a shorter path than the students in the other three clusters, and most of student in these two clusters failed to find the perfect solution. If we compare Cluster 1 and 2, we can see that the students in Cluster 2 have a poor initial solution because the average approaching index of Cluster 2 is around 125 that is evidently larger than any of the other clusters. This finding means that the students in Cluster 1 may have poor conceptual and procedural knowledge to convert the problem to the conceptual task, or they are not familiar with the programming language. The overall upward trend in Cluster 1 and 2 demonstrates most students in the two clusters neither iteratively improve their program nor find the perfect solution. In contrast, the average approaching indicators in Cluster 3 and 4 are declined smoothly, which demonstrates most students in these clusters refined there program step by step. Also, if we compare Cluster 3 and 4, we can find that students in Cluster 3 have a better start point and the average approaching indicator is around 30 that is very close to the perfect solution. The last cluster we found is Cluster 5, which has the longest programming path compared to other clusters. The average approaching indicator of start solution in Cluster 5 is around 50, which is similar to Cluster 2 and 4. Surprisingly, while students in Cluster 5 have a reasonably good initial state, their average approaching indexes are neither increased nor decreased during the programming process, which may mean they did not make a drastic revision to their program and most of them also could not complete the task perfectly.

4. CONCLUSION

We have shown how a programming progress can be transformed into a real-valued sequence based on TED and DTW. Then we demonstrated that the DTW distance matrix can be used in an agglomerative clustering algorithm to find the path pattern. We implemented this idea on a sampled dataset with 20,000 programming paths and found five different programming patterns. In the future work, we will improve the proposed method with other TEDs and sequence similarity metrics, and extend it to larger programming datasets. We also plan to feed the approaching indicator sequence into recurrent neural network to build a knowledge tracing model that is based on the programming paths within a single task.

[1]https://code.org/research

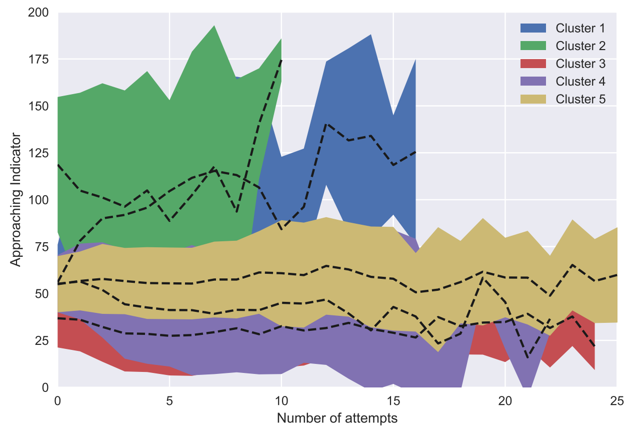

Figure 1: Programming path clusters. The black dashed line shows the change of average approaching indicator in each cluster

5. ACKNOWLEDGMENTS

This work is partly supported by the National Natural Science Foundation of China under Grant No. 61503340.

6. REFERENCES

[1] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[2] C. Piech, M. Sahami, J. Huang, and L. Guibas. Autonomously generating hints by inferring problem solving policies. In *Proceedings of the Second (2015) ACM Conference on Learning @ Scale - L@S '15*, pages 195–204, New York, New York, USA, 2015. ACM Press.

[3] C. Piech, M. Sahami, D. Koller, S. Cooper, and P. Blikstein. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160, 2012.

[4] K. Rivers and K. R. Koedinger. Automating Hint Generation with Solution Space Path Construction. In *Proceedings of International Conference on Intelligent Tutoring Systems*, pages 329–339, 2014.

[5] S. Schwarz, M. Pawlik, and N. Augsten. A new perspective on the tree edit distance. In *International Conference on Similarity Search and Applications*, pages 156–170. Springer, 2017.

[6] L. Wang, A. Sy, L. Liu, and C. Piech. Deep Knowledge Tracing On Programming Exercises. In *Proceedings of the 4th ACM Conference on Learning@ Scale*, pages 201–204, 2017.

[7] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.