Automatic Game Progression Design through Analysis of Solution Features

Eric Butler[1], **Erik Andersen**[2], **Adam M. Smith**[1], **Sumit Gulwani**[3], **and Zoran Popović**[1]

[1]Center for Game Science
Computer Science & Engineering
University of Washington
{edbutler,amsmith,zoran}@cs.washington.edu

[2]Computer Science
Cornell University
eland@cs.cornell.edu

[3]Microsoft Research
Redmond, WA
sumitg@microsoft.com

ABSTRACT

A long-term goal of game design research is to achieve end-to-end automation of much of the design process, one aspect of which is creating effective level progressions. A key difficulty is getting the player to practice with interesting combinations of learned skills while maintaining their engagement. Although recent work in task generation and sequencing has reduced this effort, we still lack end-to-end automation of the entire content design process. We approach this goal by incorporating ideas from intelligent tutoring systems and proposing progression strategies that seek to achieve mastery of not only base concepts but arbitrary combinations of these concepts. The input to our system is a model of what the player needs to do to complete each level, expressed as either an imperative procedure for producing solutions or a representation of features common to all solutions. The output is a progression of levels that can be adjusted by changing high-level parameters. We apply our framework to a popular math puzzle game and present results from 2,377 players showing that our automatic level progression is comparable to expert-crafted progression after a few design iterations based on a key engagement metric.

Author Keywords

games; procedural content generation; education

ACM Classification Keywords

H.5.0 Information interfaces and presentation: General

INTRODUCTION

For many types of games, engagement is closely linked to the quality of the *level progression*, or how the game introduces new concepts and grows in complexity as the player progresses. Several game designers have written about the link between level progressions and player engagement. Game designer Daniel Cook claims that many players derive fun from "the act of mastering knowledge, skills and tools," and designs games by considering the sequence of skills that the player masters throughout the game [11]. Others have written about Flow Theory [13] and the importance of engaging players by providing content that appropriately matches players' skill as it grows over time [4]. Producing game content with an effective progression is difficult to do effectively and is typically done by hand. As a result, designers may be reluctant to revise their progression designs as new information about what players find interesting or challenging becomes available.

Intelligent tutoring systems (ITS) [29, 22] have advanced the teaching potential of computers through techniques that track the knowledge of students and select the most appropriate practice problems [8, 12]. We believe that we can leverage techniques from ITS to more effectively structure learning in games, which could increase engagement. However, the open-ended problem structure of many games does not easily lend itself to many existing ITS techniques.

One goal of game design research is end-to-end automation of the game design process. We take a step towards this goal through a framework that reduces the need for explicit expertise in learner modeling. We aim to enable a different method of game design that shifts away from manual design of levels and level progressions, towards modeling the solution space and tweaking high-level parameters that control pacing and ordering of concepts. A key component of many games is in the combination of basic concepts. What is interesting and engaging is not just the individual game rules and interactions, but how the progression combines them in increasingly complex ways. There is a combinatorial explosion in the number of ways to mix basic concepts, which, while allowing for many deep and interesting game experiences, makes creating progressions a challenge. Theories such as Vygotsky's zone of proximal development [34] motivate introduction of concepts at a controlled rate to allow the player time to master them. We propose that tracking and seeking mastery of combinations of basic elements of solutions is an effective way to organize and control the design space.

Recently, Andersen et al. [1] proposed a theory to automatically estimate the difficulty of *procedural* problems by analyzing features of how these problems are solved. *Procedural* problems are those that can be solved by following a well-known solution procedure, such as solving integer division problems by hand using long division. The system treats these procedures as computer code and the human solvers as computers. Given such a procedure and set of problems, the system considers the code paths that a solver would follow when executing the procedure for a particular problem, for example, how many times a particular loop must be executed

CHI 2015, April 18–23, 2015, Seoul, Republic of Korea.

ACM 978-1-4503-3145-6/15/04 ...$15.00.
http://dx.doi.org/10.1145/2702123.2702330

or which branch of a conditional statement is taken. We call these paths *procedural traces*. Using this information as an estimate of difficulty, the framework specifies a partial ordering of these problems. While the previous framework can analyze existing progressions and generate problems, it cannot synthesize full progressions. This is mainly because Andersen's framework does not account for pacing. In contrast, our system allows a designer to control the rate of increase in complexity, the length of time spent reinforcing concepts before introducing new ones, and the frequency and order in which unrelated concepts are combined together to construct composite problems. Furthermore, Andersen's system can only handle highly-constrained procedural tasks. Games often have open-ended problem structures with many valid solutions and solution methods, necessitating an alternative approach.

In this paper, we extend the ideas from procedural traces to automatically synthesize an entire progression for a popular puzzle game, *Refraction*. The game *Refraction* has no unique or preferred procedure for solving each puzzle. Therefore, we extend the partial ordering theory of procedural traces to work in a non-procedural domain. We present a system that controls the pace at which concepts are introduced to synthesize a large space of puzzles and the full progression for the game.

We evaluated whether this system can produce a game able to engage players in a real-world setting. We ran a study with 2,377 players and found no significant difference in time played compared to the original, expert-crafted version of the game. This original version found success and popularity on free game websites (played over one million times), and thus these empirical results suggest our framework captures important aspects of game progression design. Our long term goal is to extend this work to use data to optimize the progressions for each player.

This work contributes to HCI by demonstrating that classifying game levels by the structure of their solutions can form the basis of automatic progression design systems that are competitive with expert human-authored content. This framework is potentially applicable to any game that relies on players exercising increasingly complex combinations of basic skills, as is common in educational games and puzzle games.

There is inevitably a learning curve for a person to do anything on computer. One of HCI's eternal questions is enabling seamless acquisition of the skills needed to use software tools effectively. Many software tools are like games in that they are open ended, allowing the user to work with many different concepts. As a result, work on automatic scaffolding and progression generation in games may apply to other software systems in the long run.

