Automatic Playtesting for Yahtzee

James Glenn
Dept. of Computer Science
Yale University
New Haven, CT, USA
james.glenn@yale.edu

Rob Brunstad
Dept. of Computer Science
Yale University
New Haven, CT, USA
charles.brunstad@yale.edu

***Abstract*—Yahtzee is a dice game with elements of skill and chance. There are many numeric parameters that govern the game's scoring rules, and varying those parameters will affect many aspects of game play, including strategic depth. We take advantage of the ease of computing the optimal policy for solitaire Yahtzee to use supervised learning to develop near-optimal agents based on neural networks. With the aim of automatically selecting scoring parameters that increase strategic depth, we measure the agent's skill as we vary the resources available to the neural networks and use metrics derived from the resulting learning curve as an indicator of strategic depth.**

***Index Terms*—games, neural networks, supervised learning**

I. Introduction

Yahtzee is a multi-player dice game with elements of both skill and chance. Players take turns rolling five six-sided dice with the aim of rolling poker-like combinations (for example, straights, three-of-a-kind, or full house). Players may reroll any of the dice twice during their turn, and at the end of their turn choose one of thirteen scoring categories. Each category may be used once per player per game, and each has its own scoring rules as follows.

- Ones...Sixes: (the upper categories) one point for each pip on dice showing a number matching the category.
- Three/Four of a Kind: the sum of the dice if at least three/four show the same number, 0 otherwise.
- Full House: 25 points if three dice show the same number and two show a different number, 0 otherwise.
- Small/Large Straight: 30/40 points if four/five of the dice show consecutive numbers, 0 otherwise.
- Chance: the sum of the dice.
- Yahtzee: 50 points for five of a kind, 0 otherwise.

In addition, there are two possible bonuses: 35 points if the sum of categories ones through sixes (the *upper categories*) is at least 35, and 100 points for each Yahtzee rolled after scoring 50 in the Yahtzee category (the extra Yahtzees are still scored in another unused category). Five-of-a-kinds that can't be used in Yahtzee can also be used as a *yahtzee joker* for the full score in Full House or the Straights when the corresponding upper category has also already been used (and with the official *forced joker* rule, the upper category must be used if available). The winner is the player with the highest score after all players have taken their thirteen turns. The game can also be played solitaire with objectives chosen by the player, such as maximizing the average score or maximum score over a sequence of games.

Varying any of the scores defined in the rules will yield a new version of the game; each new version plays differently. For example, lowering the value of the upper bonus from 35 will reduce the value of putting high scoring rolls in the upper categories in favor of placing them in the lower categories. Raising the value of the upper bonus will reward decisions that lead to higher scores in the upper categories. Raising or lowering the threshold at which the upper bonus is earned will have similar effects, and changing the other scoring rules affect decision making for the corresponding categories.

A game designer will endeavor to find variations on the rules that result in games that are rewarding for humans to play, where "rewarding" can mean many things, including deep, clear, original, or aesthetically pleasing. Traditionally, such optimization of game rules towards these characteristics has been done through manual playtesting, in which designers gather subjective feedback from playtesters.

We focus on strategic depth – the extent to which there are many different levels of skill exhibited by players – and evaluate metrics that we hope capture information about strategic depth by comparing the performance of agents for solitaire Yahtzee that have access to differing levels of computational resources to the performance of an optimal agent.

II. Background

With advances in processing power and AI techniques have come advances in automatic playtesting systems. Browne, for example, devised a system to evaluate characteristics of game play using a general game-playing system, correlated those characteristics with subjective ratings of games, and used those results to guide a system that automatically generates games, resulting in the highly-rated game Yavalath [2].

Togelius et al. survey procedural content generation, including simulation-based evaluation of generated content using intelligent agents [11], with examples of quality measures (fitness) of content including number of moves to solve [1] [8], variation in progress by a neural network controller [10], and neural networks' predictions of emotional state [9]. More recently, Liu et al. use performance of a general video game playing agent against a simple agent as the objective function when optimizing the parameters of a space-battle game [7].

