Using Rating Arrays to Estimate Score Distributions for Player-versus-Level Matchmaking

Anurag Sarkar
Northeastern University
sarkar.an@husky.neu.edu

Seth Cooper
Northeastern University
se.cooper@northeastern.edu

ABSTRACT

Rating systems (like Elo and Glicko-2) have previously been used for predicting the expected score that a player will achieve on a level. We present an approach that predicts not a single score, but an approximate cumulative distribution function over possible scores. This approach assigns each level an array of multiple ratings for different score thresholds. Our long-term goal is twofold: first, to dynamically change level difficulty for each player by using this CDF to tailor the target score required to complete a level; second, in human computation games (HCGs), to identify players capable of setting new high scores that could correspond to improved solutions to underlying tasks. To move towards this goal, we explore the rating array approach using two datasets: one gathered from the HCG *Paradox*, and one generated from idealized players and levels. We examine the accuracy of the CDF and the expected scores it predicts, as well as its use in serving levels to players who could set new high scores.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

rating systems; matchmaking; human computation games; score prediction; dynamic difficulty adjustment

ACM Reference Format:
Anurag Sarkar and Seth Cooper. 2019. Using Rating Arrays to Estimate Score Distributions for Player-versus-Level Matchmaking. In *The Fourteenth International Conference on the Foundations of Digital Games (FDG '19), August 26–30, 2019, San Luis Obispo, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3337722.3337758

1 INTRODUCTION

Human computation games (HCGs) are games that seek to leverage the collective abilities of humans in order to help solve problems that are computationally intractable or cannot readily be solved using automated methods. Previous work [6] has demonstrated the feasibility of using rating systems for matchmaking between players and levels of HCGs as a means of overcoming the difficulty balancing challenges inherent within such games. These challenges stem from levels in HCGs not being suitable to modification for difficulty adjustment since they model real world problems and any changes could compromise the validity of a level's link to the underlying problem. More recent work [25, 28] has attempted to circumvent these issues by assigning Glicko-2 ratings to players based on skill and to levels based on difficulty and then using a matchmaking algorithm to pair players and levels of comparable skills and difficulties respectively. This achieves difficulty balancing by modifying the order in which levels are served rather than by modifying the levels themselves. While these methods have shown improved player engagement, they require fixing a single target score cutoff for each level to determine if a player wins or loses based on if they score higher or lower than this cutoff.

In this work, we introduce the concept of rating arrays to avoid having to use a single predetermined score cutoff for a level to decide win/loss outcomes. The idea behind this approach is to treat each level not as a single player, as in past work, but as a group of multiple players, each representing a different score threshold. Assigning a rating to each such threshold for a level (using the rating array), rather than to the level as a whole, allows us to perform matchmaking not just between players and levels as before, but between players and *(level, score threshold)* pairs. This further enables us to match players with levels at dynamically assigned thresholds based on player skill rather than with levels at the same fixed cutoffs for all players, as in past work. Additionally, having an assigned rating for each score threshold of a level allows us to model a cumulative distribution function (CDF) over possible scores that the player can achieve on that level. This gives us the added ability of predicting not just a single score that a player is likely to achieve on a level, but their probability of achieving any possible score. These benefits can prove particularly useful in the context of HCGs where new high scores for levels represent novel and/or improved solutions to the underlying problems. Furthermore, combining the rating array method with past approaches for ratings-based matchmaking [25, 28] in HCGs could also identify players able to set new high scores and match them with levels at appropriate score thresholds while additionally performing dynamic difficulty adjustment.

In order to test these potential benefits and evaluate the accuracy of the proposed CDF, we applied the rating array approach to match data from the HCG *Paradox* (shown in Figure 1) as well as to synthetically generated match data. This work contributes 1) a description of the new rating array-based approach to player-versus-level matchmaking and 2) a three-part evaluation of the approach in terms of its accuracy and usefulness for player-versus-level matchmaking.

FDG '19, August 26–30, 2019, San Luis Obispo, CA, USA

ACM ISBN 978-1-4503-7217-6/19/08…$15.00
https://doi.org/10.1145/3337722.3337758

2 BACKGROUND

2.1 Player Rating Systems

Rating systems such as Elo [9], Glicko [10] and Glicko-2 [11] assign skill-based ratings to players in order to produce fair matches by pairing up players of similar skill. While primarily employed in competitive player-versus-player settings like chess, such systems have found increasing use in other situations involving pairwise [13] or setwise [27] comparisons.

Additionally, these systems have also been used in the player-versus-level (PvL) domain for matching players with levels of a certain difficulty rather than with other players of a certain skill. For example, the platformer *Jumpcraft* [29] applies Microsoft's TrueSkill [14] rating system on the outcomes of player attempts for ordering levels according to difficulty. In such PvL usage, each level is assigned a single rating indicating how difficult it is to complete that level. In this work, we extend this by assigning multiple ratings to a level, with each rating indicating how difficult it is to achieve a corresponding score on a level.