RELATED WORK

Human Computer Interaction

Previous work in HCI has employed a variety of methods to increase engagement in games. Some have used theoretically-grounded or interview-based approaches. For example, Linehan et al. [25] draw on ideas from behavioral psychology to provide a series of guidelines for designing effective educational games. Isbister et al. [19] interviewed design experts to gain insights for creating learning games. Also providing guidance and opportunities to game developers, our work offers automatic methods which can allow designers to iterate on their designs as they seek to engage players.

Others have attempted to optimize engagement or learning through large-scale design experiments and data-driven methods. Andersen et al. measured the effectiveness of tutorials [3] and aesthetics [2] through A/B tests, examining metrics such as time played, levels played, and return rate. Lomas et al. [27] ran a large experiment that explored several design dimensions of an educational game to maximize engagement. Harpstead et al. [18] demonstrate a toolkit and method for analyzing player learning in games based on replaying players' games. Our apporach allows designers to tweak high-level parameters and quickly produce machine-generated playable games, which could pair with these data-driven apporaches to more efficiently compare alternative designs. Bauckhage et al. [5] looked at large datasets of how long players played several games and developed techniques to analyze player data and draw conclusions about player engagement. In this work, we do only a simple statistical comparison between our experimental conditions, though a more sophisticated analysis could be performed in future work.

Intelligent Tutoring Systems

Intelligent Tutoring Systems researchers have explored several models for capturing student knowledge and selecting problems [14]. Cognitive Tutors, such as Knowledge Tracing tutors [12], model student knowledge in terms of knowledge components: the underlying facts, principles, or skills a student puts to use in larger problem-solving tasks. They predict from student data whether students know particular concepts or have particular misconceptions. In contrast, systems using Knowledge Space Theory [16] use only observable actions. Our system uses a simple model of observable variables, though it can be extended to integrate more sophisticated models. Many of these tutors use *mastery learning* when choosing content; they give students repeated problems on a particular concept until the model has 95% confidence in their mastery, then move to the next concept [22].

A key part of progression generation is ordering problems appropriately. While many systems require experts to manually order content, several researchers have explored automatically learning the relationships between concepts, typically from user data [15]. Our work concerns capturing the effects of composite concepts; for example, if X and Y are concepts, a problem requiring X then Y has a composite concept XY. Liu et al. introduces a technique to learn relationships of composite concepts [26]. Standard models can capture composite concepts (e.g., XY) by modeling them as separate concepts, using prerequisite relationships to preserve ordering.

In games, there are often non-linear dependencies between base skills and skill combinations. For example, a player might be able to handle X or Y independently but struggles when doing them together. Therefore, consideration of concept combinations is crucial both for verifying mastery

and also for driving the increase in complexity. However, if all combinations of base skills are modeled as an independent skill, this space grows exponentially large. Additionally, analysis of this joint conceptual space is often reduced to someone manually selecting combinations of skills to treat as standalone concepts. Therefore, a structured and automatic way of sampling this space is desirable.

Our system, rather than depending on a pre-existing database of user data or hand-designed relationships, determines the relationships from the structure of the solutions to problems, both for procedural and some non-procedural domains. Stern et al. describe a system that sequences problems using a scoring function of concepts and their prerequisites [33]. We build on this work in our sequencing policy. Another important aspect of creating progressions is controlling pacing. Beck et al. [6] explore choosing problems of appropriate difficulty by ranking problems based on the number of required skills and the student model's prediction of proficiency at those skills. We take a similar approach when choosing appropriate problems in our framework.

We do not directly innovate ITS methods in our work, nor do we use the most sophisticated ones. Instead, we seek to demonstrate that a framework inspired by ITS principles, when combined with some additional machinery to drive the growth of complexity and reasonable parameter settings, can produce engaging game content that is comparable to content produced by an expert game designer. Researchers have previously looked into bringing ITS techniques into educational games [17]. The primary difficulty stems from an open-ended problem nature in most games that does not easily lend itself to ITS-based structures that are geared towards mastery of concepts rather than a large space of creative recombinations of concepts and skills required to solve an explorative problem. We address this in our framework by focusing on the ability to use combinations of concepts to reach a solution.

Games

Content for games has traditionally consisted of hand-crafted progressions and levels. Game design researchers have explored procedural content generation to automatically create games. Several systems aim to automatically generate game levels. To measure the quality or diversity of generated content, many approaches measure static properties of the content [24] or affect of players [35]. Other approaches focus on what the player must do, measuring properties related to player action. For example, Launchpad characterizes levels by the rhythm of the actions the player must perform [32]. We generalize such ideas; for example, actions in a platform game can be viewed as segments of a trace of a procedure for solving platformer levels.

Generated levels are not useful in isolation; they must be sequenced into some design-relevant order, called a *progression*. A good game progression is critical for game design [11]. For generated games, if only a small number of global parameters need to be controlled, the problem can be treated as dynamic difficulty adjustment, such as in the game Polymorph [21]. However, our domains have many independent conceptual building blocks, and adjusting a few parameters fails to capture the many different dimensions by which levels create difficulty. In the platform game Endless Web [32], the player explores a large design space of levels that are classified based on various features. The player makes choices that influence what levels they will receive. Perhaps the closest work to our approach is *Square Logic*[1] by Everyday Genius, a Sudoku-style puzzle game that features 20,000 automatically constructed puzzles that are arranged into a progression based on which logical inference rules are required to solve them. Few details of the techniques used are available, but one difference with our problem is that *Refraction* does not have a clear set of inference rules used to form solutions. Further, many *Refraction* levels have more than one possible solution (though these solutions will often share high-level structure and concepts).

Several researchers have proposed mixed-initiative editors in which the system and human designer take turns producing the content [31, 24]. These systems were limited to the creation of single levels. Butler et al. [10] proposed a mixed-initiative system for end-to-end game production allowing for manual designer intervention at all stages of content generation. By contrast, the present work borrows inspiration from ITS to drive conceptual growth automatically, reducing the need for expertise in the form of manual authoring. Using our system, the designer does not manually edit levels or the progression on a per-level basis, but instead controls them through high-level parameters that describe the overall goals of the progression. These parameters allow the designer to make sweeping, purposeful changes to the progression.

