Representational Sensitivity for Divide the Dollar Playing Agents

Andrew Dong and Daniel Ashlock

***Abstract*—Divide the Dollar is a two-player simultaneous game derived from the bargaining game invented by John Nash. This game is interesting because its strategy space contains an entire subspace of Nash equilibria. The large number of moves in the game means that it is somewhat challenging to design agents to play the game. This study examines the problem of designing representations for divide the dollar playing agents and tests a number of representations. It was found that representation and resources allotted were significant factors affecting the agents' performance and that fitness and resources used appeared to be negatively correlated during evolution. This study places agents with different representations into direct competition and finds that there are representations with a strong competitive advantage. Both the choice of representation and the allocation of resources to that representation are found to impact the ability of agents to make bargains and to compete with one another.**

Index terms — Representation, Representational Sensitivity, Mathematical Games, Game Theory, Divide-the-Dollar, Co-evolution.

I. Introduction

In earlier studies on the iterated prisoner's dilemma [8] it was found that the representation of agents in simple experiments with the prisoner's dilemma had a dominant effect on the outcome [6]. In this study we extend this work to a more complex mathematical game, Divide the Dollar, using a variety of representations enumerated in Section III. The goal of the study is to check if representation has a large role in the outcome of evolutionary experiments with divide the dollar playing agents, and we find it does. Failure to control for the effect of representation is a major lacuna in many published papers on evolving agents to play mathematical games and this study continued the process of demonstrating such control is critically necessary.

The game *divide the dollar* is a simplification of the bargaining game proposed by John Nash [15]. The game is a simultaneous two-player game in which each player makes a bid. If the bids have total of one dollar or less, each player receives their bid, otherwise they receive nothing. This game is supposed to model the process of making a bargain where the bargain can fail or be made. Each player is at risk of leaving some of the potential profit behind. The space of moves for divide the dollar is the set of pairs of positive fractions of a dollar, with the pairs summing to no more than one, yielding non-negative payoffs. A feature that makes this game interesting is that there is an entire subspace, pairs of moves that sum to exactly one dollar, that are all Nash equilibria. The game is similar to the ultimatum game, but the ultimatum game enjoys serial rather than simultaneous play.

Recall that a *Nash equilibrium* of a game is a set of strategies for players so that no player may improve their score by changing their strategy unilaterally. The Nash equilibria of a game are places where, in classical game theory, players are likely to settle. Having an entire connected subspace of Nash equilibria means that player behavior is harder to predict. There is, *a priori*, no game-theoretic reason to prefer an even division of the dollar to an uneven split. Human actors see an even split as being desirable, but this is not a notion encoded by the definition of Nash equilibria and would have to be engineered into software agents.

In this study we examine a variety of different agent representations, including simple reactive agents, agents based on artificial neural nets, state conditioned agents, and agents based on a linear genetic programming representation.

The remainder of the study is structured as follows: section II examines the history of both divide the dollar and of representation in evolutionary game theory. Section III gives the design of experiments. Section IV gives and discusses results, while Section V draws conclusions.

II. Background

The earlier work on the representational sensitivity of the iterated prisoner's dilemma [6] showed that, by changing the agent representation, it was possible to drive the fraction of cooperative populations arising from evolution from 0% to 93%. This means that representation plays a dominant role. Worse, the cited study found neural nets to be largely uncooperative while other researchers found that neural nets did learn to cooperate [12]. This seemed to amplify representational sensitivity to representational instability, an even worse state of affairs.

An investigation into a possible reason for this failure to replicate the prisoner's dilemma behavior of neural nets [13] led to the discovery that not only representation, but the resources, both computational and informational made available to agents with a given representation, could substantially affect which behaviors arose under the influence of evolution. The types of resources varied included the number of hidden neurons in a neural net, the number of states in a finite state agent, the time depth of past play in a lookup table, and the granularity of probability in a Markov chain based agent.