2.2 Difficulty Balancing

Due to the previously discussed constraints imposed by HCGs, common methods for dynamic difficulty adjustment such as modifying in-game parameters [17], changing the design of a level [7], generating level segments of specific difficulties [20] and player modeling [33] are not readily applicable within HCGs. Aside from using rating system-based matchmaking, other techniques have been used to balance difficulty in such games. For example, both *Xylem* [22] and *Binary Fission* [4] used task size as a rough heuristic to determine the difficulty of levels. Relatedly, though educational games benefit from having known solutions to levels (which is not the case for HCGs), they suffer from similar obstacles to difficulty adjustment in having to balance fun and difficulty with a specific underlying objective. Within such games, Butler et al. [3] combined player data with solution features for levels to dynamically order levels in increasing difficulty commensurate with player skill. Similarly, Liu et al. [21] balanced difficulty using a system that adaptively generates new levels based on player performance on earlier levels. Though such progression-based balancing may not be directly applicable within HCGs, it may be utilized for onboarding and improving tutorial sections, as shown by Horn et al. [15, 16] who combined learning progressions with the skill chain model [5] to analyze tutorial sections for both *Paradox* and the HCG *Foldit*.

Difficulty balancing in HCGs using rating systems involves assigning ratings to players and levels and performing matchmaking to serve levels to players accordingly [25, 28]. Thus, based on a player's rating, they are dynamically served a level based on how difficult it is expected to be for that player to complete that level. Though effective, this approach does not give us information about how difficult it is for players to partially complete levels, which in an HCG would represent partial solutions to problems and thus may still be useful. Using rating arrays as proposed in this work enables us to determine how difficult a level is for a player at various stages of completion. Hence, this may enable more granular dynamic difficulty adjustment.

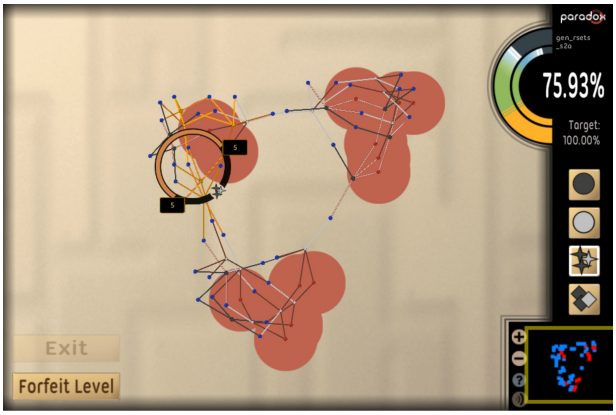

Figure 1: A screenshot of *Paradox*.

2.3 Score Prediction and Difficulty Estimation

In addition to matchmaking, rating systems have also been used to predict player scores on HCG levels. Williams et al. [32] used both Glicko-2 and Elo to predict outcomes of player-versus-level matches in *Paradox*, finding that Glicko-2 outperformed Elo and other baseline measures in terms of prediction accuracy. While we use a similar method for making predictions to test the accuracy of the CDF, the rating array approach gives the added ability of estimating the probability that a player will achieve a certain score (e.g. a new high score) on a given level. Outside of HCGs, Isaksen et al. [18] used survival analysis to predict a player's likelihood of achieving a certain score which in turn was used to estimate hazard rate and level difficulty. This approach is similar to our use of the CDF to estimate the probability of a player getting less than or equal to a given score. Similar to score prediction, Mourato et al. [23] predicted player success and failure rates for completing platformer levels using estimates of challenge posed by level elements. In general, score prediction and difficulty estimation are inherently related since player scores on levels depend on the degree of challenge offered by the levels. Difficulty estimation is an important game design problem [1] and has seen a wide variety of research across different types of games. In educational games, Szabo et al. [30] estimated the difficulty of in-game tasks using a method based on Bayesian probability theory. Puzzle games in particular have proven to be a popular testbed for difficulty estimation research. Ashlock et al. [2] used evolution to determine the difficulty of Sokoban levels, using mean time-to-solution and number of failures of the evolutionary algorithm as heuristics for estimating difficulty. Isaksen et al. [19] used models of strategy and dexterity to perform score analysis on the puzzle games *Tetris* and *Puzzle Bobble* as a means of difficulty estimation. On the other hand, van Kreveld et al. [31] analyzed various features of puzzle game levels to estimate their difficulty. Sarkar et al. [24] used a similar methodology to estimate level difficulty within the context of HCGs. Focusing on the difficulty of tasks in general rather than game levels, Guid et al. [12] modeled difficulty estimation as a problem of searching through solution alternatives and the relations between them, evaluating their model on chess.

3 METHOD

Here we describe our approach to using a rating array to estimate and update score cumulative distribution functions (CDFs). We assume possible scores are in the range $[0, 1]$. For a player-level pairing, the score CDF gives a function from a score to the probability that the player will not get higher than that score on that level. Thus, given a player p, and level l, we want to determine the cumulative distribution of the score $s_{p,l}$ that player p achieves on level l (for clarity, we omit the subscripts p, l from s), given as:

$$F_s(x) = P(s \leq x) \tag{1}$$

Our approach is to use a rating system to predict the probability of a player passing certain score thresholds on a level. We used the Glicko-2 rating system [10, 11] to predict these probabilities (although other systems could be used). We use θ to refer to a set of Glicko-2 rating parameters and μ for the *rating mean*.

