Entropy as a Measure of Puzzle Difficulty

by

You Chen Eugene Chen

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Evaluating and ranking the difficulty and enjoyment of puzzles is important in game design. Typically, such rankings are constructed manually for each specific game, which can be time consuming, subject to designer bias, and requires extensive play testing. An approach to ranking that generalizes across multiple puzzle games is even more challenging because of their variation in factors like rules and goals. This thesis introduces two general approaches to compute puzzle entropy, and uses them to evaluate puzzles that players enjoy. The resulting uncertainty score is equivalent to the number of bits of data necessary to communicate the solution of a puzzle to a player of a given skill level. We apply our new approaches to puzzles from the 2016 game, *The Witness*. The computed entropy scores largely reproduce the order of a set of puzzles that introduce a new mechanic in the game. The scores are also positively correlated with the user ratings of user-created Witness puzzles, providing evidence that our approach captures notions of puzzle difficulty and enjoyment. Our approach is designed to exploit game-specific knowledge in a general way and thus can extended to provide automatic rankings or curricula in a variety of applications.

Preface

This thesis is an extension of a paper we published at the AAAI conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). This is a joint work with the paper co-authors Nathan Sturtevant and Adam White. Antonie Bodley helped to edit parts of the thesis, primarily in Chapters 1 and 2.

To my wife and son

For the endless patience and support they provide.

The first ~~bowl of soup~~ puzzle was too ~~cold~~ easy, the second was too ~~hot~~ difficult, but this one is just right.

– AI Goldilocks, AI Goldilocks and the Three B~~ears~~illion Puzzles .

Acknowledgements

This thesis, and the graduate journey leading up to its creation, was made possible with the help of many faculty members and students at the University of Alberta. My supervisors Nathan Sturtevant and Adam White ensured that the research and writing of the paper extended by this thesis met the standards of excellence for its acceptance and publication at AIIDE. They introduced me to the requirements of graduate, scientific research, and patiently provided space and guided me to completing this thesis as I juggled full-time work and other responsibilities outside my Masters program.

In addition, Adam helped to expand my learning in other ways. His willingness to thoroughly explain concepts in my first graduate class, specifically on Reinforcement Learning, instilled useful new concepts that were foreign to me after a decade of software development. He guided me through an individual study course that included my interests in visualizations and explainable AI, resulting the creation of mdp.ai [11], and provided me the opportunity to work on the Reinforcement Learning Specialization course on Coursera [17].

Nathan's questions and insights helped direct several research experiments and outcomes. His background in games, puzzles, and work in making his research more accessible by providing demos online [12] inspired me to do the same [18]. Several pivots took place especially in the early stages of this research, and without his recommendations and thoughtful discussions, I would have likely dived deeper into unproductive rabbit holes. Students from his Moving AI Lab provided constructive feedback at multiple venues and meetings, and helped me reconsider different approaches within the research and in their explanations. Last but not least, Nathan's ability to clearly and succinctly explain complex concepts, including ones presented in this thesis, is

something I will continue to aspire towards developing, and I appreciate his guidance in helping me achieve it.

Levis Lelis helped lead me to this research by offering the opportunity to work with him on the now-published research [10] that utilized puzzles from *The Witness* videogame. I had not heard of *The Witness* until he presented his research idea at one of Michael Bowling's research lab meetings. Together with other researchers, we created a user-study to test the research findings. I am also thankful to Levi for his suggestions in navigating graduate research, and for allowing me to audit his fascinating course on writing programs that write their own programs.

I want to thank Vadim Bulitko for being a part of my examining committee, and also for teaching the course that led a few of us students to explore how evolutionary AI could affect observational learning in A-life (artificial life) [25]. Separately, Matthew Guzdial introduced me to many concepts on procedural generation for games in his course, culminating in our published paper on turning images into game levels [4], and he also ran intriguing workshops that explored ideas of gaming that came from the research of workshop attendees. Carrie Demmans Epp was incredibly gracious in imparting her wisdom on the goals of graduate research and invalidating my incorrect beliefs about education research through the joint work we did with Levi.

My interactions with Martha White, whether it was auditing her course on Machine Learning or in conversations around developing the Courseara Reinforcement Learning course, have helped me better understand complex concepts because of her willingness to explore the thoughts I and others bring up. Throughout my program, Richard Sutton showed a steadfast dedication in empowering the new generation of students in both in his Reinforcement Learning course and supporting programs for graduate students, and I benefited from his willingness to share his knowledge.

Finally, for faculty, a special thanks to Denilson Barbosa, without whom I would not have even considered pursuing a Masters program. His willingness to help me with my early queries led me to pursue the program, and his advice early on helped direct my studies to topics that were new and interesting for

my background.

I have been helped by, and learned a lot from, many students throughout my program. Many are co-authors of work, others from different research labs that provided suggestions both on research and graduate life. There are too many to list, and I thank them all sincerely here. A few specific students left very positive impressions. Cody Rosevear spent many hours with me debating assignment approaches, textbook definitions, and then research content for joint work on A-life research, and I am thankful for his steady and objective lens when we worked together. Sheila Schoepp and I helped one another understand concepts through multiple courses, and has always been thoughtful and considered in her responses. Arash Pourzarabi and Pouneh Gorji were two of the kindest and smartest students I have ever met, offering suggestions and insights for assignments, in preparing for exams together, or in trying to organize social events for grad students. I will never forget how patient and kind Pouneh was in spending time individually with my young son, and then listening to him slowly read a story to her.

The staff at the Alberta Machine Institute of Intelligence (Amii) have helped enrich the experience of graduate students in computing science, including my own. I have benefited from their multiple offerings of classes and opportunities to learn important material. Additionally, both Nathan and Adam are Amii Directors, and the Amii network has helped expose me to further opportunities outside of the University of Alberta. I am grateful for the role Amii has and continues to play for students like me.

Part of my Masters program was done while working at Darkhorse Analytics, a consulting data analytics and visualization firm. Progress in the thesis would not have been possible without their support, and I am thankful for their willingness to let me discuss some of my learnings with the team. I specifically want to thank Nancy Ho for her willingness to listen to what can sometimes sound like incoherent explanations, and for her continued grace and excitement for my progress.

As noted earlier, without the support of my wife and son, none of this would be possible.

Finally, this work was supported by the National Science and Engineering Research Council of Canada Discovery Grant Program and the Canada CIFAR AI Chairs Program. I thank them for supporting this work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An approach to compute the difficulty of a puzzle and output a numerical score can be very useful for puzzle game designers. For example, puzzles can be procedurally generated and measured for difficulty, and the resulting difficulty scores can help designers select suitable puzzles to incorporate at different stages in their game. Such an approach can be faster than a designer's manual crafting of puzzles. Calculating difficulty is challenging for many reasons, especially for a general approach that can apply to different puzzle games, which is why existing measures to calculating difficulty are tailored to individual games. In our research, we derive two entropy-focused general measures to computing puzzle difficulty and find our best measure, called ReMUSE, is positively correlated with user ratings on a set of puzzles based on the videogame *The Witness*.

1.1 The Benefits of Scoring Puzzle Difficulty

An algorithm that computes puzzle difficulty can help designers more objectively gauge the difficulty of the puzzles they craft. It can help them explore different puzzle designs and determine which should belong to the various stages within their games. Such choices must be made because players may stop playing the game when faced with puzzles that are inappropriately difficult for them. For example, very difficult puzzles introduced to new players of a puzzle game can cause them enough frustration to stop playing. A difficulty score can help designers avoid such a situation by only picking puzzles that

have low difficulty scores for these beginners, and the score could be computed while a puzzle is being designed or edited through a editor.

Puzzle difficulty scores can also help speed up game development. Puzzles can be generated *procedurally*: that is, generated algorithmically instead of being crafted manually by a game designer. When these generated puzzles are be computed for difficulty, designers can match them to the different stages of their games instead of designing puzzles manually for difficulty. A matching method after procedural generation can be much faster than manual creating puzzles for appropriate game stages, especially as puzzles can get increasingly large and complex to design for.

The combination of procedural generation and a computable difficulty score can help designers create challenging puzzles that they would not otherwise have crafted. The building blocks for many puzzles can often be changed or rearranged in many ways, resulting in astronomical variations. If the intention is to create some of the most challenging puzzles of a given size, it is unlikely that designers would have crafted those puzzles without procedural generation and a difficulty scoring approach.

A difficulty score can help develop a *curriculum* of puzzles that require players to gain knowledge for further progression. This refers to providing a series of puzzles ordered by increasing difficulty, often introducing new mechanics or concepts players must learn. Take, for example, a game introducing Chess to new players and starts with only a few pieces that can be found on a chess board. As players progress through the stages of the Chess-introducing game, more pieces are introduced to the player, as well as the mechanics of how the pieces move and interact. The goal of such curriculum is to let players learn and feel more comfortable with the new material through experiencing easier puzzles early on. As player confidence and familiarity with these concepts increase, more difficult puzzles are gradually introduced. The success of a curriculum can be measured by a player's ability to solve more challenging puzzles after completing the curriculum, and a difficulty score can be used to order puzzles by difficulty for forming a curriculum.

Ordering puzzles by increasing difficulty scores can also increase player en-

joyment of a game by helping to induce states of positive optimal experience, or flow [6]. Flow occurs when people perform tasks that are slightly challenging compared to tasks that are too easy or too difficult, leading to boredom or frustration. The same concept applies to games and players, where overly challenging experiences can cause players to stop playing a game out of frustration, and overly easy ones can cause boredom. To achieve a state of flow, the ideal experience is to provide some challenge to players, but not too much.

An approach that accounts for players' different abilities to solve puzzles can help designers craft player-specific experiences of difficulty. For example, a puzzle could be considered difficult by a new player who has yet to acquire the knowledge or experience necessary to easily solve the puzzle. An experienced player might consider the same puzzle trivial to solve. If puzzles can be scored and sorted for difficulty based on the profile of a player's skill, then designers can pick simpler ones that are slightly challenging for the earlier stages of a game, followed by increasingly challenging ones as players gain more experience.

When the computation of puzzle difficulty scores can be done quickly, it becomes possible for a game to automatically create puzzles that are tailored to a player's skill level if the creation is also quick. For example, an expert player could enter a game and select an option to be presented with difficult puzzles, and the game would dynamically generate them. Game designers would not have to manually craft puzzles for such games. They could instead focus on other elements of the game, like tweaking mechanics that can vary puzzle difficulty.

Quick procedural generation and a method to quickly compute puzzle difficulty scores make it possible for a game to actively adapt to any player's variable ability to solve puzzles. Let us assume that the longer it takes a player to solve a puzzle, the more difficult it is for that player. Making another assumption that a game designer aims to have their players achieve a state of flow when playing their game, the game could dynamically adjust the difficulty of each subsequent puzzle presented to the player. If a puzzle is solved too quickly, the next generated one will be more difficult. Conversely,

if a puzzle takes too long to solve, a simpler one is presented next.

Such an adaptive system can also help to determine if players have the requisite knowledge to solve puzzles with specific mechanics. If a player always takes longer to solve puzzles involving a specific mechanic, the system could provide simpler variants. This would provide more opportunities for the player to learn the mechanics through exposure to its most basic elements. Players gain both confidence and experience by solving these simpler puzzles and can progress to more challenging versions afterwards.

A general measure that can compute puzzle difficulty scores across different puzzle games is beneficial in several ways. First, existing measures to scoring difficulty are crafted for the game they have been designed for because puzzle games vary in their rules and goals. A standard measure can derive a difficulty score without having to craft and test game-specific methods to arrive at such a score. This can save a substantial amount of effort. Second, a general measure can compare puzzle difficulty across different games. Such a measure can lead to several helpful analyses, like comparing how difficulty changes during the various stages of different games. Finally, a standard measure to scoring difficulty can help game designers craft and test mechanics. As designers create their games, they design puzzles with different mechanics. These mechanics can impact difficulty scores, and designers can decide to add, remove, or modify these mechanics due to their impact on difficulty.

1.2 Relating Puzzle Difficulty Scoring to Education and Skill Acquisition

A general measure to scoring difficulty in puzzle games can potentially be modified to work in other domains, from non-puzzle games to the teaching of subjects like Math in schools. A puzzle game can be considered as having a primary goal of entertaining players by providing challenging puzzles that they can solve with their skills. These skills are obtained by learning and experiencing new concepts and methods that are necessary to solve puzzles. This is particularly true for games that introduce additional mechanics in their

puzzles as part of player progression. Thus, to entertain players with a series of challenging puzzles, the game must find ways to teach players concepts to address the additional mechanics meant to make the puzzles more challenging. The introduction of these additional mechanics to keep players challenged can induce a state of flow.