APPLICATION

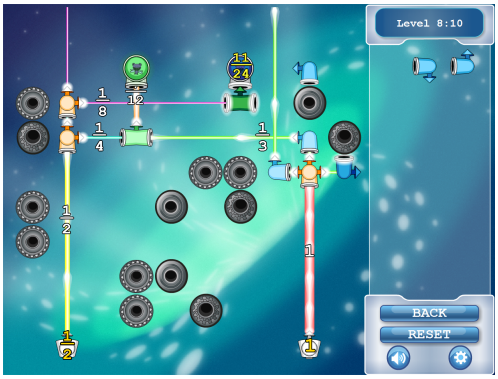


Figure 1. A level of *Refraction*. The goal is to use the pieces on the right to split lasers into fractional values and redirect them to satisfy the target spaceships. The user can pick up and put down pieces by clicking on them. The grid interface, pipe-flow game mechanics, and spatial reasoning puzzles are similar to many other puzzle games.

Before describing the technical details of our system, we describe *Refraction*, the application with which we performed our study. In *Refraction*, players solve spatial puzzles by splitting virtual lasers into fractional amounts. Each puzzle is played on a grid that contains laser sources, target spaceships, and asteroids which obstruct lasers, as shown in Figure 1. Each target spaceship requires a fractional amount of laser power, indicated by a yellow number on the ship. The

[1] http://www.squarelogicgame.com/

player can satisfy the targets by placing pieces that change the laser direction and pieces that split the laser into two or three equal parts. All targets must be correctly satisfied at the same time to complete the puzzle. Although *Refraction* was built to teach fractions to schoolchildren, it has found popularity with players of all ages and has been played over one million times. *Refraction* is freely available and can be played by anyone with a web browser and Adobe Flash.

The game relies primarily on the quality of puzzles to engage players. The original version of *Refraction* contains 61 puzzles that game designers created by hand over many dozens of hours. As the player progresses, the puzzles gradually increase in difficulty through the introduction of new concepts, such as benders, splitters, combiners, multiple sources, and multiple targets. The puzzles also increase in difficulty by combining base concepts together in way that necessitates more complex search processes, such as puzzles that require bending a laser clockwise and then counterclockwise to circumvent an obstacle, or puzzles that require splitting a laser into halves and then into thirds in order to produce one sixth. Describing a player's exact solution process for *Refraction* is very difficult, as there are many different procedures that may lead to the correct answer. Furthermore, there are often multiple configurations of pieces that can solve a puzzle correctly. Although most of these solutions are slight positional perturbations of other solutions, solutions can also differ in qualitative ways. When the network of laser beams is examined as a graph, different solutions may involve different connectivity patterns or even omit different sets of pieces from usage. For this reason, we cannot directly model solutions as unique sequences as in the previous procedural trace framework [1].

SYSTEM OVERVIEW

We first summaraize the components of our system, then go into detail about each component in subsequent sections. In this work, we use the game-termonology *levels* to mean any completable chunk of content, such as an single puzzle in Sudoku, a math problem in a workbook, or a one of the stages in Nintendo's *Super Mario Bros*. For our application, *Refraction*, a level is an individual puzzle. A *level progression* is a sequence of levels that create an entire game.

In order to arrange a given set of levels into a progression, we need to be able to extract features from the levels that can be used to create an intelligent ordering. In this work, we propose the use of *solution features*, which are properties of the solution to a level. In *Refraction*, a solution is a particular board configuration, and the features are deduced from the structure of the board pieces in the solution. The techniques we describe, while containing game-specific components, can be applied to a variety of games.

Of course, before we can automatically arrange levels into a progression, we fisrt have be able to create individual levels automatically. Creating levels is entirely game-specific, and we discuss the techniques used to create *Refraction* levels for our study.

Finally, the system must have a method for using the solution features to arrange levels into a progression. The method we present treats features indistinguishably and therefore applies generally to any game for which solution features are computed.

EXTRACTING SOLUTION FEATURES

Andersen et al. [1] proposed the use of n-grams (subsequence of length n from a larger sequence) to abstract procedural traces, allowing the creation of a natural partial ordering on traces. In this section we describe how we extend this to extract features from non-procedural games such as *Refraction*.

Procedural Traces and n-grams

We begin with a review of traces and n-grams for procedural domains. Consider the algorithm for subtraction detailed in Algorithm 1. We indicate traces through this algorithm as *sequences* of letters; these letters are output when the program executes commented lines in the above procedure. A few problems and their traces are shown in Table 1.

