Assessing the effectiveness of an automated problem generator to develop course content rapidly and minimize student cheating

Dr. Philip Jackson, University of Florida

Dr. Philip B. Jackson earned B.S. degrees in Aerospace Engineering and Mechanical Engineering as well as an M.S. and Ph.D. in Mechanical Engineering, all from the University of Florida. He is currently a faculty member at the Institute for Excellence in Engineering Education at the University of Florida. There he specializes in implementing innovative methods of instruction in undergraduate courses on dynamics, heat transfer, and thermodynamics. His research interests include numerical heat transfer, fluids, and magnetohydrodynamic simulations and facilitating undergraduate students to engage in similar projects. He is also focused in the implementation of engineering freshman design experiences.

Ricker Lamphier, University of Florida

Assessing the effectiveness of an automated problem generator to develop course content rapidly and minimize student cheating

Abstract

The education environment has taken a dramatic shift in the last decade with a greater focus in online delivery. In online and traditional classes alike, engineering faculty rely on textbooks, online publishers' content, and problem repositories for class exercises as writing new problems every semester is prohibitive. The problem with this is that both students and faculty have access to the same information on the web. With digital solution manuals or solution websites, the traditional delivery of homework has lost its integrity as the challenge of homework is no longer figuring out how to do the problem, it is now where does one find the online solution to a specific problem.

This project documents the development, implementation, and testing of a true random problem generator in Python to create original content quickly whose solutions are not ubiquitous or accessible via some online solution manual. With true random problem generation, the inputs define the problem as an archetype (where archetype is defined as a specific problem type (e.g. projectile motion, ideal gas piston cylinder, etc.)) and can have virtually an infinite set of problems generated from it. The script will read a user defined input file, randomize parameters, randomize values for those parameters, test for logical correctness (fidelity), and finally use a natural language algorithm to generate both problem text and solution.

To assess the effectiveness of problems created in this way, a survey was conducted having students rate the generator's problems based on usefulness, clarity, and other qualitative features,

as well as compare them to traditional, hand-written problems. A separate survey was conducted to score the accuracy of the generator's difficulty heuristics by comparing the calculated difficulty of the software with the average of that perceived by the students. The assessment of the reduction of cheating was measured via blind survey and performance analysis. All-together, the functionality of this automated problem generator is established based on the qualitative measure of the aforementioned criteria, and overall perception of the potential future of online learning.

Introduction

The educational environment is changing now more than ever, the internet has created a new approach to instruction which causes one to question the best approach to education as a whole. The flipped model classroom, where students watch a recorded or digital lecture before coming to class and then do problems in class, has seen traction with respect to student performance and retention of material (Butt, 2014; Baker, 2000). With a more online focused learning model, students are also turning to online resources to assist them in their education (Rowe 2004). Resources like YouTube and Khan Academy provide students with tutorials and help with navigating specific problem types or just deepening a student's understanding of fundamental topics covered in their online classrooms.

Antithesis to all these great online resources, there is a side to the world of online learning which at abstraction is harmless and even beneficial but has grown to be nothing more than a large crutch, rather than aid, in a student's repertoire of online resources. Online test banks, solutions manuals, and answer forums, like Chegg and Course Hero are increasing in popularity (CHGG Stock) at the expense of students' educations (Rowe 2004). These resources provide students

with the answer to a problem alongside a usually detailed solution providing the evidence to the answer. Moreover, students tend to escape, learning how to do the problem by using these detailed solutions and applying them to similar problems. The problem is that students often do not view this is as cheating because they pay for this additional type of "instruction" similar to that of supplemental material. (Raines, 2011). In the instance of online education, students view the internet as their instructor rather than the professor providing the content online to the online classroom, as a result student believe that the entire internet is their notebook and a resource for them to complete the assignments rather than the materials provided to them. Some would argue this is being resourceful but in reality, students are not developing the skills needed to complete assignments rather they are relying on the skills of others to complete their assignments, in the same way as one would copy another student's work in the traditional thought.

The online classroom has so much potential for educating the masses. Online classrooms have the ability to handle more students at a time, but the resources are not yet developed fully for this experience. Where traditional classrooms are hindered by a limited number of seats or space in a particular location, the online environment allows for students to watch lectures and learn remotely, give an opportunity to have more students in a class. However, there comes the issue of how to manage assignments for a larger number of students. Resources like automatic grading through multiple choice graders are one option, or the use of online eBooks, like WileyPlus and Pearson Cengage give professors the opportunity to give students assignments that are automatically graded and deliver students similar "random" problems.

These “random” problems only tend to randomize numbers but still deliver the same problem which the solution can still be found on the aforementioned resources. Leaving professors and instructors alike to develop new unique problems every time a course is taught to prevent students from finding solutions to problems online. This task is strenuous and tedious as developing unique problems may be near impossible.