Each player p has a rating θ_p, with μ_p initialized to the Glicko-2 default rating of 1500.

Instead of a single rating, each level l has an array of n ratings θ_l^t, using the superscript $t \in \{1, 2, ..., n\}$ as an index into this array. Across all levels, there is an associated array of score thresholds τ^t for these ratings. We assume that the score thresholds in this array are valid scores but not the maximum possible score ($\tau^t \in [0, 1)$) and strictly increasing ($\tau^t < \tau^{t+1}$).

In this work, we used an array of size $n = 10$, and score thresholds evenly spaced at increments 0.1 apart ($\tau^t = (t-1)/10$). We initialized the level rating array for a level l as a smoothly increasing curve around 1500, using the function $\mu_l^t = 1500 - 260\ln((1-\tau^t)/\tau^t)$.

Given a player's rating and a level rating array, we can begin to construct the score CDF with the probabilities that the player will pass each of the score thresholds. We use $\hat{F}_s^t$ to refer to the probability that score threshold τ^t will not be passed by the player (that is, $\hat{F}_s^t = P(s \leq \tau^t)$). These can be considered as paired comparisons between the player and a level's score thresholds, giving the probability that the player will not pass each score threshold. To estimate the outcome of the comparison between the player and a level threshold, we use the Glicko-2 E_{gl2} function:

$$\hat{F}_s^t = E_{gl2}(y|\mu_l^t, \mu_p, \phi_p^2) = \frac{1}{1 + e^{-g(\phi_p^2)(\mu_l^t - \mu_p)}} \tag{2}$$

where the rating means μ and variances ϕ^2—both parts of the rating θ—as well as the outcome y and the function g are all as used by the Glicko-2 system [11].

Since it is not possible for the player to pass the maximum score for a level ($P(s \leq 1) = 1$), we define an additional threshold, with $\tau^{n+1} = 1$ and $\hat{F}_s^{n+1} = 1$, for convenience.

Given all the $\hat{F}_s^t$, we construct $F_s(x)$ by linear interpolation between them:

$$F_s(x) = (1-\alpha)\hat{F}_s^t + \alpha\hat{F}_s^{t+1} \tag{3}$$

where $\tau^t \leq x \leq \tau^{t+1}$ and $\alpha = (x - \tau^t)/(\tau^{t+1} - \tau_t)$.

After a player attempt at a level, we update the player's rating and all the ratings in the level's rating array as though the player had played simultaneous matches against all the thresholds; i.e., if the player scores s, the player loses against all ratings with thresholds where $s \leq \tau^t$ and wins against all ratings with thresholds where $s > \tau^t$. In practice, we found that this update could sometimes lead to cases where the ratings for score thresholds were not strictly increasing (that is, could lead to $\mu_l^t \geq \mu_l^{t+1}$). In order to ensure that ratings were strictly increasing, we performed a post-processing step that adjusted ratings if needed. The post-processing increased ratings for thresholds above s and decreased ratings for thresholds below s (if needed) such that $\mu_l^t + 1 \leq \mu_l^{t+1}$. More specifically, for thresholds below s, if we had an instance of $\mu_l^t \geq \mu_l^{t+1}$, we set $\mu_l^t = \mu_l^{t+1} - 1$. Similarly, for thresholds above s, if we had an instance of $\mu_l^{t+1} \leq \mu_l^t$, we set $\mu_l^{t+1} = \mu_l^t + 1$.

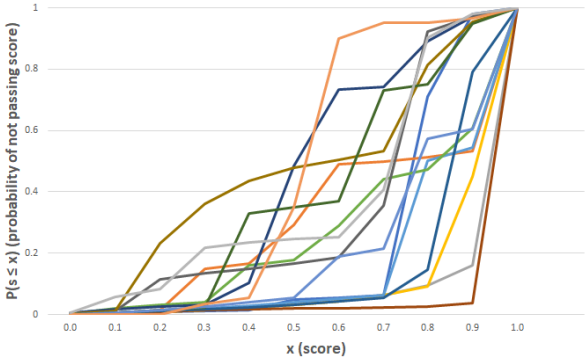

Figure 2: Example score CDFs for a player from *Paradox*. Each line is the CDF of a different level for that player.

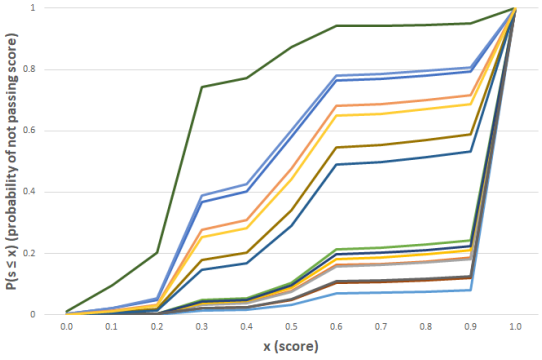

Figure 3: Example score CDFs for a level from *Paradox*. Each line is the CDF of a different player for that level.

