Generalized Entropy and Solution Information for Measuring Puzzle Difficulty

by

Junwen Shen

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Metrics for problem difficulty are used by many puzzle generation algorithms, as well as by adaptive algorithms that are expected to provide players with the puzzles at the correct level of difficulty. A recently proposed general metric, puzzle entropy, combines an analysis of game mechanics with a model of player knowledge in the form of inference rules to predict problem difficulty. The entropy of a puzzle is the amount of information required, given a player's knowledge about the puzzle, to describe a solution to a puzzle. This thesis generalizes the concepts of puzzle entropy and solution information, providing a better foundation for the previous work and creating new algorithms, Minimum Solution Information and Total Solution Information. While functionally similar to past work, the new algorithms allow knowledge about a puzzle to be represented as a policy. We then evaluate the impact of inference rules, policies, and player knowledge in the 2016 game, *The Witness*.

Preface

This thesis extends upon the research paper [1] initially submitted at the 20th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), co-authored with Nathan R. Sturtevant.

Even if the ending has been predetermined, that's fine.

There are countless things that humans cannot change.

But before that, on the road towards the end,

there are still many things that we can do.

And because of this, the "end" will thus reveal

a completely different meaning.

This is the meaning of "journey".

All those things, beautiful before, are still so now.

And I believe...

It will still bloom at the end of the Nihility,

until we meet again beneath the sun's rays.

- Acheron

Acknowledgements

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 express my sincere gratitude to them for their support of this work.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Developing a generic algorithm to measure puzzle difficulty is a complex task [2]. Different games possess various features that may be challenging to quantify, then compute and standardize [3]. Nevertheless, an accurate difficulty metric can be highly beneficial. Puzzle designers could use it to determine the quality and suitability of a puzzle within a game [4] and predict player enjoyment. It could also aid in generating and analyzing puzzle curricula, allowing players to acquire game-specific knowledge more easily [5]. Our work is closely related to measuring puzzle difficulty and its relationship with player engagement.

1.1 Related Works

Puzzle difficulty represents the challenges that players face and directly influences their enjoyment [6]. Introducing appropriately challenging puzzles is crucial for maintaining player engagement, as overly difficult puzzles early in the game can lead to player frustration and abandonment. Difficulty scoring also supports procedural puzzle generation, enabling the rapid creation and assessment of new puzzles. Automated difficulty evaluation can significantly accelerate the development process by allowing designers to quickly integrate user-specific and suitable puzzles into their games [2]. Moreover, by dynamically adjusting puzzles based on player performance and feedback, designers can maintain an engaging and satisfying player experience.

In *Sokoban*-type games like *Fling!* [7], simple metrics such as the number of reachable states or problem decomposition by subproblem solution length have proven effective [8, 9]. Similarly, minimum solution length is used in exhaustive procedural content generation (EPCG) [10] for incrementally designing levels in *Snakebird* [11, 12] or sliding puzzles like *Rush Hour* [13, 4].

Constraint-based algorithms, such as answer set programming, have been employed to assess maze-like puzzles, where mazes are represented as grid-embedded trees with constraints on reachability and path length [14, 15]. Logic puzzles like *Sudoku* can also be treated as constraint satisfaction problems, using corresponding algorithms such as arc consistency with domain splitting to analyze difficulties [16].

However, these metrics may not be applicable to puzzles that involve real-time physics, such as *Cut The Rope* [17] and *Angry Birds* [18]. Additionally, difficulty varies among players with different levels of knowledge and skills. Consequently, more complex methods, such as simulating playtests using deep reinforcement learning, have been developed and applied to *Angry Birds* [19].

Recent work by Chen et al. introduced the notion of puzzle difficulty based on the amount of uncertainty a player might face [20], which can be broadly applied across different puzzle games [21]. Their approaches rely on information entropy, representing the amount of communication required to describe the solutions of a single-player turn-based puzzle. Published results demonstrated a positive correlation between puzzle difficulty, as measured by their algorithms, and user ratings on a subset of puzzles and constraints from *The Witness* [22].

1.2 Generalizing the Recent Approach

Our work aims to understand the methods proposed by Chen et al. [20] and their limits. Therefore, we propose to study the nature of the underlying algorithms, consider enhanced player models, and expand the experiments conducted in previous studies.

This thesis establishes the mathematical connection between the previous MUSE approach and the probability of solving a puzzle. Our new approaches, minimum solution information (MSI) and total solution information (TSI), are derived which have a stronger mathematical foundation, while maintaining the generality of the previous methods. But, in addition to modeling players with inference rules, it also allows modeling policies. We then develop additional inference rules that can be applied to broader types of puzzles in the game, *The Witness*. Consequently, we are able to expand the puzzle datasets used for evaluation. Empirically, we find that our approach, total solution information (TSI), is as effective as previous methods across different datasets. We hypothesize that using too many or too powerful inference rules overestimates players' abilities, resulting in a poorer correlation between puzzle difficulty and engagement. We evaluate the policy from a machine learning (ML) model and show that it has learned puzzle difficulty, but not as well as our inference rules. Moreover, we verify that using too many or too powerful inference rules overestimates players' abilities, resulting in a poorer correlation between puzzle difficulty and engagement. Finally, we create a puzzle-design tool with a web GUI, enabling designers to invoke exhaustive procedural content generation (EPCG) queries that generate suggested designs sorted by difficulty from our approach.

Chapter 2

Background

This thesis extends algorithms proposed by Chen et al. [20, 23] that use heuristic search and information entropy for measuring difficulty of turn-based puzzle games, evaluated on puzzles in *The Witness*. In the following sections, we describe how to formally represent puzzles using mathematical notations, the concept of information entropy and how to apply it on puzzles, and puzzles from *The Witness* for evaluation.

2.1 Heuristic Search

A puzzle P can be formally defined as $P = (S, A, T, s_0, G)$ [5], where S is a set of states that includes the initial state s_0, and G is a subset of S that represents the set of goal states. The action function A returns a set of possible actions at a given state s. If a state s is terminal, $A(s) = \varnothing$. The deterministic transition function T takes a state s and an action a as inputs and produces a new state s' that results from applying action a to state s. Thus, the solution path of goal state g can be represented as a sequence of state-action pairs $\tau_g = \{(s_i, a_i) \mid T(s_i, a_i) = s_{i+1} \text{ and } T(s_m, a_m) = g \in G\}_{i=0}^{m}$, and the player's policy can be represented by a function $\pi(a, s) : A(S) \times S \to [0, 1]$ that returns the probability of taking action a at state s. The search space can be represented as a tree, where each node corresponds to a state in S, and the tree is rooted at s_0. A directed edge exists from state s to state s' if $T(s, a) = s'$ for some action $a \in A(s)$. Furthermore, the successor function is denoted as $\sigma(s) = T(s, A(s))$

and returns a set of successor states of state s.

2.2 Entropy of a Puzzle

Information entropy [24] serves as a fundamental metric in information theory, allowing for the quantification of uncertainty in stochastic processes. Entropy represents the expected amount of information content given a discrete random variable X and its probability distribution $P : \mathcal{X} \to [0, 1]$ over the sample space $\mathcal{X}$.

$$H(X) = \mathbb{E}\left[I(X)\right] = -\sum_{x \in \mathcal{X}} P(x) \log_2 P(x) \tag{2.1}$$

For example, consider the case of tossing a fair six-sided die. The entropy of this event would be $H = -6 \times \frac{1}{6} \log_2 \frac{1}{6} = \log_2 6 \approx 2.585$ Sh[1], indicating that the information content of each outcome is $\log_2 6$ Sh.

To gauge puzzle difficulty, we adopt the minimum uniform entropy (MUSE) and relative minimum uniform entropy (ReMUSE) metrics proposed by Chen et al. [20]. These metrics quantify the amount of information an oracle would need to provide for a player to solve the puzzle. MUSE directly calculates the minimum amount of uniform entropy based on the number of choices at each step along the solution path. On the other hand, ReMUSE utilizes the Kullback-Leibler (KL) Divergence [26], also known as relative entropy, as shown in Equation 2.2, to capture the information of other solutions from successors of the current state.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log_2 \frac{P(x)}{Q(x)} \tag{2.2}$$

Here, P and Q are discrete probability distributions defined on the same sample space $\mathcal{X}$. In ReMUSE, P is the softmin of direct successors' uniform entropy, Q is a uniform distribution as player's policy, and $|P| = |Q| = |A(s)|$. MUSE and ReMUSE values increase with more complex puzzles, as they require the player to

[1]We use the *Shannon* (binary unit) as the unit of information instead of *bit* (binary digit) according to the International System of Quantities [25] to avoid ambiguity

make more decisions. Previous work [20] demonstrated that MUSE and ReMUSE are effective tools for assessing puzzle difficulty. Moreover, the positive correlation between these metrics and user ratings in puzzles from *The Witness* further supports the relationship between difficulty and player satisfaction.