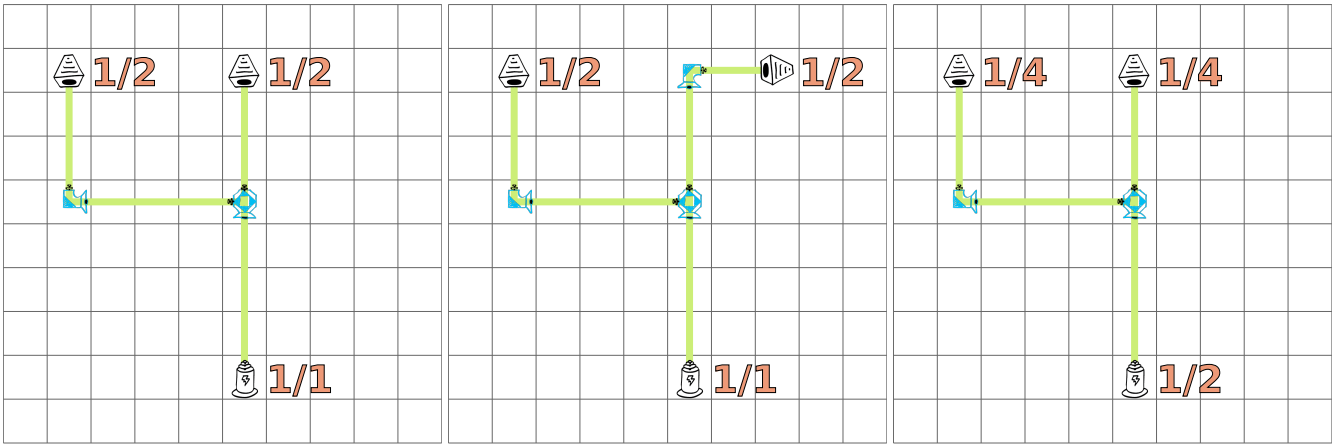
Algorithm 1 Subtraction: Given as input a minuend p and subtrahend q, both sequences of digits $n_m, ..., n_0$:

```
1:  procedure SUBTRACT(p, q)
2:      for i := 0 to len(p) − 1 do                    ▷ Each digit (D)
3:          if i < len(q) then                  ▷ Subt. digit present (S)
4:              if q[i] > p[i] then                      ▷ Must borrow (B)
5:                  p[i] := p[i] + 10
6:                  j := 0
7:                  while p[i + j] = 0 do                      ▷ Zero (Z)
8:                      p[i + j] := 9
9:                      j := j + 1
10:                 end while
11:                 p[i + j] := p[i + j] − 1
12:             end if
13:             a[i] := q[i] − p[i]
14:         else                                          ▷ Copy down (C)
15:             a[i] := q[i]
16:         end if
17:     end for
18: end procedure
```

Subtraction Problem	Trace
$1 - 1$	DS
$11 - 11$	DSDS
$11 - 1$	DSDC
$11 - 2$	DSBDC
$101 - 2$	DSBZDCDC

Table 1. Example subtraction problems used in elementary-school mathematics and their corresponding traces.

Given a problem and its trace, the *n-grams* of a trace are the n-length substrings of the trace. For example, in the trace $ABABCABAB$, the 1-grams are $\{A, B, C\}$, the 2-grams are $\{AB, BA, BC, CA\}$, and the 3-grams are $\{ABA, BAB, ABC, BCA, CAB\}$. Intuitively, 1-grams represent fundamental concepts, and higher-value n-grams represent the sequential interaction, not captured by 1-grams, of using multiple concepts together. n-grams are intended as a method for computing comparable features for traces. We



(a) (b) (c)

Figure 2. Three example levels for *Refraction*, to illustrate solution features and graphlets. When considering the *solution graph*, levels (a) and (b) have the same 1-graphlets. They differ in 2-graphlets: (a) has a splitter-target chain, while (b) does not. They differ on several 3-graphlets; notably, (a) has a splitter branching into a bender and target while (b) has a splitter branching into 2 benders. However, the solution graph is insufficient to distinguish (a) from (c). We need the additional *math graph* to capture differences between them with graphlets.

can represent a problem by the n-grams of its trace and use this to compare problems' relative complexity. We typically will only compute n-grams up to a small, fixed n, such as 3 or 4. The choice of n is a trade-off: a small maximum n will not capture salient interactions between concepts, but a large n will distinguish problems with only a slight difference in traces.

Solution Features in Refraction

Because *Refraction* does not have a preferred solution procedure, we cannot directly apply n-gram analysis. Rather than attempting to describe the procedure players use, we model features of solutions (henceforth, *solution features*), that players necessarily encounter. For now, we assume that ever puzzle has a reference solution that is representitive of all possible solutions. In the section on level generation, we describe how we can extract features even though *Refraction* puzzles often have many valid solutions.

As the players are forced to connect the pieces into a graph, one obvious source of solution features is the reference solution's laser graph, which we call the *solution graph*. As the analogue to n-grams for procedures, we use *graphlets*, which are small connected non-isomorphic induced subgraphs of a larger graph [28]. Figure 2 shows some example *Refraction* levels to explain graphlets. In *Refraction*, graphlets on the solution graph capture ideas such as the following: an example of a 1-graphlet is that a two-splitter is used; an example 2-graphlet is a laser edge linking a bender and a two-splitter; an example 3-graphlet is a length-3 chain including two two-splitters and a three-splitter.

While the solution graph captures many features of the solution, particularly the spatial components, we model additional graphs to capture other mechanics. For example, we can construct a graph whose nodes are the fraction values of the source and target pieces, where two nodes are connected with an edge if, in the reference solution, that source has a path to that target. We select graphlets on both this auxiliary math graph and the solution graph. Although we do not claim to understand how the spatial and mathematical challenges interact when players solve the puzzle, by using multiple graphs we are able to capture those features we find relevant for progression design in sufficient depth.

AUTOMATICALLY GENERATING LEVELS

We applied the level generation process described by Smith et al. [30] to generate a large and diverse database of puzzles. In each puzzle, we enforced the constraint that all potential solutions to the level share the same solution graph. The generation process additionally produces a reference solution. Since all solutions are guaranteed to share the same graph, we can safely extract general solution features from the single reference solution produced by the generator. This analysis could be extended to puzzles with multiple, significantly different solutions by considering which subset of features must show up and which optionally may be used to solve the problem, but we omitted this for sake of simplicity.

CREATING A PROGRESSION

In this section we descirbe how to take a set of levels describe by solution features and create a full progression. We use the term *n-grams* in this section, but everything described applies euqally to graphlets.

Given a diverse set of a problems and their traces, the solution features suggest a natural partial ordering of these problems, described by Andersen et al. [1].To summarize, given two levels L_1 and L_2, L_1 is considered conceptually simpler if, for some positive integer n, the set of n-grams of L_1 is a strict subset of the set of n-grams of L_2. Other approaches use data collected from players to determine the complexity relationships between problems (e.g., [15]). However, for spaces with a large set of possible traces, learning all these relationships from data or specifying them manually can be infeasible. Our technique uses the structure of the solutions to compute these

relationships without data, which has been shown to correlate with users' perception of difficulty in the domain of an educational algebra game [1].