4 DATA SETS

We evaluated the rating array approach on two sets of match data: one gathered from *Paradox* and another generated from idealized players and levels using Elo. Each data set consisted of a time-ordered list of matches with each match consisting of a player, a level, the result (i.e. complete/forfeit/skip) and the player's score on that level.

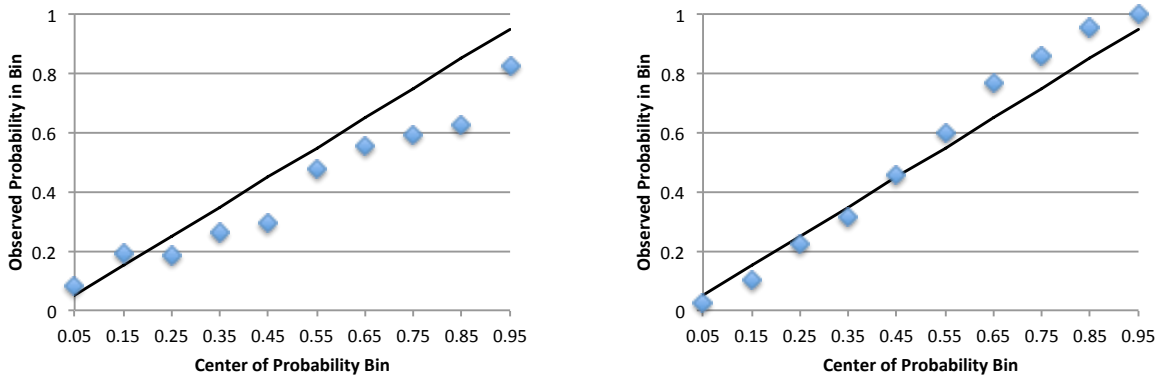

Figure 4: Plots of the centers of predicted probability bins and the observed probabilities for those bins for the *Paradox* (left) and Elo (right) data sets.

4.1 Paradox

Paradox is a 2D puzzle game where each level represents a MAX-SAT constraint satisfaction problem. The player's score for a level corresponds to the percentage of constraints in the underlying problem that they are able to satisfy by assigning boolean values using different tools. A player completes a level by reaching the target score for that level. More detailed descriptions of the gameplay and how levels model MAX-SAT problems are given in prior work [8, 28].

To gather match data from *Paradox*, we ran a Human Intelligence Task (HIT) on Amazon Mechanical Turk. We recruited 100 players with each being paid $2 upon completing the HIT. Players had to complete eight tutorial levels. These levels were used to familiarize players with the mechanics of the game and data from these levels was not used in our evaluations. After the tutorial phase, players proceeded to the challenge phase which consisted of 50 levels served in random order, each of which the players could skip (i.e. move to the next level without making a move), complete (i.e. reach the target score for that level) or forfeit (i.e. move to the next level having made at least one move but fail to complete the level). Each player could see each level only once during the playthrough. In the version of *Paradox* used in this work, the target score for each challenge level was set to 100%, which was not attainable for some of the levels. Players were able to exit the game, complete the HIT and receive payment once they had skipped/forfeited at least 5 challenge levels. Out of the 100 recruited players, we excluded from our evaluations one player who had errors in their data and another who just skipped through all 50 levels. Thus, the final gathered data set consisted of 98 players taking part in a total of 691 matches.

For the match data, each instance of a player seeing a level was treated as a match. Scoring in *Paradox* is based on the number of constraints a player has satisfied. For the purposes of analysis in this work, we normalized scores using the highest score a player achieved on a level such that having the initial number of constraints solved (the level's in-game starting score) was a score of 0 and solving all the constraints (an in-game score of 100%) was a score of 1. This ensured that scores remained in the range [0, 1] and allowed players to cover a wider range of possible scores for a level. If a player skipped a level, their score for that match was taken to be 0.

4.2 Idealized Elo

We also used a data set generated by simulating idealized players and levels with the Elo rating system [9]. We chose the Elo rating system to generate idealized data as it is a similar rating system to Glicko-2. To create this data set, we generated 100 players and 50 levels with Elo ratings selected uniformly at random between 900 and 2100. We then simulated 1000 matches by randomly selecting a player and level. The player's score for the match was their Elo expected score based on the Elo rating of the player (R_p) and level (R_l). This score is given by the equation: $1/(1+10^{(R_l-R_p)/400})$. There were no skips in this data set.

5 EVALUATIONS

To evaluate the rating array method, we looked into: 1) measuring the accuracy of the cumulative distribution function in predicting probabilities of events; 2) computing the error between expected and actual scores; and 3) deciding if we could serve a player a level that they would set a new high score on.

