Gameful Intelligent Tutoring
Project Proposal

COMS W3998 sec:028

Fall 2018

Robby Costales

Wei Han

Joe Huang

Yu Li

Yicheng Shen

Alina Ying

September 20, 2018

I - Abstract

The goal of this paper is to outline our plans for continuing development under the Gameful Intelligent Tutoring epic. More specifically, the features we are focusing on are Programming Behavior Detection 1.1 and Intelligent Hinting MVP. Within intelligent hinting, our main focus will be to implement an on-demand hinting system, and increase the complexity, variety, and customization of hints that students will receive. For behavior detection we plan on improving the functionality of the API and also integrate with hinting system to provide data for model training.

II - Introduction

“The Social Addictive Gameful Engineering (SAGE) research project extends the popular drag and drop based programming language, Scratch, which allows developers to create games, puzzles, animations and stories through an interactive interface. SAGE builds on top of Scratch by adding functionality that allows teachers to create puzzles and exercises to promote game based learning for 6-8 grade students. These features are aimed at motivating computational thinking in students at an early age which develops logical, analytical, creative, recursive, and algorithmic skills.” [2]

The current hinting system in Intelligent Tutor exclusively incorporates automatic hinting. According to a previous study by Razzaq and Heffernan [1], middle school students - which is our target user group - “working on algebra problems did significantly better with hints-on-demand and having control over when to see a hint compared to being shown a hint when they made an error…” This is intuitively explained by the reasoning that for students who are proactive at seeking help, the existence of

on-demand hints provides this option for these students, thus increasing the upside for mastery of knowledge. Hence, within the realm of the Intelligent Tutor, and given the current development stage of the hinting system, developing an on-demand hinting system has great significance.

Behavior detection is a fundamental step needed to power the hinting system. While students are currently being classified, the classification information is not accessible to the rest of the SAGE platform (only the behaviors are). We plan to implement the storage of the classifications so that the information is accessible to the rest of SAGE, and we then can use that information for outer loop recommendations, and potentially to improve the inner loop as well.

III - Related Works

1. **Existing Intelligent Hinting System**

   Intelligent hinting has gone through several changes in attempt to improve the intelligent automatic hint generation. In the recent project, a Hidden Markov Model (HMM) is implemented to model the learning path of students, and Poisson Path and Dijkstra's Algorithm are utilized to generate hints based on the result of HMM [5]. To train the model, mock data is generated assuming the students could be classified into the following 4 categories: stopper, mover, extreme mover, and tinker; each kind of students has his/her own behavioral characteristic, for example, extreme movers move fast without much consideration. In addition, the project also set up the methods in RESTful API for future integration with SAGE front-end.

2. **On-Demand Hinting**

Since it is oftentimes difficult for a tutoring system to determine the optimal timing to offer hints, on-demand hinting renders students more control in the learning process in comparison to automatic hinting. As noted by Burton and Brown, untiming interruptions hinder students' development of the cognitive skills to "detect and use their own errors" [3]. In a randomized controlled experiment conducted by Razzaq and Hefferman, it was demonstrated that hints-on-demand were significantly more helpful than proactive hints [1].

3. Multi-Layer Hinting

The developed intelligent tutoring system has a relatively simple and inflexible hinting policies, which offers only one prescribed problem-solving strategy. If the hint offered is too specific to the particular problem, the students might not understand the more general concepts underlying this problem; on the other hand, if the hint provided is too general, it might lead to students' frustration and consequently disengagement from learning because they are unable to solve the particular problem.

Therefore, to help improve students' cognitive ability and learning efficacy, Anohina in her paper has advised a multi-layer hinting system giving students hints from the most general to the most specific [8]. The upper layer broadly divides hints into three categories, whereas the lower layer hints are grouped under the three general categories of hints. As shown in the figure below.