For the prisoner's dilemma it was found that both representation and the more fine-grained issue of resources provided within a representation both had a significant effect on the

Andrew Dong is an independent researcher in Guelph, Ontario, Canada, dongandrew99@gmail.com

Daniel Ashlock is with the Department of Mathematics and Statistics at the University of Guelph, in Guelph, Ontario, Canada, N1G 2W1, dashlock@uoguelph.ca

The authors thank the University of Guelph for supporting this work.

978-1-6654-3886-5/21/$31.00 2021 IEEE

agent behaviors that arose during training of agents with an evolutionary algorithm. Every sort of resource tested, informational or computational, exhibited a measurable impact on behavior when varied. Changing the number of states in a state conditioned agent from 8 to 80, for example, completely reversed some agent behaviors [2].

In a pair of papers that used genetic programming [9] to train agents to play divide the dollar [1], [7] it was found, first, that evolved agents favor near even splits when they are drawn from a single breeding population. The second result was that when the players were spit into two breeding populations, with all play between agents from distinct populations, that the tendency to split the dollar nearly evenly decreased substantially. The evolution of fairer solutions was probably a kinship effect [11]. This is another nail in the coffin of the notion that simple simulations will provide a reliable way of modelling behavior. The details of the simulation are critical.

In [3] a generalization of divide the dollar was proposed that permitted visual design of additional bargaining games with controllable properties. This generalization replaces the set of coordinated bids in standard divide the dollar,

$$C = \{(x, y) : 0 \leq x, y \leq 1, x + y \leq 1\}, \tag{1}$$

with arbitrary sets. Any two dimensional subset of the positive quadrant with positive area specifies a generalization of divide the dollar. Each agent specifies a coordinate of a point (their bid) and if the point jointly specified by the player's bids lands in the coordination set, they receive their bids. A picture of the coordination set permits many of the properties of the resulting game to be deduced. A variety of possible scoring sets were explored in [5].

This version of divide the dollar generalizes naturally to higher dimensions, though the ability to retrieve properties of the game from visualizations of the set diminishes. A paper modelling undependable government subsidies with three-player generalized divide the dollar appears in [4]. This paper not only generalized to a three player version of the game, but also explored dynamic coordination sets. The unreliability of government subsidies was modelled by having a favourable portion of the coordination set that sometimes disappeared, modelling the failure of a government to fund industrial subsidies.

The current study intersects these two lines of research, into representation and resources for game playing agents on one hand and into divide the dollar one the other. For the sake of simplicity, this initial foray into representation for divide the dollar remains with the original divide the dollar game, where the scoring set is as in (1). The impact of representation on generalizations of divide the dollar is left for the future.

III. Design of Experiments

We begin by stating the fitness function we will use to drive evolution and compare different agents.

Definition 1: The *score* of a gene G against another gene H is the average payout when playing iterated Divide the Dollar with 50 iterations. This score is always between 0 and 1.

The number of rounds used for iterated play was chosen based on earlier studies [3], [4], [5]. Too small a number of rounds does not permit an agents ability to play to be assessed well, while large numbers of rounds of play take too long. Fifty is a compromise value arrived at in the earlier work. When comparing agents, we refer to such a direct comparison of G and H as a *match* between G and H.

When evolving agents, the fitness of a gene at a point in time is its average match score against all other population members. The fitness evaluation method is a round-robin tournament of all population members.

A. The Agent Representations

Each representation tested in this study has a binary variation operator, crossover, and a unary variation operation, mutation. All new population members are produced by applying the crossover operator to copies of population members selected from a 24 member elite. A single instance of mutation is applied to each new agent after it is produced.

1) Simple Agents: An agent that depends only on the previous bids by the two players was tested, called the *simple agent*, which is determined by three parameters: the first bid F, the aggression A, and the volatility V. The idea is to try and coordinate by moving in the direction of a deal if coordination did not happen in the last round, and to score slightly better if a deal happened in the last round.

