5-18-2021

A Highly-Parameterized Ensemble to Play Gin Rummy

Masayuki Nagai '22
DePauw University

Kavya Shrivastava '23
DePauw University

Kien Ta '22
DePauw University

Steven Bogaerts
DePauw University, stevenbogaerts@depauw.edu

Chad Byers
DePauw University, cbyers@depauw.edu

A Highly-Parameterized Ensemble to Play Gin Rummy

Masayuki Nagai, Kavya Shrivastava, Kien Ta, Steven Bogaerts, Chad Byers
DePauw University
Greencastle, IN 46135, U.S.A.
{masayukinagai_2022, kshrivastava_2023, kienta_2022, stevenbogaerts, cbyers}@depauw.edu

Abstract

This paper describes the design and training of a computer Gin Rummy player. The system includes three main components to make decisions about drawing cards, discarding, and ending the game, with numerous parameters controlling behavior. In particular, an ensemble approach is explored in the discard decision. Finally, three sets of parameter tuning and performance experiments are analyzed.

1 Introduction

Gin Rummy is a two-player card game played over a series of *rounds*. A round begins with each player receiving 10 cards from a standard deck. One additional card is placed face-up to form the initial *discard pile*, with the remaining placed face-down to form the *stock pile*. Players alternate turns, drawing a card from either pile and then discarding a card face-up in the discard pile, attempting to form *melds* in their hand. A meld can be either a *rank meld* with 3+ cards of the same rank, or a *run meld* with 3+ cards of the same suit and consecutive ranks (with aces low). Unmelded cards are called *deadwood*, with a cost equivalent to rank, with aces worth 1 point and face cards worth 10. A player with 10 or fewer deadwood points can *knock* by revealing their melds to the opponent. The total deadwood points of unmelded cards in each player's hand is calculated. The player with the least deadwood wins the round, with significant additional bonuses for having 0 deadwood (*going gin*) or having less deadwood than a knocking player (an *undercut*). For more details about rules, see (McLeod 2020).

This paper describes a computer player for Gin Rummy. We begin with a discussion of the system's tracking of the state of the round. Next, the measures of *obtainability* and *meldability* are presented. We then describe the system components, and finally three sets of parameter tuning and performance experiments.

2 Tracking the State of a Round

Let *self* represent the computer player and *opp* the opponent. In this paper, opp is always the same: the *simple player* that only draws a card when it can immediately form a meld, considers only the minimization of deadwood in forming melds, and knocks as soon as possible.

Building upon the work in (Mark 2017), self tracks each card's current *state* in the round from its own perspective. (Recall in Section 1 that a "game" consists of multiple "rounds".) For cards known by self to be held by a player, the state includes which player holds the card, where it came from (initial deal, stock, or discard pile), and whether opp also knows about the card. For discarded cards, the state includes which is on top, which are irretrievable, and which player discarded the card.

The remaining cards are in the `unknown` state. In that case we also estimate the probability $p(c)$ that c is in the stock pile. Since an `unknown` card must be either in stock or in opp's hand, the probability that c is in opp's hand is $1 - p(c)$. As the round progresses, adjustments to $p(c)$ for various c are made by observing opp's actions.

Let S_t and O_t represent the set of cards in stock and the set of cards in opp's hand that self has not seen, respectively, at turn t. The set U_t of cards in the `unknown` state can be formed by $U_t = S_t \cup O_t$. At the start of a round, $|S_0| = 31$ and $|U_0| = 41$, since self only knows about its own hand and the single card in the discard pile. We therefore initialize the probabilities: $\forall c \in U_0,\ p(c) = |S_0|/|U_0| = 31/41$.

For a given card c, define an *adjacent* card as one of the same rank, or 1-2 ranks away of the same suit. Let A_c be the set of cards adjacent to c that have state `unknown`. When self observes certain actions involving c, *adjacency updates* are made to the estimated probabilities of the cards in A_c.

If opp draws face-up card c, opp likely already has some of the cards in A_c. Self therefore decreases the estimated probability that the cards in A_c are in stock, corresponding to an increased expectation that they are in opp's hand:

$$\forall a \in A_c,\ p(a) \leftarrow p(a) - p(a) \cdot \beta$$

for a parameter[1] β in $(0, 1)$.

Similarly, opp may show disinterest in a card c, either by declining to draw it when face-up, or by discarding it. In either case, we may expect opp to be less likely to have cards in A_c. Thus we increase our estimate that they are in stock:

$$\forall a \in A_c,\ p(a) \leftarrow p(a) + (1 - p(a)) \cdot \beta$$

[1] β is actually a temporary placeholder for a more expressive parameterization (Section 5).

In addition to the adjacency updates defined above, we also employ a *baseline* update based on the number of `unknown` cards. The baseline update continues the pattern introduced in the initialization of $p(c)$ for each $c \in U_0$ as described above. At each turn t, the estimated probabilities of remaining `unknown` cards are updated by calculating the change in probability (Δ_t) of a card being in stock:

$$\Delta_t = \frac{|S_t|}{|U_t|} - \frac{|S_{t-1}|}{|U_{t-1}|} = \frac{|S_t|}{|S_t| + |O_t|} - \frac{|S_{t-1}|}{|S_{t-1}| + |O_{t-1}|}$$
$$\forall c \in U_t,\ p(c) \leftarrow p(c) + \Delta_t \quad (1)$$

There are multiple circumstances in which $|S_t|$ and/or $|O_t|$ will change from turn $t-1$. Δ_t is negative when self or opp draws from stock. Δ_t is positive when opp draws from the discard pile and discards a previously `unknown` card. Finally, Δ_t is zero when self draws from the discard pile or opp draws from the discard pile and discards a known card. The adjacency and baseline updates are refined and experimentally validated in Section 5.1.