2.3 Puzzles from The Witness

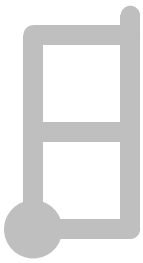
The Witness is a single-player puzzle solving game published by Thekla, Inc. in 2016. We refer them henceforth as "Witness puzzles". Most puzzles in the game are rectangular grids of size $w \times h$. In Witness puzzles, the player must draw a non-self-crossing path along the grid lines starting from the designated start junction and ending at any of the goal junctions. The start junction is represented by a round circle, and the end junction is indicated by a notch extended from the grid line (Figure 2.1a). When the path reaches the end junction, the notch is automatically filled.

The game includes two classes of constraints: path constraints and region constraints. Path constraints are placed on the grid lines, while region constraints are present in the blocks surrounded by the grid. Figure 2.1b and Figure 2.1c illustrate examples of path constraints and region constraints with their corresponding solutions. Additionally, Figure 2.2a presents a more complex puzzle that includes every type of constraint studied in this research. The requirements for satisfying these constraints are explained in Table 2.1. It is essential for the solution path to comply with all the constraints in the puzzle.

Using the definitions in previous sections, the solution path for the puzzle in Figure 2.1b can be represented as $\{(s_0, a_{\text{up}}), (s_1, a_{\text{right}}), (s_2, a_{\text{up}})\}$, and its MUSE value without using inference rules is calculated as $\log_2 |A(s_0)| + \log_2 |A(s_1)| + \log_2 |A(s_2)| = 1 + 1 + 0 = 2$ Sh. Similarly, the solution path for the puzzle in Figure 2.1c is $\{(s_0, a_{\text{right}}), (s_1, a_{\text{up}}), (s_2, a_{\text{left}}), (s_3, a_{\text{up}}), (s_4, a_{\text{right}})\}$, and its MUSE value is 1 Sh instead. Based on these calculations, puzzle in Figure 2.1c is easier than the one in Figure 2.1b.

Icon and Name	Requirements
 Must-Cross	The path must cross the constraint
 Cannot-Cross	The path cannot cross the constraint
 Separation	The constraint must be separated from different coloured separation constraints in the same region
 Star	The containing area must have exactly one other region constraint with the same colour
 Triangle	The path around it must occupy the same number of edges as the number of triangles

Table 2.1: Constraints and their requirements

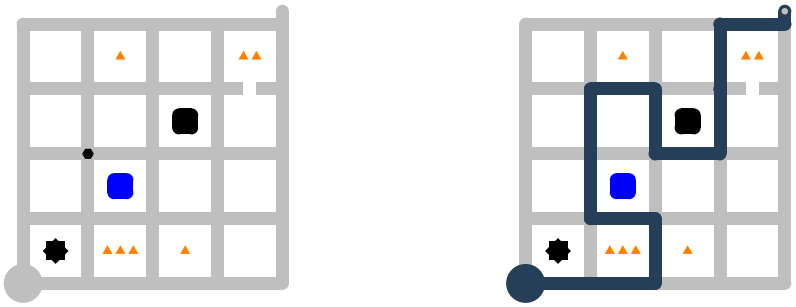


(a) A 1×2 puzzle with no constraint

(b) A 1×2 puzzle with 2 region constraints and a solution

(c) A 1×2 puzzle with a path constraint and a solution

Figure 2.1: Example The Witness puzzles with size 1×2



(a) A 4×4 puzzle with 5 types of constraints

(b) The solution to the puzzle in (a)

Figure 2.2: Example 4×4 complex Witness puzzle and its solution

2.4 Procedural Content Generation

Procedural content generation (PCG) refers to the automatic creation of content through algorithms, primarily employed in video games, simulations, and other interactive media to produce complex and varied environments, levels, and assets. One of its primary advantages is its ability to dynamically produce large amounts of content, reducing the need for manual design, offering novel experiences for players with each interaction by leveraging computational techniques [27]. Exhaustive Procedural Content Generation (EPCG), a type of search-based PCG [28], aims to generate and evaluate all possible content configurations [10]. EPCG ensures comprehensive coverage of all potential variations, thereby enhancing the robustness and diversity of generated content. By systematically evaluating every possible outcome, EPCG can identify optimal solutions, uncover hidden patterns, and ensure that generated content meets specific design criteria and quality standards. For puzzles, difficulty is one common metric used for evaluation [4]. This approach is particularly valuable in research and development contexts, where extensive exploration of content possibilities can lead to significant insights and innovations.

Chapter 3

Mathematical Foundations

We study the nature of MUSE by examining its connection to the probability of finding a solution according to player's policy. In this chapter, we present mathematical theorems that reveal the underlying principles of these algorithms.

3.1 MUSE and Probability

We denote the MUSE of puzzle X as $\mu(X)$. In $\mu(X)$, it is assumed that the probability distribution of player actions is uniform, and $|A(s)|$ represents the number of available actions at each state s. Therefore, in the policy π_μ, $\pi_\mu(a, s) = |A(s)|^{-1}$ and the local entropy at s is $\log_2 |A(s)|$ [20]. Moreover, let $\tau_m = \{(s_i, a_i)\}_{i=0}^{m}$ be the sequence of state-action pairs starting at the root of the tree s_0 and ending at terminal state $T(s_m, a_m) = s$, then $\pi(s) = \prod_{(s_i,a_i)\in\tau_m} \pi(a_i, s_i)$ and $\pi(g)$ represents the probability of reaching the goal state g through the solution path, τ_g, according to π. Similarly, $\pi_\mu(g)$ is the probability of reaching the goal state g by uniform random. Now, we can derive the following theorem.

Theorem 1 *$\mu(X)$ is proportional to the probability of finding the most likely solution according to the player's policy (uniform random), where each solution path to the goal state $g \in G$ is a state-action sequence τ_g. It can be calculated by:*

$$\mu(X) = -\log_2 \max_{g\in G} \pi_\mu(g) \tag{3.1}$$

Proof. According to the recursive definition, μ of a non-terminal state, $\mu(s)$, is obtained by summing the local entropy and the minimum entropy of its successors. The local entropy is $\log_2 |A(s_i)|$ since $\pi_\mu(s) = |A(s_i)|^{-1}$. Hence, we have the following cases:

$$\mu(s) = \begin{cases} 0 & \text{if } s \in G \\ \min_{s_i \in \sigma(s)} \mu(s_i) + \log_2 |A(s_i)| & \text{if } A(s_i) \neq \varnothing \\ \infty & \text{otherwise} \end{cases}$$

By simplifying the definition without recursion, we obtain:

$$\mu(X) = \min_{g \in G} \sum_{s_i \in \tau_g} \log_2 |A(s_i)|$$

where each solution path to the goal state $g \in G$ is a state-action sequence $\tau_g = \{(s_i, a_i) \mid T(s_i, a_i) = s_{i+1} \text{ and } T(s_{m_g}, a_{m_g}) = g\}_{i=0}^{m_g}$ and $s_i \in \tau_g$ stands for $s_i \in \{s_i \mid (s_i, a_i) \in \tau_g\}$.

Using the properties of logarithms, where $\log_2 x = -\log_2 x^{-1}$ and the logarithm function is monotone, we have:

$$\begin{aligned} \mu(X) &= \min_{g \in G} \sum_{s_i \in \tau_g} \log_2 |A(s_i)| \\ &= \min_{g \in G} \log_2 \prod_{s_i \in \tau_g} |A(s_i)| \\ &= \min_{g \in G} \left[-\log_2 \prod_{s_i \in \tau_g} |A(s_i)|^{-1} \right] \\ &= \min_{g \in G} \left[-\log_2 \prod_{(s_i, a_i) \in \tau_g} \pi_\mu(a_i, s_i) \right] \\ &= \min_{g \in G} \left[-\log_2 \pi_\mu(g) \right] \\ &= -\log_2 \max_{g \in G} \pi_\mu(g) \end{aligned}$$

Thus, we can conclude that $\mu(X)$ is proportional to the probability of finding the most likely solution when sampling a random trajectory from the policy. ■

That is, MUSE is equivalent to the logarithm of the probability of the most likely solution path, with a uniform probability on all actions.

3.2 Minimum Solution Information

Now we extend MUSE to cover the case where the player's policy is not uniform, and the local entropy is not $\log_2 |A(s)|$. For example, suppose $|A(s)| = 2$. This is like a fair coin flip, where getting heads represents taking the action leading to the goal. If the coin is fair, then the entropy of this event is $H = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$ Sh, which equals to the information content of each outcome. However, if the coin is not fair and the probability of heads is 0.1, then the entropy of this event is $H = -0.1 \log_2 0.1 - 0.9 \log_2 0.9 \approx 0.469$ Sh. This results shows that tossing this unfair coin is less uncertain than tossing a fair one, since the outcome is highly unlikely to be heads. Therefore, the outcome of heads is more informative in this event, which is $I = -\log_2 P(\text{Head}) = -\log_2 0.1 \approx 3.322$ Sh. Similarly, if an oracle needs to describe the solution to the player in this case, it also needs to communicate 3.22 Sh to the player.