A solution to this could be having a software develop problems based on already written problems. While online teaching resources randomize the numbers, a way to develop new problems is just by randomizing the givens and finds of problems, for example, given a problem which asks the question one way, if one were to reverse the problem, taking the solutions as the inputs and the old givens are now the new finds produces a new permutation of the problem. Additional development in this area, aside from randomizing number, and givens and finds, would be to randomize the verbiage of the problem. Taking specific words to the problem and generating additional words that act in the same way but do not give a solution when searched on the internet. For example, changing the object of a problem from a ball to particle, to a disk, in a kinematics problem all these objects can be treated similarly. Furthermore, changing the action of the object can prevent access to solutions, for example, instead of throwing or tossing, an object could be launched in a kinematics problem.

Methods for furthering online education need to be implemented to continue to maintain the rigor and integrity of higher education. Especially in a world where it is easy to hide behind the screen and create a façade which lacks the honor of the face to face interaction of success or failure in the traditional classroom.

Algorithm and Methodology

An elusive goal in education is to curb student cheating and force students to complete homework problem sets without the ability to simply look solutions up on the internet. Using online sources like solutions manuals or chegg.com might be useful for students to work their way through problem solutions step-by-step. However, when solutions to exact homework problems are available there is no way to guarantee the student will use these sources as learning tools rather than sources to copy, cheat, or plagiarize. The only fool-proof method to prevent a student cheating (from online solution sources) is to generate new problems for each new semester's homework. This is prohibitively costly in time.

To generate a constantly changing set of homework problems an automated algorithm was designed to create fresh content by randomizing problem inputs, output, solutions, and diagrams (Jackson, 2018). This is a complicated, sophisticated process that uses multiple technologies. The problem must first be defined in such a way that all aspects of the problem may be randomized. Consider a common problem in undergraduate dynamics such as two-dimensional projectile motion or particle-to-particle impact. In such typical problems there are a finite number of variables that might be used to completely define the problem. Positions, velocities, accelerations, forces, etc. may all appear in homework problems as either given information or values to be found by calculation. A problem archetype is created and encoded in such a way that an automated algorithm may be able to create a set of random variables specific to the problem for both sets of given information and information to be found.

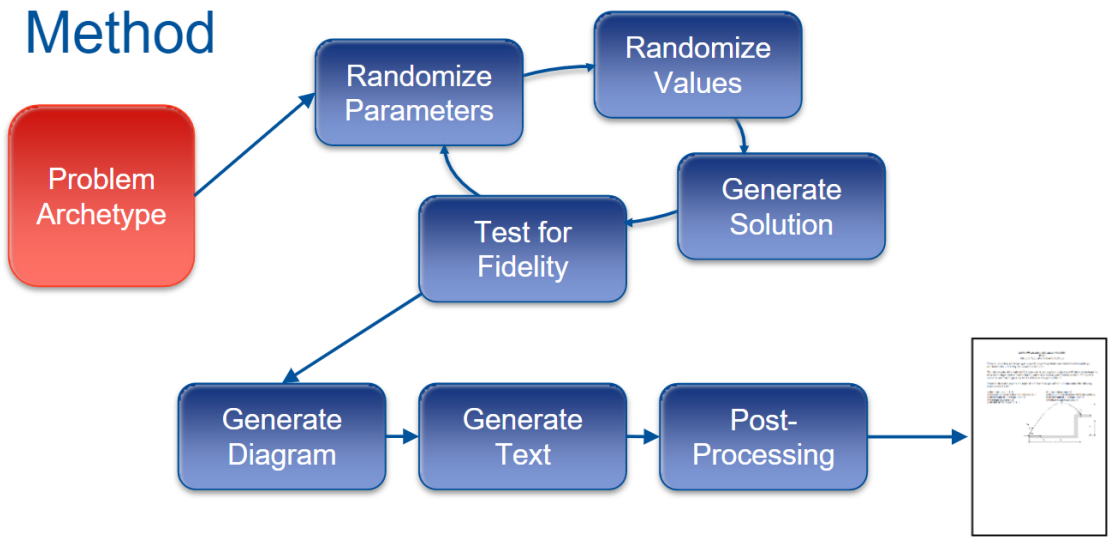

Figure 1: Flow diagram of randomized algorithm to generate fresh and correct problem inputs, outputs, solutions, diagrams, and text

Next, numerical values are chosen for each variable of the parameter set that is to comprise the given information in the problem. Some values are chosen from uniform random variables and some from gaussian distributed random variables. Each parameter has defined minimum and maximum values based on what might be reasonable for a problem. A test for fidelity or efficacy is then performed by solving the system. Using a computer algebra system and a set of the pertinent equations to relate all the archetype's variables, the algorithm iterates through all the unknown parameters and solves for all that may be found with the given set of inputs. A computer algebra system is used in this way, not simply to find such solutions but also to may out those solutions step-by-step in a human-readable format. The outcome of such a randomized algorithm is not guaranteed and must be tested for correctness. For example, a randomly generated set of problem inputs might all be linearly dependent and not lead to any solvable values. The problem may be over-constrained, under-constrained, or unreasonably simple. The algorithm iteratively either corrects for these problems by removing constraints and resolving,

adding constraints and resolving, or starting over until a problem of the desired difficulty is achieved.