Our system must have a sequencing policy for creating a full progression, allowing the designer to control pacing and ordering. A simple approach would be to traverse the graph of partially-ordered problems in an order consistent with a topological sorting of the graph. This would ensure problems would be introduced before any of their (more complex) descendants. This approach has several drawbacks. It does not allow for control over the rate at which complexity increases or control over the balance between when new concepts (1-grams) are introduced versus combinations of concepts (2-grams and 3-grams). Moreover, this method would attempt to show every problem. Games cover a deliberately chosen subset of the entire (combinatorily large) space of possible problems, but this method lacks an intelligent way to decide which subset to use. As discussed, rather than trying to fully order a set of problems, we instead will use a large set of generated levels as a library from which to select an appropriate progression.

Systematically Tracking and Introducing n-grams

To create progressions, we propose tracking players by estimating their mastery of concepts (i.e., knowledge components) identified by the n-grams found in level solutions and pairing this with a mastery-learning-based sequencing policy to teach content. That is, the system's goal is to systematically introduce each of the possible solution features. The model should assign, for each concept we want to introduce, a measure of whether the student has had success over problems involving that concept. This, along with a method of quantifying the expected difficulty of future problems, gives us a basis for choosing the next problem to present. With our diverse set of labeled problems, the system will be able to carve nearly any possible path through this space, allowing the model to choose the most appropriate content at each stage. This ordering may be done offline to create a static progression, or online to create an adaptive game that responds to player performance. As it was not the focus of this work, we chose a simple method for modeling players in this system that does not reflect the state-of-the-art in student modeling. This framework allows for the insertion of more sophisticated models, such as Bayesian Knowledge Tracing [12].

In our system, the model tracks, for each component, whether the user successfully completed a problem containing that component. Note that we usually do not want to track *all* the concepts. As n increases, the number of n-grams tends to increase, sometimes very quickly. Therefore, it generally makes sense only to track all n-grams up to a small n.

The model is updated every time the player completes a problem. If the player is successful, then we mark as successful *all* of the n-grams contained within the problem that was just given. If the player fails, then it is less clear what to do. If we have no knowledge of how or why the player failed that might allow us to update the model, then the strictest option is to mark all components contained within the problem as unsuccessful. If we have more knowledge of where the player failed in the trace, we can update this more accurately. If we know that the player failed on a specific 1-gram, then we mark as unsuccessful all n-grams containing that basic concept.

Choosing the Next Problem

Given a set of problems, the domain of concepts, and a player model, the framework must generate a sequence of problems for the player. We want this choice to respect the partial ordering determined from the n-grams, but allow the designer and player model to control the pacing of content. The specific task is: given the current model state and a set of problems, we must choose the most appropriate next problem. Our progression generator builds progression one level at a time, choosing the next based on history of previous problems and data from player performance. We use a dynamic cost model, which allows us to control the pacing and respect the ordering. The cost of a particular problem p is a weighted sum of the n-grams in the trace of p. These weights have 2 components: one based on what the model knows about the player, and one designer-specified weight.

First, at each point in the progression, for a given n-gram x, the player model assigns a cost $k(x)$. This cost should be high for unencountered n-grams and low for ones already mastered, which will ensure more complex problems have a higher cost than simpler ones with respect to the player's history. This is used to respect the partial ordering of problems. The cost of a problem can intuitively be considered inversely proportional to the chance of the modeled player succesfully completing the problem using only their currently mastered knowledge.

The second component is designed to allow the designer control over pacing and relative order of basic concepts. The system treats concepts indistinguishably as opaque solution features. On the other hand, as expert designers, we know (possibly from data gathered during user tests) that particular concepts are more challenging than others or should otherwise appear later in the progression. For example, we may like to have some control over the ordering in which the system introduces 1-grams, or have the system give a higher cost to 1-grams than 2-grams, to prefer increasing complexity over adding new concepts. To this end, the second component of our cost function is a designer specified weight $w(x)$, which can be used to influence the relative order of otherwise independent n-grams.

Thus, given a library of problems $\mathcal{P}$, choosing the next problem p_{next} consists of finding the problem with the closest cost to some target value T. This is shown in Equation 1. Intuitively, a minimum cost problem would be the easiest for the player to successfully complete, but it would not stretch their knowledge into new areas at all.

$$p_{\text{next}} = \arg\min_{p \in \mathcal{P}} \left| T - \sum_{x \in \text{ngrams}(p)} w(x)k(x) \right| \quad (1)$$

The target value and weighting functions can be based on the player model. For example, an adaptive progression may change the target value based on how well it perceives the player to be performing. Note that (for any positive weight

function) this cost function respects the partial ordering because for any levels $L_1 \prec L_2$, L_1 has a strict subset of the n-grams of L_2 and therefore less cost.

In order for this sequencing policy to be effective, it requires a large library of levels from which to choose. The levels should be diverse, covering the space of all interesting solution features. Furthermore, in order to enable effective control of pacing, this space should be dense enough that the progression can advance without large conceptual jumps. That is, for any level in the space, we want there to be a potential next level that adds few new n-grams. We accomplished this by ensuring that our level generation procedure attempted to create levels for every possible solution graph up to a particular size.

Level Selection in Refraction

During live gameplay, the library of levels embedded into the game is repeatedly consulted to determine (using the described cost function) which level to give to the player next. The graphlet-specific weights ($w(x)$ in the cost function) were largely left at a default value (1.0). It might seem strange to assign the feature of using a single bender the same weight as forming a chain of three benders in a row. However, graphlets have similar properties of containment as n-grams: for example, every level that posses the triple-bender-chain graphlet will also contain the double-bender-chain graphlet. Since our cost function respects the subset-based partial ordering (defined identically for graphlets as for n-grams), we can be certain that levels that combine independent concepts or explore embedded repetitions of those concepts have higher cost than the levels on which they build (causing them to occur later in generated progressions). We only adjusted the weight for four of the features, usually associated with graphlets of size 1. This very sparse advice allowed the progression system to know that, all else being equal, it is easier to introduce benders than splitters or that it is easier to introduce two-splitters than three-splitters.