For both data sets, we played back the matches, and used our rating array approach to update the ratings for the players and level arrays. For evaluation, predictions were only made for matches in which both the player and level had been involved in 3 prior attempted matches (i.e. matches where the player made at least one move). Note that during playback, the ratings updates and CDF computations were done using all the matches up until that point of the playback, and predictions were made only on future matches whose results had not yet been incorporated into the updates and computations. That is, the matches until the current point of playback when the computations and updates were calculated can be viewed as the training data and the future matches which were used for evaluation can be viewed as the test data. An example of some resulting score CDFs are shown in Figures 2 and 3. Each evaluation method is described below.

	Err_{cdf}	Err_{gl2}	$Diff_{cdf-gl2}$
Paradox	0.407	0.401	0.058
Elo	0.115	0.126	0.066

Table 1: Root mean squared difference (RMSD) of expected score predictions.

5.1 CDF Accuracy

To examine the accuracy of the CDF, we wanted to examine if scores happened as often as they were predicted to by the CDF. In order to do so, we looked at the probability that scores would fall into ranges as predicted by the CDF. For each predicted match, we looked at all ranges of scores between intervals at multiples of 0.1. If the lower end of the interval was 0 it was inclusive, otherwise it was exclusive. Thus, we looked at the score ranges [0.0, 0.1], [0.0, 0.2], [0.0, 0.3], ... (0.8, 0.9], (0.8, 1.0], (0.9, 1.0]. We then used the CDF to compute the probability that the score would fall in each range. We then grouped these probabilities into bins by 10% and counted the percent of times each happened—i.e. counting how often scores that were predicted to happen between 0–10%, 10–20%, ..., 90–100% of the time actually happened within the predicted range.

As evaluation, we compared the center of the predicted probabilities within each bin (e.g. 5% or 0.05 for the bin from 0–10%) with the observed probabilities within that bin. A plot for each data set is given in Figure 4. We computed the Pearson's correlation between the center of the predicted probably bin and observed probabilities for each data set. For the *Paradox* data set, we found a correlation of 0.980 ($p < 0.001$), and for the Elo data set, we found a correlation of 0.995 ($p < 0.001$).

5.2 Estimating Scores

In addition to determining the accuracy of the CDF, we also wanted to examine the accuracy of the player scores predicted using it, specifically in comparison to using a single Glicko-2 rating per level as in Williams et al. [32].

For this evaluation, for both data sets, we computed the root mean square difference (RMSD) between the actual score achieved by players on levels and both the expected score $E(s)$ predicted by the score CDF as well as the expected score predicted using a single Glicko-2 rating per level, denoted as Err_{cdf} and Err_{gl2} respectively. We also looked directly at the RMSD between the CDF and Glicko-2 score predictions, denoted as $Diff_{cdf-gl2}$.

The expected score *E(s)* predicted using the CDF is given by:

$$E(s) = \int_0^1 (1 - F_s(x))\, dx \tag{4}$$

Since our construction of $E(s)$ is piecewise linear, we can compute its integral as the sum of trapezoid areas:

$$E(s) = \sum_1^n 0.5(2 - \hat{F}_s^{t+1} - \hat{F}_s^t)(\tau^{t+1} - \tau^t) \tag{5}$$

Results are given in Table 1.

5.3 Setting New High Scores

For our final evaluation, we wanted to look at how well the rating arrays can be used to decide how to serve levels to players with the aim of setting new high scores, while also performing dynamic difficulty adjustment. This is particularly useful in HCGs as new high scores for levels could represent potential novel solutions to the underlying problems that levels represent.

In previous work, dynamic difficulty adjustment in *Paradox* has been done by adjusting the "desired loss rate" (DLR) for players. The DLR for a player is the likelihood of losing against levels that we would like for that player and is used to match players with levels of appropriate difficulty. The player's DLR is computed based on their skill as indicated by their current Glicko-2 rating. As a player's rating goes up, so does their DLR, causing them to be matched up with harder levels. This loss rate is discussed in more detail in prior work involving *Paradox* [25, 28].

A benefit of using a score CDF is that we could adjust the difficulty of any given level for a player by changing the target score required for that player to complete that level when we serve it to them (e.g. a player with a higher rating will have a higher target score and thus a higher DLR for that level). Used in-game, such a target score represents the score a player needs to pass in order to complete the level; thus, we might expect that players would work to pass their given target score for a level and then move on to the next level if they are successful. Thus, we are interested in target scores that are greater than the maximum score seen on a level since if the player were to achieve them, it would be a new high score.

We explored two possible approaches to selecting the next level to serve to a player with regard to setting a new high score. First, a level could be served to a player simply if their expected score for that level s_{exp}, as predicted by the CDF using Equation 5, is greater than that level's current high score s_{max}. This approach may work well for setting new high scores, but does not take into account the player's DLR for dynamically adjusting the difficulty. To account for this rate, we also consider that a level could be served to a player if both their expected score is greater than the level's current high score *and* their desired loss rate score s_{dlr} is greater than or equal to the level's high score s_{max} (as a player must pass s_{dlr} to complete the level). This additional constraint on the s_{dlr} means that in some cases, we would not serve levels to players where we expect them to set a new high score since a low target score may cause them to stop playing before they achieve a high score. Thus, we might miss out on some potential new high scores, but might also benefit from engagement effects of dynamic difficulty adjustment when incorporating the s_{dlr}.