3 Obtainability and Meldability

We now define two measures that use state and probability information: *obtainability* and *meldability*.

3.1 Obtainability

We define the *obtainability* $ob(c)$ of a card c as an estimated likelihood that self can obtain c. The value $ob(c)$ is computed according to c's state. If c is currently in self's hand, then $ob(c) = 1$, with one exception described in Section 3.2. If c is the top card of the discard pile, then self can obtain the card, but at the cost of discarding another, and so we distinguish from cards in-hand with $ob(c) = 0.99$. If opp obtained c from the discard pile, then it seems less likely that the card will be discarded later, and so we use a low value $ob(c) = \alpha_{m0}$ with parameter α_{m0} in $[0, 0.5]$ to be tuned in Section 5. If c is buried in the discard pile, it cannot be obtained, and so $ob(c) = 0$. Finally, if c has state `unknown`, then self may be able to obtain c, provided that the card is in stock and not in opp's hand. Therefore, $ob(c)$ is a fraction of the probability of the card being in stock. We define $ob(c) = \alpha_{m1} \cdot p(c)$, with parameter α_{m1} in $(0, 1)$ representing this fraction, to be tuned in Section 5.

3.2 Meldability

We define the overall *meldability* $m(c)$ of a card c as the estimated likelihood that c will eventually be part of a meld in self's hand. Meldability is calculated using obtainability for rank and run melds, as defined below.

The *rank meldability* $m_{rank}(c)$ of a card c estimates the likelihood of c being in a rank meld. If c is already in a rank meld, then let $m_{rank}(c) = 100$, a comparatively high value. Otherwise, $m_{rank}(c)$ is calculated according to the obtainability of cards with the same rank as c. Given a card c_1 and cards c_2, c_3, and c_4 with the same rank, we define:

$$m_{rank}(c_1) = ob(c_2)ob(c_3) + ob(c_3)ob(c_4) + ob(c_2)ob(c_4)$$

This makes an analogy to probability theory, since c_1 may be combined with any two of $\{c_2, c_3, c_4\}$ to form a rank meld.

The *run meldability* $m_{run}(c)$ is defined similarly. Given a card c and same-suit adjacent cards c_a, c_b, c_d, and c_e in increasing rank order, we define:

$$m_{run}(c) = ob(c_a)ob(c_b) + ob(c_b)ob(c_d) + ob(c_d)ob(c_e)$$

Recall that if c is currently in self's hand, then $ob(c) = 1$. However, if c is in a meld already, then we set $ob(c) = 0$ in the meldability calculations of any other potential melds, since c is not available for multiple melds simultaneously.

We can now define the overall meldability of c:

$$m(c) = \alpha_{m2} \cdot m_{rank}(c) + \alpha_{m3} \cdot m_{run}(c)$$

with parameter weights α_{m2} and α_{m3} summing to 1, to be tuned in Section 5.

4 Three Components of a Player

In Gin Rummy, a player has three decisions to make at each turn: (1) whether to *draw* from the stock or discard pile, (2) which card to *discard*, and (3) whether to *knock*. For each decision we describe below a parameterized *decider*. We can therefore define a computer player as a given set of parameter values controlling these three deciders.

4.1 Deciding From Which Pile to Draw

In a turn, the player chooses whether to draw the face-up card c_u from the discard pile or the face-down card from the stock pile. Let $d(c)$ represent the deadwood value of a card c, and let c_h be the unmelded card in the player's hand that maximizes $d(\cdot)$. The player might use one of the following criteria to choose to draw c_u: (1) c_u can immediately be part of a meld, (2) c_u cannot immediately be part of a meld but $d(c_u) < d(c_h)$, or (3) c_u cannot immediately be part of a meld but the player expects this to change in the future.

Clearly every player should draw when criterion (1) is met. Meldability may be leveraged for criterion (3), as considered briefly in Section 6. Here, we turn our attention to criterion (2): no meld with c_u; $d(c_u) < d(c_h)$.

To *always draw* c_u when it meets criterion (2) leads to frequent face-up draws, particularly early in the round when there are few melds and $d(c_h)$ is likely high. This is problematic because face-up draws give the opponent information about cards in hand. Furthermore, to quickly discard cards with high deadwood value may lead to the removal of highly meldable cards – particularly if the opponent is following a similar strategy. Thus, to always draw c_u when it meets criterion (2) would be unwise. On the other hand, to *never draw* c_u when it meets criterion (2) would also be unwise. Even when melds cannot be formed, the player may benefit from replacing c_h with c_u when $d(c_u) \ll d(c_h)$.

A hybrid approach seems best. Indeed, human players often advise holding high deadwood cards early in the round, but replacing them later (Solutions 2019). This corresponds to *not* drawing c_u under criterion (2) early in the round, but doing so later in the round. Thus we divide a round into two *stages* with a threshold α_{d0} (tuned in Section 5) for the number of turns (t) that have occurred. In the early stage of a round ($t < \alpha_{d0}$), criterion (1) must be met in order to draw c_u (i.e. c_u must form a meld). In the later stage

($t \geq \alpha_{d0}$), criterion (2) is also sufficient to draw c_u (i.e. c_u reduces deadwood). Note that $\alpha_{d0} = 0$ is equivalent to the strategy *always draw* if there is a reduction in deadwood, as described above. Similarly, $\alpha_{d0} = \infty$ is the strategy *never draw* given a mere reduction in deadwood.