Our player model tracked, for each graphlet, the number of levels the player successfully completed containing that graphlet, minus the number of unsuccessful attempts (but never less than 0). For each each graphlet x, call this value $c(x)$. We assigned cost $k(x) = 1/(1+c(x))$, so that graphlets would move towards zero cost as they were successfully encountered.

One additional constraint we imposed on level selection in *Refraction* is that new levels were required, if possible, to include at least one graphlet that had $c(x) = 0$ (i.e., had either never been seen or had recent unsuccessful attempts). This guaranteed that the progression always chose something with an unmastered challenge: it would otherwise be possible (albeit unlikely) that the level with the closest cost to the target value is actually easier than the most-recently beaten level. Furthermore, in the interest of presenting the player with a series of interesting challenges, we ensure that no level is ever selected twice.

As the construction of auxiliary graphs, selection of graphlet features from them, and assignment of weights is an open-ended design task, we iterated on our choices for part of the system several times. The time to re-label a library of levels and re-build a version of the game using the new labels was under 30 seconds, allowing us to rapidly explore many different trajectories through our diverse library without re-running the level generator.

EVALUATION

With our evaluation, we hope to demonstrate that this framework is capable of producing a game of comparable engagement to an expert-designed version. Specifically, we hope players would play for comparable amounts of time. In order to accomplish this, we generated an entire progression for an existing successful game, *Refraction*, and performed a between-participants experiment against the original version of the game. The original's puzzles were hand-crafted by game desigers. We believe that the original version of the game serves as a fair comparison against which to test the generated version. The game found a large amount of success (being played over a million times since its release) and relies heavily on the quality of the puzzles. Therefore, even though it is prone to bias, we argue the puzzle design in the original version can be considered to be of high enough quality that it serves as a reasonable target for our automated system.

We deployed both versions of *Refraction* and performed a between-participants experiment to measure engagement. We posted the game on a popular Flash game website, Newgrounds[2], as well as though MochiMedia[3], which distributes through the main MochiGames[4] website and dozens of other affiliated game websites. The game was released under the title *Infinite Refraction*. We gathered data from 2,377 players in this experiment. 1,221 randomly selected players played the version driven by our framework and 1,156 players played a version that included the exact, expert-designed level progression from the original *Refraction*.

When deciding what to measure, we are interested in actual player behavior rather than player opinions, so we measure player performance directly. We performed our test "in the wild," without notifying players that they were part of an experiment. We did not collect any personally identifiable information from players. Because we did not ask players directly, we must rely on proxy metrics to estimate engagement. One of the primary metrics previous "in the wild" experiments have used to estimate engagement is total play time [2, 3, 27], which we use here. Because the platforms for this experiment were websites with thousands of free games that players could be choosing to play instead of *Refraction*, getting players to spend a significant amount of time on a game suggests that the game is at least somewhat engaging.

We first performed a series of statistical tests on the distribution of time played in each condition. Our data was not normally distributed, so we relied on a nonparametric test, the Wilcoxon-Kruskal-Wallis test. Despite having over one thousand players in each condition, the test showed no significant

[2] www.newgrounds.com
[3] www.mochimedia.com
[4] www.mochigames.com

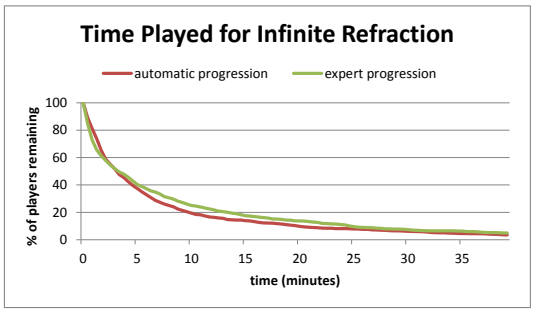


Figure 3. Comparison of our automatically-generated progression with the expert-generated progression from *Refraction*. The x-axis is time in minutes and the y-axis is the percentage of players who played for at least that much time. The median values are very similar: approximately 3 minutes. Although our framework's progression performs slightly worse, it is certainly *comparable* to the expert progression. These results suggest that our framework was able to create an experience that engages players approximately as long as one crafted by hand with design expertise. In contrast to the original design process, which included many hours of crafting and organizing puzzles by hand, our framework requires the designer to specify the rules of the game, a process for solving the game that our framework deconstructs into base conceptual units, and a few designer-tuned weights that control the order and speed of introduction of these units.

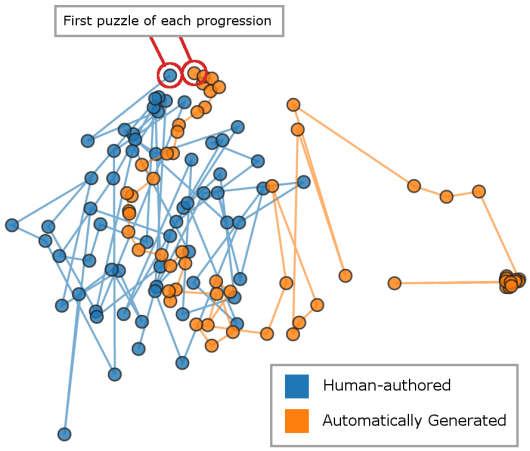


Figure 4. Visualization of the first 61 levels in the two progressions in our experiment, the human-authored (blue) and automatically genreated (orange). Nodes represent levels and edges are drawn between consecutive levels. Levels are represented as feature vectors of concepts and projected into two dimensions using Multidimensional Scaling. Nodes that are close together represent puzzles that are conceptually similar according to our solution features.