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

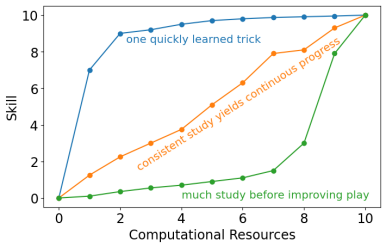

Fig. 1: Three hypothetical strategy ladders.

Lantz et al. propose a metric d of the number of steps of improvement in an artificial agent's quality of play as the computational resources available to it increase [6]. The number of steps is counted by generating a *strategy ladder*: a graph of the performance of an agent against the computational resources available, where computational resources could be computing time, memory, or any other quantity for which increasing the value could be expected to result in an increase in performance. The number of steps is then determined by the number of increments of resources available that result in increases in performance above the previous threshold; after each such step the threshold is reset to a fixed amount above the current level of performance. That metric d could be construed as an approximation of the strategic depth of a game – the extent to which continuous careful study of a game yields insights that generalize well and lead to constant improvement in skill. Chess and go are regarded as games with great strategic depth, so should have high d values. A game with a randomly generated game tree should have a low d value since playing well comes from memorization rather than generalizable insights.

Volkovas et al. use the same idea of varying computational resources and measuring the resulting performance [13]. They choose the number of hidden neurons in a neural network as their variable resources and measure how well the networks can predict the next state in a puzzle. Rather than assigning a measure of quality to the shape of the learning curve, their tool allows the puzzle designer to specify a target learning curve and the tool, having precomputed the curves for all versions of the puzzle, then picks the puzzle with the best match.

Although solitaire Yahtzee is not as strategically deep as chess or go, it may be interesting enough to serve as a testbed for automatic playtesting techniques. Yahtzee is a finite game with a finite state space, and so, in theory, an optimal strategy could be computed by solving the Bellman equations $V(s) = \max_a \sum_{s'} (P_a(s,s') \cdot (R_a(s,s') + V(s'))$, where $P_a(s,s')$ is the probability action a from state s leads to state s' and $R_a(s,s')$ is the reward earned by that transition. For a finite game, the equations can be solved in one pass by dynamic programming, ordering the states by distance to the end of the game. But in practice, the state space for even two-player Yahtzee is too large to explore exhaustively. The solitaire game, however, has a small enough state space so that the optimal strategy for certain objectives can be computed easily on current systems. Verhoeff [12], Glenn [4] [5], Woodward [14], and others computed the optimal policy when the goal is to maximize the expected score as long as 20 years ago. Other objectives have also been considered, such as maximizing the probability of exceeding a target score [3].

III. Experimental Design and Results

We consider techniques that leverage the optimal policy in order to evaluate the results of varying the numeric parameters of the game on its strategic depth. In particular, we investigate the effect on strategic depth of the parameters of the upper bonus: both the upper total threshold required to earn it, and its value. It is clear that these parameters affect players' strategy: when the first turn ends with the roll 56666 then the optimal policy chooses to score the roll 56666 in Four of a Kind if there is no upper bonus, and the difference between that and the next best option (Sixes) is about 5.5 points. But in the official game the decision swings the other way, with a difference between those options (still the two best) of 3.7 points.