We generalize MUSE to handle arbitrary policies by defining the minimum solution information (MSI) of a puzzle X as $I^*(X)$ shown in Eq 3.2, which represents the minimum information needed to describe a specific goal from puzzle X.

$$I^*(X) = -\log_2 \max_{g \in G} \pi(g) \tag{3.2}$$

If player's policy is uniform, then $\pi = \pi_\mu$ and $I^*(X) = \mu(X)$.

3.3 Total Solution Information

Similar to ReMUSE, if a puzzle has multiple solutions (goal states) and the player does not have a preference for a specific one, then the oracle does not have to communicate an exact solution, but can minimize the expected communication needed.

Considering the state-space tree of possible moves, minimizing the expected amount of information for every node in the tree requires minimizing the sum of the communication needed at that node and the expected amount of information necessary

for each direct successor. It is worth nothing that after receiving the communication from the oracle, the player's policy will be changed. For example, in MUSE or MSI, following the communication, the player's policy is updated such that $\pi'(a, s) = 1$ if and only if $s_a = T(a, s)$ is on the most likely solution path and 0 otherwise, as shown in Figure 3.1. Therefore, the amount of information required for the communication at the current node can be also calculated by $D_{KL}(\pi', \pi)$.

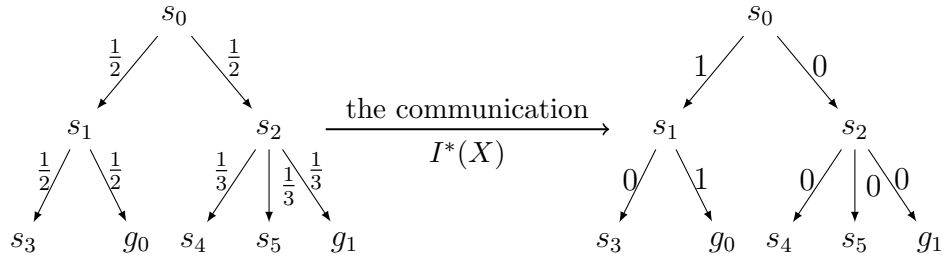


Figure 3.1: How the communication shifts the player's policy in MUSE and MSI

Recall that $\sigma(s)$ is the successor function for s. Then, we define the total solution information (TSI) at state s as

$$\psi(s) = \begin{cases} 0 & \text{if } s \in G \\ \min F(\pi'(s)) & \text{if } A(s) \neq \varnothing \\ \infty & \text{otherwise} \end{cases} \tag{3.3}$$

where F is defined as

$$F(\pi') = D_{\mathrm{KL}}(\pi', \pi) + \mathbb{E}_{\pi'}\left[I\left(\sigma(s)\right)\right] \tag{3.4}$$

$D_{\mathrm{KL}}(\pi', \pi)$ captures the amount of information required for shifting the player's policy from π to π' using KL divergence. $\mathbb{E}_{\pi'}\left[I\left(\sigma(s)\right)\right]$ is the expected amount of information for successors of s under the new policy π'.

After carefully examining $\psi(s)$, we propose the following theorem:

Theorem 2 *$F(\pi'(s))$ is minimal if*

$$\pi'(a, s) = \frac{\pi(a, s) \cdot P_\pi(g_a \mid s_a)}{\sum_{a \in A(s)} \left[\pi(a, s) \cdot P_\pi(g_a \mid s_a)\right]}$$

and

$$\min F(\pi'(s)) = \sum_{a \in A(s)} \pi(a, s) \cdot P_\pi(g_a \mid s_a) \tag{3.5}$$

where $P_\pi(g_a \mid s_a)$ represents the probability of reaching a goal g_a from the successor state $s_a = T(s, a)$ of state s according to the player's original policy π.

Proof. Suppose the probability of reaching a goal g_a from the successor state $s_a = T(s, a)$ is $P(g_a \mid s_a)$. For simplicity, we let $p(a) = \pi'(a, s)$, $q(a) = \pi(a, s)$ and $r(a) = P_\pi(g_a \mid s_a)$, then the corresponding information content of a successor state is $I(s_a) = -\log_2 r(a)$ and

$$\begin{aligned} F(\pi') &= D_{\mathrm{KL}}(\pi', \pi) + \mathbb{E}_\pi \left[I(\sigma(s)) \right] \\ &= \sum_{a \in A} p(a) \log_2 \frac{p(a)}{q(a)} + \sum_{a \in A} p(a) I(s_a) \\ &= \sum_{a \in A} p(a) \log_2 \frac{p(a)}{q(a)} - \sum_{a \in A} p(a) \log_2 r(a) \\ &= \sum_{a \in A} p(a) \log_2 \frac{p(a)}{q(a) r(a)} \end{aligned}$$

Since π' is a policy, we have $\sum_{a \in A} p(a) = 1$. According to the method of Lagrange multiplier, we define the Lagrange function $\mathcal{L}$ as

$$\mathcal{L} = F(\pi') + \lambda (1 - \sum_{a \in A} p(a))$$

Let $\frac{\partial \mathcal{L}}{\partial p(a)} = 0$. We can proceed as follows:

$$\begin{aligned} \frac{\partial \mathcal{L}(\pi', \lambda)}{\partial p(a)} &= 0 \\ \log_2 \frac{p(a)}{q(a) r(a)} + \frac{1}{\ln 2} - \lambda &= 0 \\ p(a) &= q(a) r(a) 2^{\lambda - \log_2 e} \end{aligned}$$

Since $\sum_{a \in A} p(a) = 1$, we have:

$$\begin{aligned} \sum_{a \in A} p(a) &= 1 \\ \sum_{a \in A} q(a) r(a) 2^{\lambda - \log_2 e} &= 1 \\ 2^{\lambda - \log_2 e} &= \frac{1}{\sum_{a \in A} q(a) r(a)} \end{aligned}$$

Therefore,

$$p(a) = \frac{q(a)r(a)}{\sum_{a \in A} q(a)r(a)}$$

which is equivalent to

$$\pi'(a, s) = \frac{\pi(a, s) \cdot P_\pi(g_a \mid s_a)}{\sum_{a \in A} \pi(a, s) \cdot P_\pi(g_a \mid s_a)}$$

Moreover, we have:

$$\begin{aligned} F(\pi') &= \sum_{a \in A} p(a) \log_2 \frac{p(a)}{q(a)r(a)} \\ &= \sum_{a \in A} p(a) \log_2 \frac{1}{\sum_{a \in A} q(a)r(a)} \\ &= \left(- \log_2 \sum_{a \in A} q(a)r(a) \right) \sum_{a \in A} p(a) \\ &= - \log_2 \sum_{a \in A} q(a)r(a) \end{aligned}$$

and

$$\forall\ a \in A,\ \frac{\partial^2 F}{\partial p(a)^2} = \frac{\log_2 e}{p(a)} > 0$$

Therefore, F is convex and its minimum occurs when $\pi'(a, s) = \frac{\pi(a,s) \cdot P_\pi(g \mid s_a)}{\sum_{a \in A} \pi(a,s) \cdot P_\pi(g \mid s_a)}$ ■

This shows that, in TSI, the communication modifies the player's policy by multiplying the probability of taking action a (original policy π) with the probability of reaching a goal g_a from the successor s_a after taking the action a under the original policy π. Figure 3.2 presents an example of the behavior of the communication for TSI within a state-space tree, similar to the one depicted in Figure 3.1. Specifically, For s_0, the original policy π is $[\frac{1}{3}, \frac{2}{3}]$ and the probabilities for reaching g_0 and g_1 from s_1 and s_2 ($P_\pi(g_0 \mid s_1)$ and $P_\pi(g_1 \mid s_2)$) are $\frac{3}{4}$ and $\frac{1}{2}$, respectively. Therefore, after receiving TSI, the policy updates to $[\frac{1}{3} \times \frac{3}{4}, \frac{2}{3} \times \frac{1}{2}]$. Upon normalization, it becomes $[\frac{3}{7}, \frac{4}{7}]$.