Using program libraries capable of drawing images, Pygame in this case, the algorithm will then generate a close-to-scale picture of the problem scenario. Using natural language generation, through simpleNLG, a word problem can be generated that includes all the given variables and the variables to be found. In the ideal scenario, the algorithm should be able to generate images and text indistinguishable from what a human professor might produce. This has been prohibitively difficult to date, but work is still being done to make the images and text that are generated more sophisticated. At this point in the algorithm's development a significant amount of post-processing is needed to create a reasonable diagram and word problem.

Problem Archetypes

For the first phase of algorithm development a single problem archetype was used to test the algorithm's efficacy. In the current state of development three unique problem archetypes have now been produced. Each problem archetype is intended for use in an undergraduate dynamics course and are as follows:

1. Two-dimensional projectile motion of particles without drag
2. Two-dimensional oblique impact of circular particles
3. Two-dimensional central impact of a particle with a rigid body pinned at one endpoint

For the first archetype, two-dimensional projectile motion, two or more points in the trajectory are usually defined (typical the start and end points of the motion). More intermediate points may

be allowed as well, such as the moment of maximum height. For each point all possible variables of kinematics become possible for the algorithm to choose as inputs, such as velocity, position, angle of attack, or time. Simple equations of constant acceleration kinematics, and other expressions of trigonometry, comprise the list of equations that may be used to solve for the unknowns.