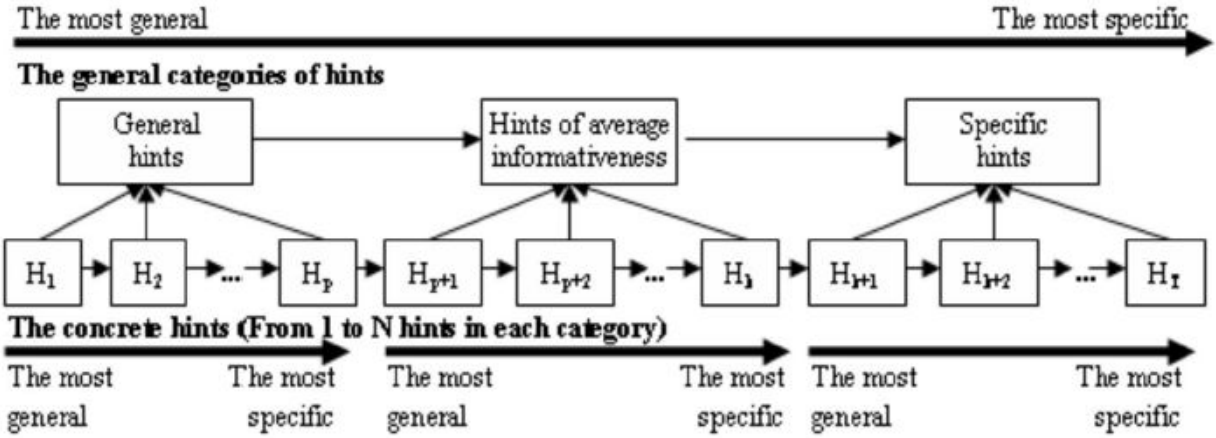


Figure 1: Multi-layer hinting mechanism

4. Behavior Detection

Last semester, an architecture has been implemented to connect the SAGE Frontend, Scratch Analyzer, SAGE Node, and mLab. Scratch Analyzer is able to extract student JSON files for each game from SAGE Node to analyze student logs, and decide whether it is advantageous to update the database.

Sambhav Anand and Allison Sawyer utilized behavior detection in their project on improving the hinting system. They used clustering algorithms to group students according to the general paths they took through an assignment. Instead of using the clusters to predict student outcomes, they used them to increase the relevance of hints generated for each cluster.

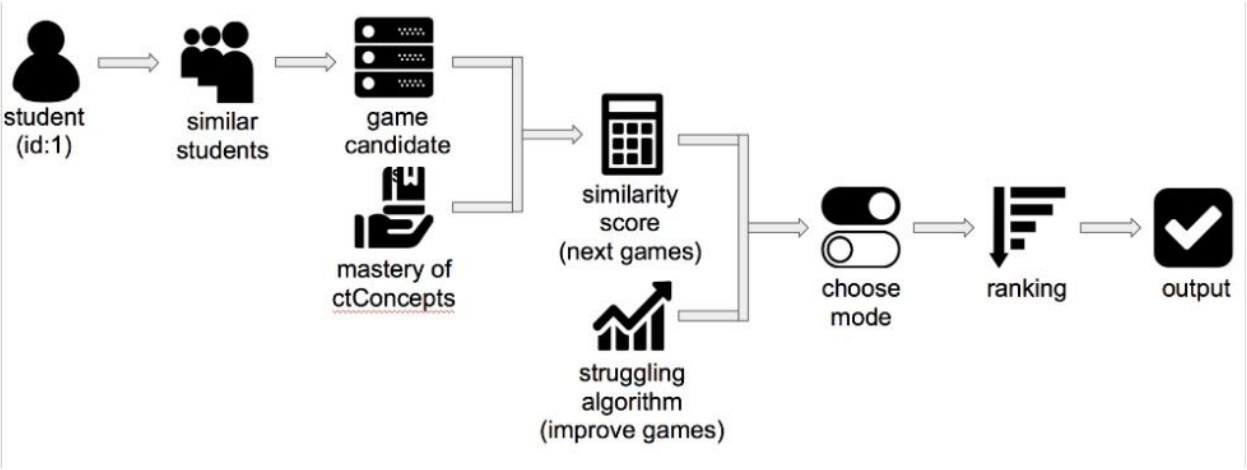
5. Bayesian Knowledge Tracing

To infer whether a student has mastered a skill from a sequence of assessment records, Anderson and Corbett proposed Bayesian Knowledge Tracing, a user modeling method extensively used in intelligent tutoring systems ever since [6]. BKT is a special case of the Hidden Markov Model which assumes that student knowledge is represented as a set of binary variables. The four types of model parameters [7] used are P_{init} (knowing the skill beforehand),

$P_{transit}$ (transitioning from unknown to known state after the opportunity to apply it), P_{slip} (making mistake when applying a known skill), P_{guess} (correctly applying a unknown skill).