To compute the player's s_{dlr}, we first compute their desired loss rate *DLR*, as in prior work involving *Paradox*, using the equation:

$$DLR = \frac{1}{1 + e^{\alpha(\beta - \mu_p)}} \tag{6}$$

where $\alpha = 0.00628$ and $\beta = 1850$. This is done so as to give the player a DLR of 10% at the starting rating of 1500.

We then use each level's rating array to determine the player's expected value of crossing each of the level's score thresholds. These expected values combined with the player's current *DLR* allows us to use interpolation to find the player's s_{dlr} for that level (i.e.

	$s_{act} > s_{max}$	$s_{act} \leq s_{max}$
$s_{exp} > s_{max}$	34	61
$s_{exp} \leq s_{max}$	13	681
$s_{exp} > s_{max} \wedge s_{dlr} \geq s_{max}$	38	57
$s_{exp} \leq s_{max} \vee s_{dlr} < s_{max}$	228	466

Table 2: Results of comparisons between predicted and actual player scores for *Paradox* match data

	$s_{act} > s_{max}$	$s_{act} \leq s_{max}$
$s_{exp} > s_{max}$	130	54
$s_{exp} \leq s_{max}$	84	2161
$s_{exp} > s_{max} \wedge s_{dlr} \geq s_{max}$	111	27
$s_{exp} \leq s_{max} \vee s_{dlr} < s_{max}$	184	2107

Table 3: Results of comparisons between predicted and actual player scores for Elo match data

$s_{dlr} = F_s^{-1}(DLR)$). The player's s_{exp} for that level is simply their expected score for the level (i.e. $s_{exp} = E(s)$).

To test if these scores are useful in deciding which levels could be served to players for setting new high scores, we played back the matches while keeping track of the highest score seen for each level s_{max}. These were initialized to 0 for each level and so it was easier to set high scores earlier on. During the processing of each match, we used the player's current rating to compute their s_{exp} and s_{dlr} against each level in the game to compare the two previously discussed approaches. For each level, as a proxy to see what would have happened if we decided to serve that level, we then checked all of that player's *future* matches to see if they went on to play that level, and if so, compared the score they achieved s_{act} to see if it would have set a new high score for that level. Results of these comparisons for both data sets are given in Tables 2 and 3. Since for each match in the data, we can make these comparisons for multiple future matches, the total number of comparisons exceeds the total number of matches. Note that instances of s_{exp} being greater than a level's s_{max} represent predictions that the player will be able to increase that level's high score. However, with the added constraint on s_{dlr} , we are no longer simply serving levels to players whenever they are expected to improve the high score since we may decide not to serve such a level for the purpose of dynamic difficulty adjustment. However, we can still evaluate the properties of when we decide we could serve a level to a player based on if they would have improved the high score. Thus, instances where we decided we could have served a level to a player because they were expected to improve the high score, and the player's s_{act} turns out to be greater or lower than s_{max}, can be regarded as true positives and false positives respectively. Similarly, decisions to not serve a level because the player is not expected to improve the high score and the player's s_{act} turns out to be greater or lower than s_{max}, can be regarded as false negatives and true negatives respectively. To better evaluate these decisions, we used the following metrics with TP, FP, TN and FN representing the total number of true positives, false positives, true negatives and false negatives respectively:

- *Precision:* probability that a player who could be served a level to improve the high score, does improve it when served that level: TP/(TP+FP)
- *Recall:* probability that a player who improved the high score could have been served that level to improve the high score: TP/(TP+FN)
- *False Omission Rate (FOR):* probability of missing a player who would have set a new high score on a level: FN/(FN+TN)
- *Accuracy:* probability that the decision to serve or not serve a level to improve the high score, was correct overall: (TP+TN)/(TP+TN+FP+FN)

Metric values for both data sets are given in Tables 4 and 5.

	$s_{exp} > s_{max}$	$s_{exp} > s_{max} \wedge s_{dlr} \geq s_{max}$
Precision	0.358	0.400
Recall	0.723	0.143
FOR	0.019	0.329
Accuracy	0.906	0.639

Table 4: Metrics for *Paradox* match data. Values range from 0 to 1.

	$s_{exp} > s_{max}$	$s_{exp} > s_{max} \wedge s_{dlr} \geq s_{max}$
Precision	0.707	0.804
Recall	0.607	0.376
FOR	0.037	0.080
Accuracy	0.943	0.913

Table 5: Metrics for Elo match data. Values range from 0 to 1.

6 DISCUSSION

Based on our evaluations, the CDF was able to predict score ranges for players with a reasonably high level of accuracy. For both data sets, the predicted probabilities were shown to be highly correlated with the observed probabilities. Moreover, based on our results for computing error in score estimation, we found that the estimation error when using the CDF was very similar to that when using a single rating per level, being slightly lower for the Elo data set and slightly higher for the *Paradox* data set. The expected scores predicted by each approach were also relatively close to each other. Thus, in demonstrating that the CDF accurately predicts player score ranges and performs similarly to using a single level rating in terms of estimating specific player scores, these results help validate the CDF, and in turn, the rating array method used to derive it.