difference between the two populations on time played, with a median of 184 seconds for the automatically-generated progression and 199 seconds for the expert-crafted progression. In an effort to quantify this small difference we observed, we looked at other engagement metrics, paricularly the number of puzzles completed in both groups. We found a small but significant difference, $r = 0.13$, $p < 0.001$, with players playing the automatically generated progression completing a median of 8 puzzles versus 10 puzzles completed for the original, expert-designed progression. Since the puzzles are not the same between conditions, they are not directly comparable, so we feel that this is not the best measure of engagement. The median time is 8% lower for the automatically generated progression than the human-authored one. Figure 3 visualizes this data in greater detail. We can see from this graph that the automatically generated progression is able to retain players at rates qualitatively and quantitatively similiar to the original game. Though we can draw no definite conclusions from this lack of difference, the results do suggest that the game automatically produced by our framework was capable of engaging players for a comparble length of time to a hand-crated, expert-designed version.

While our experiment has a condition that represents a reasonable upper bound on progression quality (the human-authored progression), one major limitation in our experiement is the lack of a condition to represent a lower bound on progression quality. With our current results, we cannot be sure the progression had any effect on player engagement at all. In a followup experiment, a 3rd condition with a completely random progression should be added to the experiment. If, in this hypotehtical experiement, the random condition was not significantly worse than the human-designed or automatically-generated progressions, it would suggest that perhaps progressions have little effect on that set of players. On the other hand, if random progressions were significantly worse than the other two conditions, it would suggest that progressions do have some effect on player engagement, and furthermore, that our automatic techniques were capturing at least some aspects of engaging progression design.

Difference Between Progressions

Figure 4 shows a visualization of the first 61 puzzles of each progression in our experiment, created using a tool by Butler et al. [9]. To compute the visulization, levels are represented as a feature vector of concepts (one for each n-gram) and projected into two dimensions using Multidimensional Scaling [23]. Thus the visualization shows how different levels are from each other in the conceptual space determined by our solution features. We must assume that this distance (i.e., Euclidean norm) in the feature space captures at least some information about the coneptual difference between two puzzles that players would percieve. Because these features are based on how a puzzle must be solved, there is enough meaning to at least note some qualititative differences between the two progressions.

One major difference is that our automatically generated progression takes much smaller steps in this conceptual space than the human-authored one. Another difference we can see is that the automatically generated progression has several puzzles clustered together. This is because, at that point in the progression, our system had run out of new concepts to introduce. Our cost model then tends to choose very similar levels over and over. This exposes a limitation in our technique; by contrast, the human progression often makes large conceptual jumps and never stays in the same spot for an extended period of time. If we wished to more closely mimic the human progression, we could try to slow the rate at which concepts are introduced or add features to the cost model that penalized choosing concecutive levels that were too close in the conceptual space.

Discussion

This work presents a key building block towards data-driven automatic optimization of game progressions. It allows us to automatically create a progression that is comparable to an expert progression with minimal manual tuning. Because it exposes key high-level control knobs (e.g., rate at which concepts are introduced, when to favor introducing new concepts over combinations), data-driven optimization could further refine these parameters, which we hope to explore in a future study.

Due to the time and and difficulty in implimenting this system for a particular game, we expect that applying this technique to a particular game is often more laborious that manual progression design. It is theoretically possible that the automated method saves hundreds of hours of effort by avoiding having to manually craft puzzles. On the other hand, designers likely have to create and test many different levels to gain enough understanding to choose good features, and the cost of creating a level generator is often non-trivial. Unlike the previous trace-based framework that only needs a solution procedure, this method requires enough insight into the mechanics to create a reasonable labeling function that can extract features. For us, when the end-goal is data-driven optimization of game progressions, the benefits are apparent. Other promising applications may include all-in-one game creation tools such as PuzzleScript[5], especially those with a constrained language for expressing rules and mechanics. In such tools, the extent of possible rules and mechanics is (somewhat) known for most or all games created within the tool. Therefore, the labeling function and other parts required for system could be included in the tool and take much less effort to apply to any particular game. However, if the goal is merely to produce a good game with a reasonable level progression, it is unclear that this system would be beneficial, even for domains well-suited to this approach.

We believe this approach could be applied to games whose content centers around a progression of concrete skills or procedural knowledge, such as puzzle games and educational games. It currently requires a game designer to provide a breakdown of the individual *skill components* of each puzzle, and thus games for which this breakdown cannot be reasonably produced will not work well with this system. Games that primarly rely on apsects other than the level progression for engagement will also likely not benefit from this system.

CONCLUSION

We presented a framework that automates level progression design by analyzing features of solutions. This framework borrowed ideas from intelligent tutors to enable a model of design where content and progression are fully generated. We applied these ideas to create an implementation for the educational game *Refraction*. Our study with 2,377 players showed that the median player was engaged for 92% as long as in an expert-designed progression, demonstrating that our framework is capable of producing content that can engage players for a comparative length of time as content designed by hand.

[5]http://www.puzzlescript.net/

In the future, we plan to further explore integrating technologies of intelligent tutoring systems in this framework. Our implementations used a simple player model, but we plan to explore using technology from cognitive tutors and other models to drive the progression. While our system can order problems based on the combination of concepts, it cannot distinguish individual concepts (the 1-grams) and must resort to designer-specified weights or arbitrary decisions. Although the parameter settings we tried for *Refraction* were inspired by the original progression, we do not have a deep understanding of why they were effective. We hope that automatic methods for learning these settings will increase the designer's ability to optimize a game for engagement. While previous work showed that ordering problems using n-grams correlated with user perception of difficulty, further studies should be performed to understand this relationship. We speculate that difficulties keeping the user engaged in ITS [20, 7] may be alleviated by deeper exploration of complexity, and the solution-trace methods described in this paper may help to accomplish this through automatic control and tracking of concept combinations.