A primary goal of traditional education in school is for students to attain the skills and knowledge taught to them. The combination of education and entertainment is *edutainment*, which is meant to help students attain knowledge and skills by keeping them engaged through entertainment. Many edutainment video games are created to facilitate student learning, and a general difficulty scoring measure can potentially be applied to identifying the difficulty of concepts being taught. It can even determine the difficulty of certain concepts for individual students or players that vary in how much they have understood in such edutainment games.

Learning and entertainment exist in both edutainment and puzzle games, with puzzle games primarily focused on entertainment and having learning be the means to the end of achieving that entertainment. The opposite can be considered true for edutainment games. They can be very similar, however. The ability to help players achieve states of flow in puzzle games through puzzle difficulty scoring can be realized for students by providing the right level of challenge of questions or material they are presented with. The stream of slightly challenging puzzles can keep students engaged, and offers more opportunities for students to learn new skills and knowledge.

1.3 The Challenges of Scoring Difficulty for Different Puzzle Games

It is hard to determine the difficulty of puzzles with a general measure that can apply to different games. One reason is that puzzle games can vary greatly in their rules and goals. For example, the game-specific rules for playing Minesweeper, Sudoku, and Sokoban puzzles are unique enough that they cannot be applied to help solve one of the other games. In Minesweeper, players

reveal non-bomb tiles, while in Sudoku, they fill up blank spots of a 9x9 grid with numbers 1 through 9, ensuring that all rows, columns and the 9 non-crossing 3×3 sections of the larger grid contain only single instances of each number. In Sokoban, players control an avatar to push boxes either horizontally or vertically to designated box locations. While all three games are examples of single-player, turn-based puzzles, it is challenging to provide an approach to scoring difficulty that can work across all three games.

Difficulty is also different for players of different knowledge and skill levels, further complicating measurement. However, such a measure can be very useful. It can be used to predict whether puzzles will be interesting to players of different skill levels, identify specific skills that render puzzles as easy or challenging, and help explain or generate the ordering of puzzles within puzzle games.

1.4 Existing Puzzle Scoring Approaches

Specific algorithms have been proposed to measure puzzle difficulty for some games. Sudoku puzzles have been measured for difficulty by mapping them into sets of constraint satisfaction problems [7]. In Sokoban difficulty has been measured by the minimum number of steps needed to decompose a puzzle into subproblems [8]. These algorithms are tied to the puzzle domains they are created to measure. We aim to design general algorithms that apply across multiple puzzle games.

Generative approaches need a way to select intriguing puzzles, and our work is related to evaluating puzzles for some desired property, such as difficulty. Grammatical evolution can be used to create levels for a clone of a puzzle game *Cut the Rope*, measuring generated levels for playability and variety [19]. Incremental Exhaustive Content Generation (EPCG) [23] can be used to produce minor variations in the design of *Snakebird* game levels to significantly increase the length of the shortest possible solution [24]. In *The Witness*, neural-guided tree search has been used to determine a set of ordered puzzles [10] that compares favourably with ordered puzzles from the game in teaching

players to solve additional test puzzles. In each of these works, problem-specific scores are designed to measure and select the most interesting content.

Difficulty in puzzles has been measured using different approaches. Search and strategic depth scores have been suggested as reasonable indicators of puzzle difficulty [15]. Others have proposed constraint satisfaction solvers as part of a deductive search approach to measure difficulty [2].

Finally, a number of simpler puzzle measures can be generally extracted and used as proxies for difficulty. For example, the number of solutions to a puzzle provides a natural measure of difficulty because more solutions implies easier puzzles. Shorter solution lengths also relate to difficulty for similar reasons. However, these measures are not sufficient for capturing difficulty in puzzles, as we will later show.

1.5 Our Entropy-based Approach

In this thesis, we contribute a new measure of puzzle difficulty based on entropy. Applied to single-player, turned-based puzzles, our entropy measure quantifies the communication that would be needed to describe a puzzle's solution, given player knowledge of the puzzle domain. Entropy is calculated from the number of choices a player faces at each step in the solution path. Using these notions, we derive two information entropy measures and then evaluate them on several puzzle test sets

Empirically, we find that our best approach, ReMUSE, is able to measure puzzle difficulty effectively, according to two experiment outcomes. First, in *The Witness*, it orders a set of mechanic-introducing puzzles very similarly to the original puzzle designers. Second, we find a positive correlation between the ReMUSE scores and user ratings for a series of online puzzles frequented by puzzle enthusiasts, indicating a relation to puzzle enjoyment and separately by difficulty. The correlation with user ratings for ReMUSE are higher than the correlation with other puzzle measures such as average solution length and a puzzle's number of solutions.

Chapter 2

Background

We design two algorithms, MUSE and ReMUSE, which take as input a single-player, turn-based puzzle and return a scalar score for the entropy of a puzzle. In the upcoming subsections, we describe how we represent puzzles as graphs, utilizing simple puzzles from the game *The Witness* to illustrate essential concepts. We then describe information entropy, KL divergence, and softmin algorithms, which are utilized in MUSE and ReMUSE.

2.1 Puzzle Representation as a Graph

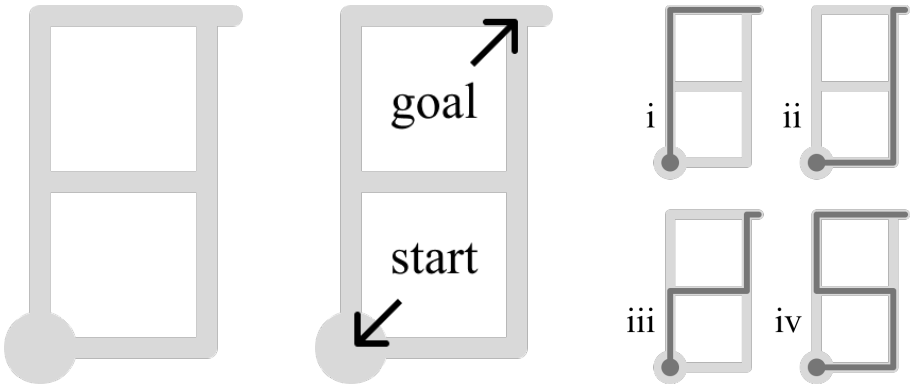
A single-player, turn-based puzzle can be defined as a directed graph $G = (V, E)$. V is a set of k vertices, also interchangeably called states s, where $V = \{s_1, ..., s_k\}$, and states can be reached through edges $E = \{e_1, e_2, ...\}$. Each directed edge can be represented as a pair of states originating from state s_i to s_j, or $e_m = \{s_i, s_j\}$. Beginning with the action of picking an initial starting state s_{start} from a set of starting states S_{start}, a solution to a puzzle is an edge-connected path from a start state to a goal state.

Edges E of the puzzle are not explicitly enumerated: instead, they are calculated with an action function $A(s)$ that returns a set of edges, interchangeable with actions a, for a given state s. A puzzle also needs a goal test function $\Gamma(s)$ that returns 1 if state s is a goal state and 0 if not.

Considered together, a puzzle can be defined by a graph, a set of start states, an action function, and a goal test function. This is represented as $puzzle = \{G, S_{start}, A, \Gamma\}$. This representation is used as the input to compute

our MUSE and ReMUSE scores.

2.2 The Witness Domain

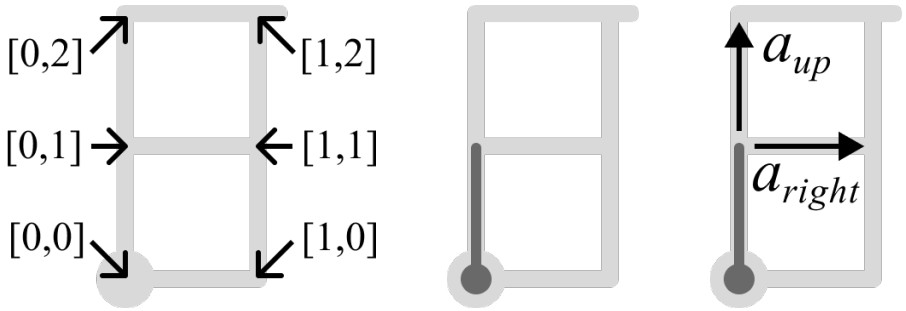


(a) Puzzle (b) Start & Goal (c) Solutions, i-iv

Figure 2.1: Example witness puzzle, annotations on the start and end goal junctions, and the puzzle solutions. Early Witness puzzles, like the one shown in a), are rectangular grids that have start junctions, indicated by grey circles, and end goal junctions, indicated by notches. Solving each puzzle minimally requires the drawing of a path along the grey grid line from a start junction to an end one. All four solutions of the puzzle in a) are shown in c).

The Witness is a 2016 single-player video game where players can explore an island filled with different biomes of themed, solvable puzzles. Early puzzles are presented as $[w \times h]$ sized rectangular grids, such as the $[1 \times 2]$ sized puzzle in Figure 2.1a. Solving any Witness puzzle will, at minimum, require a non-crossing path to be drawn along the light-grey grid from special start junctions, indicated by round circles, to any goal junctions, indicated by notches. The start and goal junctions for our puzzle are displayed in Figure 2.1b, and all four solutions for the puzzle are shown in Figure 2.1c. Note that when a non-crossing path is drawn and reaches the end goal junction, the notch indicating the end goal junction is automatically filled as part of the drawn path.

Formally, the state of a Witness puzzle is the path drawn during play, which can be represented as a sequence of junction positions. For example, junction positions for the Witness puzzle shown throughout Figure 2.1 and 2.2 are listed in Figure 2.2a. State s from Figure 2.2b is $\{[0, 0], [0, 1]\}$, starting with

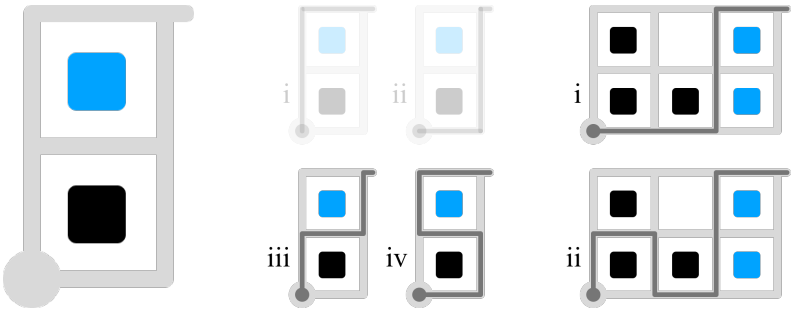


(a) Junctions and their positions (b) Sample state s (c) Actions from $A(s)$

Figure 2.2: Witness puzzle states are the history of $[x, y]$ positions taken to reach different junctions within the grid. These junctions are denoted in a). A sample state s shown in b) is $\{[0, 0], [0, 1]\}$, which includes the starting position at $[0, 0]$. $A(s)$ returns valid cardinal actions that can be taken to further draw the line to a next state, and as shown in c), $A(s)$ for that state would return $\{a_{up}, a_{down}\}$.

the only start junction in this puzzle, with $S_{start} = \{[0, 0]\}$. Graph edges or actions in Witness puzzles represent cardinal (up, down, left, right) directions used to draw a path to a next state, so action edge-detection function $A(s)$ returns $\{a_{up}, a_{right}\}$ for state s, as shown in Figure 2.2c. The next state will return $\{[0, 0], [0, 1], [0, 2]\}$ if a_{up} is taken at s, or $\{[0, 0], [0, 1], [1, 1]\}$ if a_{right} is taken instead. Goal test function $\Gamma(s)$ returns 0 for state s in Figure 2.2b, and 1 for each state shown in Figure 2.1c.

Unlike the puzzle in Figure 2.1a, most Witness puzzles also contain **constraints** that must be fulfilled when a non-crossing path is drawn from a start to goal junction. The puzzle in Figure 2.3a contains such a constraint: different coloured squares within the grid are associated with constraints that require that squares of only one colour exist within the regions separated by a start-to-goal non-crossing path. As a result $\Gamma(s)$ returns 1 only for states shown in iii) and iv) in Figure 2.3b since the different coloured squares are separated into two regions with the path. $\Gamma(s)$ returns 0 for states shown in i) and ii) of Figure 2.3b. Both states depicted in Figure 2.3c return 1 through $\Gamma(s)$. For i) in Figure 2.3c, the path separates two regions of coloured squares (3 black squares left, 2 blue squares right), while three regions (containing 1



(a) Puzzle w constraints

(b) Solutions, iii & iv

(c) 3x2 puzzle & solutions

Figure 2.3: Witness puzzles often contain constraints, and the puzzles shown here include different colour square constraints. In a), two squares of different colours reside between the gridlines, and solutions must satisfy their constraints. Squares of different colours must be separated by the drawn path, so only iii) and iv) in b) are considered solutions. Two solutions for a different puzzle are further demonstrated in c).

black, 2 black, and 2 blue squares) also separate squares of different colours in ii).