Moreover, Equation 3.5 shows that the minimum amount of information required for communicating the solution is proportional to the probability of reaching a goal from the current state s. Thus we can simplify the definition without recursion as

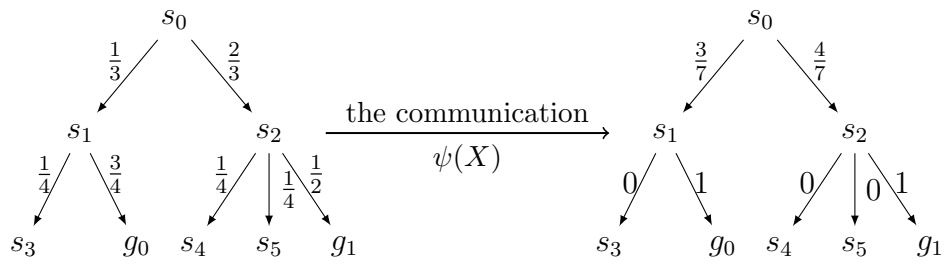


Figure 3.2: How the communication shifts the player's policy in TSI

follows:

$$\psi(X) = -\log_2 \sum_{g \in G} \pi(g) \tag{3.6}$$

Therefore, the TSI of a puzzle, $\psi(X)$, is derived from the logarithm of the sum of the probabilities of reaching goals according to player's policy π.

In contrast, ReMUSE employs a softmin function with base e for calculating the probability distribution of the uniform entropy of the immediate children [20], which effectively shifts the distribution of reaching goals since uniform entropy is derived from the log of base 2 instead of e. Moreover, it is not possible to derive an equation for ReMUSE without recursion. Even if we know a player's policy and the corresponding probability of reaching the goal(s), calculating ReMUSE still requires recursively traversing the state space of the puzzle.

Algorithm	Calculation	Result type	Player policy	Proposed by
MUSE	single solution	entropy	uniform	Chen et al. [20]
ReMUSE	all solutions	relative entropy	uniform	Chen et al. [20]
MSI	single solution	information	any	Shen
TSI	all solutions	information	any	Shen

Table 3.1: Comparison between different algorithms

To summarize, as shown in Table 3.1, MUSE computes the information needed from an oracle to find *a single* solution to a puzzle, given a uniform *a priori* policy. MSI generalizes this to find the information needed for a single solution given any *a*

priori policy. TSI computes the minimum expected information needed to find *any* solution, given any *a priori* policy. TSI can be expressed more simply that ReMUSE, as TSI is the logarithm of the probability of reaching a solution with a given policy.

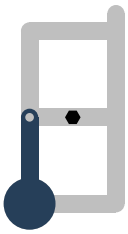
Chapter 4

Inference Rules

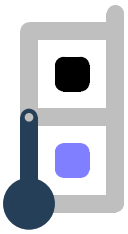
When solving logic puzzle games, players often deduce sets of inference rules, which are guidelines that can solve or simplify the puzzle [20]. Formally, an inference rule is a function that determines whether a specific action should be taken at a given state. When a player's policy incorporates inference rules, the probability of actions that cannot be taken is reduced to zero. Conversely, if an action must be taken, then the probability of that action is one, and the probability of all other actions is zero. For actions in Witness puzzles (a_{up}, a_{right}, a_{down}, a_{left}), each action can be classified as unknown (UNKNOWN), cannot be taken (CANNOT_TAKE), or must be taken (MUST_TAKE). These classifications reflect the probability of solving the puzzle by taking the corresponding actions. For instance, CANNOT_TAKE actions either directly violate the puzzle constraints or lead to situations where constraints cannot be satisfied later. Similarly, necessary actions to satisfy constraints are classified as MUST_TAKE. If there are multiple MUST_TAKE actions, then the puzzle is not solvable at that state since the player can take only one action at each state.

Note that inference rules can be directly incorporated into policies; however, a policy cannot always be converted into an inference rule. It is useful to reason explicitly about inference rules separately from policies, as the policy itself is a probability distribution that may not always be explainable. An example of integrating inference rules into the uniform policy, π_μ is shown in Algorithm 1.

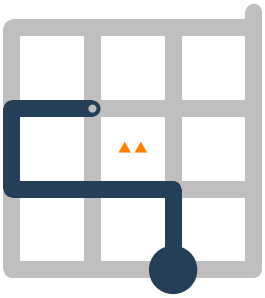
Chen et al. proposed two inference rules for must-cross and separation constraints in their work [20], and another one for triangle constraints in a follow-up thesis [23]. We briefly explain these rules in Figure 4.1.



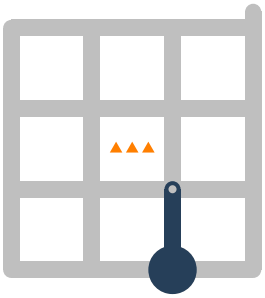
(a) a_{right} is MUST_TAKE, since a must-cross constraint is adjacent to the current junction



(b) a_{right} is MUST_TAKE, since two separation constraints are adjacent to each other and the path must go between them



(c) a_{right} is MUST_TAKE, since the double triangle cannot be satisfied later by taking a_{up}



(d) a_{right} is CANNOT_TAKE, since the triple triangle cannot be satisfied later by taking it

Figure 4.1: Inference rules proposed in previous studies [20, 23]

4.1 Additional Inference Rules

To expand the dataset for testing algorithms and study the effect of different player models, we propose the following additional inference rules that could be applied broadly across Witness puzzles containing different types of constraints.

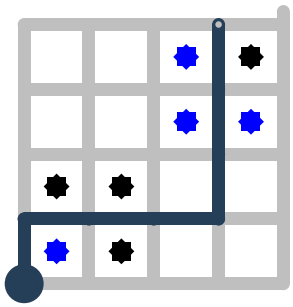
We derived these inference rules during the development of the puzzle editor introduced in Chapter 5. A group of lab members was tasked informally with designing puzzles with high TSI. However, we discovered that these puzzles were not sufficiently challenging, prompting us to formalize the inferences we employed as inference rules.

Algorithm 1 π_μ with inference rules

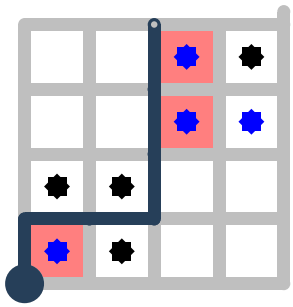
function $\pi_\mu(a,\ s)$
  Initialize empty map M
  for each $a' \in A(s)$ **do**
    $M[a'] =$ UNKNOWN
  end for
  for each $f \in \{\text{All inference rules}\}$ **do**
    if $f(a, s) =$ CANNOT_TAKE **then**
      return 0
    else if $f(a, s) =$ MUST_TAKE **then**
      return 1
    end if
    for each $a' \in A(s)$ **do**
      if $M[a'] \neq$ UNKNOWN **and** $M[a'] \neq f(a', s)$ **then**
        return 0 ▷ *conflicts between rules*
      end if
      $M[a'] \leftarrow f(a', s)$
    end for
  end for
  $A' \leftarrow \emptyset$
  for each $a' \in A(s)$ **do**
    if $M[a'] =$ MUST_TAKE **and** $a' \neq a$ **then**
      return 0
    else if $M[a'] =$ CANNOT_TAKE **then**
      continue
    end if
    $A' \leftarrow A' \cup \{a'\}$ ▷ *append* a' *to* A'
  end for
  return $|A'|^{-1}$
end function

4.1.1 Along-the-path Rule (APR)

For any state that forms a sub-sequence of a solution sequence, meaning it is part of the solution, region constraints adjacent to the same side of the path should not be violated. They are guaranteed to be in the same region no matter the rest of the solution, thus, they can be evaluated immediately to determine whether a partial solution is valid. For example, in Figure 4.2a, there are 8 star constraints adjacent the path, and none of them are violated. In contrast, there are 3 violated star constraints adjacent to the path in Figure 4.2b, thus this path is not part of any solution. The implementation of this inference rule is provided in Algorithm 2.



(a) A puzzle with a path that does not violate any region constraints next to it



(b) The same puzzle as (a) but with a path that violates star constraints highlighted in red on its right-hand side

Figure 4.2: Example puzzles with similar paths

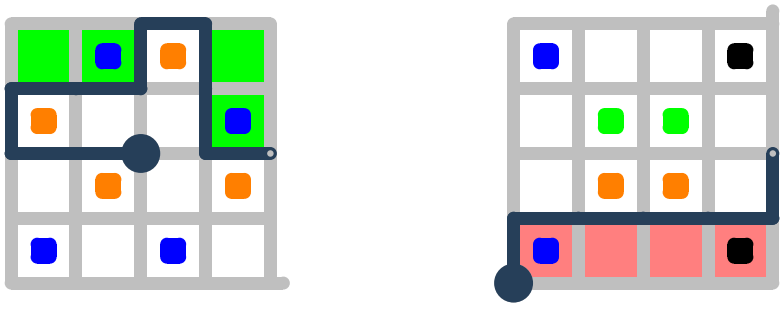
4.1.2 Region-completion Rule (RCR)

When the path intersects the boundary of the puzzle twice, it forms a new closed region. A closed region that does not contain the end junction cannot be changed, as shown in Figure 4.3. Every constraint within a completed region must be satisfied; otherwise, these constraints cannot be satisfied later. Algorithm 3 provides the details of this inference rule.