4.2 Deciding Which Card to Discard

Having drawn a card from the stock or discard pile, the player next chooses a card to discard from the current hand of 11. To make this decision, we consider the 11 *candidate hands* of 10 cards that could be formed. Let h_i represent the hand formed by discarding card c_i, with integer $i \in [1, 11]$. We evaluate each h_i with an ensemble of hand evaluation policies. The ensemble's evaluation $eval(h)$ of a hand h is a weighted sum of the evaluation $eval_j(h)$ of each member j:

$$eval(h) = \sum_j eval_j(h) \cdot \alpha_{wj} \tag{2}$$

We require $\sum_j \alpha_{wj} = 1$ and $\forall j \ \forall h, \ eval_j(h) \in [0, 1]$. The ideal α_{wj} values are tuned as described in Section 5. Given this ensemble evaluation function we can find the h_i that maximizes $eval(\cdot)$, and discard c_i.

We now describe each hand evaluation policy $eval_j$ in the ensemble. It is not the case that each hand evaluation policy would be effective on its own, but in combination they make a discard decision that considers many factors.

Hand Evaluation Policy 1: Seek High Meldability Recall that the card meldability $m(c)$ estimates the likelihood that card c will eventually be part of a meld for self. Thus, cards with high meldability should be kept. We define the *hand meldability* $hm(h)$ of a hand of 10 cards (h):

$$hm(h) = \sum_{c \in h} m(c) \tag{3}$$

We normalize this by dividing by the maximum value hm_{max} observed over 10,000 games. Thus $eval_1(h) = hm(h)/hm_{max}$.

Hand Evaluation Policy 2: Lower Deadwood This policy evaluates a hand h based on its total deadwood score $d(h)$. The score is normalized by the maximum amount of deadwood possible, 98. The result is subtracted from 1 to obtain a value to be maximized:

$$eval_2(h) = 1 - d(h)/98 \tag{4}$$

Hand Evaluation Policy 3: Penalize Deadwood by Turn As discussed in Section 4.1, it may be beneficial early in the round to keep high-rank cards if they seem likely to become part of a meld. However, such a strategy becomes more dangerous as the round proceeds, due to these cards' high deadwood values. So, while formula (4) penalizes deadwood consistently throughout the round, here we apply a further penalty as the round proceeds. We compute $eval_3(h)$ exactly as $eval_2(h)$. However, we use an ensemble weight $\alpha_{w3,t}$ that increases as turn t increases:

$$\alpha_{w3,t} = \alpha_{w3,0} + \alpha_{ts} \cdot t^{\alpha_{te}} \tag{5}$$

where $\alpha_{w3,0}$ is the weight at the start of the round (i.e. the initial α_{w3} in formula (2)) and parameters α_{ts} and α_{te} shape the weight adjustment function, with $0 \leq \alpha_{ts} \leq 1$ and $0 \leq \alpha_{te} \leq 4$. Thus at turn t we set α_{w3} in formula (2) to $\alpha_{w3,t}$ and renormalize all ensemble weights to again sum to 1. α_{ts} and α_{te} are tuned in Section 5.

Hand Evaluation Policy 4: Penalize Deadwood by Known Opponent's Cards When opp draws a face-up card, this provides information to self. Opp would likely choose to avoid this unless the face-up card is deemed sufficiently useful. Thus, the more face-up cards opp has drawn, the closer opp may be to knocking, and the more dangerous it is for self to keep high-deadwood cards.

While formula (5) penalizes deadwood according to turn t, this policy penalizes deadwood by increasing the ensemble member's weight according to the number of face-up cards k that self knows opp has:

$$\alpha_{w4,k} = \alpha_{w4,0} + \alpha_{os} \cdot k^{\alpha_{oe}}$$

In every other way this policy behaves analogously to the previous. Parameters will be tuned in Section 5.

Hand Evaluation Policy 5: Bonus for Aces and Twos Aces, twos, queens, and kings inherently have lower run meldability than cards of other ranks, because other ranks have four cards ($\pm$ two ranks) that could form a run meld of length three. Aces and kings only have two such cards, while twos and queens have three. We ignore queens and kings here due to their high deadwood values. In light of the low deadwood values of aces and twos, however, this ensemble member is designed to give them a small bonus to counteract the run meldability penalty:

$$eval_5(h) = n/8$$

where n represents the number of aces and twos in h with rank 1 or 2, which is then normalized by the maximum number of these cards possible in a hand (8). While this policy is ineffective on its own, it serves as a potentially useful additional voice in an ensemble.

Hand Evaluation Policy 6: Seek High Adjacency Recall that hand meldability (formula (3)) considers not just cards in the hand but also estimated probabilities of obtaining useful cards *not* in the hand. This simpler "adjacency" policy considers only cards *in hand*. We first define the *card adjacency* $a(c)$ for card c:

$$a(c) = n_{c,0} \cdot \alpha_{c0} + n_{c,1} \cdot \alpha_{c1} + n_{c,2} \cdot \alpha_{c2}$$

where $n_{c,0}$ is the number of cards with the same rank as c, and $n_{c,1}$ and $n_{c,2}$ are the number of cards with the same suit and 1 or 2 ranks away, respectively. The parameters α_{c0}, α_{c1}, and α_{c2} (tuned in Section 5) represent the respective weight of each adjacency type. Each is constrained to the range [0,1], and they sum to 1. We also require $\alpha_{c1} > \alpha_{c2}$ since a card 1 rank away provides more meld possibilities than a card 2 ranks away.