2.3 Information Entropy

In information theory, entropy is a measure of the uncertainty of a random variable Z, where Z has k possible outcomes $\{z_1, ..., z_k\}$ [20]. If the probability of an outcome z is given by a probability function $P(z)$, the **information entropy** is defined as:

$$H(Z) \doteq \sum_{n=1}^{k} P(z_n) \log_2 \frac{1}{P(z_n)} \tag{2.1}$$

For example, for an unfair double-sided coin that always turns up heads there is no uncertainty in the outcome. Mathematically, given the random variable $Z_{\text{unfair}} = \{z_{\text{heads}}, z_{\text{tails}}\}$, the probability $q = P(z_{\text{heads}}) = 1$ and $P(z_{\text{tails}}) = 0$, and so its entropy is $H(Z_{\text{unfair}}) = 1 \log_2 \frac{1}{1} + 0 \log_2 \frac{1}{0} = 0$ bits.

With a fair coin, where there is an equal chance that heads or tails is the outcome of the flipped coin, the probabilities $P(z_{\text{heads}}) = 0.5$ and $P(z_{\text{tails}}) = 0.5$ result in the entropy of the outcomes being $H(Z_{\text{fair}}) = 0.5 \log_2 \frac{1}{0.5} +$

$0.5 \log_2 \frac{1}{0.5} = 1$ bit. In other words, it would take 1 bit of information to communicate the outcomes of a flipped, fair coin.

2.3.1 KL Divergence

Kullback–Leibler (KL) Divergence [9], often used in machine learning models to calculate loss [16, 21], is a measure that calculates the difference between two probability distributions. We use the **relative entropy** interpretation of KL divergence, or the resulting uncertainty from using a probability distribution Q to represent a true probability distribution P:

$$D_{KL}(P||Q) \doteq \sum_{x \in X} P(x) log_2 \frac{P(x)}{Q(x)} \tag{2.2}$$

Keeping with our example of a fair coin where the $P(Z_{fair}) = \{P(z_{\text{heads}}), P(z_{\text{tails}})\} = \{0.5, 0.5\}$, the KL divergence from using a probability distribution Q that estimates heads turning up 90% of the time is $D_{KL}(P||Q) = 0.5 \log_2 \frac{0.5}{0.9} + 0.5 \log_2 \frac{0.5}{0.1} = 0.74$. If Q had the same probability distribution as P, then the KL divergence would work out to be 0, reflecting no uncertainty from using Q to represent P.

2.3.2 Softmin Function

The softmax function [1] is widely used to normalize vectors of real numbers into a probability distribution on outcomes. It is, for instance, often used as the last activation function of a neural network to turn network outputs into a probability distribution for a set of output classes, and is defined as:

$$\text{Softmax}(z_i) \doteq \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, 2, \ldots, K \tag{2.3}$$

In our work we want the highest value in the incoming distribution to have the lowest probability, so we use the similar softmin function, which uses $-z_i$ in place of z_i in the calculation.

Chapter 3

Encoding Puzzle Uncertainty as Entropy

MUSE and ReMUSE measure the uncertainty of a puzzle, specifically encoding the uncertainty encountered in solving a puzzle. This can also be viewed as the amount of information that an *oracle*, who knows the solution to a puzzle, needs to communicate to a player so that they can make the correct decision at each step when solving a puzzle.

We utilize information entropy to encode the uncertainty encountered at each state, similar to how uncertainty is encoded for the outcomes of coins or dice. We use the number of legal actions of a puzzle at a given state s, $|A(s)|$, to determine the entropy encountered at that state.

In the Witness there are at most four cardinal actions a per state s and thus the amount of *uniform* entropy is simply:

$$H(Z_{A(s)}) = \begin{cases} 0, & \text{if } |A(s)| = 1 \\ 1, & \text{if } |A(s)| = 2 \\ 1.6, & \text{if } |A(s)| = 3 \\ 2, & \text{if } |A(s)| = 4 \\ \infty, & \text{if } |A(s)| = 0 \end{cases} \tag{3.1}$$

There are two important special cases to consider. If only one legal action is possible, then there is no uncertainty regarding which action must be taken, so entropy is 0. If, however, there are no possible actions, or $|A(s)| = 0$, then no amount of bits can encode for the solution path from state s since there are no future outcomes or successor states, and we assign ∞ for the entropy

of such states.

3.1 Minimum Uniform Solution Entropy (MUSE)

Our first measure, called Minimum Uniform Solution Entropy (MUSE), computes a score for a single-player, turn-based puzzle that reflects the uncertainty encountered when the puzzle is solved. More specifically, it computes the entropy of the solution with the smallest entropy. It achieves this through a simple recursive function depicted in lines **??**-21 of Algorithm 1. The entropy of any state s is the sum of the *local* entropy resulting from the number of actions $|A(s)|$ and the entropy belonging to the child state with the smallest entropy. The algorithm returns the entropy of the solution with the smallest entropy, which is particularly relevant if there is more than one start state, or $|S_{\text{start}}| > 1$.

Computed recursively, the base cases (lines 12-13) handle if the goal is reached (returning 0 entropy) or if no actions are available (returning ∞ entropy). The general case recursively computes the entropy of each child (lines 14-18) and then returns the local entropy of the current state plus the minimum entropy of the children (line 20). Note that we will later generalize the action function passed in, so σ is the action function in this pseudo-code.

Table 3.1 provides a concrete example of applying Algorithm 1 to a puzzle from the Witness. State 1 from the table is also the puzzle's start state $s_{start} = \{[0, 0]\}$, which has actions a_{up} and a_{right} returned by the action function $A(s)$. These lead to States 2a and 3a, or $\{[0, 0], [0, 1]\}$ and $\{[0, 0], [1, 0]\}$ respectively, since states in Witness puzzles are the non-crossing paths that can be represented by a series of junction positions.

MUSE computes the sum of the local entropy encountered at State 1 plus the minimum sum of entropy from the next state with the lowest entropy. This would be $H(Z_{|A(s)|}) = 1$ bit of entropy for the $|A(s)| = 2$ actions at State 1 plus the lower of the two entropies from State 2a and 3a, which themselves would need to be similarly calculated in a recursive fashion.

Algorithm 1 MUSE: Minimum Uniform Solution Entropy

```
// Note: σ is either A or L, depending on approach.
// If L, then all σ(s) are replaced with L(s).
// Additionally, if an n-step lookahead is used, L(s, n) and Γ(s, n) applies
instead of L(s) and Γ(s).

function MUSE(S_start, σ, n)
    uses ← [ ]
    for each s_start ∈ S_start do                    ▷ For every start state
        Append MinPathEnt(s_start, σ, n) to uses
    end for
    return min(uses)
end function

function MinPathEnt(s, σ, n)
    if Γ(s) = 1 then return 0                                  ▷ Solved
    if |σ(s)| = 0 then return ∞                                ▷ Unsolvable
    childEnts ← [ ]
    for each a ∈ σ(s) do                                ▷ For every action
        s' ← Apply a to s
        Append MinPathEnt(s', σ, n) to childEnts
    end for
    localEnt ← H(Z_|σ(s)|)
    return min(childEnts) +localEnt
end function
```

We can easily calculate the entropy at State 2a and 3a through three observations. Only two solutions exist for the puzzle, and as a result all other states not shown at Table 3.1 will have an entropy of ∞, since all other states eventually end up at a terminal, non-goal state where $|A(s)| = 0$. Only goal states of 2d and 3h would return 0 entropy to parent states, and because the minimum child entropy is picked at each state, all other child entropy resulting in ∞ would be ignored. Because States 2a and 3a do not branch into multiple solutions, we sum child entropy together. Thus the entropy at State 2a is its local entropy plus the entropy of its descendant states, or $1+1.6+1 = 3.6$ bits of entropy. Similarly, the entropy at State 3a is $1+0+1+1+0+0+0 = 3$ bits of entropy.

Together, the MUSE for the puzzle is $1 + \min(3.6, 3) = 4$ bits of entropy, from reaching the goal state at 3h, or $s_{\text{goal}} = \{[0,0],[1,0],...,[1,2],[2,2]\}$.

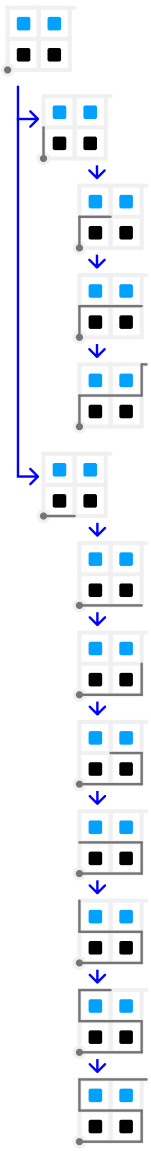
ID	State	Position	$\|A\|$	$H(Z_{\|A\|})$	$\|L\|$	$H(Z_{\|L\|})$
1		[0,0]	2	1	2	1
2a		[0,1]	2	1	1	0
2b		[1,1]	3	1.6	1	0
2c		[2,1]	2	1	1	0
2d		[2,2]	0	0	0	0
3a		[1,0]	2	1	2	1
3b		[2,0]	1	0	1	0
3c		[2,1]	2	1	1	0
3d		[1,1]	2	1	1	0
3e		[0,1]	1	0	1	0
3f		[0,2]	1	0	1	0
3g		[1,2]	1	0	1	0
3h		[2,2]	0	0	0	0

Table 3.1: Entropy encountered at each state for puzzle solution reached at states 2d and 3h using valid action function return $A(s)$ and logic action function return $L(s)$.

MUSE is a measure that computes a score for a single-player, turn-based game that reflects the uncertainty encountered while a puzzle is solved. The uncertainty is encountered by human players, but so far we have treated any action returned by $A(s)$ as a source of uncertainty, when players with skill, knowledge, and experience might not see all actions as equally valid in achieving a solution. We can improve our approach by incorporating player knowl-

edge and skills within MUSE, and to do that, we introduce logic-driven inference rules.

3.2 Logic Puzzles and Inference Rules

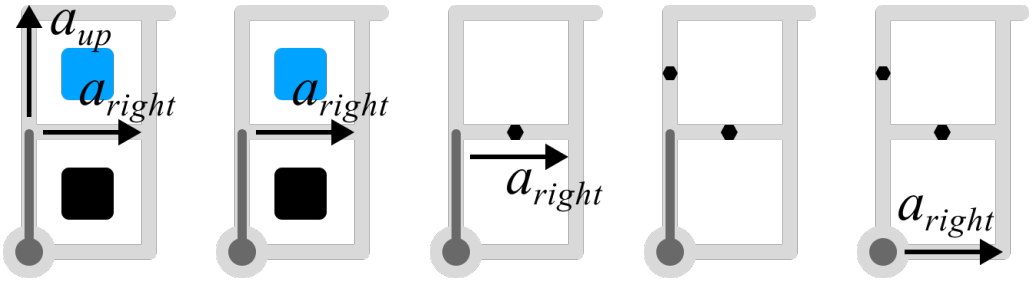
In logic puzzle games, players often deduce sets of rules that can simplify or solve a puzzle. For example, with typical rectangular jigsaw puzzles, players can intuit that puzzle pieces containing two straight edges are its corners pieces, and those corner pieces must be adjacently connected with other pieces that have a straight edge. Accurate deductions can be translated into **inference rules**, or a set of rules that should be followed for the puzzle to remain solvable.

Inference rules have been extensively created to explain and teach methods of solving puzzles in logic games like Sudoku, with further research used to rank the difficulty in self-learning and applying these rules for puzzle-solving [5]. While these rules can be created automatically [14, 22], in this work we manually created sets of inference rules for Witness puzzles to generate specific entropy-related scores based on the uncertainty found in solving a puzzle.

3.3 Inference Rules for Witness Puzzles

Inference rules exist for different constraints found in Witness puzzles. For instance, while it is possible to extend the path either up or right for the next part of the path for the puzzle in Figure 3.1a, we can apply an inference rule that eliminates a_{up} as a potential action to take. The rule applies as follows: at a junction adjacent to two different colour squares, the action extending the path to immediately divide these squares **must** be taken. Not doing so would result in the puzzle not being solvable since the pieces have to be in separate regions, and if we do not separate them immediately, there will be no way to do so later.