Algorithm 2 Along-the-path Rule

```
function PathTest(s)
    ▷ Record blocks on the left-hand side and right-hand side of the path that are
      not in a completed region, into lists lhs and rhs, respectively. ◁
    lhs, rhs ← GetLeftRightBlocks(s)
    if ∃c_region ∈ lhs and c_region is not satisfied then
        return false
    end if
    if ∃c_region ∈ rhs and c_region is not satisfied then
        return false
    end if
    return true
end function

function AlongThePathRule(a, s)
    Apply Action a to s
    r ← PathTest(s)
    Undo Action a
    if r = true then
        ▷ cannot be determined by this rule ◁
        return UNKNOWN
    else
        return CANNOT_TAKE
    end if
end function
```



(a) A puzzle with 2 completed regions highlighted in green, where all separation constraints inside them are satisfied

(b) A puzzle with 1 completed region highlighted in red, where all separation constraints inside it are not satisfied

Figure 4.3: Example puzzles with completed regions highlighted

Algorithm 3 Region-completion Rule

```
function RegionTest(s)
    R ← GetCompletedRegions(s)
    for each r ∈ R do                                  ▷ Check each completed region
        if ∃c_region ∈ r  and  c_region is not satisfied then
            return false
        end if
    end for
    return true
end function

function RegionCompletionRule(a, s)
    Apply action a to s
    if a region is completed then
        r ← RegionTest(s)
    else
        r ← true
    end if
    Undo action a
    if r = true then                                   ▷ Cannot be determined by this rule
        return UNKNOWN
    else
        return CANNOT_TAKE
    end if
end function
```

Chapter 5

Integration with EPCG

Procedural content generation (PCG) is a technique that involves the algorithmic creation of content rather than manual creation, aiming to reduce content production costs and enable the production of vast, diverse, and dynamic environments, characters, and scenarios [4]. EPCG extends this concept by striving to explore all possible permutations and variations within a defined set of parameters. An EPCG procedure requires two inputs shown in Algorithm 4 [10]: a generator and an evaluator. The generator accepts a problem description and generates states according to a predefined generation algorithm. In contrast, the evaluator takes a state and compute a numerical score for it, to determine of which states worth attention. Moreover, to ensure the production of all possible content, the generator employs a *rank* function that convert integers into content, an *unrank* function that convert content back into integers [29], and a *maxRank* function that compute the size of the complete state space.

The Windmill [30] is a website where users can design and publish Witness puzzles, offering a Witness puzzle editor[1]. However, this editor only supports solution correctness analysis. We have developed a novel Witness puzzle editor shown in Figure 5.1 that not only adheres to the constraints listed in Table 2.1, but also provides comprehensive puzzle analysis and querying capabilities for EPCG. This chapter details the puzzle editor's functionalities.

[1]https://windmill.thefifthmatt.com/build

Algorithm 4 EPCG procedure [10]

Input: Generator G, Evaluator E
Output: best puzzle b
$b \leftarrow G.\text{unrank}(0)$
for $i = 1, \ldots, G.\text{maxRank}() - 1$ **do**
    $t \leftarrow G.\text{unrank}(i)$
    if $E(t) > E(b)$ **then**
        $b = t$
    end if
end for

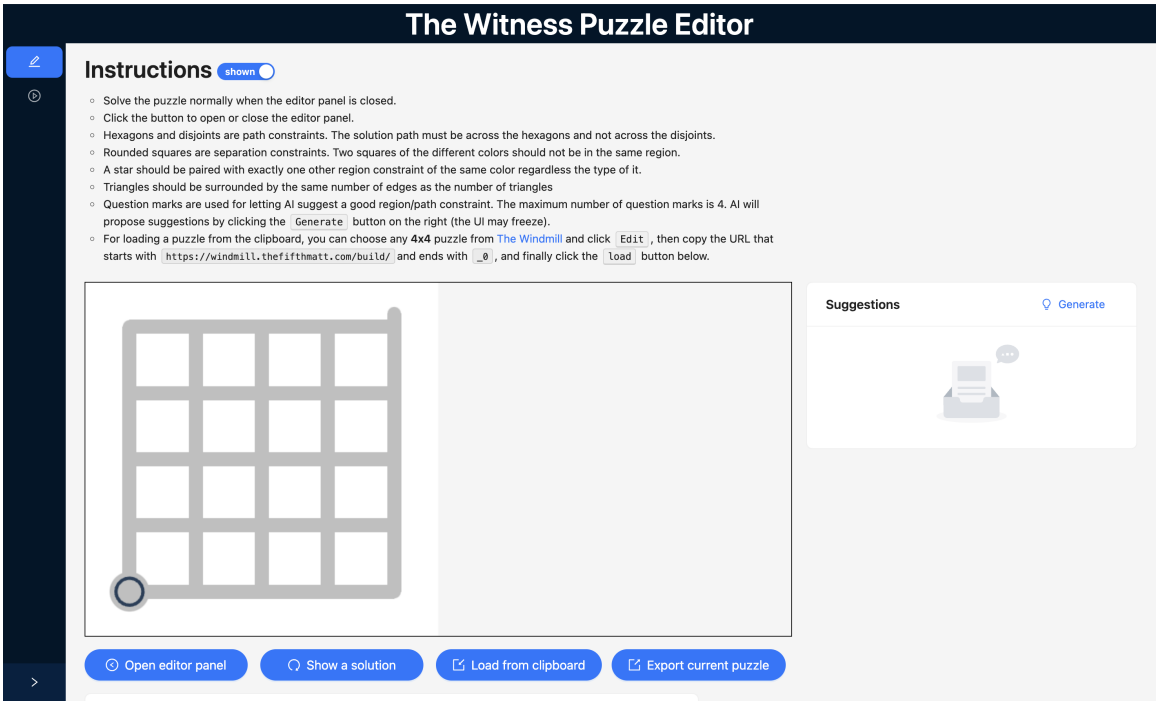
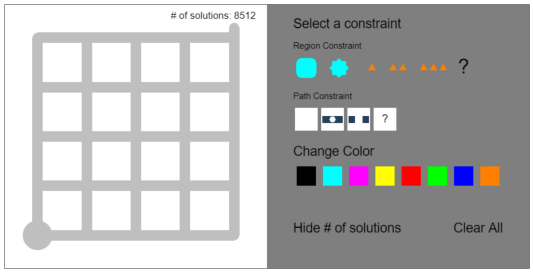


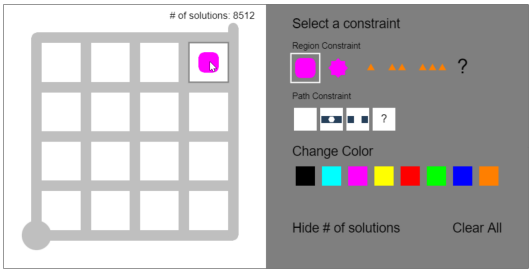
Figure 5.1: Screenshot of the web page of the puzzle editor

5.1 Designing Puzzles

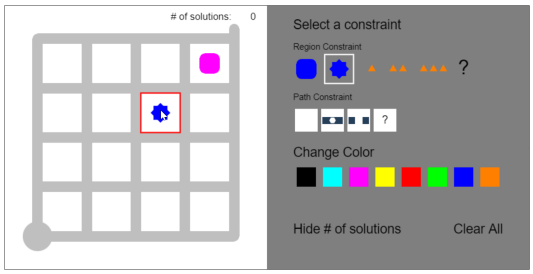
By default, the web page provides an empty 4×4 puzzle, as shown in Figure 5.2a. The designer can toggle the editor panel using the button or the shortcut key e. Using the editor panel, the designer can select a region or path constraint, change its color (for separation and star constraints only), and place it on the puzzle panel, as shown in Figure 5.2b. Upon updating the puzzle panel, the number of solutions will be updated accordingly. If placing the constraint makes the puzzle unsolvable, the indicating frame will turn red, as shown in Figure 5.2c. Additionally, the editor can display a solution to the current puzzle as needed, allowing the designer to iterate through each one by clicking the corresponding button. The editor also allows designers to import puzzles from *The Windmill*. Once the design is finalized, the puzzle can be transferred back to *The Windmill* for publication.