We leverage the ability to compute the optimal solitaire strategy for versions of Yahtzee with different scoring rules to train a set of architecturally similar neural networks. The training examples used in the supervised learning are sampled from games played by the optimal solitaire player. The neural network has 2 hidden layers with default sizes of 100 and 200 nodes. The current state is encoded as the input, and includes the state of the scoresheet, the current roll, and the number of rolls left in the turn. The scoresheet part of the state generally only includes whether a category is used or not, and not the score earned in the category, since the score earned in the past does not affect future strategy. The exceptions are the upper categories, where the upper bonus requires the total to be recorded in the state (so 2 points in twos and 6 in threes is equivalent to 8 in twos and 0 in threes), and Yahtzee, where the Yahtzee bonus requires a distinction between unused, zero, and 50. There are then 22 inputs to the neural network: twelve binary inputs to indicate whether the categories other than Yahtzee are used or unused; two binary inputs for the Yahtzee category (00 meaning unused, 10 meaning zero, and 01 meaning 50); six to count how many of each possible number are in the current roll, normalized to the range $[0,1]$; one for the current upper total, clipped and normalized to $[0,1]$ by $\frac{\min(total,threshold)}{threshold}$; and one for the number of rolls left, normalized to $[0,1]$. The network has categorical outputs that correspond to meta-actions. A meta-action is a function that takes a game state and outputs a game action, which will be a subset of the dice to keep for states with rolls left, and unused categories for states at the end of a turn. We use meta-actions to reduce the number of outputs: instead of outputs for the up to 32 different choices of dice to keep and 13 choices of category to score in, we have only ten meta-actions. The meta-actions are hand-coded functions that attempt to capture the intent of players – what category they are trying to achieve a good score in. For example, if the current roll is 12235 and the player keeps 123 then they are probably trying to complete a straight. The complete list of meta-actions follows.

- Ones,...,Sixes: one meta-action for each upper category that, when there are rolls left, chooses to keep only the dice that match the category, and at the end of a turn chooses the corresponding category.
- N-of-a-kind: when rolling, keep the largest subset of equal dice, breaking ties in favor of higher numbers, and keeping other high numbers when there is no advantage to making a larger group. Ending a turn, score in the highest available n-of-a-kind category in which the roll would earn a non-zero score. If there is no such category, score zero in the highest available n-of-a-kind category.
- Full House: when rolling, keep all dice that are not singletons. Ending a turn, score in Full House (whether or not the score earned is non-zero).
- Straight: when rolling, keep dice to maximize the chances of completing the longest available straight category. Ending a turn, score greedily in the straights; if a non-zero score is impossible then zero the longest available.
- Chance: keep 5s and 6s with two rolls left, and also 4s with one roll left; at the end of a turn score in Chance.

In all cases, if at the end of a turn the meta-action indicates a category that has already been used, the meta-action with the next highest output value is chosen.

The optimal policy and corresponding value of the game when actions are restricted to these meta-actions can be computed by restricting a in the Bellman equations to game actions output by at least one of the meta-actions. In all variations considered, the difference between the optimal policy when restricted to meta-actions and the unrestricted optimal policy is less than 0.2%. For example, the unrestricted optimal policy for the official game (with the forced joker rule) earns an expected 254.588 points, and the optimal policy restricted to meta-actions has an expectation of 254.205.

We generate the training set for supervised learning by simulating games played using the (unrestricted) optimal policy and sampling one state per game. We then determine the value of the game actions selected by each meta-action and label the state with the meta-action with the maximum value. Often, different meta-actions will output the same game action in a given position. For example, if the current roll is 12666 then Sixes, N-of-a-Kind, Full House, and Chance will all output 666. In such cases, we label the example with the meta-action with the smallest set of possible outputs over all inputs. In the previous example that would be Sixes since there are only six possible outputs from that meta-action.

The supervised learning approach works quite well: with two hidden layers of 100 and 200 nodes and 100,000 training examples, the mean score of the trained neural network for the official game is 248.73, which is 97.7% of optimal.

Following the suggestion of Lantz et al. [6] to measure performance as the availability of computational resources changes, we vary the size of the training set and the number of hidden nodes in the neural network. For each combination of training parameters, we evaluate the performance of the trained neural network by simulating game play. There are three stochastic processes involved: sampling game states to build the training set, training the neural network, and, of course, playing the game; we repeat each step multiple times to reduce the variance in our final evaluation. We vary the number of training inputs while holding the size of the neural network fixed at 50 and 100 nodes per hidden layer, and then for holding the number of training inputs fixed at 50,000 while varying the total number of nodes, repeating the process for selected values of the upper bonus threshold. The fixed values were chosen as a compromise between performance of the trained neural network and speed of training and evaluation. Unlike an environment in which the skill is measured by a value whose range doesn't change as parameters are varied, such as percentage of games won or puzzled solved, for Yahtzee the range of possible scores will vary as the parameters of the game change. For example, the optimal policy with an upper threshold of 1 has an expected score of 272.44, which decreases to 237.47 as the threshold increases to 84. To account for this, we calculate each neural network's performance as a percentage of the optimal policy's expected score. The results are shown in Figure 2.