This inference rule can be formalized as follows. We introduce a logic function $L(s)$ that incorporates the inference rule for immediately separating



(a) A(s) actions (b) L(s) action (c) L(s) action (d) L(s) actions (e) L(s,n) with n=1

Figure 3.1: $A(s)$ returns more actions than logic infused $L(s)$. Inference rules used in $L(s)$ for the same puzzle in both a) and b) does not include a_{up} that $A(s)$ does, since taking a_{up} would not allow the constraint to later be resolved. In c) we see how $L(s)$ similarly determines a_{right} to be the only action to take to possibly solve the puzzle due to the must-cross constraint. In d), $L(s)$ returns no actions since it is impossible to fulfill both must-cross constraints that must be drawn to from that state, while in e), using a 1 step lookahead from the start state at $\{[0,0]\}$ only returns a_{right} since the state reached in d) would result in no further actions to solve the puzzle.

different colour squares. Consider the puzzle shown in Figure 3.1a with $A(s) = \{a_{\text{up}}, a_{\text{right}}\}$. In state $s = \{[0,0],[0,1]\}$ at Figure 3.1b, $L(s)$ would return a_{right} as the only action that could lead to the goal state, or $\Gamma(s) = 1$ for a future state s.

Another inference rule can be written for *must-cross* constraints. These constraints are black hexagons (which look like black dots in our smaller diagrams) along the grid lines of Witness puzzles: an example is shown in Figure 3.1c. When must-cross constraints are present, they must be crossed over by a path for the constraints to be fulfilled. The inference rule for must-cross constraints is simple: at a junction where a must-cross constraint lies between two junctions, the action extending the path to cross that must-cross constraint must be immediately taken. As a result, $L(s)$ returns only a_{right} for the puzzle state shown in Figure 3.1c.

$L(s)$ can return no actions if a combination of inference rules suggest no future states beyond s can lead to a goal state, even though $A(s)$ would return a set of actions. This is the case of the puzzle in Figure 3.1d, where inference rules require both a_{up} and a_{right} to reach a possible goal state. Since that is

not possible from s, $L(s)$ returns no actions.

Additional inference rules for Witness puzzles were developed to identify the actions that would be returned for our logic function $L(s)$ at a given state s. We have included all inference rules in Appendix A for reference, since our work is supported by these puzzle-specific rules..

Modeling player lookahead

Uncertainty can be further reduced if we account for a player's ability to consider the consequences of actions a few steps ahead in turn-based games, as good chess players typically do [3]. In our case, we extend $L(s)$ to $L(s, n)$ to return a set of logic-driven actions at s that check if the puzzle is still solvable after a n-step depth first search (DFS), or is solved in the DFS, suggesting 0 entropy. Having already established that the puzzle state from Figure 3.1d cannot result in reaching future goal states, $L(s, 1)$ for the start state shown in Figure 3.1e will only return a_{right}.

3.4 MUSE with a modified action function

We augment MUSE with inference rules by simply replacing valid action function A with logic action function L for σ as shown in Algorithm 1. In some cases $A(s)$ and $L(s)$ are exactly the same: in the example puzzle shown in Table 3.1 on page 16, no local inference rules apply to L at State 1, so $|A(s)| = |L(s)| = 1$. However, in many other states like State 2a or 2b, the inference rule pertaining to coloured squares makes a_{right} the only logical action that can be taken, $L(s) = \{a_{\text{right}}\}$, and the local entropy at that state, not accounting for child entropies, is 0.

As a result, the uncertainty encountered in solving the puzzle, or the puzzle MUSE score, is reduced. This can be seen for the solutions reached at States 2d and 3h in Table 3.1 on page 16. Similar to the earlier derivation of the MUSE score, we can first sum up their local and descendant entropies: for State 2a this is $0 + 0 + 0 = 0$ and for State 3a it is $1 + 0 + 0 + 0 + 0 = 1$. Therefore, the puzzle's MUSE score is the local entropy at State 1 plus its

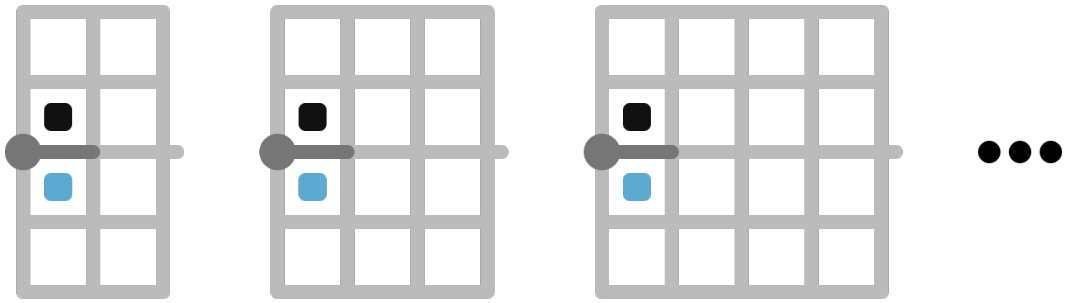


Figure 3.2: Witness puzzles can increase in grid size, but not necessarily in complexity or difficulty. The puzzles shown above indicate examples of a puzzle that is effectively solved from the state of $[0, 2], [1, 2]$, since there is no further complexity from that state no matter how much larger the grid increases in size from the x axis. MUSE scores would continue to increase in these larger puzzles, which would not make sense.

smallest child entropy, or MUSE $= 1 + \min(0, 1) = 1$, which is smaller than the entropy of 4 computed using only the valid action function A.

3.4.1 ReMUSE: Relative MUSE

MUSE with a modified logical action function, which we will simply refer to as our general MUSE algorithm from here, is an improvement by more accurately encoding the perceived uncertainty of a puzzle by a human player. This uncertainty is measured locally and uniformly at each state, but it likely does not reflect the uncertainty that would be experienced from future states. To account for this, we introduce our ReMUSE approach.

Several cases need to be accounted for in the design of our ReMUSE algorithm. Consider a state s where three actions $|L(s)| = 3$ exist, but the eventual outcome of all three actions is that the puzzle will be solved. MUSE returns a local entropy score of 1.6 for state s despite the intuition that in such a state, no further information needs to be encoded or communicated to the player to solve the puzzle. The entropy should be 0 in such a state. A similar situation occurs if all children have the same entropy: the choice again does not matter, and no information needs to be communicated to describe the solution. Finally, consider if the three actions have only an ϵ difference in entropy: even if a player takes the wrong action, the entropy of the rest of the

puzzle is essentially unchanged.

Expanding slightly on the first motivation, where the eventual outcome of actions of a state result in a solved puzzle, our example in Figure 3.2 shows variations of a puzzle at such a state. The puzzles can increase in horizontal size to the right, and MUSE would suggest an ever-increasing complexity of the puzzle as the grid expands. This is despite the fact that the expansions do not contain any constraints and thus no increase in difficulty to the puzzle, and all future actions will result in solving the puzzle. This assumes inference rules that players would use to not draw paths that loop themselves into dead ends where all actions would lead to previously visited junctions. An algorithm that encodes difficulty should model the human intuition that the increases in empty grid space of Witness puzzles where the constraints are already solved would result in no or little additional difficulty. MUSE would instead output an exponential growth of entropy.

ReMUSE utilizes KL Divergence to account for the *relative entropy* between two probability distributions when computing the local entropy of a given state. The first distribution is a uniform probability distribution based on the number of actions. The second distribution is the softmin of the entropy from the immediate children. The larger the KL divergence measure, the larger the divergence from the assumption of uniform probability. [1]

This approach only differs from the MUSE algorithm shown in Algorithm 1 by a single line (line 19), highlighted in Algorithm 2. Instead of setting a state's local entropy as the uniform information entropy of the state, we replace it with a KL divergence comparison between the softmin of the entropy of the children and the uniform distribution.

Algorithm 2 ReMUSE: Relative MUSE

...

19: $immedEnt \leftarrow$
$\quad\quad \mathrm{KL}(\mathrm{softmin}(childEnts), (Z_{|L(s)|}))$

...

[1] We use a uniform prior distribution to model the probability of player actions. If a player has a different prior distribution or policy, we can use that instead.

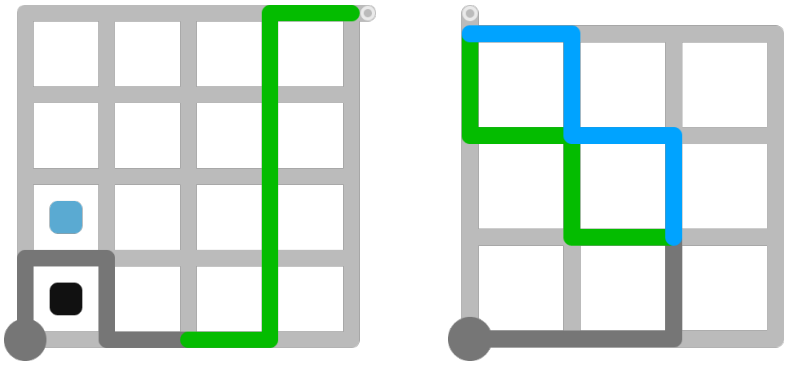
3.4.2 Other Puzzle Measures

Single player, turn-based puzzles from different puzzle games share a set of general measures with which we can compare our entropy-related scores from MUSE and ReMUSE. The number of solutions and the average length of solutions provide reasonable measures of puzzle difficulty. The number of solutions can be determined by the number of states that return $\Gamma(s, 0) = 1$ after a brute-force traversal of every path. The average length of solutions takes the average number of actions needed to arrive at a solution for the puzzle. We expect that for puzzles of a given $[w \times h]$ size, the more difficult puzzles are ones with fewer number of solutions and larger average length of solutions.

3.5 Logic-Driven MUSE with Straight Line Policy

The inference rules used in our logic function $L(s)$ consist of local constraint checks, whereas human players often create their own non-local (global) inference rules that can greatly reduce the uncertainty experienced in a puzzle. As we will discuss later in the section on the limitations of our approach, creating code-equivalents of global inference rules that human players naturally and easily adopt can be difficult. Some of these inference rules may be simple enough to implement, which is what we do by augmenting MUSE with a straight line policy.

The intuition for the straight line policy arises from the quick visual check players can perform in solving Witness puzzles: if a straight-line path from the current state is drawn to the end vertex, is the puzzle solved? This check can be performed quickly and with no regard to existing constraints from the puzzle, unless the constraints block such a path from being drawn. This quick check is akin to a low effort short-cut: if a straight line appears to solve the puzzle, there is no further need to consider other constraints within the puzzle, and no further search is needed. In practice, the approach can make puzzle solving much simpler, and is straightforward to implement.



(a) Straight-line global lookahead in green

(b) Two straight-line global lookaheads: green and blue

Figure 3.3: Non-local straight-line policies reflect the behaviour that human players can (and likely do) plan by drawing a straight line from the current state to the end junction. Example lines are demonstrated in a) and b), where in b) two paths are included from that state.

Implementing a straight line policy check requires three key considerations to our MUSE algorithm. First, instead of local iterative searches for next actions as implemented in lines 15-18 in Algorithm 1 (page 15, this check effectively generates paths, or chains of directional actions from a given state, to an end vertex, and then checks if such paths solve the puzzle. This results in a *global lookahead* in the direction of the end vertex. Second, we have to determine what a straight line looks like in the context of drawing paths for Witness puzzles. As shown by the examples in Figure 3.3, there are different ways of drawing one or more paths of what can be considered a straight line from an existing state to the end vertex. We could explain the approach of deducing these lines algorithmically, but referring back to our earlier claim that certain global inference rules are intuitive for humans to grasp and can be challenging to reproduce and explain, we hope that the visual examples in the figure provide the intuition of these generated lines.

Algorithm 3 MUSE with Straight Line Policy

...

12: **if** $\Gamma(s) = 1$ **or** $\pi(s) = 1$ **then return** 0 ▷ Solved

...

Most critically, even though the approach of utilizing policies to model

global inference rules should apply to other puzzle games, we acknowledge that this straight line policy is dependent on the rules of Witness puzzles. This general approach can be applied to the MUSE algorithm by replacing a single line, highlighted by line 12 in Algorithm 3, where we introduce a general function $\pi(s)$ that checks one or more policies to see if the puzzle can be solved. If so, like in the case where the puzzle is solved at the current state with $\Gamma(s) = 1$, no entropy is determined to exist at the current state, and no further iterative searches from the actions of the current state need to be done. In the case of our implementation with respect to Witness puzzles, only a straight line policy is implemented and checked.