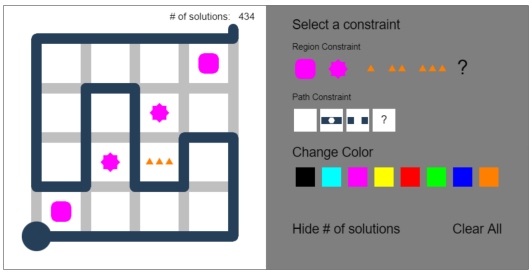
(a) The default puzzle editor



(b) The puzzle editor when placing a new constraint



(c) The puzzle editor when the new puzzle has no solution



(d) The puzzle editor when it displays the solution

Figure 5.2: Screenshots of the main part of the puzzle editor

5.2 Querying EPCG

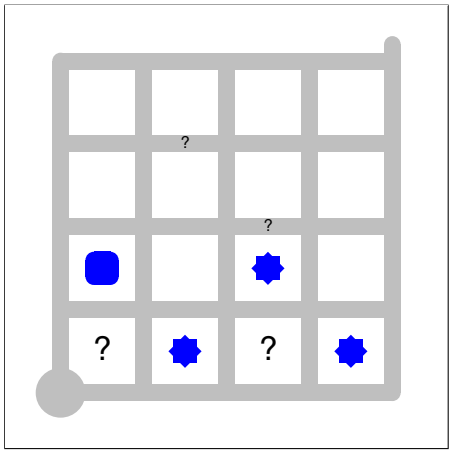
The editor panel provides special placeholder constraints for both the path constraint and the region constraint, indicated by a question mark icon, as shown in Figure 5.3a. If no constraint is placed on placeholders, then this puzzle has 1919 solutions with TSI of 0.08 Sh. The designer can place up to 4 placeholder constraints on the puzzle. Upon clicking the *Generate* button, the generator will exhaustively attempt all possible constraints and colors at the specified locations. Specifically, if there is a placeholder for path constraints, there are three placement options: no constraint, must-cross constraint, and cannot-cross constraint. For placeholders involving region constraints, the generator will attempt to apply either no constraint, any of three triangle constraints, a separation constraint, or a star constraint. Furthermore, since separation and star constraints can have varying colors, the number of new colors introduced is two if the puzzle initially contains no colors and there are at least two placeholder constraints. If the puzzle initially contains more than two colors, no new colors will be added. In all other cases, exactly one new color will be introduced. Therefore, the total number of placement configurations is given by:

$$(3 + 2 \times n_{\text{color}})^{n_{\text{region}}} \times 3^{4-n_{\text{region}}}$$

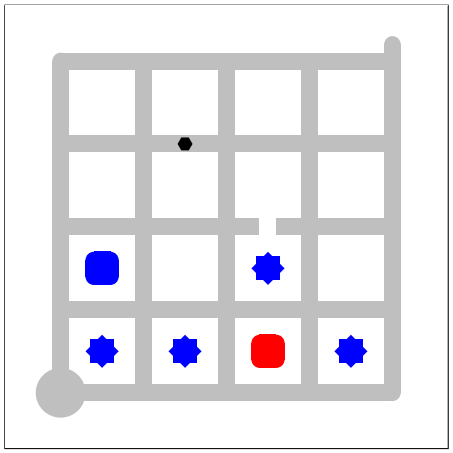
Using the puzzle in Figure 5.3a as an example, the total number of configurations is calculated as $(3 + 2 \times 2)^2 \times 3^2 = 441$ (including unsolvable puzzles).

The evaluator will assess each placement configuration using the TSI. The time required for this process varies depending on the number of placeholder constraints and the performance of the designer's CPU, as the computation is executed locally in the browser with parallelization. Typically, the process should complete within one minute. Once the generation process is completed, the editor returns the resulting solvable puzzles, sorted by TSI, along with suggestions based on puzzle statistics such as the number of constraints, the number of solutions, and TSI values. The designer can review each suggested puzzle, select a preferred option, and proceed with further

design refinements. Figure 5.3b shows the one generated puzzle for the puzzle in Figure 5.3a. The generated puzzle has 24 solutions with TSI of 7.33 Sh.



(a) The puzzle with placeholders in the editor



(b) The puzzle generated by EPCG procedure

Figure 5.3: EPCG example puzzle

The editor has been useful for informally testing these constraints and the effectiveness of TSI. Future research could focus on formal user studies with designers.

Chapter 6

Experiments

In this chapter, we present a comprehensive analysis of the behavior of proposed algorithms and inference rules. Our experiments aim to address questions derived from the study and uncover insights into the factors that influence puzzle difficulty and player engagement by leveraging various collections of puzzles. This chapter details the experimental setup, thorough comparisons of different algorithms, inference rules, and the machine learning policy, laying the groundwork for a deeper understanding of player engagement modeling in Witness puzzles.

6.1 Experiment Setup

The Windmill [30] allows users to upvote or downvote published Witness puzzles based on their enjoyment. We assume that users on the website are advanced players of *The Witness* and generally upvote challenging puzzles while downvoting trivial ones [20]. To assess the relationship between player enjoyment and puzzle difficulty, as well as the quality of difficulty assessment, we compare the Pearson Correlation Coefficients (r-values) derived from simple linear regression of puzzle difficulty and the number of upvotes. The corresponding probabilities of the F-statistic (p-values) represent the likelihood of failing to reject the null hypothesis; in other words, they indicate that the variable has no effect. Additionally, the number of upvotes for each puzzle is normalized based on the linear relationship between upvotes and timestamps

within the test set to account for temporal variations, as the number of active users on the website decreases over time. Furthermore, for reproducibility, we implemented MUSE, ReMUSE and their inference rules individually and assumed player policies to be uniform after applying inference rules.

The complete puzzle dataset comprises every 4×4 puzzle prior to November 29, 2022, consistent with previous studies [20]. This fixed size limits the maximum difficulty and eliminates the variable of puzzle size. We excluded all puzzles that are included in the original game or duplication, applying different filters to obtain subsets of puzzles for various experiments.

6.2 Comparing Different Algorithms

The first question to answer is that how TSI performs comparing to ReMUSE on different puzzle datasets, therefore we compose the first test set that is identical to the one used in the previous study [20]. It contains 104 puzzles that incorporate only separation and path constraints. Table 6.1 shows the correlation computed from different algorithms, and Figure 6.1 provides details of the difficulty score of each puzzle and its linear correlation with the number of upvotes. In this case MSI is identical to MUSE, because a uniform default policy is assumed, so we don't report results for MSI separately. TSI and ReMUSE, however, are not the same, but TSI is very similar to ReMUSE on this test set.

6.3 Applying Additional Inference Rules

The second question we want to address is that how the quality of the player model affects the puzzle difficulty calculation and its correlation with player enjoyment. Since we proposed more inference rules that could be used on broader Witness puzzles, we make the test sets containing different types of constraints. The baseline inferences we use are the same inference rules used in previous study [23] that are briefly explained

Approach	Correlation	p-value
MUSE (Reported)	0.410	1.10×10^{-5}
MUSE (Reproduction)	0.412	1.53×10^{-5}
ReMUSE (Reported)	0.570	1.57×10^{-10}
ReMUSE (Reproduction)	0.583	1.02×10^{-10}
TSI	0.563	5.78×10^{-10}

Table 6.1: Correlation and p-value for different approaches evaluated on the dataset used in the previous study [20]

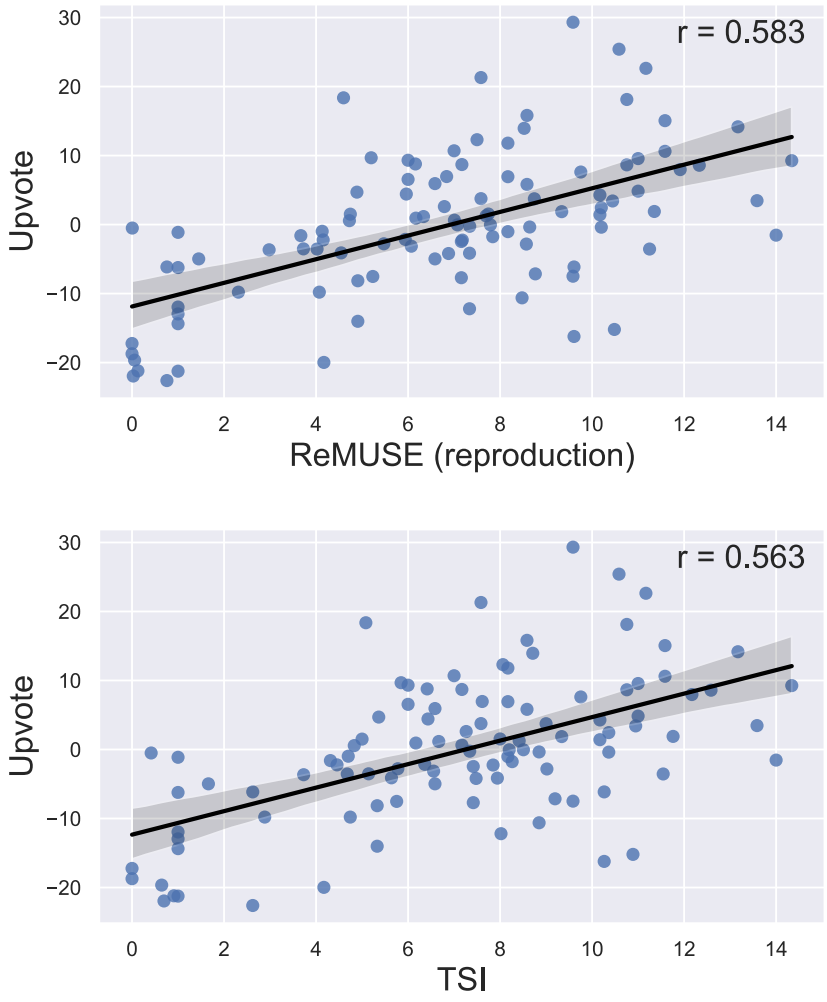


Figure 6.1: The number of upvote of puzzles from the dataset used by the previous study [20] and their difficulty measured by ReMUSE and TSI respectively

in Figure 4.1. The second test set (T-only set) comprises 179 puzzles, each containing at least one triangle constraint and possibly one or both path constraints. The first half of Table 6.2 provides details of how inference rules affect the correlation. The absolute correlations without applying any additional inference rule are lower than in previous test sets, as expected [20]. More importantly, adding the along-the-path rule improves the correlation, while adding the region-completion rule reduced correlation. This suggests that players tend to use along-the-path rules when the puzzle contains triangles.