6. Recommendation

In the previous semesters, a hybrid collaborative filtering algorithm, which combines item-based and user-based collaborative filtering [4], has been adopted to implement the game recommendation engine for students. The workflow of the recommendation engine is shown below.

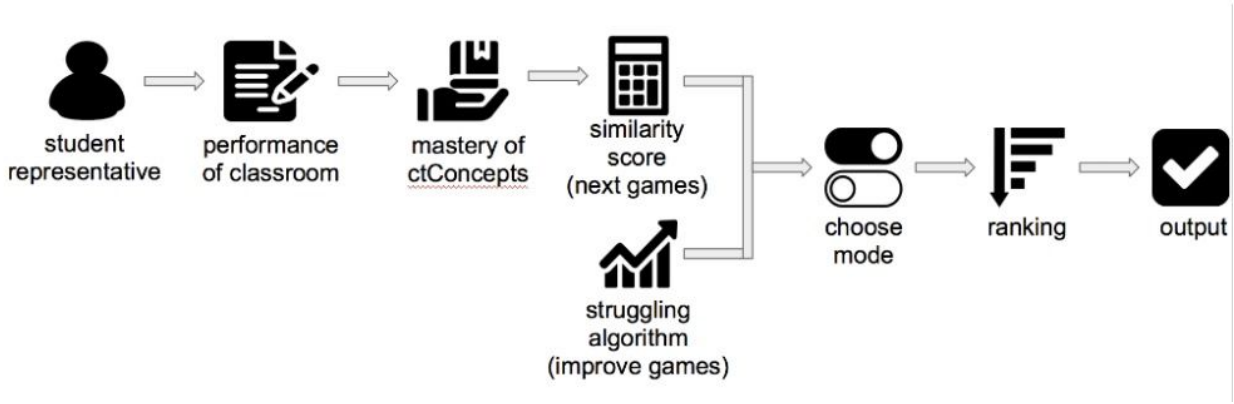


The similarity score between users is calculated based on the following formula to find the k-nearest neighbors, where a, b are users(students), p is the item(game), s_a, s_b are the ratings(scores) of the two users to the same game.

$$sim(a,b) = \frac{\sum_{p \in P}(s_{a,p} - \bar{s}_a)(s_{b,p} - \bar{s}_b)}{\sqrt{\sum_{p \in P}(s_{a,p} - \bar{s}_a)^2}\sqrt{\sum_{p \in P}(s_{b,p} - \bar{s}_b)^2}}$$

The similarity score between games is calculated based on the average of mastery level (the highest score the student has ever earned on the particular ctConcept) similarities of each ctConcept in the game.

The game recommendation engine for instructors is constructed in a similar manner.



IV - Proposal

1. **Hinting**
    a. As the previous intelligent hinting system is currently offline, we could integrate the project with Scratch Analyzer and provide hinting functionality to SAGE UI using the existing API.
    b. After the completion of integrating Intelligent Hinting 1.1 into SAGE UI, on-demand hinting can be presented to students via a click on the hint button, which becomes available when an appropriate hint is generated based on the inferred student classification via the existing Hidden Markov Model and Poison Path.
    c. After on-demand hinting has been implemented and persisted to mLab via SAGE Node, multi-layer hinting could be integrated into on-demand hinting by progressively providing more and more specific hints according to the types of errors and inferred student classification.
    d. The usage of on-demand hints provides more information than the fact that a particular student needs help with a specific project. We may be able to learn a student's help-seeking behavior [4], and provide more

accurate and timely proactive hints based on this behavior. In addition, we can study the pattern of on-demand hints. The variation in the number of on-demand hints across different projects from a single student, for example, could indicate the student's level of mastery on different topics.

e. We may also want to prevent students from abusing the on-demand hints by penalizing excessive hint usage or regulating the availability of hints.