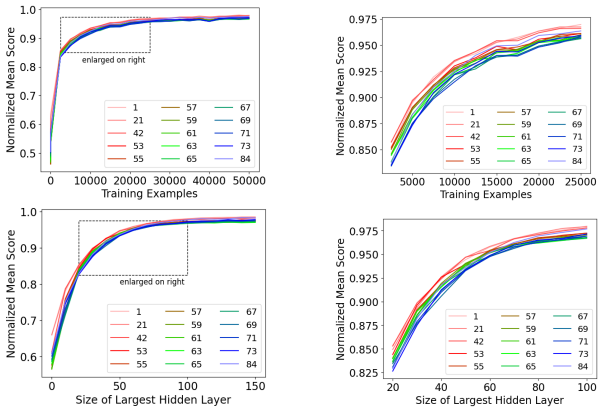

Fig. 2: Strategy ladders for different upper bonus thresholds.

Unfortunately, the shapes of the strategy ladders are so similar that the d metric can't make useful distinctions between the different versions of the game given the step size we've initially chosen for computational resources. It is possible that by adding more steps along the steepest part of the curves we could get some useful measurements. It is much less computationally intensive to measure the difference between the performance of the neural network with the lowest amount of resources and with the highest amount. This does not capture any information about the shape of the curve, as d is intended to do, but since our shapes are similar the simple difference may yield meaningful information. We have computed the difference Δ between an untrained neural network (0 training examples) and a neural network with two hidden layers of 100 and 200 nodes with 100,000 training examples for various upper bonus thresholds, keeping the value of the bonus fixed at 35, and for various values of the upper bonus, keeping the threshold fixed at 63. The results are shown in Figure 3. The difference between the peak and the lowest point on the curve

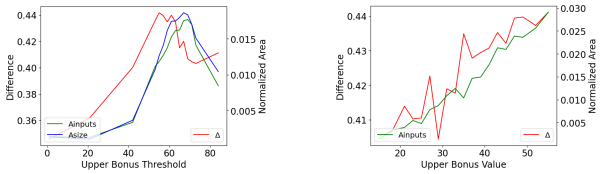

Fig. 3: Depth metrics for upper bonus parameters.

for the upper bonus threshold is over 25 standard deviations.

Figure 3 also lists a new metric that may carry useful information about the shape of the curve: A, the area between the curve for a particular variation of the game and the upper envelope, the maximum over all variations of the performance for a particular level of resources. This matches the intent of the d metric by rewarding games that allow for consistent improvement as resources increase. But A also rewards games for which there is improvement only after a high level of resources are reached. That is not an issue for Yahtzee. The A metric also rewards games for which the neural network with the highest level of resources has a relatively low level of performance. If we assume that the gap between the performance of the neural network with the highest level of resources and the performance of the optimal policy represents some interesting strategy that the neural network hasn't learned, then it makes sense to have a metric like A that is influenced by that value.

We hope to capture something about the strategic depth of the Yahtzee variants with both Δ and A. It is encouraging that both match what we expect from our subjective evaluation of the upper bonus threshold: for low values, the bonus is easy to earn and so there is less strategic depth, and for high values the bonus is too hard to earn, again resulting in lower strategic depth. The consistency between the values for A computed when varying the number of training examples and when varying the size of the hidden layers also supports the notion that A is capturing some intrinsic property of the game variants. Finally, both Δ and A show peaks relatively close to the values of the upper bonus threshold and value in the official game, as one would expect under the assumptions that Yahtzee was reasonably well playtested, and that Δ and A are reasonable measures of strategic depth.