For comparison, we conduct a follow-up experiment on the complementary set of the second test set (non-T set): the set of 226 puzzles that do not contain triangle constraints (they contain separation constraints, star constraints, and path constraints). The second half of Table 6.2 demonstrates that incorporating more inference rules, resulting in a more informed model, does not necessarily lead to a better correlation between difficulty scores and player upvotes. One might infer that users on *The Windmill* are not necessarily using all of the inference rules that we developed. However, a puzzle we created using procedural tools, which we discuss further at the end of the experimental results, suggests a different conclusion: that good puzzles will have inference rules that help guide players to the solution. Regardless, it is crucial to model player knowledge accurately when measuring puzzle difficulty.

6.4 Difficulty Calculated by ML Policy

The last question we want to address is that how policies from machine learning (ML) compare to our manually created inference rule. Since MSI and TSI are derived from the probability of reaching goals, it is possible to use a ML policy instead of manually creating inference rules. Lelis et al.[5] utilized a modified Bootstrap model [31], which takes a puzzle instance as input and returns a policy, to be used as a policy in Levin tree search [32] for generating an equidistant curriculum of specific types of Witness puzzles. This curriculum is designed to ease player learning. The model was trained

Puzzle Set	Inference Rule	Correlation	p-value
T-only	Baseline	0.366	4.67×10^{-7}
T-only	APR	0.433	1.40×10^{-9}
T-only	RCR	0.252	6.58×10^{-4}
T-only	Both	0.322	1.09×10^{-5}
non-T	Baseline	0.515	1.03×10^{-16}
non-T	APR	0.494	2.62×10^{-15}
non-T	RCR	0.497	1.71×10^{-15}
non-T	Both	0.479	2.16×10^{-14}

Table 6.2: Correlation and p-value for for different player models evaluated on triangle-only and non-triangle Witness puzzles

on Witness puzzles that contain only separation constraints with two colors, referred to as Black and White Square (BWS) puzzles. The authors shared the source code and corresponding datasets with us, and we trained their model independently. The complete dataset contains 24 BWS puzzles. We collect all solutions for each of them. To compute the MSI, we calculate the logarithm of the probability of reaching the most likely solution as dictated by the model's policy using Equation 3.2. For the TSI, we calculate the logarithm of the sum of the probabilities of reaching each solution (goal state) using Equation 3.6. The baseline inference rule in these puzzles is the separation rule (SR), explained in Figure 4.1b, since other inference rules proposed previously [23] do not affect puzzles in this test set. The results shown in Table 6.3 indicate that the MSI and TSI derived from the ML policy are very close to each other. Notably, the TSI derived from the ML policy surpasses that derived from inference rules on these simple puzzles. Additionally, the results show that applying extra inference rules worsens the correlation, consistent with the previous experiment.

To verify that the ML policy learns inferences, we conduct a supplementary experiment. We first collect states along all solution sequences of each puzzle that have actions pruned by the separation rule shown in Figure 4.1b. We then calculate the

Player Model	MSI		TSI	
	Correlation	p-value	Correlation	p-value
SR (baseline)	0.521	8.99×10^{-3}	0.844	2.23×10^{-7}
APR	0.401	5.23×10^{-2}	0.720	7.20×10^{-5}
RCR	0.405	4.94×10^{-2}	0.449	2.77×10^{-2}
SR and APR	0.341	1.03×10^{-1}	0.769	1.15×10^{-5}
SR and RCR	0.490	1.51×10^{-2}	0.827	6.26×10^{-7}
APR and RCR	0.377	6.92×10^{-2}	0.739	3.69×10^{-5}
All Inference Rules	0.299	1.55×10^{-1}	0.771	1.02×10^{-5}
ML Policy	0.751	2.35×10^{-5}	0.787	5.16×10^{-6}

Table 6.3: Correlation and p-value for MSI and TSI with different player models evaluated on BWS Witness puzzles [5]

expected probability of the ML policy taking CANNOT_TAKE actions. Additionally, we calculate the expected probability of taking the correct action, as the separation rule selects actions that are MUST_TAKE rather than explicitly identifying other actions as CANNOT_TAKE. In general, if the policy learns the inference that prunes actions, then the probability of taking these pruned actions should be close to 0. Similarly, if the policy learns the inference that selects actions, then the probability of taking these selected actions should be close to 1. Furthermore, we collect states along all solution sequences of every puzzle that the separation rule cannot be applied but other two new inference rules can be applied respectively. The results are provided in Table 6.4. They show that the ML model adequately learned the along-the-path rule; however, it did not learn the region-completion rule effectively, which is also reflected in Table 6.3. Consequently, we conclude that the ML model is capable of learning inference, which is also suggested by Stevens et al. [33]. Moreover, they can be effectively utilized in both MSI and TSI as well.

Method	Probability
uniform	0.497
policy	0.002
inference	0.000

(a) Method and expected probabilities of taking CANNOT_TAKE actions pruned by the separation rule in total of 11050 states

Method	Probability
uniform	0.497
policy	0.994
inference	1.000

(b) Method and expected probabilities of taking MUST_TAKE actions selected by the separation rule in total of 11050 states

Method	Probability
uniform	0.465
policy	0.091
inference	0.000

(c) Method and expected probabilities of taking CANNOT_TAKE actions pruned by the along-the-path rule in total of 4518 states

Method	Probability
uniform	0.418
policy	0.331
inference	0.000

(d) Method and expected probabilities of taking CANNOT_TAKE actions pruned by the region-completion rule in total of 2998 states

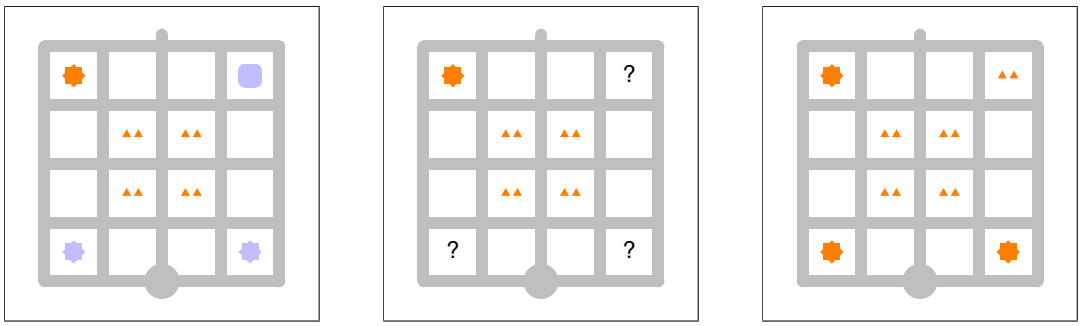
Table 6.4: The expected probabilities of taking CANNOT_TAKE actions and MUST_-TAKE actions

6.5 EPCG Verification

As an independent experiment, in March 2024, we selected the latest puzzle[1] (Figure 6.2a) posted on *The Windmill* and used our editor to generate a similar puzzle[2] with a higher TSI (Figure 6.2c). We then published it alongside the original one. Three months later, the original puzzle had 207 solves and 4 upvotes, while our puzzle had only 113 solves and 5 upvotes.

In going back and trying to solve this puzzle again, we find that it is difficult for us to solve, because there are very few inferences we can make to guide our own problem-solving process. We end up solving it more by random exploration than by reasoned exploration. TSI is high because no inference rules reduce the search. Our experience is reflected in the relative low total number of solves.

We must be careful not to draw too broad of conclusions from this single puzzle. This result suggests that instead of just looking for puzzles with high TSI, we may be interested in finding puzzles that have a large difference in TSI depending on the inference rules used. This would indicate puzzles that can be more easily solved if you know the inference rule, but are hard otherwise. This approach has been used recently in curriculum generation [34]. Regardless, more work is needed to understand the interplay between TSI, player models, and puzzle enjoyment.