2. Assessment

a. The previous project modeled the learning process by Poisson process, but the mock projects it generated lack variation as they were created by rearranging or omitting blocks from successful projects. We can improve the algorithm to generate more variation of paths to success or failure.

b. The project also mocked student data with strong correlation between student type and performance. Because all mock data highly depend on researchers' beliefs in how students learn, which can be very different from reality, they are less reliable in evaluating the hinting algorithm. We can better assess the hinting system if we implement a feature to collect snapshots from real student practices.

c. When do we capture snapshots? Researchers at Stanford learned how students solved programming assignments by taking snapshots whenever students compiled their code [9]. While this method helps capture a student's progress in a natural way, it also misses some important details along a student's path to solution. First of all, it relies heavily on the compile action (i.e. human behavior). Some students compile very frequently, and some may not compile at all until they are completely satisfied with their solutions. Secondly, even if two students give identical snapshots in two consecutive compiles, the difference between two compiles can be constructed in very different ways. Therefore, this method cannot guarantee a high "resolution" of paths to solution- smaller progress

can be lost, and they could potentially miss some important steps that help classify students. Fortunately, for Scratch programs, there are easier ways to identify “units” of work. For instance, in Parson’s Puzzle, moving a block or filling a black can be count as a “unit” of work. Ideally, we want to be able to capture all “units” of work, based on which we can determine students’ milestones. The smaller the “unit” of work, the more accurate we can calculate the dissimilarity of two students. However, the “unit” of work should be reasonable and not very noisy so that we can see a general pattern in student approaches to problems. We can tune the “resolution” of paths to solution based on the performance of the hinting algorithm and how good the clusters are.

d. Additionally, the snapshots of paths to solution convey valuable information about student behaviors. We know how frequently a student makes a move according to timestamps on snapshots, and we can infer whether a student is trying different things, going back and forth, getting stuck, etc.. Thus, we can learn what type a student is more accurately. An accurate classification is important because it affects how the hinting system generates hints. Of course, this also entails that we need to update the algorithm that clusters students by type. Finally, this information can potentially be used to decide student’s level of mastery and the type of hint and study students’ behavior from educators’ perspective.

3. Behavior Detection

a. It is important to record student actions like submission attempts, types of errors, the number of times the student demands hints, and time spent on each game to analyze a student’s mastery of a certain unit of games.

b. It would be very helpful for user behavior analysis if a trigger could be implemented such that important student events could be automatically recorded and saved to log files.
c. Bayesian knowledge tracing could be implemented by appropriate modification to the existing Hidden Markov Model.

V - References

[1] Razzaq, L. and Hefferman, N.: *"Hints: Is It Better to Give or Wait to Be Asked?"*

[2] Schaal, Y. and Lien, J *"Gameful Constructionism Project Proposal COMS 6901 E sec 028 Spring 2018"* https://github.com/cu-sage/Documents/blob/master/2018_2_Summer/Proposal_Parsons_Puzzles.pdf

[3] Burton, R.R., Brown, J.S.: "*An Investigation of Computer Coaching for Informal Learning Activities*". In: Sleeman, D.H., Brown, J.S. (eds.) Intelligent Tutoring Systems. Academic Press, New York (1982)

[4] Guanwen Yao and Lifeng Cai. *"User-based and Item-based Collaborative Filtering Recommendation Algorithms Design"*. University of California, San Diego.

[5] Ding, Y., Luo, W., Zhang, J. "*Intelligent Hinting 1.1 Final Report*" https://github.com/cu-sage/Documents/blob/master/2018_1_Spring/Final_SAGE_git_intelligent_Hinting_1.1.pdf

[6] Corbett, A. T. and Anderson, J. R.: *“Knowledge tracing: Modeling the acquisition of procedural knowledge”*. User Modeling and User-Adapted Interaction, 4(4), 253-278. (1995)

[7] Yudelson, Koedinger, Gordon. *“Individualized Bayesian Knowledge Tracing Models”*. Carnegie Mellon University.

[8] Anohina. *“The Problem-Solving Modes and a Two-Layer Model of Hints in the Intelligent Tutoring System for Minimax Algorithm”*.

[9] Piech C, et al.. 2012. Modeling How Students Learn to Program. *SIGCSE’12*: 153-160.