In terms of decisions about serving levels to players to set new high scores, we found that the metrics in Tables 4 and 5 (except for *Precision*) get worse upon adding the additional constraint involving s_{dlr}. As mentioned in the previous section, we would expect this since by including the s_{dlr}-based constraint for dynamic difficulty adjustment, we withhold serving certain levels where we expect players to improve the high score. Thus, there is a trade-off between

increased accuracy using only s_{exp} and the ability to perform dynamic matchmaking when also using s_{dlr}. Ultimately, the goal of player-versus-level matchmaking systems in HCGs is to balance player engagement with finding new solutions to problems. Hence, in addition to helping derive CDFs and estimating player scores, the rating array method was also motivated by its potential in helping identify players able to set new high scores while also keeping the game sufficiently challenging. Though the s_{exp}-based method may more accurately suggest which levels players are capable of setting new high scores for, in not taking the player's current skill into account, it ignores the fact that a suggested level may be too difficult or not difficult enough for the player. Thus, combining s_{exp} and s_{dlr} allows us to retain the ability to adapt the game's difficulty to the player while still serving players with levels whose high scores they are expected to improve.

7 CONCLUSION AND FUTURE WORK

This work presented the concept of using level rating arrays for the purpose of improved player-versus-level score prediction and matchmaking, particularly within the context of human computation games. Using this approach enables us to derive cumulative distribution functions for estimating probabilities of player scores on levels as well as decide if a level should be served to a player to try to set a new high score.

Since this work focused on describing the approach and testing it by playing back previously gathered match data, necessary future work is to evaluate this approach via an online experiment that uses rating arrays to perform matchmaking. The motivation for incorporating rating arrays into existing matchmaking systems for HCGs is to be better able to find new solutions by serving players levels that are both comparable to their skill as well as ones whose high scores they are likely to improve. Thus, performing a live matchmaking experiment using rating arrays to determine which levels to serve to players would be the natural next step for this research. This could involve more precisely defining the process of dynamically setting target scores for levels and of updating player and level ratings when players succeed or fail in reaching said targets, as discussed earlier in this paper, ultimately helping in validating and/or improving these processes. Examining players in this dynamic setting could also offer important insights into player behavior such as if players keep playing after reaching the target and if they exhibit more engagement when we use dynamic target scores rather than fixed targets like in past work.

Future work could also look at improving upon the prediction metrics when incorporating the desired loss rate. In addition to the loss rate function given in this paper, prior work in *Paradox* has examined other, potentially more engaging, difficulty curves [26] which might lead to better results when used with rating arrays.

Finally, we applied the rating array method to just *Paradox* so it would be worth investigating how this method works when applied to other games. It would also be interesting to explore if rating arrays could be applied in other types of games. For example, it may be possible to apply rating arrays to levels in educational games where increasing score thresholds could represent progressively harder concepts to be learned.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under grant no. 1652537. We would like to thank the players, and the University of Washington's Center for Game Science for initial *Paradox* development.