We then define the *hand adjacency* $a(h)$ for hand h:

$$a(h) = \sum_{c \in h} a(c)$$

We normalize this by dividing by the maximum value a_{max} observed over 10,000 games. Thus $eval_6(h) = a(h)/a_{max}$.

4.3 Deciding When to Knock

If for hand h, $d(h) \leq 10$, then the player must make a third decision: whether or not to knock. If $d(h) = 0$, knocking is guaranteed to be beneficial; otherwise, there are tradeoffs to consider. A knock with high deadwood (but still ≤ 10) risks an undercut, but may be advantageous if the opponent has even more deadwood. To wait for a hand with lower deadwood may reduce undercut risk, but also gives the opponent more time to obtain a better hand.

In an effort to address this, we consider the round to be in one of two stages. Let t be the number of elapsed turns and define a parameter α_{t2}. The round is in the *early* stage when $t < \alpha_{t2}$, and the *late* stage otherwise. Let $kn(t)$ be the *knocking threshold*, in which the player will only knock with hand h at turn t if $d(h) \leq kn(t)$. Define:

$$kn(t) = \begin{cases} \alpha_{t0} & \text{if } t < \alpha_{t2} \text{ (early stage)} \\ \alpha_{t1} & \text{otherwise (late stage)} \end{cases}$$

with additional parameters α_{t0} and α_{t1}. We require $\alpha_{t0} > \alpha_{t1}$ since self should be more willing to make higher-deadwood knocks at the early stage than the late stage.

Since t is always greater than 0, we can also set $\alpha_{t2} = 0$ and vary α_{t1} as desired to obtain a *one-stage* strategy. We evaluate various knock strategies in Section 5.

5 Parameter Tuning Experiments

We use genetic algorithms (Holland 1992) and grid searches to tune the parameters. A player is created for evaluation with a set of parameter values, determining the behavior of the draw, discard, and knock deciders.

Since many strategies depend on accurate state tracking, we tune the state tracking parameters first, using a genetic algorithm with a fitness function that isolates those parameters from the influence of others. We then tune the knock decider parameters with a grid search, since those too can be tuned in isolation from the others. Finally, we tune the remaining parameters through additional grid searches.

5.1 A Genetic Algorithm to Tune State Tracking

In this section, we discuss the tuning of state tracking parameter β, introduced in Section 2. Recall that state tracking includes the estimation of $p(c)$ for `unknown` cards, where $p(c)$ is the estimated probability that c is in stock, and $1 - p(c)$ is the estimated probability that c is in opp's hand. As described previously, these probabilities can be updated through *baseline* updates at each turn, and also by *adjacency* updates when self observes opp's actions. There are three actions opp may take that lead to adjacency updates: (1) draw a face-up card, (2) decline a face-up card, and (3) discard a card. In combination with the three types of adjacent cards, there are $3 \cdot 3 = 9$ distinct probability update scenarios. Thus what was introduced in Section 2 as one adjacency update parameter β is actually nine – one for each scenario.

For a given card c, let $A_{c,0}$ be the set of `unknown` cards with the same rank as c, and $A_{c,1}$ and $A_{c,2}$ the sets of `unknown` cards of the same suit that are 1 or 2 ranks away, respectively. When opp draws face-up card c, the in-stock probabilities of adjacent cards are decreased. For example:

$$\forall a \in A_{c,0},\ p(a) \leftarrow p(a) - p(a) \cdot \alpha_{s0}$$

with parameter α_{s0}. A similar update occurs for $A_{c,1}$ and $A_{c,2}$ with parameters α_{s1} and α_{s2}, respectively.

When opp declines face-up card c, the in-stock probabilities of adjacent cards are increased. For example:

$$\forall a \in A_{c,0},\ p(a) \leftarrow p(a) + (1 - p(a)) \cdot \alpha_{s3}$$

This occurs similarly for $A_{c,1}$ and $A_{c,2}$ with parameters α_{s4} and α_{s5}, respectively. Similarly, the probabilities of the three sets of adjacent cards are also increased when opp discards c, with parameters α_{s6}, α_{s7}, and α_{s8}.

Again, we call the above probability updates *adjacency updates*, to distinguish from the *baseline updates* based only on the number of `unknown` cards (Equation 1). Recall that these occur only for adjacent cards with state `unknown`. For all other cards, no probability is tracked, because the state of the card is known with certainty. Also note that each parameter above is constrained to the range $(0, 1)$. Finally, we constrain $\alpha_{s1} \geq \alpha_{s2}$, $\alpha_{s4} \geq \alpha_{s5}$, $\alpha_{s7} \geq \alpha_{s8}$ because intuitively cards that are 1 rank away from c should be influenced more strongly than cards that are 2 ranks away.

It might be argued that some of the above scenarios can or should use the same parameter value. This nine-parameter approach gives us greater expressiveness at the cost of higher dimensionality. However, since the state tracking parameters are considered in isolation (as justified below), we choose greater expressiveness in this case.