(a) Original puzzle with TSI of 6.67 Sh (b) Puzzle with placeholders in the editor (c) Generated puzzle using our editor with TSI of 8.58 Sh

Figure 6.2: Screenshot of original puzzle and its harder alternatives

[1] https://windmill.thefifthmatt.com/zmxfp88
[2] https://windmill.thefifthmatt.com/7nw1t3r

Chapter 7

Conclusions and Future Work

In this thesis, we generalized entropy and solution information and proposed new algorithms, MSI and TSI, as general metrics for measuring puzzle difficulty. These novel approaches facilitate analysis involving both inference rules and player policies. After validating these metrics with two additional inference rules through extensive experiments on various puzzle sets and player models, we conclude that our approach is as effective as previous methods while offering improved mathematical foundation and generality. Furthermore, we demonstrated that accurate player modeling is essential. Finally, we developed a Witness puzzle editor that supports EPCG queries.

Future research could focus on studying other region constraints in *The Witness*, such as Tetris-shaped constraints, which are also crucial mechanics of the game. This would involve enhancing the player model and conducting experiments to test the TSI.

Additionally, incorporating ML techniques to create explainable inference rules for puzzles and their solutions, as suggested by previous studies [33], represents another promising area of exploration. The effects of these learned inference rules on the player model could also be investigated.

Moreover, the data from *The Windmill* presents challenges due to noise and outliers, including puzzles from the original game and spam test puzzles, which impact experimental outcomes. Future work could conduct a formal user study to develop

a clean dataset and then examine the correlation between TSI and player enjoyment based on it.

Finally, this approach as a generic algorithm could be extend to different puzzle games using corresponding player policies to evaluate its broader applicability and effectiveness.

References

[1] J. Shen and N. R. Sturtevant, "Generalized entropy and solution information for measuring puzzle difficulty," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2024.

[2] M. van Kreveld, M. Loffler, and P. Mutser, "Automated puzzle difficulty estimation," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, Aug. 2015. DOI: 10.1109/cig.2015.7317913.

[3] A. Frazer, "Towards better gameplay in educational computer games: A phd thesis," Ph.D. dissertation, University of Southampton, Nov. 2010. [Online]. Available: https://eprints.soton.ac.uk/172421/.

[4] B. De Kegel and M. Haahr, "Procedural puzzle generation: A survey," *IEEE Transactions on Games*, vol. 12, no. 1, pp. 21–40, 2020. DOI: 10.1109/TG.2019.2917792.

[5] L. H. S. Lelis, J. G. G. V. Nova, E. Y. C. Chen, N. R. Sturtevant, C. D. Epp, and M. Bowling, "Learning curricula for humans: An empirical study with puzzles from the witness," in *International Joint Conference on Artificial Intelligence*, 2022. [Online]. Available: https://www.ijcai.org/proceedings/2022/538.

[6] S. Abuhamdeh and M. Csíkszentmihályi, "The importance of challenge for the enjoyment of intrinsically motivated, goal-directed activities," *Personality & social psychology bulletin*, 2012. DOI: 10.1177/0146167211427147.

[7] CandyCane Software, *Fling!* 2011. [Online]. Available: https://www.candycaneapps.com/fling/.

[8] N. Sturtevant, "An argument for large-scale breadth-first search for game design and content generation via a case study of fling!" In *AI in the Game Design Process (AIIDE workshop)*, 2013, pp. 28–33.

[9] P. Jarušek and R. Pelánek, "Difficulty rating of sokoban puzzle," in *Proceedings of the 2010 Conference on STAIRS 2010: Proceedings of the Fifth Starting AI Researchers' Symposium*, NLD: IOS Press, 2010, pp. 140–150, ISBN: 9781607506751.

[10] N. Sturtevant and M. Ota, "Exhaustive and semi-exhaustive procedural content generation," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 14, no. 1, pp. 109–115, Sep. 2018. DOI: 10.1609/aiide.v14i1.13020. [Online]. Available: https://ojs.aaai.org/index.php/AIIDE/article/view/13020.

[11] Noumenon Games, *Snakebird*, 2015. [Online]. Available: https://store.steampowered.com/app/357300/Snakebird/.

[12] N. R. Sturtevant, N. Decroocq, A. Tripodi, and M. Guzdial, "The unexpected consequence of incremental design changes," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2020, pp. 130–136. [Online]. Available: http://www.cs.ualberta.ca/~nathanst/papers/sturtevant2020incremental.pdf.

[13] ThinkFun Games, 1996. [Online]. Available: https://www.thinkfun.com/rush-hour-online-play/.

[14] A. M. Smith, E. Andersen, M. Mateas, and Z. Popović, "A case study of expressively constrainable level design automation tools for a puzzle game," in *Proceedings of the International Conference on the Foundations of Digital Games*, ser. FDG'12, ACM, May 2012. DOI: 10.1145/2282338.2282370.

[15] M. J. Nelson and A. M. Smith, "Asp with applications to mazes and levels," in *Computational Synthesis and Creative Systems*. Springer International Publishing, 2016, pp. 143–157, ISBN: 9783319427164. DOI: 10.1007/978-3-319-42716-4_8.

[16] B. Fatemi, S. M. Kazemi, and N. Mehrasa, "Rating and generating sudoku puzzles based on constraint satisfaction problems," *International Journal of Computer and Information Engineering*, vol. 8, no. 10, pp. 1816–1821, 2014. [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fc78d54ae5d5234c9fb229d6a2cd2ef5af181e70.

[17] ZeptoLab UK Limited, *Cut the rope*, 2010. [Online]. Available: https://www.cuttherope.net/.

[18] Rovio Entertainment Corporation, *Angry birds*, 2009. [Online]. Available: https://www.angrybirds.com/.

[19] S. Roohi, C. Guckelsberger, A. Relas, H. Heiskanen, J. Takatalo, and P. Hämäläinen, "Predicting game difficulty and engagement using ai players," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, no. CHI PLAY, Oct. 2021. DOI: 10.1145/3474658. [Online]. Available: https://doi.org/10.1145/3474658.

[20] E. Y. C. Chen, A. White, and N. R. Sturtevant, "Entropy as a measure of puzzle difficulty," *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, no. 1, pp. 34–42, Oct. 2023, ISSN: 2326-909X. DOI: 10.1609/aiide.v19i1.27499.

[21] J. A. del Solar-Zavala, H. Schaa, and N. A. Barriga, "Using entropy for modeling difficulty in the asteroid escape sliding puzzle," in *2023 IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, IEEE, Dec. 2023. DOI: 10.1109/chilecon60335.2023.10418707.

[22] Thekla, Inc., *The witness*, Jan. 2016. [Online]. Available: https://store.steampowered.com/app/210970/The_Witness/.

[23] Chen, You Chen Eugene, "Entropy as a measure of puzzle difficulty," M.S. thesis, University of Alberta, 2023. DOI: 10.7939/R3-9RSM-EG27.

[24] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, Jul. 1948, ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[25] International Organization for Standardization, *Quantities and units – part 13: Information science and technology*, Apr. 2008. [Online]. Available: https://www.iso.org/standard/31898.html.

[26] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[27] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup, "Procedural content generation for games: A survey," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 9, no. 1, Feb. 2013, ISSN: 1551-6857. DOI: 10.1145/2422956.2422957. [Online]. Available: https://doi.org/10.1145/2422956.2422957.

[28] A. Summerville *et al.*, "Procedural content generation via machine learning (pcgml)," *IEEE Transactions on Games*, vol. 10, no. 3, pp. 257–270, 2018. DOI: 10.1109/TG.2018.2846639.

[29] W. Myrvold and F. Ruskey, "Ranking and unranking permutations in linear time," *Information Processing Letters*, vol. 79, no. 6, pp. 281–284, 2001, ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(01)00141-7.

[30] M. Gruen, *The windmill*, May 2016. [Online]. Available: https://windmill.thefifthmatt.com.

[31] S. Jabbari Arfaee, S. Zilles, and R. C. Holte, "Learning heuristic functions for large state spaces," *Artificial Intelligence*, vol. 175, no. 16–17, pp. 2075–2098, Oct. 2011, ISSN: 0004-3702. DOI: 10.1016/j.artint.2011.08.001.

[32] L. Orseau, L. H. S. Lelis, T. Lattimore, and T. Weber, "Single-agent policy tree search with guarantees," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, Montréal, Canada: Curran Associates Inc., 2018, pp. 3205–3215.

[33] J. Stevens, V. Bulitko, and D. Thue, *Solving witness-type triangle puzzles faster with an automatically learned human-explainable predicate*, 2023. DOI: 10.48550/ARXIV.2308.02666.

[34] Y. Mahmoud and N. R. Sturtevant, "Using epcg for designing a hexagon tangram puzzle," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2024.