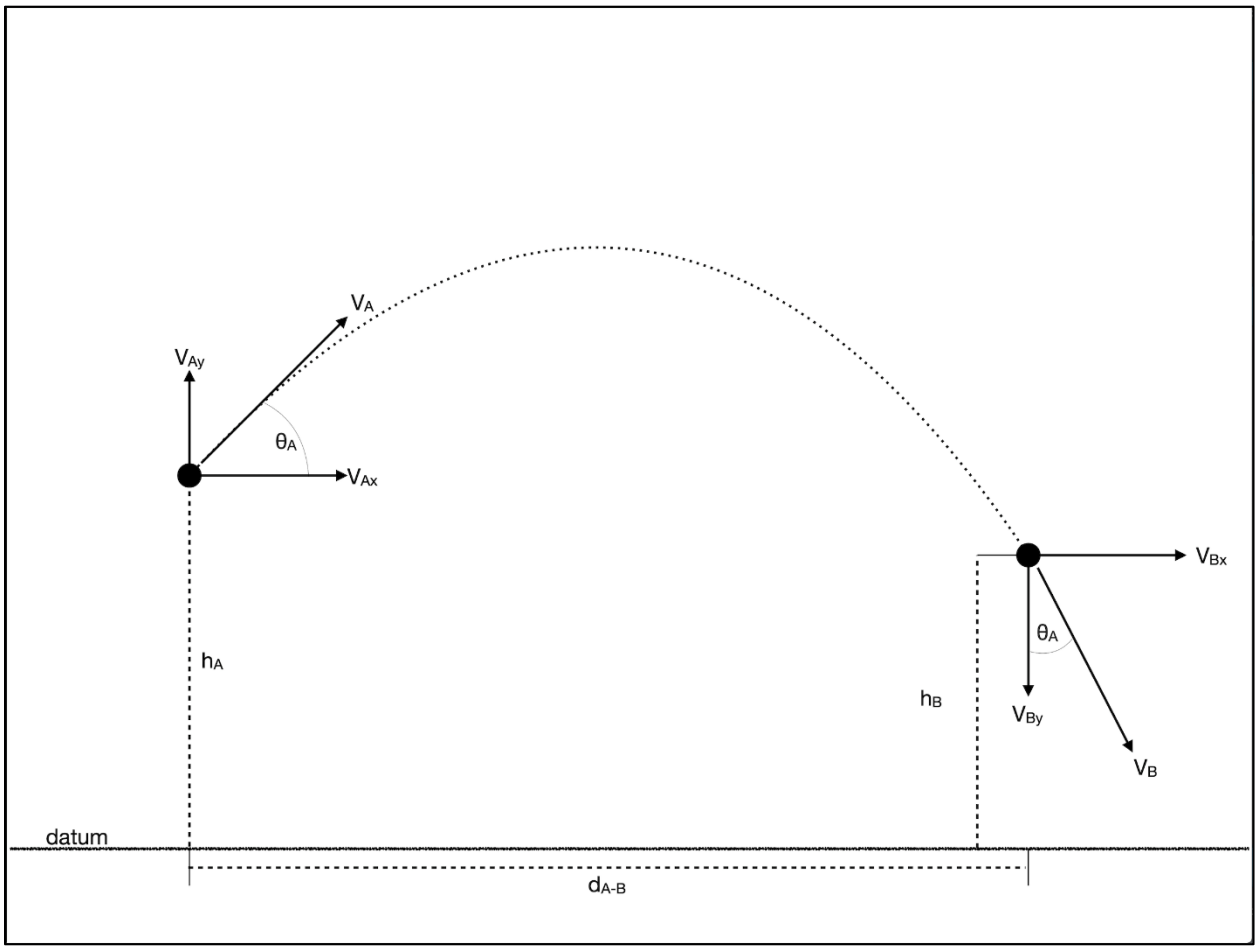

Figure 2: Problem archetype for generalized projectile motion of a particle in two-dimensions.

The second archetype developed is a simpler problem where two particles impact one another in an oblique fashion. The problem is intended to describe the phenomenon of impact only and in each randomly generated scenario the problem begins at the moment the particles touch and ends after the restitution phase of particle impact.

Initial velocities (both in scalar and vector representations), final velocities, impulses, directions of travel, coefficient of restitution, and masses are all included as variables in the problem archetype. The problem is solved through the typical equations of geometry and conservation of momentum.

Usually particle impact is defined by giving the two initial velocities and solving for the final velocities after the particles bounce off one another. Randomization of this archetype, however, found some interesting problem configurations. A single initial velocity and an x-component of a final velocity might be chosen as input variables. The number of possible permutations of problems that can be generated in this way are astronomical and require students to view simple problems from a multitude of perspectives.