We tune these parameters with a genetic algorithm. An individual is nine numbers, corresponding to the nine parameter values, subject to the constraints above. An initial population of 100 random individuals (100 "selfs") is created. At each generation, each individual plays 2,000 games against a player who behaves the same as the simple player (Section 2) but knocks only on gin to prolong each round and generate more data.

Let U_o be the `unknown` cards that are actually in opp's hand (not in stock). Self attempts to guess the cards in U_o based on the in-stock probability estimates, and the fitness function counts the number of correct guesses. Instead of always requiring that self guess precisely the elements of U_o (a very difficult task), self chooses the $(|U_o| + ad)$ cards with the lowest in-stock $p(c)$ values, where $ad \geq 0$ is the number of additional cards self considers. These selected cards form the set G_{ad}, on which the fitness is calculated:

$$Fitness_{ad}(G_{ad}) = |U_o \cap G_{ad}| / |U_o|$$

Thus, the higher ad is, the greater allowance there is for some error in the in-stock probability estimates. We discuss different values of ad below.

The genetic algorithm computes this fitness for each individual. Individuals are selected by tournament selection (Miller et al. 1995). That is, 3% of the population is chosen randomly, and the fittest individual among them is selected. This process is repeated to select many individuals in sequence, consecutive pairs of which undergo two-point crossover. The children are mutated with a 2% chance per gene per child, to create the next generation. Finally, we also

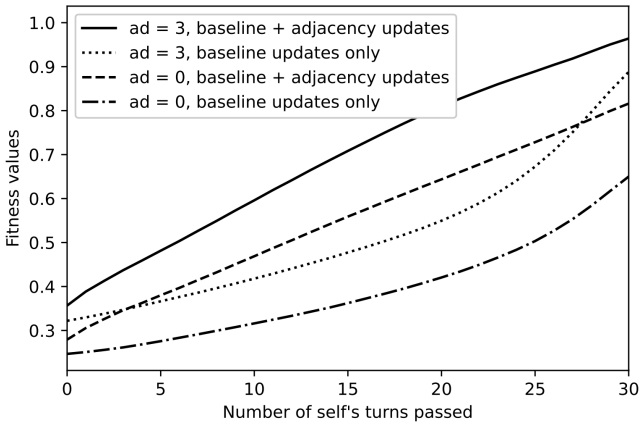

Figure 1: In tuning the state tracking parameters, the average fitness of the best individual based on number of turns, for two fitness functions ($ad = 0$ and 3) and two probability update types.

apply *elitism* (Whitley and Sutton 2012), in which the single highest-fitness individual from one generation proceeds to the next without any crossover or mutation adjustments.

Again, in this procedure opp is the simple player, which has no parameters. Since U_o depends only on opp's actions and random chance, this fitness calculation is not influenced by any other parameters. Thus it is reasonable to consider these nine parameters in isolation.

Results and Discussion In the experiment, we apply the above genetic algorithm in its entirety in six different ways: with either $ad = 0$ or 3, and with different times and frequencies of measuring fitness. To compare the results of the six separate genetic algorithm runs, we make six players, each with the corresponding run's best set of parameter values, and two additional players that use only the baseline update with either $ad = 0$ or 3. These eight players each then play 24,000 games against the simple player.

Selected results are shown in Figure 1. Experiments using other fitness-measuring times and frequencies lead to nearly identical results, and thus are omitted. In all cases, as the number of turns increases, fitness increases. This suggests that the baseline probability adjustment is effective. Comparing the two lines with $ad = 0$, we see that the adjacency updates result in a consistently higher fitness than the corresponding baseline. Similar results are seen for $ad = 3$. This suggests that adjacency updates provide additional accuracy over the baseline. Nevertheless note that fitness is rather low earlier in the game, reflecting the uncertainty of predictions about `unknown` cards. The consequences of this uncertainty are explored further in Section 5.3.

With the results showing a similar growth pattern for both considered ad values, we arbitrarily choose the parameter values from the $ad = 3$ experiment that uses adjacency updates. These values are fixed for the tuning experiments that follow. While other state tracking parameter values may perform better against other (non-simple) players, such a consideration is beyond the scope of this paper.

		α_{t1}									
		0	1	2	3	4	5	6	7	8	9
α_{t0}	1	0.48									
	2	0.50	0.45								
	3	0.51	0.46	0.430							
	4	0.50	0.45	0.40	0.40						
	5	0.50	0.47	0.41	0.40	0.36					
	6	0.52	0.45	0.43	0.39	0.37	0.33				
	7	0.51	0.49	0.44	0.44	0.40	0.35	0.33			
	8	0.54	0.50	0.45	0.43	0.39	0.34	0.33	0.32		
	9	0.54	0.53	0.47	0.45	0.41	0.37	0.36	0.34	0.32	
	10	0.57	0.53	0.50	0.48	0.43	0.39	0.38	0.34	0.33	0.32

Figure 2: In tuning the knock decider parameters, win rate for each allowed α_{t0} and α_{t1} pair, for fixed $\alpha_{t2} = 6$.

5.2 Grid Search to Decide When to Knock

To explore knocking strategies, we first consider a search with a one-stage knock decider (Section 4.3). Specifically, we set $\alpha_{t2} = 0$ and vary α_{t1}, so that the player always knocks with hand h when $d(h) \leq \alpha_{t1}$. In all other ways the player behaves the same as the simple player: draw face-up only to form a meld, discard only to lower deadwood.