For further consideration, both puzzles shown in Figure 3.3 would have resulted in a MUSE score of 0 with the straight line policy implemented by changing line 12 noted in Algorithm 3 for MUSE. This means that the inference rule would be checked from the start junction at $[0, 0]$. In the case of the puzzle in Figure 3.3a, a similar staircase-shaped line shown by the two straight line paths in Figure 3.3b, except drawn from the starting bottom left vertex to the top right, instead of drawing towards the top left corner. The path beginning with moving up and right, then repeating until the end vertex is reach, would separate the two different colour squares on the bottom left corner, resulting in fulfilling their constraints and leading to a solved puzzle. For the puzzle in Figure 3.3b, a straight vertical line from the bottom left to the top left corner of the puzzle would be drawn, solving the puzzle.

Chapter 4

Experiment Setup and Results

We setup our experiments to answer a number of questions. First, can our proposed puzzle scores predict puzzle difficulty on user-submitted puzzles from a site frequented by puzzle enthusiasts? How do MUSE and ReMUSE compare with measures like solution length in capturing puzzle difficulty? We also compare our ReMUSE-based orderings of puzzles to those found within the Witness game, as well as the neural-ordered equidistant puzzles from prior work on puzzle ordering [10].

4.1 Interesting Puzzles from the Windmill

We downloaded every single puzzle from the Windmill up to November 29th, 2022, and then filtered puzzles by a number of criteria. First, we only computed puzzle scores for puzzles containing one or a combination of must-cross, different colour squares, and cannot-cross (briefly described in the final paragraph of this section) constraints, since these were the constraints our puzzle solver could account for, and ones we had inference rules written into our code. We also filtered specifically for puzzles with a 4×4 size. This was done for a number of reasons: a fixed grid size allows for a consistent assessment and evaluation of puzzles that have similar number of states, thereby removing the element of variable scoring due to different puzzle sizes. This grid size also limits a puzzle's maximum entropy, as we believe that players do not enjoy puzzles with entropy that is too high. We also removed exact copies of puzzles that appear in the game. Finally, we removed five outlier puzzles with user

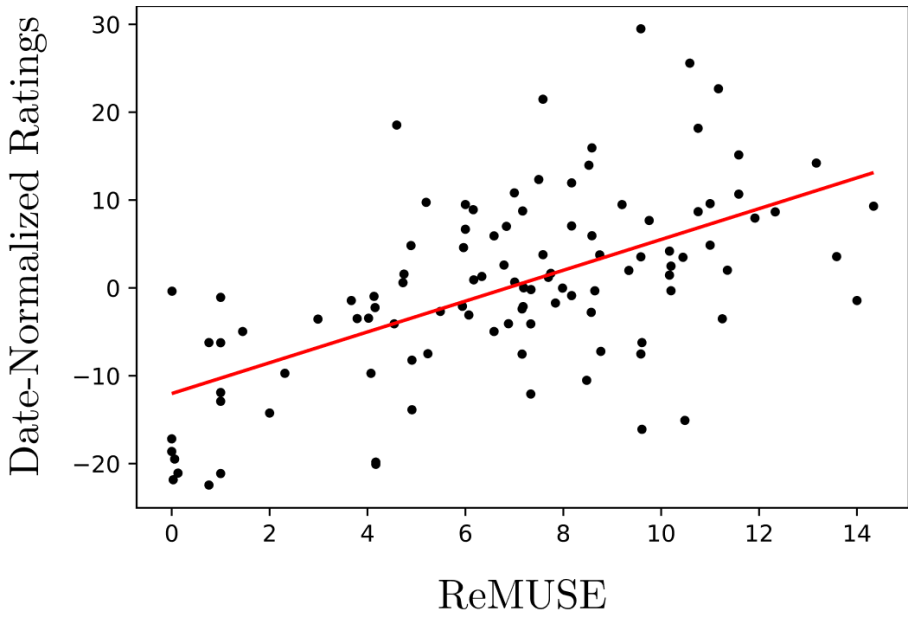


Figure 4.1: User ratings of puzzles and their ReMUSE scores. A ReMUSE Pearson Correlation Coefficient of 0.57 is achieved in this analysis.

ratings of above 40: these were created and voted upon early on when the Windmill was launched.

After filtering, 104 puzzles fit our criteria for testing with an average of 6.4 upvotes. We also adjusted user ratings to reflect the smaller number of players participating on the Windmill site over time by identifying the correlation between user ratings and time, creating a temporally adjusted user rating for each puzzle.

Approach	Correlation (PCC)	p-value
MUSE, no lookahead	0.41	1.1×10^{-5}
MUSE, n=1 step lookahead	0.40	2.4×10^{-5}
MUSE, n=2 step lookahead	0.40	2.4×10^{-5}
MUSE, n=2, w/ Straight Line Policy	0.50	3.7×10^{-8}
ReMUSE, no lookahead	**0.57**	1.5×10^{-10}
ReMUSE, n=1 step lookahead	0.56	4.9×10^{-10}
ReMUSE, n=2 step lookahead	0.56	4.2×10^{-10}
Number of Solutions	0.32	9.7×10^{-4}
Average Solution Length	0.47	3.9×10^{-7}

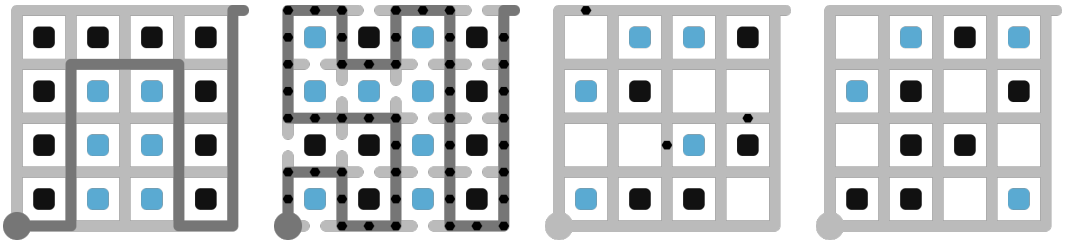
Table 4.1: Pearson Correlation Coefficients (PCC) and p-values for user ratings with Windmill puzzles

The scores obtained from applying different approaches to user-generated puzzles from the Windmill provide a number of interesting insights. As seen on Table 4.1, MUSE scores on puzzles correlate moderately with the puzzle's user ratings, with ReMUSE scores leaning towards a strong correlation at 0.57. MUSE scores with the straight line policy also has a positive correlation with user ratings at 0.50. ReMUSE scores for every puzzle are visualized in Figure 4.1. Unintuitively, it seems that performing n-step lookaheads with either MUSE or ReMUSE approaches lowers the entropy scores' correlation with user ratings. Looking at other measures, the number of solutions for puzzles had a low correlation to user rating. We expect that puzzles with fewer solutions would be more strongly correlated to higher user ratings since fewer solutions imply more difficult puzzles.

These results suggest that the uncertainty-based MUSE and ReMUSE approaches can be used to predict puzzle enjoyment, and by extension we argue that it can predict puzzle difficulty. When we see a positive correlation with user ratings, we are likely seeing a proxy of encountered difficulty since players from the Windmill are experienced and likely derive more enjoyment from challenging puzzles, suggesting that our approaches can be used to assess puzzle difficulty.

We do not expect that the positive correlation between our entropy-based scores and user ratings would continue as puzzle grid size increases. Our results are based on 4×4 puzzles, which are not large enough to create extremely complex puzzles that might frustrate the experience players from the Windmill. For example, difficult 500×500 puzzles could exist, but may prove to be too difficult for human players to solve, and we would expect user ratings for such puzzles to be low even as MUSE and ReMUSE scores increase.

Comparing the ReMUSE measure to the measures of the number or the length of solutions yields two additional findings. Our approaches compare well, if not notably better, in correlating with user ratings. Given that these other measures are often associated with predictors of difficulty, this is a good result. Perhaps more meaningful, however, is the discovery that our puzzle scores can vary greatly even when those other measures are held constant.



(a) 14 actions, ReMUSE: 1 (b) 24 actions: ReMUSE: 0 (c) 24 actions: ReMUSE: 6 (d) 20 actions: ReMUSE: 11

Figure 4.2: Set of puzzles on the Windmill with only 1 solution, but vary in the number of actions taken to solve the puzzle and their ReMUSE scores. The expectation that puzzles are more difficult if they have longer solutions that require more actions is disproved with these puzzles. Puzzle b) requires the most number of actions (24) but is trivial to solve and has a ReMUSE score of 0. The puzzle in c) shares the same number of actions but has a higher ReMUSE score, and the puzzle in d) has fewer actions and the highest ReMUSE score. In our experience, the ReMUSE scores reflect the ranking of difficulty in solving for these puzzles.

For example, puzzles from Figure 4.2 have only one single solution but vary in the number of actions necessary to solve them, as well as their ReMUSE scores. Puzzles a) and b) are solved, noting that for b) the gaps/breaks in the grid mean that no path can be drawn across the gap: these are also known as *must-not cross constraints*. Puzzle a) is simple, but b) is comparatively trivial to solve despite requiring more actions to solve it. Like Puzzle b), Puzzle c) also requires 24 actions to reach the only solution for the puzzle, but is more challenging than b) and has a ReMUSE score of 6. Puzzle d) has fewer actions necessary than b) or c), but has the highest ReMUSE score of the four puzzles, suggesting a higher level of difficulty. For reference, puzzles a) to d) have respective user rating scores of -6, -15, 15, and 28.

4.1.1 Case: Puzzles with Must-Cross Constraints At Every Vertex

Through further analyses, we found a specific set of puzzles that had higher entropy scores and low user ratings. These puzzles had must-cross constraints at every vertex except the start and end vertices, a few of which are shown in Figure 4.3. Placed in this manner, the must-have constraints did not trigger

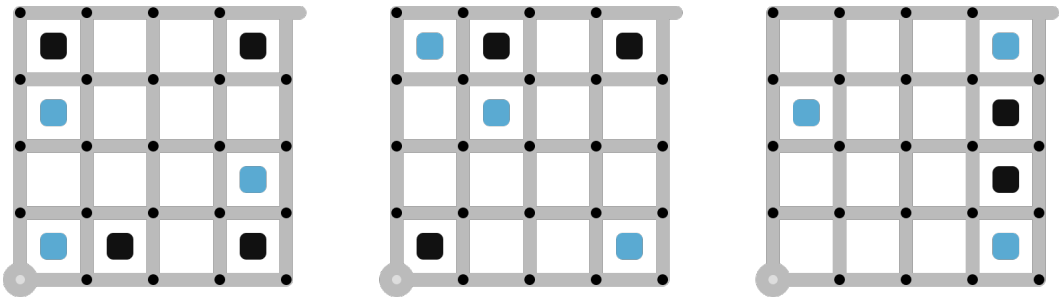


Figure 4.3: Sample of puzzles with must-cross constraints at each non-starting or non-ending vertex. Taking away such puzzles from our data results in a larger coefficient between ReMUSE and user ratings.

our inference rules in reducing the set of possible actions to a smaller set of logical ones. Entropy remained high as a result. Playing the puzzles ourselves, we noted that it was not until only a few final states remained that we could see if we could solve the puzzle. Specifically, early moves in each attempt had a major impact in allowing each must-cross constraint to be fulfilled in a path drawn from the beginning to the end vertex. The impact of these early moves cannot be determined until the player was almost done drawing a path for the rest of the puzzle.

We propose an explanation for the high entropy and low user ratings for this class of puzzles. Puzzles with these specific constraints and placement greatly reduces the effect that logic has for puzzle solving through the solving process. Inference rules can still apply to all other constraints, but no matter how logical a player is in trying to solve the puzzle, we don't have an inference rule that helps identify initial moves that help to draw a path that crosses all vertices. This forces the puzzle to be similar to the experience of solving a large maze: players can get very close to reaching the end of the maze but only realize they cannot solve it until close to the end. Unless a proper inference rule, assuming it exists, is learned by the player, these puzzles will be more frustrating to solve.

When these puzzles are filtered out our our Windmill dataset, we find 14 of them and end up with a smaller subset of 92 puzzles in our analysis. The Pearson Correlation Coefficient for no-lookahead ReMUSE is **increased to 0.67**, with a p-value of 4.04E-15, indicating statistical significance. These results are

promising, since the avoidance of puzzles with particular characteristics can help with crafting enjoyable puzzles.

4.1.2 Case: MUSE with a Straight-Line Policy

Extending our MUSE algorithm with policies that implement global inference rules, specifically using a straight line policy for Witness puzzles, we found that using MUSE with an $n = 2$ lookhead plus the straight line policy resulted in a Pearson Correlation Coefficient of **0.50** and a p-value of 4.64E-08 with user ratings. This is a notable increase from the best MUSE correlation coefficient of 0.41 and suggests that building in additional policies to reflect global inference rules can help to further predict puzzle difficult and enjoyability.