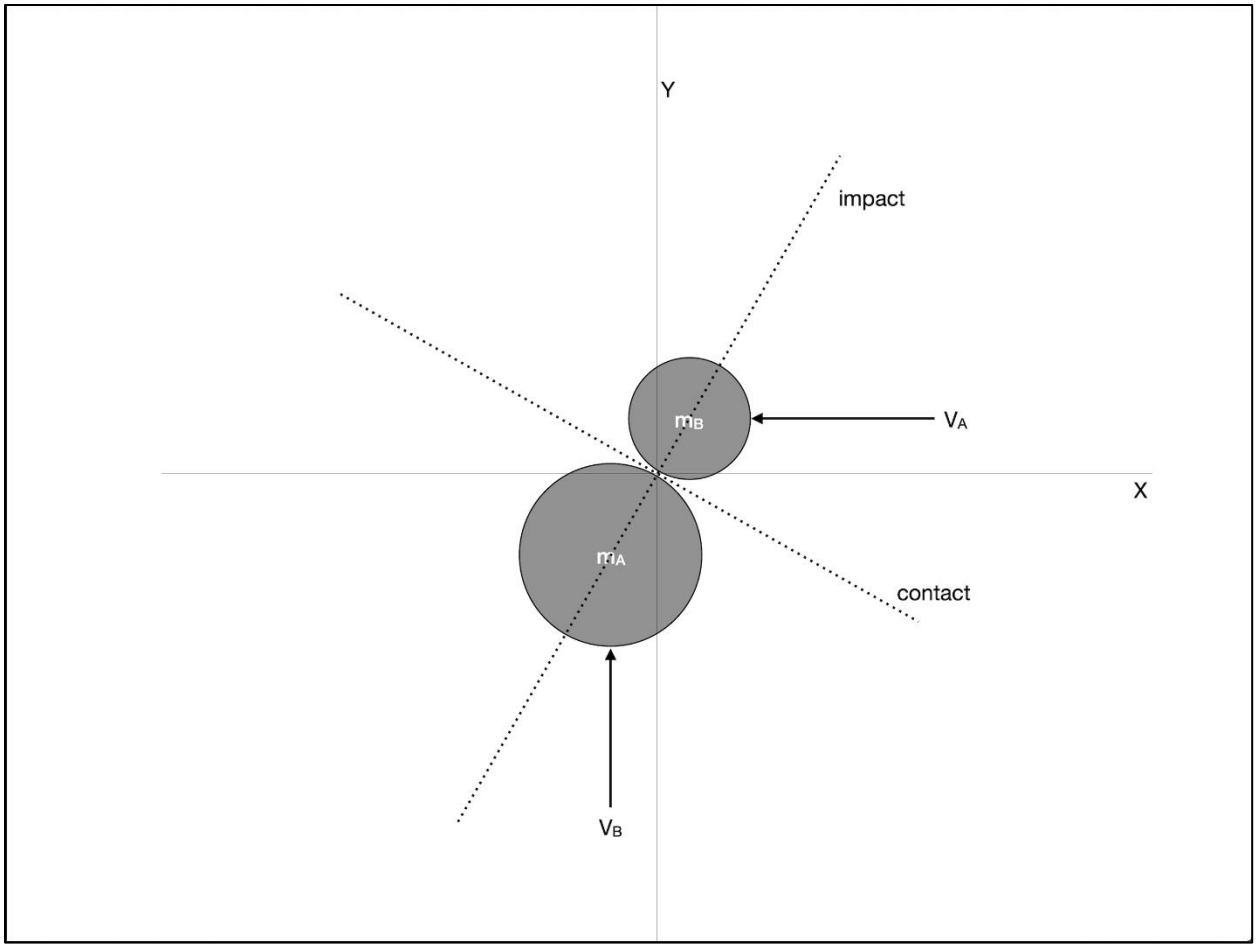

Figure 3: Problem archetype for two-dimensional oblique impact of two point masses

The final problem archetype that was developed describes the particle-rigid body impact that occurs if the body is a slender bar with one end simply pinned. Typical variables of velocity, mass, etc, similar to the previous archetype, are defined. In this case, the length of the bar, the moment of inertia, radius of gyration, position of the center of gravity, and position of impact now become randomizable quantities.

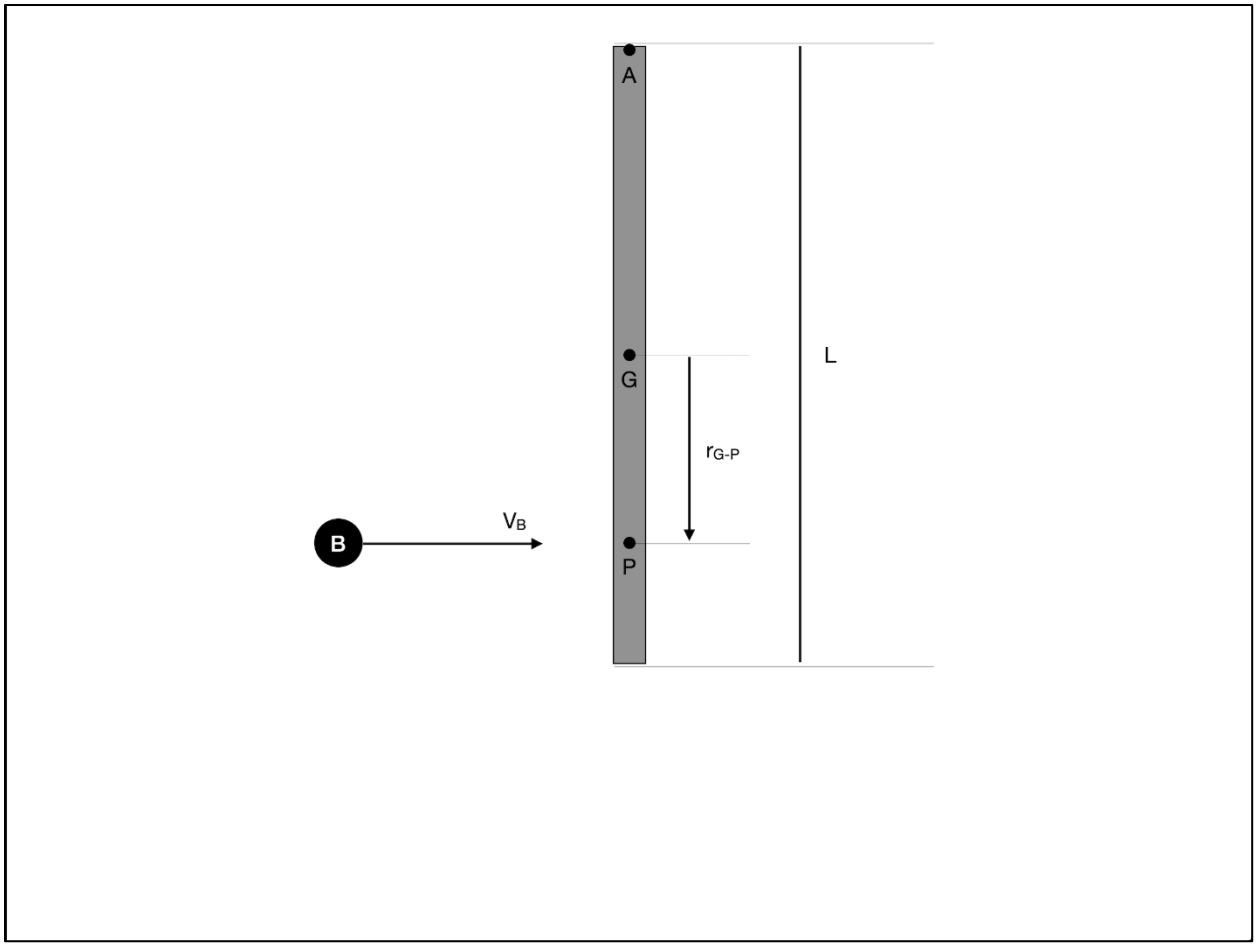

Figure 4: Problem archetype for generalized impact between a particle and a rigid body pinned at one end.