In a preliminary experiment we consider every combination of two such players: $11 \cdot 11 = 121$ pairings corresponding to the 11 possible values for each player's α_{t1}. Each pair of players plays 2,000 games. Results follow a very clear pattern. For any fixed α_{t1} value for player 1, player 0 performs best with $\alpha_{t1} = 0$, and mostly consistently worse for each higher α_{t1} value. These results suggest that the best one-stage knocking strategy is $\alpha_{t1} = 0$: knock only on gin.

Having established that, we move on to the main knock decider experiment: a search on a two-stage knock decider. Recall that this means the player will use threshold α_{t0} to be met for early-stage knocking and α_{t1} for late-stage knocking, with α_{t2} representing the number of turns to transition between the early and late stages. Again, two modified simple players are created: one using a two-stage knock decider, and one knocking only on gin ($\alpha_{t2} = 0$, $\alpha_{t1} = 0$). For the two-stage player we consider 825 different possibilities by varying the integer parameters with $\alpha_{t2} \in [5, 19]$ and $\alpha_{t0}, \alpha_{t1} \in [0, 10]$ subject to $\alpha_{t1} < \alpha_{t0}$. Each pair of these players plays 2,000 games.

Results and Discussion In this experiment, the highest winning rate, 0.57, is found with (α_{t0}, α_{t1}, α_{t2}) = (10, 0, 6). This corresponds to knocking as soon as possible (even with 10 deadwood points) in the first 5 turns of the round, and otherwise knocking only on gin (0 deadwood points). This strategy makes intuitive sense, as early in the round a sudden knock may catch the opponent unprepared, but later in the round the knock-on-gin approach is preferable. Also note that the best two-stage player is able to beat the best one-stage player (knock-on-gin) 57% of the time, suggesting that two-stage knocking with the correct parameterization is more effective than one-stage knocking. For fixed $\alpha_{t2} = 6$ as in Figure 2, for example, we observe that the two-stage player's win rate increases fairly consistently as α_{t0} reaches 10, and decreases fairly consistently as α_{t1} reaches 10, with the best win rate at $\alpha_{t0} = 10$, $\alpha_{t1} = 0$.

We can also explore whether these (10, 0, 6) values are ideal against a one-stage opponent with other α_{t0} knocking

thresholds. When considering $\alpha_{t0} = 2$, 4, 6, and 8, we find that (10, 0, 6) is consistently the best, ± 1 for α_{t2} (when to change stages). Thus, we conclude that (10, 0, 6) are the ideal two-stage knocking parameter values.

5.3 Tuning by Grid Search

We tune the remaining parameters in a series of grid searches. To constrain the search space, we divide the parameters into subsets by system component: (1) meldability $\{\alpha_{m0}, \alpha_{m1}, \alpha_{m2}, \alpha_{m3}\}$, (2) adjacency $\{\alpha_{c0}, \alpha_{c1}, \alpha_{c2}\}$, (3) deadwood penalty by turn count $\{\alpha_{ts}, \alpha_{te}\}$, (4) deadwood penalty by opp cards known $\{\alpha_{os}, \alpha_{oe}\}$, and (5) ensemble weights $\{\alpha_{w1}, \alpha_{w2}, \alpha_{w3}, \alpha_{w4}, \alpha_{w5}, \alpha_{w6}\}$.

Let α be a single parameter, and $\mathcal{A}$ the set of all parameters. Let $\mathcal{A}_s \subset \mathcal{A}$ be the subset of parameters that will have values varied in a search $s \in \{1, 2, 3, 4, 5\}$ as listed above. In a search, each $\alpha \in \mathcal{A}_s$ is varied within a given range according to a given step size; the precise variations depend on the parameter and are omitted here. Each possible combination of values for each $\alpha \in \mathcal{A}_s$ is considered, except those that violate any constraints on ordering (e.g. $\alpha_{c1} > \alpha_{c2}$) or total value (e.g. $\alpha_{c0} + \alpha_{c1} + \alpha_{c2} = 1$). The total number of parameter value combinations considered varies from hundreds to thousands for each $\mathcal{A}_s$, depending on s. Parameters in $\mathcal{A} - \mathcal{A}_s$ are held constant at values specified in $base_s$, defined below.

Finally, let v be an assignment of values to the parameters in $\mathcal{A}$, and let V_s be the complete set of assignments generated for search s according to the procedure above.

To begin the search process, we form an initial set $base_1$ of parameter values. We first fix the parameters related to state tracking and the knock decider to the optimal results found in Sections 5.1 and 5.2. We then initialize the remaining parameters to intuitively-reasonable values.

Given this initialization of $base_1$, we can describe the execution of search s. The $base_s$ values and the variation of values of each $\alpha \in \mathcal{A}_s$ generates a set of value assignments V_s. For each $v \in V_s$, a player $player_v$ is created and plays 5,000 games against the simple player (Section 2). This large number of games is used to reduce the influence of random chance according to cards dealt. (In our results for this experiment we also formally measure statistical significance.) Denote the corresponding proportion of wins for $player_v$ as $winRate(player_v)$. We seek the parameter values that maximize this win proportion. These values become the base values for the subsequent search $s+1$. That is: $base_{s+1} = \texttt{arg max}_{v \in V_s} winRate(player_v)$ With this definition, the process repeats for search $s + 1$.

Upon completing search $s = 5$, we then return to $s = 1$ again, and thus $base_1 = \texttt{arg max}_{v \in V_5} winRate(player_v)$. In this way the searches are completed in sequence for multiple *iterations*. We allow for this due to the possibility that the optimal values of parameters across various $\mathcal{A}_s$ are not independent – that after adjusting one set of parameters, another may benefit from further adjustment.