REFERENCES

[1] Maria-Virginia Aponte, Guillaume Levieux, and Stephane Natkin. 2011. Measuring the level of difficulty in single player video games. *Entertainment Computing* 2 (2011).
[2] Daniel Ashlock and Justin Schonfeld. 2010. Evolution for automatic assessment of the difficulty of Sokoban boards. In *2010 IEEE Congress on Evolutionary Computation (CEC)*. 1–8.
[3] Eric Butler, Erik Andersen, Adam M. Smith, Sumit Gulwani, and Zoran Popović. 2015. Automatic game progression design through analysis of solution features. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. 2407–2416.
[4] Kate Compton, Heather Logas, Joseph C. Osborn, Chandranil Chakrabortti, Kelsey Coffman, Daniel Fava, Dylan Lederle-Ensign, Zhongpeng Lin, Jo Mazeika, Afshin Mobramaein, Johnathan Pagnutti, Husacar Sanchez, Jim Whitehead, and Brenda Laurel. 2016. Design lessons from Binary Fission: a crowd sourced game for precondition discovery. In *Proceedings of the 1st International Joint Conference of DiGRA and FDG*.
[5] Daniel Cook. 2007. The chemistry of game design. (2007). http://www.gamasutra.com/view/feature/1524/the_chemistry_of_game_design.php
[6] Seth Cooper, Sebastian Deterding, and Theo Tsapakos. 2016. Player rating systems for balancing human computation games: testing the effect of bipartiteness. In *Proceedings of the 1st International Joint Conference of DiGRA and FDG*.
[7] Valve Corporation. 2008. *Left 4 Dead*. Game.
[8] Drew Dean, Sean Gaurino, Leonard Eusebi, Andrew Keplinger, Tim Pavlik, Ronald Watro, Aaron Cammarata, John Murray, Kelly McLaughlin, John Cheng, and Thomas Maddern. 2015. Lessons learned in game development for crowdsourced software formal verification. In *Proceedings of the 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education*.
[9] Arpad E. Elo. 1978. *The rating of chessplayers, past and present*. Arco.
[10] Mark E. Glickman. 1999. Parameter estimation in large dynamic paired comparison experiments. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 48, 3 (1999), 377–394.
[11] Mark E. Glickman. 2001. Dynamic paired comparison models with stochastic variances. *Journal of Applied Statistics* 28, 6 (Aug. 2001), 673–689.
[12] Matej Guid and Ivan Bratko. 2013. Search-based estimation of problem difficulty for humans. In *International Conference on Artificial Intelligence in Education*. 860–863.
[13] Severin Hacker and Luis von Ahn. 2009. Matchin: eliciting user preferences with an online game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1207–1216.
[14] Ralf Herbrich, Tom Minka, and Thore Graepel. 2007. TrueSkill(TM): a Bayesian skill rating system. In *Advances in Neural Information Processing Systems 20*. 569–576.
[15] Britton Horn, Seth Cooper, and Sebastian Deterding. 2017. Adapting cognitive task analysis to elicit the skill chain of a game. In *Proceedings of the Annual Symposium on Computer-Human Interaction in Play*. 277–289.
[16] Britton Horn, Josh Aaron Miller, Gillian Smith, and Seth Cooper. 2018. A Monte Carlo approach to skill-based automated playtesting. In *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.
[17] Robin Hunicke. 2005. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACM, 429–433.
[18] Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. 2018. Exploring game space of minimal action games via parameter tuning and survival analysis. *IEEE Transactions on Games* 10, 2 (2018), 182–194.
[19] Aaron Isaksen, Drew Wallace, Adam Finkelstein, and Andy Nealen. 2017. Simulating strategy and dexterity for puzzle games. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. 142–149.
[20] Martin Jennings-Teats, Gillian Smith, and Noah Wardrip-Fruin. 2010. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. 11:1–11:4.
[21] Yun-En Liu, Christy Ballweber, Eleanor O'Rourke, Eric Butler, Phonraphee Thummaphan, and Zoran Popović. 2015. Large-scale educational campaigns. *ACM Transactions on Computer-Human Interaction* 22, 2 (March 2015), 8:1–8:24.
[22] Heather Logas, Jim Whitehead, Michael Mateas, Richard Vallejos, Lauren Scott, Dan Shapiro, John Murray, Kate Compton, Joseph Osborn, Orlando Salvatore, Zhongpeng Lin, Huascar Sanchez, Michael Shavlovsky, Daniel Cetina, Shayne Clementi, and Chris Lewis. 2014. Software verification games: designing Xylem,

The Code of Plants. In *Proceedings of the 9th International Conference on the Foundations of Digital Games*.

[23] Fausto Mourato, Fernando Birra, and Manuel Próspero dos Santos. 2014. Difficulty in action based challenges: success prediction, players' strategies and profiling. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*.

[24] Anurag Sarkar and Seth Cooper. 2017. Level Difficulty and Player Skill Prediction in Human Computation Games. In *Proceedings of the Thirteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

[25] Anurag Sarkar and Seth Cooper. 2018. Meet Your Match Rating: Providing Skill Information and Choice in Player-versus-Level Matchmaking. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*.

[26] Anurag Sarkar and Seth Cooper. 2019. Transforming Game Difficulty Curves Using Function Composition. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 551:1–551:7.

[27] Advait Sarkar, Cecily Morrison, Jonas F. Dorn, Rishi Bedi, Saskia Steinheimer, Jacques Boisvert, Jessica Burggraaff, Marcus D'Souza, Peter Kontschieder, Samuel Rota Bulò, Lorcan Walsh, Christian P. Kamm, Yordan Zaykov, Abigail Sellen, and Siân Lindley. 2016. Setwise comparison: consistent, scalable, continuum labels for computer vision. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 261–271.

[28] Anurag Sarkar, Michael Williams, Sebastian Deterding, and Seth Cooper. 2017. Engagement effects of player rating system-based matchmaking for level ordering in human computation games. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

[29] Alex Cho Snyder and Mario Izquierdo. 2014. *Jumpcraft*. Game.

[30] Máté Szabó, Krisztián Dániel Pomázi, Bertalan Radostyán, Luca Szegletes, and Bertalan Forstner. 2016. Estimating task difficulty in educational games. In *2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 397–402.

[31] Mark van Kreveld, Maarten Loffler, and Paul Mutser. 2015. Automated Puzzle Dificulty Estimation. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*. 415–422.

[32] Michael Williams, Anurag Sarkar, and Seth Cooper. 2017. Predicting Human Computation Game Scores with Player Rating Systems. In *Predicting Human Computation Game Scores with Player Rating Systems. In: Munekata N., Kunita I., Hoshino J. (eds) Entertainment Computing – ICEC 2017. ICEC 2017.*

[33] Alexander Zook and Mark Riedl. 2012. A Temporal Data-Driven Player Model for Dynamic Difficulty Adjustment. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.