4.2 Patterns from Ordered Witness Puzzle Sets

We consider puzzles from two sets of Witness puzzles, with the first being the starting slate of puzzles within *The Witness* that introduce constraints to the player. We start by computing ReMUSE scores for these puzzles. Sorting puzzles by the ReMUSE scores, we can look for patterns that can help to explain the ordering of puzzles with the game.

We also evaluate Witness puzzles and compare their ordering with the set of *equidistant* puzzles from work on learning curriculum [10]. Given that the equidistant curriculum was generated and shown as effective in helping players learn to solve later puzzles, a similar ordering of the puzzles can provide a possible relationship between of the perceived entropy of a puzzle and its difficulty as modeled through their approach.

Comparing the order of the introductory constraint puzzles from the Witness game with the ordering from ReMUSE provides a number of insights. First, as seen in Figure 4.4, the overall order is very similar. ReMUSE scores the first three puzzles as trivial and equivalent in uncertainty, while puzzles vi-ix have the exact same order that the ReMUSE scores for. Only one change in order occurs as a swap for puzzle iv and v. Both have the same MUSE score

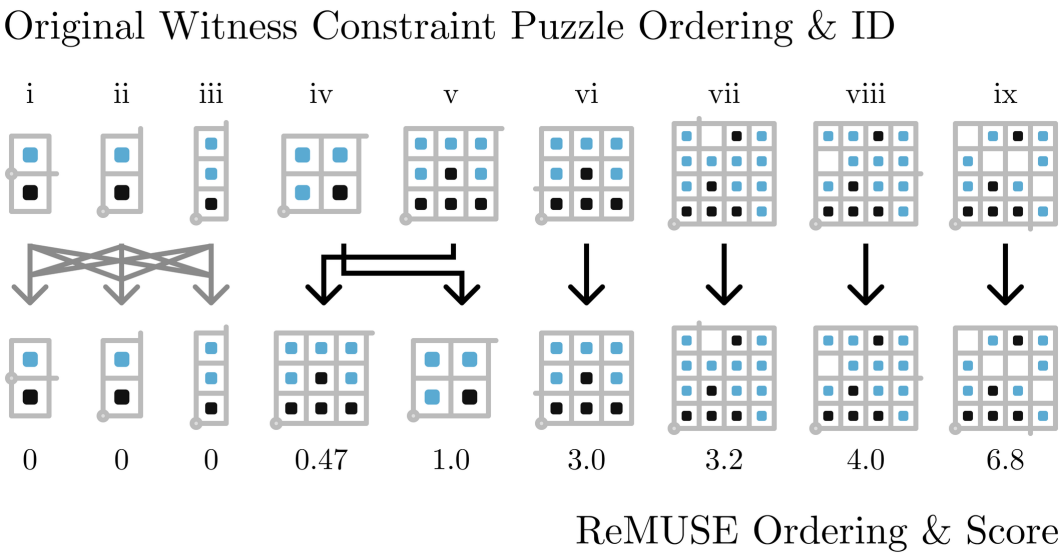


Figure 4.4: Puzzle order of introductory constraint puzzles in The Witness compared with ReMUSE score-driven order. They are quite similar, with initial puzzles deemed as equivalent in puzzle ordering using ReMUSE scores, and only iv) and v) are flipped. This similar ordering indicates that ReMUSE could be used to help rank puzzles for difficulty, and can help to explain the choice of puzzle ordering.

of 1 bit of entropy that occurs at state $s_{\text{start}} = \{[0, 0]\}$, but the KL divergence of the softmin in the ReMUSE measure accounts for the two possible solutions (by first taking either a_{up} or a_{right}), so even if the shorter solution is missed by taking a_{up} the player can still solve the puzzle, resulting in lower overall entropy at s_{start} with ReMUSE. Note that we assume the player knows the constraints, but they do not when these puzzles are first introduced to them. This may account for the difference in the ordering of earlier puzzles.

The similar ordering is a promising result, as it suggests that ReMUSE can be used to create sets of increasingly challenging puzzles to teach concepts. This is what happens in *The Witness*: the game designer creates a variety of puzzles and an ordering of the puzzles to allow players to learn inference rules about the game. This ordering of increasingly challenging puzzles should not deter the player from continued play, either because the puzzles are too simple or too difficult. As demonstrated here, the ReMUSE approach is able to credibly rank puzzles for difficulty.

The ordering of puzzles through our ReMUSE measure can also help to

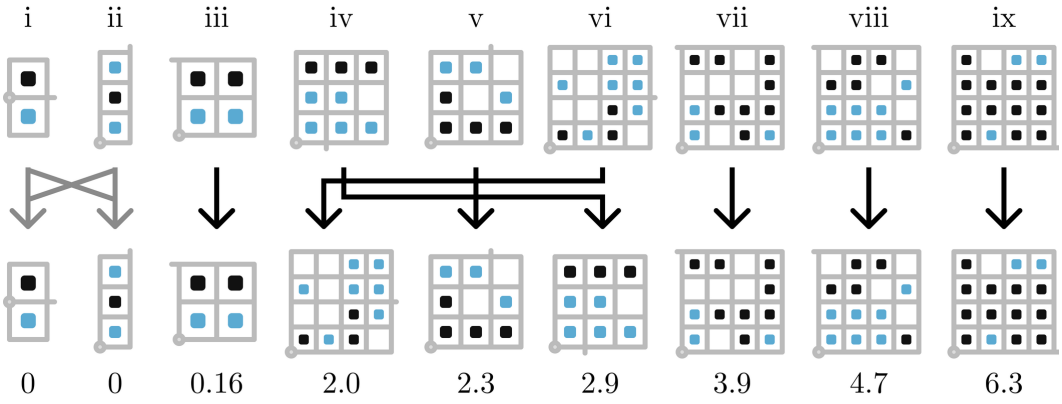


Figure 4.5: Puzzle order of equidistant puzzles compared with ReMUSE score-driven order. The similar ordering here indicates the potential for ReMUSE to be used in sorting puzzles by difficulty for curriculum development.

explain the choice of puzzle ordering by a game designer. Puzzles iv and vi in Figure 4.4, as well as vii through ix, are similar to one another: in all five puzzles, four black square constraints sit beside the start junction of the puzzle. What mostly changes is the position of the exit junction, and it is done to force the player to evaluate another, more uncertain path to solve the puzzle. For example, instead of taking path of least uncertainty to a solution by first taking a_{up} at s_{start} for puzzle v, puzzle vi removes the option for taking that path, requiring the player to encounter uncertainty for the first three states by taking the solution path starting with a_{right}.

The similar ordering for the equidistant curriculum puzzles in Figure 4.5 further reinforces the validity of using ReMUSE as a measure of difficulty and for ordering puzzles for learning. A neural-guided tree search modeled the difficulty of a set of generated puzzles, and 9 were selected for their equally increasing gaps in difficulty. They were successfully tested as possible replacement puzzles for the introductory constraint puzzles in the Witness game, and implies that ReMUSE would have done well at ordering another set of puzzles to teach concepts and logic to players.

These puzzles also suggest insights about entropy in different size puzzles,

work that we explore further by exhaustively generating Witness puzzles of different grid sizes and comparing their ReMUSE score distributions. First, ReMUSE scores vary across different size puzzles, and as shown in both sets of ordered Witness puzzles, smaller puzzles can have larger ReMUSE scores than their larger counterparts. This is promising as it suggests that ReMUSE can identify more challenging puzzles that have a smaller size rather than relying on grid size to determine difficulty. Additionally, 0-entropy puzzles exist on any puzzle size, and the larger the puzzle the longer the maximum path, and the greater the maximum entropy.

4.3 ReMUSE Distribution across Puzzle Sizes

We exhaustively generated every single-solution puzzle that contain at least two differently coloured square constraints, of two colours, for 3×3 and 3×4 sized grids where the starting vertex is at the bottom left and the ending vertex is at the top right. This was done to answer three questions. Keeping constant the number of solutions per puzzle, 1 in this case, are there patterns in the distribution of such puzzles relating to their ReMUSE score? Secondly, are there noticeable patterns and insights from the groupings of puzzles sorted by their ReMUSE scores? Finally, can the insights derived from the groups of puzzles provide intuitions for the use of ReMUSE in puzzle generation, the effectiveness of ReMUSE scores, and the potential for its adoption in puzzle co-creation and exhaustive puzzle generation, or Exhaustive PCG (Procedural Content Generation)?

The distributions of 3×3 sized Witness puzzles with at least two square constraints of having two different colours and having only one solution is shown in Figure 4.6, with the 3×4 sized puzzles shown in Figure 4.7. 1,056 such puzzles were generated for the 3×3 grids, with 16,636 such puzzles generated for the 3×4 grids. Both distributions show that the proportion of puzzles with increasingly large ReMUSE scores will decrease relative to the earlier puzzles, noting the increasing rarity of such single-solution puzzles. Puzzles with ReMUSE scores landing on single integer scores are proportionally higher

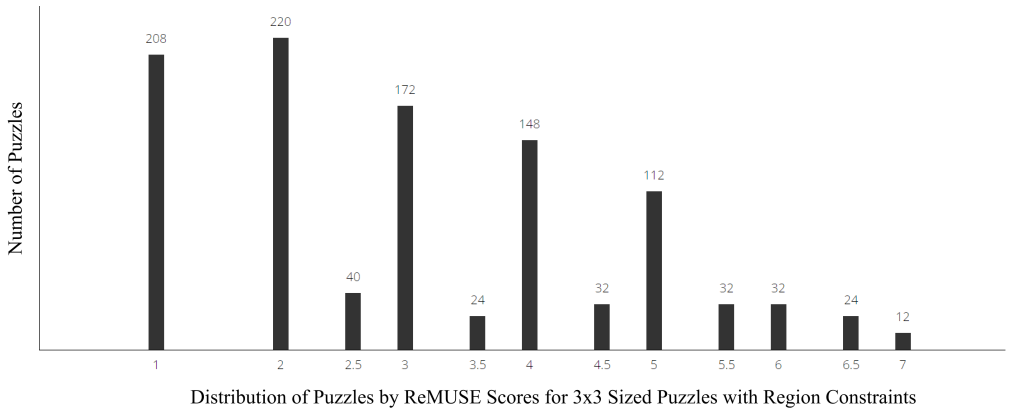


Figure 4.6: Distribution of all 3×3 puzzles with only one solution using two-coloured square constraints. Note the pattern in the decreasing number of puzzles of higher ReMUSE scores, suggesting that challenging puzzles within a given puzzle grid size will be of a smaller proportion compared to easier puzzles.

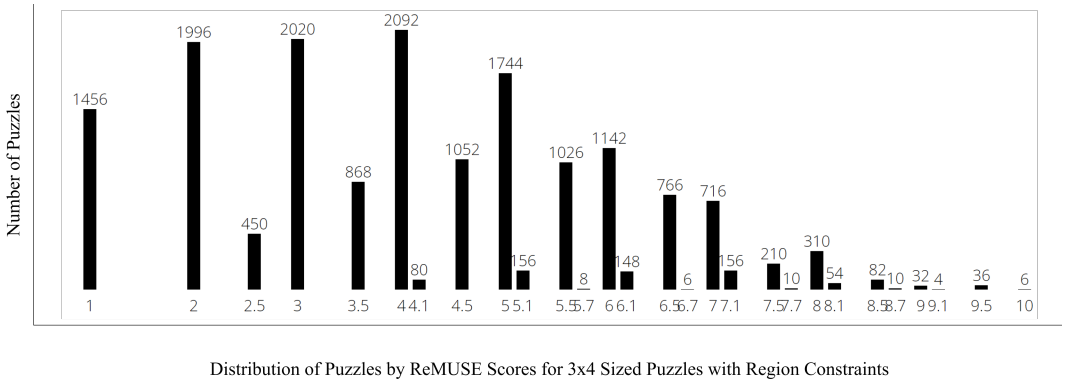
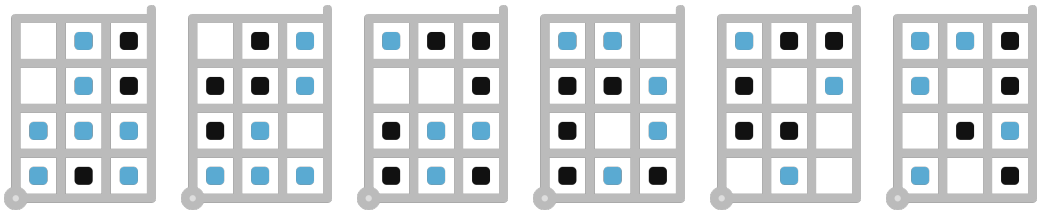


Figure 4.7: Distribution of all 3×4 puzzles with only one solution using two-coloured square constraints. Similar analyses can be arrived at from our 3×3 puzzles. Additionally, note that despite the increased grid size, it is possible to find puzzles with lower ReMUSE scores compared to their 3×3 sized counterparts, suggesting that increased grid size is not a guarantee of ensuring that more difficult puzzles are created.

than their non-integer counterparts, and it was not clear they were marginally more challenging than their closest integer counterparts.