Results and Discussion Recall that at the start of the first iteration, $base_1$ represents the initial set of parameter values, with the results from the state tracking and knock decider tuning, and "reasonable" guesses for the remaining parameter values. These values earn a win rate of 0.701. This fairly high initial win rate likely comes from multiple factors: (1) the inherent effectiveness of the combined two-stage draw decider, 6-member ensemble discard decider, and two-stage knock decider, (2) the benefit of the prior state tracking and knock decider tuning, both essential for successful play, and (3) the reasonableness of the guesses for the remaining parameter values. Thus it is not surprising to find only modest improvements in this last set of tuning experiments. This makes it particularly notable, then, when we do see a significant effect on some parameter value changes. To clearly identify these situations, we performed a chi-square test on the win rate of the best parameter value assignment before and after *each* search (1 through 5) through *each* iteration (1 through 3), with a strict $\alpha = 0.01$ requirement for significance. Space allows consideration of only the most statistically significant results here.

First, consider the results before and after the entire first iteration of experiments, in which each experiment s from 1 through 5 is conducted. After completing iteration 1, the win rate of the best parameter values is 0.739 – a statistically significant improvement ($\chi^2(1, N = 5000) = 34.4, p < .001$) over the 0.701 win rate using $base_1$. It is also interesting to note that *after* this first iteration, there were no additional significant changes in win rates. This suggests that the parameter subsets $\mathcal{A}_s$ for each experiment s are fairly independent – that finding the ideal values for one subset does not then require a re-tuning of previously-tuned subsets. Thus, in the discussion that follows, we explore the significant results of experiments *within* iteration 1.

In meldability parameter tuning (experiment 1), we obtain a 0.720 win rate, which is significantly higher ($\chi^2(1, N = 5000) = 8.0, p = .004$) than the 0.701 win rate using $base_1$. This experiment found that the best values for α_{m2} and α_{m3} (the weight of rank and run melds; Section 3.2) are 0.4 and 0.6 respectively. Intuitively, the two types of melds are equally important; however, a given card can be in more distinct run melds (of length ≥ 3) than rank melds. Thus it makes sense that the experimental results slightly favor run melds.

Next, consider this experiment's tuning of α_{m0} (Section 3.1), in which we find that the value appears to have little influence on win rate. Recall that α_{m0} determines the obtainability of a card c when it has been drawn by opp from the discard pile. But for opp to draw c from the discard pile, self must first discard it (except in the first turn). Given self's prior discard decision, it is unlikely to want c again, and therefore any estimate on the obtainability of c, whether high or low, is inconsequential. Therefore it is not surprising that the value of α_{m0} also appears to be inconsequential.

Finally, consider experiment 1's tuning of α_{m1}, influencing the estimated probability of obtaining an `unknown` card (Section 3.1). Results show that this parameter's value also has little effect on the win rate. One could consider a high α_{m1} value as reflecting consistent "optimism" about obtaining an `unknown` card, and a low value reflecting consistent "pessimism". Recall, however, the considerable uncertainty of any such prediction. In Section 5.1 we found that de-

spite the demonstrated effectiveness of adjacency updates to probabilities, predictions remain only modestly accurate until late in a round. Whether self can obtain an `unknown` card depends on what cards opp holds but will discard, how many stock cards will be drawn in a round, and who will draw which cards from stock. For each drawn stock card there is a fairly equal chance that it goes to either player. Thus at times optimism is rewarded significantly when the card is drawn, and keeping adjacent cards was therefore wise. This significant reward offsets the penalties that sometimes come from keeping adjacent cards when the desired card is never obtained. Similarly, pessimism will lead a player to drop adjacent cards – perhaps a safer move if they have high deadwood value, but this may make it harder to form melds. So optimism may be a high risk / high reward approach, and pessimism low risk / low reward. So should a player keep or drop a card adjacent to an `unknown` card? The minimal effect of α_{m1} on win rate suggests that the decision is a wash given the information the system has. This reflects the dilemma each human Gin Rummy player faces a well.

This keep/drop dilemma ultimately may be made not by obtainability predictions (as controlled by α_{m0} and α_{m1}), but by considering deadwood points. This hypothesis is confirmed in experiment 5, tuning the discard decider ensemble weights α_{w1} through α_{w6}. The win rate of 0.739 is statistically significantly higher than the prior win rate ($\chi^2(1, N = 5000) = 9.7, p = .001$), and the resulting parameter values are illuminating. Recall that hand evaluation policies 2, 3, and 4 (Section 4.2) each aim to lower deadwood, under differing conditions. The hypothesized preeminence of reducing deadwood is supported in these results, in which the combined weight of these three policies $\alpha_{w2} + \alpha_{w3} + \alpha_{w4} = 0.6$. This is not to say that probability estimation is useless, however; despite the demonstrated uncertainty, its effectiveness is again supported with the non-zero weight of meldability (policy 1), $\alpha_{w1} = 0.2$. In these results we also find ideal weights $\alpha_{w5} = 0.1$ for the ace-two bonus (policy 5) and $\alpha_{w6} = 0.1$ for in-hand adjacency (policy 6). In summary, given the uncertainty of obtaining desired `unknown` cards and the comparative ease of reducing deadwood, the factors that make cards desirable (meldability, adjacency, etc.) have a minority but non-zero influence compared to deadwood reduction.