Determining Problem Difficulty and Collecting Student Data

Of chief importance in this study in developing problem sets and course content that are equitable to all students in terms of difficulty. Determining the difficulty of any randomized problem, then, becomes indispensable knowledge before problems may be dispensed to students. The algorithm uses a simple heuristic to gauge difficulty. Rather than ask the algorithm to generate a problem of a desired difficulty the algorithm instead continues to generate problems of random difficulty until one of the desired level is found. All problems, solutions, and diagrams are stored in a database to be used to improve the algorithm as development progresses. Once a

problem's input and output parameters are determined, the solution is generated, checked for correctness and only then is the difficulty of the result determined. Since generation of new problems occurs rapidly this was determined as the best method to collect a problem using the chosen difficulty heuristic (which may change later as more tests are completed).

The first difficulty heuristic chosen was determined as it was the simplest logical choice (keep it simple stupid). The most difficult problems are those that require more variables to be solved for and have more steps in the solution. While this is a naïve approach to determining problem difficulty, it is also simple and may yield results sufficiently close to what is desired. For each problem generated, the number of variables asked for added to the number of steps found in the algorithm's solution become the uncorrected difficulty score. A thousand problems are generated in succession and the maximum uncorrected difficulty score becomes the value about which to normalize. The following equations then summarizes a problem's final difficulty score

$$D_{unadjusted} = \left(\sum \text{variables to find} + \sum \text{steps in solution}\right)$$

$$D = \text{ceil}\left[\left(\sum variables\ to\ find + \sum steps\ in\ solution\right) \times \frac{10}{D_{unadjusted,max}}\right]$$

All fractional values are rounded up to the nearest whole number using a ceiling operation to adjust the difficulty scores to be between 1 and 10. Any value lower than one is rounded up to unity.

The logic of a difficulty rating determined in this way roughly assigns one "difficulty point" to each calculation that needs to be performed and one point to each logical solution step. This sum is then adjusted to a ten-point scale. Obviously, some calculations and some logical solution steps are more difficult than others. A more appropriate or accurate difficulty rating might adjust each step or variable with some additional weighting factor as shown below:

$$D = \text{ceil}\left[\left(\sum a_i * variables\ to\ find + \sum b_i * steps\ in\ solution\right) \times \frac{10}{D_{unadjusted,max}}\right]$$

Proper weighting values of such features, while more accurate, may be more difficult to determine and applied when all solution steps are created in an automated fashion.

To determine the efficacy of the difficulty estimates, undergraduate students were given a series of problems generated by the algorithm and asked to solve and score them. Those results were then compared to the algorithms own difficulty calculation.

The algorithm was used to generate four series of four problems each. For each series a single problem was chosen with a difficulty randomly between 1 and 3, two problems were chosen with a difficulty randomly between 3 and 8, and the last problem was chosen with a random difficulty between 8 and 10. Each student in an undergraduate dynamics course were then assigned four problems as extra credit. The class held about 100 students and about 20 students were assigned to each of the four series. The 20 remaining students comprised a control group and were given random problems from the course textbook all of which had solutions that could readily be found online.

Each student's problems were then graded for correctness and were given an educational survey to determine how they would score the difficulty of each problem. Students were asked the following questions:

1. What was the difficulty of each problem, rated from 1 to 10, 1 being the easiest and 10 being the hardest?
2. Was the problem either too difficult or too easy for you to be useful to learn about projectile motion?
3. How would you describe your skill level at solving projectile motion problems before completing this exercise?
4. Out of 100 points for each problem, how do you feel you scored?
5. Demographic information

Results and Conclusions

Conclusions for the efficacy of the random generator to produce problems consistent with desired difficulties was determined based on two factors. The performance of students solving the problems, and their answers to the educational survey (which involved self-estimation of their own performance). For a sample size of almost 100 students, five individual groups were defined with about 20 students in each group. Five problem sets were generated, one for each group. Four of the problem sets were generated by the randomized algorithm and the fifth was a collection of problems from the course textbook, whose solutions can easily be found online.

Students were asked to solve each problem and then answer a series of survey questions about each one. Figure 5 shows the results of the first question on the survey which asks the students to estimate the difficulty of each problem. Data is shown for each of the four randomly generated problem sets comparing the algorithm's difficulty score with that of the students. Each student difficulty score, in red, was an average of the answers from all roughly 20 students of that set. Difficulty scores calculated by the algorithm are in blue.