Several insights were drawn from assessing the puzzles grouped by ReMUSE scores. Puzzles with the lowest ReMUSE scores were, in our opinion as researches who have already solved many Witness puzzles, trivial to solve intuitively. This appeared to be true for every puzzle with low ReMUSE scores, and the puzzles appeared more difficult to solve as we assessed puzzles with higher ReMUSE scores. With these groupings, we considered ways to further categorize the scores by some order of difficulty to demonstrate these differences, and we came up with groupings of Easy, Medium, and Hard based on the ReMUSE scores of the puzzles. We provide examples of these puzzles, specifically 3×4 sized puzzles, in Figure 4.8. 3×4 sized puzzles are fairly small and simple to solve, and we suggest that you attempt to solve them to assess their implied difficulty. It is important to note that although every puzzle we visited with low ReMUSE scores were trivial to solve, puzzles with high ReMUSE scores did not guarantee an increase in puzzle challenge. The overall proportion of puzzles that reflected their level of difficulty decreased from their specific groupings of ReMUSE scores, but you were much more likely to find challenging puzzles from looking at puzzles with higher ReMUSE scores than finding them from lower ones.

These insights and the distributions spurred further realizations. For one, most generated puzzles would result in having lower ReMUSE scores, suggesting that challenging puzzles are going to be a smaller proportion of all generated puzzles. Not surprisingly, the increase in size of the grid for Witness puzzles, or an increase in the state space of particular puzzles generally in games, can result in an exponential increase in the number of puzzles to generate and assess, lending further credibility to the use of automated approaches to measure difficulty. Our findings also suggest that if finding challenging puzzles is a primary goal of a puzzle designer, our approach can effectively filter out trivial puzzles by identifying them as ones with low ReMUSE scores. Our approach can also help identify groupings of puzzles with potential higher difficulty, and we suspect that it can become more accurate at predicting puz-



(a) Easy, ReMUSE: 1 (b) Easy, ReMUSE: 1 (c) Medium, ReMUSE: 4.5 (d) Medium, ReMUSE: 4.5 (e) Hard, ReMUSE: 9.5 (f) Hard, ReMUSE: 9.5

Figure 4.8: Examples of generated 3×4 puzzles that have only a single solution, but vary in their ReMUSE scores. We attached arbitrary levels of difficulty to these puzzles based on their ReMUSE scores, and found the difficulty levels to be accurate in our own experiences of solving these puzzles.

zle difficulty by implementing more inference rules that more accurately reflect what a human player can apply to simplify the action space they would have to otherwise navigate. Finally, deriving ReMUSE scores extensively across thousands of puzzles provides us confidence in building our algorithm into puzzle co-creation tools since the approach is simple, deterministic, and relatively quick to return a score.

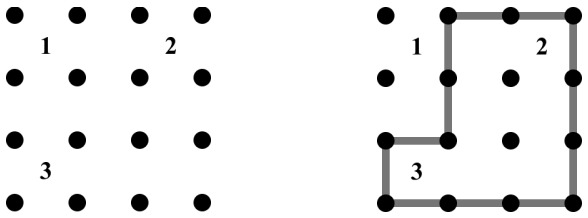
4.4 Slitherlink Puzzles and Triangle Constraints

We pick a separate test set of puzzles from the Windmill to continue to validate the effectiveness of our approach on other games. Specifically, we considered a specific set of constraints from the Witness puzzles that have not been used in our experiments thus far, and together with the rules with Witness puzzles these specific constraints provide a close analogue to another puzzle game called Slitherlink.

4.4.1 Equivalences between Witness and Slitherlink Puzzles

Slitherlink, also known as Fences or Takegaki, is a puzzle game where are dots arranged in a grid-like fashion and numbers 0-3 can appear in the center of four dots. An example of a Slitherlink puzzle is shown in Figure 4.9a.

Solving these puzzles requires players to draw a single, closed loop of vertical or horizontal lines between these dots while ensuring that the number of lines connecting to the dots around a number are equivalent to the number itself. Comparing these to Witness puzzles, instead of grid junctions the vertices are visualized as dots, and the number of edges immediately surrounding a number must be equivalent to the number itself.



(a) Slitherlink puzzle

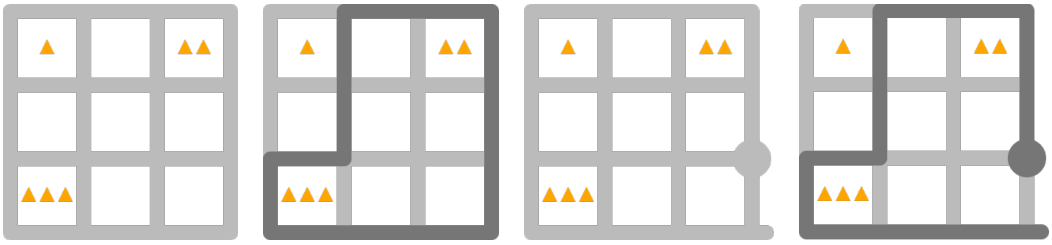
(b) Solution for puzzle

Figure 4.9: Slitherlink puzzle and the corresponding solution. Puzzle solutions require that the number of lines drawn along the black dots or junctions immediately surrounding the number equal the number itself. These lines must connect in a closed grid too. The solution in b) provides three examples of these constraints being fulfilled. The number 1 has one line draw to its immediate right, the number 2 has one above and to the right, and number 3 has lines drawn around it except to its right.

A solution to the sample Slitherlink puzzle in Figure 4.9a is shown in Figure 4.9b. A single loop comprising of edges connecting dots appears, and each number at the center of any four dots has exactly the same number of edges immediately surrounding it. The number 1 has one edge to its right, the number 2 has immediate edges on the top and right, while the number 3 has edges around it except for the one the right.

Witness puzzles can contain triangle constraints that have similar requirements to the numbers found within Slitherlink puzzles. Instead of numbers, Witness puzzles can include the number of triangles that must have the same number of surrounding edges for that triangle constraint to be fulfilled. If we replace the requirement that solutions to Witness puzzles must include a path drawn from a start junction to an end one with drawing a closed-loop path, we can turn the Slitherlink puzzle in Figure 4.9a into a Witness puzzle shown in Figure 4.10a. Similarly, the solution to the Slitherlink puzzle in Figure 4.9b

applies the same way to the modified Witness puzzle in Figure 4.10b.



(a) Close equivalence to Slitherlink puzzle in Figure 4.9a

(b) Solution to a closed-loop puzzle in Figure 4.10a

(c) A Witness puzzle modeled after Figure 4.9a

(d) Solution to puzzle in Figure 4.10c

Figure 4.10: Witness puzzles similar to the Slitherlink puzzle in Figure 4.9a. These puzzles are not equivalent to the Slitherlink puzzles, but are quite similar. Instead of using numbers, the number of triangles in Witness puzzles are used instead to indicate the number of lines that must be immediately drawn around it, so the Witness puzzle in a) without start and end junctions as shown in c) is most similar to the Slitherlink puzzle in Figure 4.9a.

4.4.2 Differences between Puzzle Games

Slitherlink and Witness puzzles containing only triangle constraints differ in a few ways. As already discussed earlier, Witness puzzles are solved by drawing a path from a start vertex to a goal vertex instead of drawing a closed loop. There are ways to make these more similar, such as the example Witness puzzle in Figure 4.10c and a consequent solution in Figure 4.10d, where the start and goal vertices are immediately adjacent either horizontally or vertically to one another. This effectively requires a closed loop to be drawn, making the puzzle games similar to one another, although this does require the closed loop path to pass through those vertices. Additionally, there are no triangle constraints that reflect a Slitherlink's use of the number 0 as a constraint at the center of four dots. In those puzzles, no edges must connect between any of the four dots, and Witness puzzles do not have a special visual representation of that constraint using triangles.

Players can also solve Slitherlink puzzles from any dot or vertex, whereas Witness puzzles are solved by drawing a path from a starting vertex. Slitherlink puzzles can also be analyzed and solved in separate component sections

that make up the entire puzzle. For example, the puzzle from Figure 4.9a could have been solved by drawing edges around each of the numbers first, and then connecting those edges to form a closed loop. In Witness puzzles edges can only be drawn from a junction that was last connected to by the latest edge, and solving the puzzle in parts is not similarly possible to Slitherlink puzzles. When Witness puzzles are being solved by human players on paper, however, they can be deconstructed and solved in parts.

We feel that there is value in testing the ReMUSE approach on Witness puzzles with triangle constraints as an analogue to a different non-Witness game despite the stated differences between the games. Puzzles from both games can be solved by connecting vertices with edges and the specific number or triangle constraints are equivalent in both games. Witness puzzles with only triangle constraints can be separated from other puzzles, providing us a separate testing dataset, and such puzzles found on the Windmill site allows us to compare the ReMUSE approach's effectiveness at predicting positive user ratings.

4.4.3 Testing Data and Outcomes

Similar to our original dataset, we retrieved Witness puzzles from the Windmill that are of a 4x4 size, and found 268 such puzzles that contain triangle constraints. We filtered the puzzles further to exclude ones containing non-triangle constraints, resulting in similar-to-Slitherlink puzzles, and further filtered ones from a specific user that had submitted dozens of such puzzles to the Windmill within two days when the average of such puzzle submissions is less than once a week. That user's puzzles, all titled as "Practice" by the user, had unusually low user ratings, likely a result of negative player feedback from having the user's puzzles spammed on the site.

The finalized dataset included 152 4x4 Witness puzzles that contained only triangle constraints, with examples of such puzzles shown in Figure 4.11. Testing on these 152 puzzles resulted in ReMUSE scores that had a Pearson Correlation Coefficient of 0.40 with user ratings and a p-value of 3.92E-07. This indicates a lower correlation to user ratings as compared to our results

of puzzles containing one or a combination of constraints, specifically coloured squares, must-cross, and cannot-cross constraints.

(a) ReMUSE: 2.0 (b) ReMUSE: 2.05 (c) ReMUSE: 8.17 (d) ReMUSE: 9.91

Figure 4.11: Puzzles containing only triangle constraints on the Windmill and their ReMUSE scores.

Several factors could have resulted in these Slitherlink-like puzzles in having a lower correlation to user ratings. For one, we acknowledge that several inference rules that can be used to represent player knowledge in our algorithms were not included due to the complexity of implementing them in code. In practice, we also found that the puzzles seemed easier to solve because the appearance of triangle constraints limited paths that could be drawn for all four edges surrounding a given constraint. Looking later at generated Slitherlink puzzles from puzzle game sites online, it was clear that the smallest puzzles started at a 5x5 grid size for simpler puzzles as opposed to our test puzzles of 4x4 grid size. This suggests that 4x4 Witness puzzles with only triangle constraints might be easier to solve, resulting in the lower user ratings found in our analysis.

Perhaps more interesting, the correlation between the average solution length for our test puzzles to user ratings is 0.02 with a p-value of 0.76, which is significantly smaller from the correlation of 0.47 found in our triangle constraint exclusive set of puzzles. It suggests that while average solution length could be correlated with challenge and user ratings for certain puzzle games, it could be ineffective for puzzle games or puzzle types with specific properties. This is the case for our test puzzles, for which our ReMUSE approach still indicated a positive, if weaker, correlation in our current implementation.

Chapter 5

Conclusion and Future Work

In this thesis we introduced MUSE and ReMUSE as measures of puzzle difficulty based on entropy. We imbued our algorithms with logical inference rules to model human players solving puzzles. We evaluated our approach on Witness puzzles, comparing our entropy scores with user ratings from a large online database. Our results suggest that our ReMUSE measure produces scores that correlate well with user ratings and better than the correlations from alternative measures like solution length and the number of solutions. Our approach produced similar orderings for puzzles in two sets of puzzle curricula, and we identified further learnings from the distribution of puzzles by ReMUSE scores across single-solution puzzles of different sizes. Finally, we had a positive correlation to user ratings for Witness puzzles that are a close equivalence to a different puzzle game.

On future work, inference rules are hard to write. A potential solution to this problem is to use inductive logic programming [13] to automatically create sets of inference rules by learning from databases of puzzles and their solutions [22]. These automatically generated inference rules could then be applied to our entropy algorithms.