The results for the value of upper bonus are quite different. Although both Δ and A (computed for inputs only; A_{size} was not calculated because of constraints on computational resources) show the same trend, it likely they are capturing something other than strategic depth: as the value increases, the learned policies' normalized scores will converge to how often they earn the bonus compared to how often the optimal policy does. If that is lower than the neural networks' performance in other categories, the trend in the difference will converge from below to a horizontal asymptote.

IV. Conclusion and Future Work

In some situations, the size of a neural network seems to be a useful notion of computational resources that can be used to estimate strategic depth in games using the strategy ladder approach – the strategy ladders are at least visibly different, even if Lantz's d metric is too coarse-grained to make useful distinctions given our choice of step sizes. For games for which the optimal policy can be determined, the number of training examples also seems to be a useful notion of resources. The area A between the upper envelope for all game variations under consideration and the curve for a particular variation may be more useful than d to make fine-grained distinctions, and should be examined in other contexts to validate the results described here. The other contexts could include attempting to optimize all the scoring parameters of Yahtzee using for example a stochastic hill climber with A as the objective function, or determining optimal parameters for other small games.

The results presented here could be further validated by comparison with those yielded by a similar method with a different notion of computational resources, perhaps a reinforcement learning algorithm with a varying number of training steps. A reinforcement learning algorithm would also benefit from not needing the optimal policy to be computed in order to produce training examples, but would still need a measure of performance that is not tied to the performance of the optimal solution as our normalized expected score is. Extending our methods to the two-player head-to-head version of Yahtzee with winning percentage against a baseline player would meet that need.

References

[1] D. Ashlock. Automatic generation of game elements via evolution. In *Proc. of the 2010 IEEE Conf. on Comp. Intelligence and Games*, pages 289–296, 2010.

[2] C. Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland Univ. of Tech., Brisbane, Au., 2008.

[3] C. J. F. Cremers. How best to beat high scores in yahtzee: A caching structure for evaluating large recurrent functions. Master's thesis, Technische Universiteit Eindhoven, Eindhoven, Netherlands, 2002.

[4] J. Glenn. An optimal strategy for yahtzee. Technical Report CS-TR-0002, Loyola University Maryland, 2006.

[5] J. R. Glenn. Computer strategies for solitaire yahtzee. In *2007 IEEE Symp. on Computational Intelligence and Games*, pages 132–139, 2007.

[6] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, and J. Togelius. Depth in strategic games. In *WS-17-01*, AAAI Workshop - Technical Report, pages 967–974. AI Access Foundation, January 2017.

[7] J. Liu, J. Togelius, D. Pérez-Liébana, and S. M. Lucas. Evolving game skill-depth using general video game ai agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2299–2307, 2017.

[8] D. Oranchak. Evolutionary algorithm for generation of entertaining shinro logic puzzles. In C. Di Chio et al., editor, *App. of Evolutionary Comp.*, pages 181–190. Springer, 2010.

[9] C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience in super mario bros. In *Proc. of the 5th Intl. Conf. on Comp. Intelligence and Games*, CIG'09, page 132–139. IEEE Press, 2009.

[10] J. Togelius, R. De Nardi, and S. M. Lucas. Towards automatic personalised content creation for racing games. In *2007 IEEE Symposium on Computational Intelligence and Games*, pages 252–259, 2007.

[11] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Trans. on Comp. Intelligence and AI in Games*, 3(3):172–186, 2011.

[12] T. Verhoeff. Solitaire yahtzee: Optimal player and proficiency test. http://www-set.win.tue.nl/~wstomv/misc/yahtzee/. Accessed: 2020-07-01.

[13] R. Volkovas, M. Fairbank, J. R. Woodward, and S. Lucas. Extracting learning curves from puzzle games. In *2019 11th Computer Science and Electronic Engineering (CEEC)*, pages 150–155, 2019.

[14] P. Woodward. Yahtzee®: The solution. *CHANCE*, 16:18–22, 09 2012.