6 Future Work

There are numerous opportunities for the continuation of this work; unfortunately space allows only the briefest of commentary here. The grid searches could be performed with smaller or even variable step sizes, or a genetic algorithm could be employed to consider the entire parameter space at once. The draw decider could require a particular amount of reduction in deadwood before drawing a card, and could also consider meldability. A dataset of state information and results from knocking could be created through simulated games, and then fed to a machine learning system, to attempt to predict the results of knocking. Ensemble approaches could be employed for the draw and knock deciders. Finally, the meldability of opp's predicted cards could be used in the discard decider, to attempt to provide cards of lesser value to opp.

7 Related Work

There does not exist much prior work in Gin Rummy. One exception is (Kotnik and Kalita 2003), which deals with optimizing a player's performance through temporal difference learning and co-evolutionary learning. Through self-play, the system's performance improves. This is in contrast to our system, in which performance is based on the collection of components and parameter tuning.

Gin Rummy is an *imperfect information* game – a broad area in which much prior work exists. For example, (Brown, Sandholm, and Amos 2018), (Seitz et al. 2019) (Billings et al. 2004) describe various tree search approaches to imperfect information games. Such an approach could potentially be combined with parts of our state tracking and decision-making components, though this would greatly increase the computational requirements. In contrast, (Brown and Sandholm 2019) discusses performance optimization of a poker player through self-play against five copies of the player using a Nash equilibrium strategy and Monte-Carlo counterfactual regret minimization.

Other Monte-Carlo approaches include (Frank, Basin, and Matsubara 1998), (Furtak and Buro 2013), (Lisý, Lanctot, and Bowling 2015). The exploration/exploitation tradeoff of such approaches is an alternative mechanism by which to handle a large search space, as we dealt with in this paper through grid search and genetic algorithms. Note that a similar tradeoff occurs in genetic algorithms, in which exploration and exploitation are balanced via the tournament size, mutation rate, and crossover method.

8 Conclusion

This research presents an effective strategy for a computer Gin Rummy player composed of three deciders: a draw decider, discard decider, and knock decider. The draw decider considers whether the face-up card is expected to be melded, and in later turns, also whether the card would simply reduce deadwood. The discard decider is an ensemble of hand evaluation policies, considering a wide range of factors including deadwood points, expected melds based on the observed state of the round, and adjacency of cards in hand. The knock decider determines whether to end the round, based on a deadwood point threshold that varies depending on whether the round is in an early or late stage. Finally, three sets of experiments were conducted to optimize the parameters that govern these decisions: (1) a genetic algorithm for tuning the state tracking parameters, (2) a grid search to find the thresholds of deadwood points for knocking at both round stages, and when to transition from early to late stage, and (3) a grid search for the parameters of the hand evaluation policies and their weights in the ensemble.

9 Acknowledgments

We express sincere thanks to Dr. Michael Roberts and Dr. Naima Shifa for assistance in statistical analysis.

References

Billings, D.; Davidson, A.; Schauenberg, T.; Burch, N.; Bowling, M.; Holte, R.; Schaeffer, J.; and Szafron, D. 2004. Game-tree search with adaptation in stochastic imperfect-information games. In *International Conference on Computers and Games*, 21–34. Springer.

Brown, N.; and Sandholm, T. 2019. Superhuman AI for multiplayer poker. *Science* 365(6456): 885–890.

Brown, N.; Sandholm, T.; and Amos, B. 2018. Depth-limited solving for imperfect-information games. In *Advances in Neural Information Processing Systems*, 7663–7674.

Frank, I.; Basin, D. A.; and Matsubara, H. 1998. Finding optimal strategies for imperfect information games. In *AAAI/IAAI*, 500–507.

Furtak, T.; and Buro, M. 2013. Recursive Monte Carlo search for imperfect information games. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, 1–8.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press. ISBN 0262082136.

Kotnik, C.; and Kalita, J. 2003. The Significance of Temporal-Difference Learning in Self-Play Training TD-Rummy versus EVO-rummy. 369–375.

Lisỳ, V.; Lanctot, M.; and Bowling, M. 2015. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *Proceedings of the 2015 international conference on autonomous agents and multiagent systems*, 27–36.

Mark, F. 2017. AI Learning Gin Rummy, Part I URL https://towardsdatascience.com/learning-gin-rummy-part-i-75aef02c94ba. Accessed: 2020-07-01.

McLeod, J. 2020. Gin Rummy. URL https://www.pagat.com/rummy/ginrummy.html. Accessed: 2020-05-15.

Miller, B. L.; Miller, B. L.; Goldberg, D. E.; and Goldberg, D. E. 1995. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems* 9: 193–212.

Seitz, D.; Kovařík, V.; Lisỳ, V.; Rudolf, J.; Sun, S.; and Ha, K. 2019. Value Functions for Depth-Limited Solving in Imperfect-Information Games beyond Poker. *arXiv preprint arXiv:1906.06412* .

Solutions, A. 2019. Gin Rummy Strategy and Tips. URL https://medium.com/@Artoonsolutions/gin-rummy-strategy-and-tips-631debf5fa82. Accessed: 2020-05-15.

Whitley, D.; and Sutton, A. M. 2012. *Genetic Algorithms — A Survey of Models and Methods*, 637–671. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-92910-9. doi:10.1007/978-3-540-92910-9_21. URL https://doi.org/10.1007/978-3-540-92910-9_21.