People also vary in knowledge and skill, so different players may find certain puzzles difficult when the puzzles may seem easy for other players. Our experiments model the strong player that does not make mistakes based on the inference rules we know, which works well for trying to predict the difficulty of a puzzle for experienced players. A natural next step is to create models

of players with different skillsets and knowledge: they may not know all the inference rules and get stuck at a puzzle unless that gap in their knowledge is filled. Exploring further non-local policies like our straight line policy would also be helpful. These can be identified by finding puzzles that are hard for existing policies to solve. The additional policies can be further tested and may better model players.

Applying our measures to other games would be a natural extension. Logic puzzle games with a small number of actions that can be taken at any state would be a natural fit, since this reflects Witness puzzles. *Sokoban* and Snakebird are examples of such games. Our measures can likely apply to games with larger state spaces like *Sudoku* and *Minesweeper* too. While the number of actions that can be taken in such games can be quite large at every turn, inference rules can also help to greatly reduce the number of actions that should be considered or taken. As mentioned above, sets of inference rules can also be varied to model player experience and ability for these games to compute their difficulty for the player.

It would make sense to test ReMUSE on broader ranges of puzzles in future work, as well as performing user studies to empirically verify whether our approaches can be used to generate enjoyable puzzles of varying uncertainty. Utilizing this research for puzzle co-creation would also be useful, allowing game designers to create, or even automatically generate, puzzles that are tailored to a player's knowledge and skill.

Finally, the approach to evaluating uncertainty can be extended to non-game settings. Injecting measurable uncertainty into educational settings could help learners better absorb lessons by providing skill-appropriate questions that are not too challenging or simple.

References

[1] John S Bridle. "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition." In: *Neurocomputing: Algorithms, architectures and applications*. Springer. 1990, pp. 227–236.

[2] Cameron Browne. "Deductive search for logic puzzles." In: *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. 2013, pp. 1–8. DOI: 10.1109/CIG.2013.6633649.

[3] Guillermo Campitelli and Fernand Gobet. "Adaptive expert decision making: Skilled chess players search more and deeper." In: *ICGA Journal* 27.4 (2004), pp. 209–216.

[4] Eugene Chen et al. "Image-to-level: Generation and repair." In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 16. 1. 2020, pp. 189–195.

[5] José Coelho. *A scale to measure the difficulty of sudoku puzzles*. Unpublished. 2007.

[6] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper & Row, 1990. ISBN: 0060162538.

[7] Mária Ercsey-Ravasz and Zoltán Toroczkai. "The chaos within Sudoku." In: *Scientific reports* 2.1 (2012), p. 725.

[8] Petr Jarušek and Radek Pelánek. "Difficulty rating of sokoban puzzle." In: *Proceedings of STAIRS 2010*. IOS Press, 2010, pp. 140–150.

[9] Solomon Kullback and Richard A Leibler. "On information and sufficiency." In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[10] Levi HS Lelis et al. "Learning Curricula for Humans: An Empirical Study with Puzzles from The Witness." In: (2022).

[11] *mdp.ai, Generates visualized Markov decision processes*. https://mdp.ai/. Accessed: 2023-09-08.

[12] *Moving AI lab website*. https://www.movingai.com/. Accessed: 2023-09-08.

[13] Stephen Muggleton and Luc De Raedt. *Inductive logic programming: Theory and methods*. 1994.

[14] Tomofumi Nakano et al. "Inducing shogi heuristics using inductive logic programming." In: *Inductive Logic Programming*. Ed. by David Page. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 155–164. ISBN: 978-3-540-69059-7.

[15] Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game Analytics: Maximizing the Value of Player Data*. Springer Publishing Company, Incorporated, 2013. ISBN: 1447147685.

[16] Xue Bin Peng et al. "Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow." In: *arXiv preprint arXiv:1810.00821* (2018).

[17] *Reinforcement Learning Specialization course on Coursera*. `https://www.coursera.org/specializations/reinforcement-learning`. Accessed: 2023-09-08.

[18] *ReMUSE interactive site*. `https://ideaowl.com/remuse/`. Accessed: 2023-09-08.

[19] Mohammad Shaker et al. "Automatic generation and analysis of physics-based puzzle games." In: *Proceedings of 2013 IEEE Conference on Computational Inteligence in Games (CIG)*. 2013, pp. 1–8. DOI: `10.1109/CIG.2013.6633633`.

[20] C. E. Shannon. "A mathematical theory of communication." In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: `10.1002/j.1538-7305.1948.tb01338.x`.

[21] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning structured output representation using deep conditional generative models." In: *Advances in neural information processing systems* 28 (2015).

[22] Justin Stevens, Vadim Bulitko, and David Thue. "Solving Witness-type Triangle Puzzles Faster with an Automatically Learned Human-Explainable Predicate." In: *arXiv preprint arXiv:2308.02666* (2023).

[23] Nathan Sturtevant and Matheus Ota. "Exhaustive and semi-exhaustive procedural content generation." In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 14. 1. 2018, pp. 109–115.

[24] Nathan Sturtevant et al. "The Unexpected Consequence of Incremental Design Changes." In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1 (Oct. 2020), pp. 130–136. DOI: `10.1609/aiide.v16i1.7421`. URL: `https://ojs.aaai.org/index.php/AIIDE/article/view/7421`.

[25] *The Effect and Evolvability of Observational Learning in A-Life*. `https://slides.com/ideaowl/ola-life`. Accessed: 2023-09-08.

Appendix A

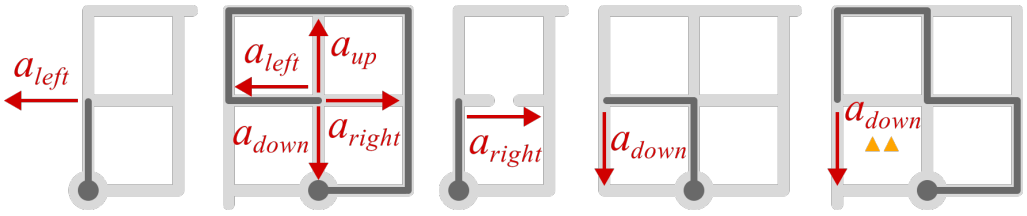
Inference Rules for Witness Puzzles

Note that the working code for MUSE and ReMUSe, which utilize $L(s)$, is available at https://ideaowl.com/remuse.

A.1 Assessing Actions

To determine logical actions from $L(s)$ for a given state s, each of the four cardinal actions (up, down, left, right) that can be taken is first assessed as being one of four types of actions. Each must be a **must not take**, **must take**, **probable**, or **probable must take** action, with each assessment indicating how taking the action affects the possibility of solving the puzzle.

Must not take actions, with examples highlighted in Figure A.1 cannot be returned by $L(s)$, since taking any of these actions will either violate puzzle



(a) Path outside the puzzle grid

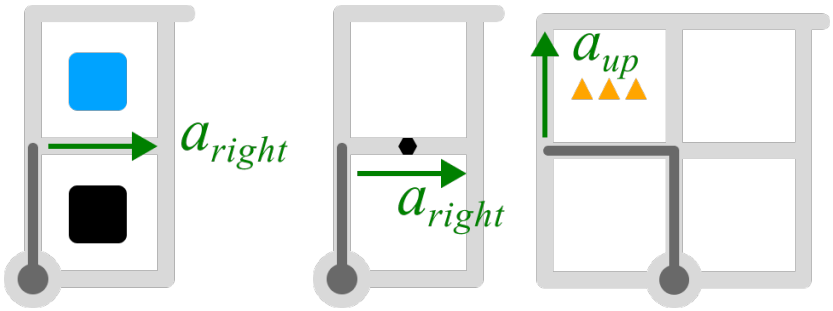
(b) Crosses previously visited path

(c) Cannot draw across gap

(d) Boxes path in with no path to end junction

(e) Triangle constraint no longer satisfiable

Figure A.1: Examples of must not take actions based on rules for Witness puzzles or from inference rules



(a) Different colour squares adjacent to the state must be immediately separated

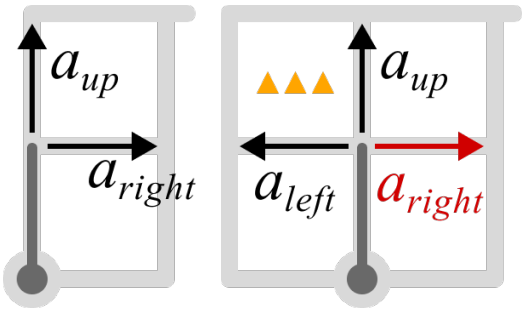
(b) Must cross constraints between junctions

(c) Triangle constraint satisfaction requires this action

Figure A.2: Examples of must take actions based on different inference rules

rules or make the puzzle unsolvable. For example, paths cannot be drawn outside of the puzzle grid, so a_{left} in Figure A.1a is considered a must not take action. Paths must be non-crossing, which renders all four actions in Figure A.1b as ones that must not be taken too, as is the case for actions that draw a path across gaps, as demonstrated in Figure A.1c. Any state that has previously visited a *wall* of the puzzle grid, specifically edges that form the perimeter of the puzzle, and immediately arrives at another wall perpendicularly, as is the case for the state in Figure A.1d, will result in having an action that will draw the path within a closed-off box that will not allow any future path to reach the end junction. These actions are considered must not take actions too. Finally, if the number of non-visited junctions of a triangle constraint is fewer than the remaining number of edges that a path needs to cross for the constraint to be satisfiable, any action that would otherwise draw the path along the edge of the triangle constraint is considered a must not take action. This is shown in Figure A.1e: three junctions have already been visited by the current path, and while taking a_{down} can help to fulfill the triangle constraint of requiring two of its edges to be part of the solution path, it needs two edges to satisfy the requirement and there are not enough non-crossing junctions to do so.

Must take actions suggest that they must be taken for the puzzle to remain solvable. As previously demonstrated in the thesis, the existence of different



(a) Actions that are not must take or must not take

(b) Special case of probable must take actions

Figure A.3: Probable and probable must take actions

colour squares adjacent to the state requires that they are immediately separated, as is the case in taking a_{right} in Figure A.2a. Similarly, must cross constraints that are in between junctions adjacent to the current state must also be fulfilled by taking the action that draws the path to cross them, as shown in Figure A.2b. Triangle constraints that have the number of remaining unvisited junctions equal to the number of remaining edges to fill will require that the action that draws a path along the triangle constraint must be taken. Figure A.2c shows such a triangle constraint and state: with only 2 unvisited junctions and 2 remaining edges to fill, a_{up} is assessed as a must take action. There is one exception to this assessment, which is the special case of the triangle constraint with three triangles, which we expand on soon.

Probable actions are ones where no inference rules exist to suggest that the action must or must not be taken, so these actions could result in solving the puzzle. The black colored arrows in Figure A.3a indicate the probable actions for that puzzle's state.

Probable must actions are slightly different from probable actions. Like probable actions, taking one of these actions could satisfy constraints to make the puzzle solvable from the puzzle state. Unlike them, however, one of these probable actions must be taken to satisfy puzzle constraints, and so any other probable actions are considered must not take actions. The puzzle state in Figure A.3b shows such a scenario. When a path is drawn to the first junction

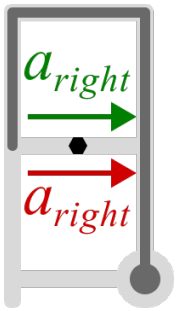


Figure A.4: When an action can be determined as both a must take and must not take action.

adjacent to a triangle constraint of 3 triangles, then 1 of the 2 actions that would draw a path along the edge of the constraint must be taken. These are considered probable must take actions. Either of the 2 actions must immediately be taken for the triangle constraint to be satisfied, otherwise the constraint cannot be satisfied later. As a result, what would originally be considered a probable action for a_{right} is now considered a must not take action, since taking it would make it impossible to solve the puzzle in any descendant state.

Note that it is possible for an action to be simultaneously determined as a must take and must not take action. Figure A.4 shows such a case, where a_{right} is assessed as a must take action since it fulfills the must cross constraint on the right, but taking a_{right} violates the rule of crossing over previously taken paths. In such cases where an action is considered both must take and must not take, the action is deemed as a must not take action.

A.2 $L(s)$ Returns After Actions are Assessed

If only a single must take action exists at state s, $L(s)$ returns only that specific action. If more than one must take action exists at state s, $L(s)$ returns no actions. This is because more than one action needs to be taken at state s for the puzzle to remain solvable, and since it is impossible to take both actions at the same time, no solution is possible from state s. $L(s)$ returns all probable actions otherwise.