ACKNOWLEDGEMENTS

This work was supported by the Office of Naval Research grant N00014-12-C-0158, the Bill and Melinda Gates Foundation grant OPP1031488, the Hewlett Foundation grant 2012-8161, Adobe, and Microsoft.

REFERENCES

1. Andersen, E., Gulwani, S., and Popovic, Z. A trace-based framework for analyzing and synthesizing educational progressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 773–782.

2. Andersen, E., Liu, Y.-E., Snider, R., Szeto, R., and Popović, Z. Placing a value on aesthetics in online casual games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 1275–1278.

3. Andersen, E., O'Rourke, E., Liu, Y.-E., Snider, R., Lowdermilk, J., Truong, D., Cooper, S., and Popovic, Z. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2012), 59–68.

4. Anthropy, A., and Clark, N. *A Game Design Vocabulary*. Addison-Wesley, 2014.

5. Bauckhage, C., Kersting, K., Sifa, R., Thurau, C., Drachen, A., and Canossa, A. How players lose interest in playing a game: An empirical study based on distributions of total playing times. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, IEEE (2012), 139–146.

6. Beck, J., Stern, M., and Woolf, B. P. Using the student model to control problem difficulty. *Courses and Lectures-International Centre for Mechanical Sciences* (1997), 277–288.

7. Bell, C., and McNamara, D. Integrating iSTART into a high school curriculum. In *Proceedings of the 29th Annual Meeting of the Cognitive Science Society*, Cognitive Science Society (Austin, TX, USA, 2007).
8. Block, J. H., Airasian, P. W., Bloom, B. S., and Carroll, J. B. *Mastery learning: Theory and practice*. Holt, Rinehart and Winston New York, 1971.
9. Butler, E., and Banerjee, R. Visualizing progressions for education and game design. 2014.
10. Butler, E., Smith, A. M., Liu, Y.-E., and Popovic, Z. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, ACM (2013), 377–386.
11. Cook, D. The chemistry of game design. *Gamasutra* (2007).
12. Corbett, A. T., and Anderson, J. R. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction 4*, 4 (1994), 253–278.
13. Csikszentmihalyi, M. *Flow: The Psychology of Optimal Experience*. Harper & Row Publishers, Inc., New York, NY, USA, 1990.
14. Desmarais, M. C., and d Baker, R. S. A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction 22*, 1-2 (2012), 9–38.
15. Desmarais, M. C., Meshkinfam, P., and Gagnon, M. Learned student models with item to item knowledge structures. *User Modeling and User-Adapted Interaction 16*, 5 (2006), 403–434.
16. Doignon, J.-P., and Falmagne, J.-C. Spaces for the assessment of knowledge. *International journal of man-machine studies 23*, 2 (1985), 175–196.
17. Easterday, M. W., Aleven, V., Scheines, R., and Carver, S. M. Using tutors to improve educational games. In *Artificial Intelligence in Education*, Springer (2011), 63–71.
18. Harpstead, E., Myers, B. A., and Aleven, V. In search of learning: facilitating data analysis in educational games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 79–88.
19. Isbister, K., Flanagan, M., and Hash, C. Designing games for learning: insights from conversations with designers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2010), 2041–2044.
20. Jackson, G. T., Dempsey, K. B., and McNamara, D. S. Short and long term benefits of enjoyment and learning within a serious game. In *Artificial Intelligence in Education*, Springer (2011), 139–146.
21. Jennings-Teats, M., Smith, G., and Wardrip-Fruin, N. Polymorph: A model for dynamic level generation. In *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference* (2010).
22. Koedinger, K., Corbett, A., et al. Cognitive tutors: Technology bringing learning science to the classroom. *The Cambridge handbook of the learning sciences* (2006), 61–78.
23. Kruskal, J. B., and Wish, M. *Multidimensional scaling*. Sage, 1978.
24. Liapis, A., Yannakakis, G. N., and Togelius, J. Sentient sketchbook: Computer-aided game level authoring. In *Procedings of the 8th International Conference on the Foundations of Digital Games* (2013), 213–220.
25. Linehan, C., Kirman, B., Lawson, S., and Chan, G. Practical, appropriate, empirically-validated guidelines for designing educational games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2011), 1979–1988.
26. Liu, C.-L. A simulation-based experience in learning structures of bayesian networks to represent how students learn composite concepts. *International Journal of Artificial Intelligence in Education 18*, 3 (2008), 237–285.
27. Lomas, D., Patel, K., Forlizzi, J. L., and Koedinger, K. R. Optimizing challenge in an educational game using large-scale design experiments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM (2013), 89–98.
28. Pržulj, N., Corneil, D. G., and Jurisica, I. Modeling interactome: scale-free or geometric? *Bioinformatics 20*, 18 (2004), 3508–3515.
29. Schulze, K., Shapiro, J., Shelby, R., Treacy, D., and Wintersgill, M. The andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education 15* (2005), 147–204.
30. Smith, A., Butler, E., and Popović, Z. Quantifying over play: Constraining undesirable solutions in puzzle design. In *Procedings of the 8th International Conference on the Foundations of Digital Games* (2013).
31. Smith, G., Treanor, M., Whitehead, J., and Mateas, M. Rhythm-based level generation for 2d platformers. In *In procedings of the 4th International Conference on the Foundations of Digital Games* (2009).
32. Smith, G., Whitehead, J., Mateas, M., Treanor, M., March, J., and Cha, M. Launchpad: A rhythm-based level generator for 2-d platformers. *Computational Intelligence and AI in Games, IEEE Transactions on 3*, 1 (2011), 1–16.
33. Stern, M. K., and Woolf, B. P. Curriculum sequencing in a web-based tutor. In *Intelligent Tutoring Systems*, Springer (1998), 574–583.
34. Vygotsky, L. S. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, November 1980 / 1930.
35. Yannakakis, G. N., and Togelius, J. Experience-driven procedural content generation. *Affective Computing, IEEE Transactions on 2*, 3 (2011), 147–161.