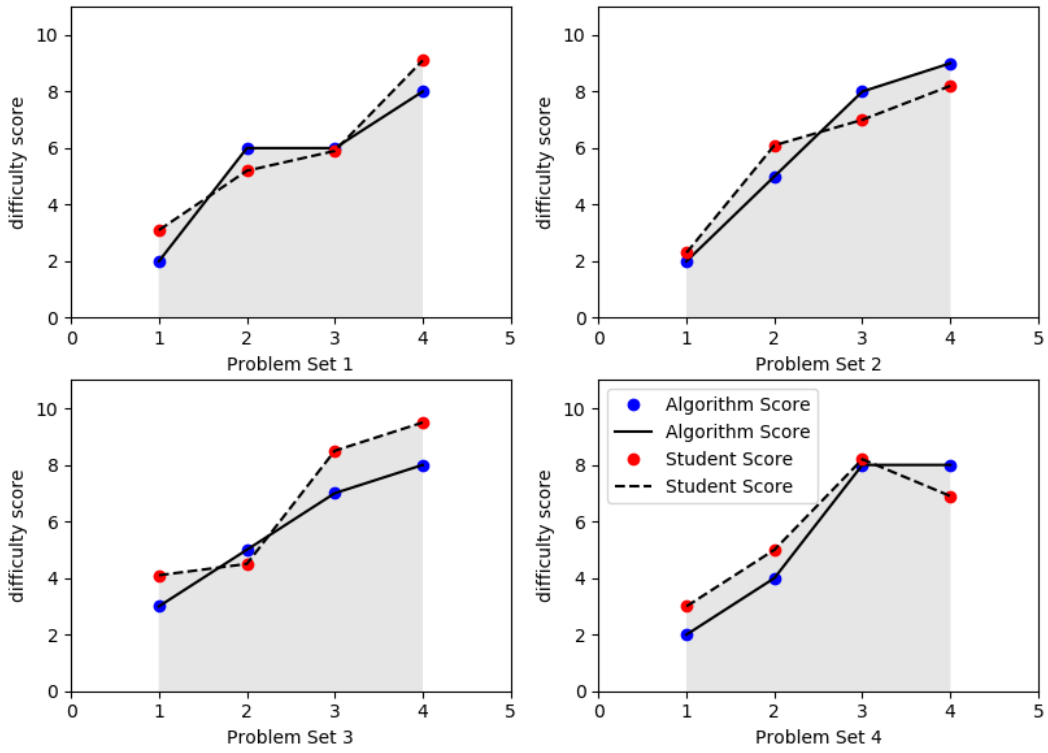

Figure 5: Comparison of student evaluated difficulty with the basic algorithm heuristic difficulty score.

As the sample space for each problem set includes only 20 students, good statistical results are lacking. However, several qualitative conclusions might be reached. First, there is significant variation between student scores both above and below the algorithm's scores. This variability was expected. Not only is the sample size small, but the accuracy and precision of student difficulty estimates are wide ranging. Students vary in ability and mastery of the material, vary in their ability to judge the needs, and therefore the difficulty, of problems they have not mastered, and, perhaps most importantly, vary in their confidence of their own skills.

Second, the student's difficulty scores and the algorithm's scores were almost always within one point of each other. This might suggest that the algorithm's heuristic is quite accurate in its estimate of difficulty. However, this may also represent a type of order bias or grouping bias. Students were not given the four problems in order of increasing or decreasing difficulty, but they all did seem to conclude that the four problems spanned the spectrum between easy at one and difficult at 10. It may then be obvious to students which problems were easier and which were more difficult, narrowing the window of estimation in which they would place each problem. This may indeed result in students placing each problem within 1 or 2 points of the algorithm's heuristic.

Lastly, there is one interesting variation in the last two problems of Set 4. Student averages rated the last two problems as 8.2 and 6.9 points respectively, while the algorithm determined both problems were an 8. This variation might be due to either the last problem of the set truly feeling easier to the students than the third or be due to the students' bias in not wanting to place two difficulty scores at the same value.

Most students seemed to underestimate their total scores from all four problems combined. Fifty-three percent of students underestimated their total scores, thirty-six percent of students overestimated their total scores and the remaining eleven percent of student estimated their final scores within one point. Most of the students who estimated their scores accurately achieved a perfect score for all problems.

Most students seemed to suggest that the exercises were helpful for learning. Most problems were rated with comments such as “not too easy and not too hard, but just right.” Close to the “Goldilocks zone” is deemed enough for a successful test.

Future Work

This work is a continuation of a project that begin in 2017. Development has proceeded steadily since then but considering the challenges outlined earlier it is not surprising that the endeavor continues to take time and will do so for some time. There are several areas where progress is chiefly centered.

First, more problem archetypes can be added to the library corresponding to as many as there are sections in a textbook on dynamics. Problem archetypes are not trivial to develop, nor do they typically work properly without testing. The intent is to create a simple encoding of equations and variables so that anyone can extend the archetype library. While creating equation sets or

variable sets is simple enough, developing the natural language or graphics that accompany them is not so simple.

Second, work continues on building sophistication into the natural language generation components of the algorithm. This difficulty falls into two categories. The first is in expanding the lexicon of dynamics-specific nouns and verbs. For a natural language algorithm to function the most robustly it must often contain a detailed dictionary of terms, their parts of speech, and basic rules on how to use them. SimpleNLG handles the grammar of the English language well enough, but for the purposes of writing word problems the language must be correlated to numerical values. For example, a projectile may either be a baseball or a bullet. It is reasonable that a baseball may travel 40 m/s and a bullet 500 m/s but not the other way around.

Third, developing customizable images to accompany the word problem is costly in time and graphic resources. While simple images that closely resemble the archetype images presented earlier are easy to produce, even with the appropriate scaling, it is more difficult to customize images more robustly. Consider the issue with expanding the lexicon to associate nouns with numbers from the previous paragraph. A similar issue exists in associating images with nouns. If a person is throwing a football, one could design an algorithm to draw a simple circle to the screen, but it is much more authentic to draw an image that resembles a football. And that football may be of many different styles, from a black-and-white outline to a photorealistic image, to anything in-between. Because of these issues, post-processing of randomly generated problem features is still required.

References

A. Butt, “Student Views on the Use of Lecture Time and their Experience with a Flipped Classroom Approach,” *SSRN Electronic Journal*, 2012.

J. W. Baker, “‘The Classroom Flip’: Using Web Course Management Tools to Become the Guide by the Side.”

Jackson, P. (2018, June), *A Study of Voluntary Problem Sets on Student Interest, Motivation, and Performance* Paper presented at 2018 ASEE Annual Conference & Exposition, Salt Lake City, Utah. https://peer.asee.org/29729

N. C. Rowe, “Cheating in Online Student Assessment: Beyond Plagiarism,” *Calhoun: The NPS Institutional Archive*, 2004.

D. Raines, S. Brown, P. Ricci, T. Eggenberger, T. Hindle, and M. Schiff, “Cheating in Online Courses: The Student Definition.,” *Journal of Effective Teaching*, 30-Nov-2010. [Online]. Available: https://eric.ed.gov/?id=EJ1092169.

Felder, R. & Brent, R. (2016) *Teaching and Learning STEM: A Practical Guide*. San Francisco, CA: John Wiley & Sons, Inc.

Michael, T.& Williams, M. (2013) Student Equity: Discouraging Cheating in Online Courses. *Administrative Issues Journal: Education, Practice, and Research*, 3(2), 1-12.

Muilenburg, L. & Berge, Z. (2007) Student Barriers to Online Learning: A Factor Analytic Study. *Distance Education*, 26:1, 29-48.

Lavieri, E. D., Jr. (2014) *A Study of Adaptive Learning for Educational Game Design*, ProQuest Dissertations Publishing (Order No. 3628679).

Murray, M. C., & Pérez, J. (2015). *Informing and performing: A study comparing adaptive learning to traditional learning*. Informing Science: the International Journal of an Emerging Transdiscipline, 18, 111-125.

Walberg, H. J., Paschal,R. A.,& Weinstein, T., (1985) *Homework’s Powerful Effects on Learning*, Educational Leadership, 85 (42):76-79.

Swanbom M. K. Moller D. W., Evans K., (2016) *Open-Source, Online Homework for Statics and Mechanics of Materials Using WeBWorK: Assessment of Student Learning*. Proceedings of the 2016 American Society for Engineering Education Annual Conference & Exposition. Paper ID 16092

Chew, K., Chen, H., Rieken, B., Turpin, A., & Sheppard, S., (2016) Improving Students’ Learning in Statics Skills: Using Homework and Exam Wrappers to Strengthen Self-Regulated Learning. Proceedings of the 2016 American Society forEngineering Education Annual Conference & Exposition. Paper ID 15770

Roberts, M., Curras, C., & Parker, P. (2006) A Homework ProblemDatabase: Design and Implementation: Proceedings of the 2006 American Soceity for Engineering Education Annual Conference & Exposition. 11.53.1 –11.53.9

Cohen, J. (2003) *Computer Algebra and Symbolic Computation*. Natick, Massachusetts: A K Peters, Ltd.

Bird, S., Klein, E. & Loper, E. (2009) Natural Language Processing with Python. Sebastopol, CA: O'Reilly.

Reiter, E. (2010). Natural Language Generation. In The Handbook of Computational Linguistics and Natural Language Processing (eds A. Clark, C. Fox and S. Lappin).