Formally, the simple agent operates as follows: its first bid is F. Let a and b be the bids of the simple agent and the opponent on a given round. Then the next bid of the simple agent is

$$a' = \begin{cases} a + A & \text{if } (a, b) \in C, \\ (1 - V)a + V(1 - b) & \text{if } (a, b) \notin C \end{cases}$$

where C denotes the set of bids that are considered to be coordinated, which is that shown in Equation 1 in this study. If a deal was not made, the SimpleAgent moves a fraction of V of the way from a to $1 - b$, since $(b, 1 - b) \in C$ for all $b \in [0, 1]$.

The binary variation operator used is uniform crossover on the three parameters that form the gene, F, A, and V. The unary variation operator, mutation, consists of modifying each of the three parameters by some Δ, chosen uniformly at random from $[-0.2, 0.2]$.

2) Artificial Neural Networks (ANNs): Each ANN agent maintains a *history vector* $\vec{v_{\text{in}}}$, which stores the bids from both agents in alternating order and serves as the input vector to the ANN to generate each bid. The vector $\vec{v_{\text{in}}}$ has dimension m, where m is an even positive integer denoting the *memory* of the agent. Before play, each element of the history vector is initialized with 0.5.

The ANN has a hidden layer of h nodes and an output layer consisting of one node. The hidden layer vector is computed by an h-by-m weight matrix W_1, a 1-by-h matrix B_1, and a transfer function f_1 chosen from one of the

following functions: Sigmoid, ReLU, arctangent, tanh, and Heaviside. The function used for each of the two layers can be different functions and are a part of the agents' genetics. All functions were normalized to have outputs in the range $[0, 1]$. Similarly, the output layer is computed with appropriate genetic parameters W_2, B_2, f_2. The equations are:

$$\vec{v_1} = f_1(W_1 \vec{v_{\text{in}}} + B_1)$$
$$\vec{v_{\text{out}}} = f_2(W_2 \vec{v_1} + B_2)$$

The transfer functions f_1, f_2 are applied element-wise the matrices. The final matrix $\vec{v_{\text{out}}}$ has dimensions 1×1, whose value is used as the bid. A complete structure of an ANN agent used in this study can be seen in figure 1.

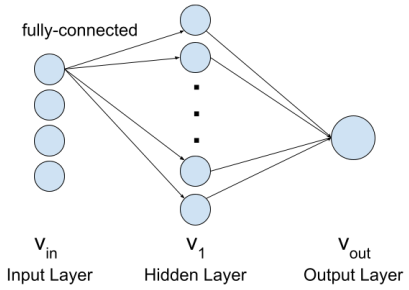

Fig. 1. A simple illustration of the Artificial Neural Network representations used in this study. An arrow from a node u to another node v indicates that the output from node u is used to compute the output of node v.

The binary variation operator is one-point crossover, producing two children, on the rows of W_1 and B_1 and the columns of the row vector W_2. The set of biases B_2 of the children is equal to the set of biases B_2 of the parents, with both orders being equally likely.

3) Moore Machines: The next agent representation used in this study are finite state Moore machines that generate bids as their output. Moore machines associate the output values they emit with their states, as opposed to Mealy machines that associate outputs with transitions. Transitions are driven by the outcomes of the previous play, using the three possible outcomes: I scored higher (H), my opponent scored at least as well as I did (L), and my opponent and I did not make a deal (F). These moves are called high, low, and fail. An example of one of these machines is shown in Figure 2.

The machine shown in in Figure 2 has only four states. Because the agent has one possible payoff *per state* the number of payoffs available is relatively small. This means that agents need to have enough states to give them an acceptable number of payoffs. In addition to payoffs and transitions, the agents need an initial play. The *starting state*, shown with a source-less arrow in Figure 2, yields the initial payoff. The starting state is always state zero.

The agents are represented as vectors of integers in the range $[0, 2^{13})$. For an s-state agent, the vector has length $4s$, with contiguous blocks of four integers specifying a state. Within a 4-tuple (n_1, n_2, n_3, n_4), we take the first three

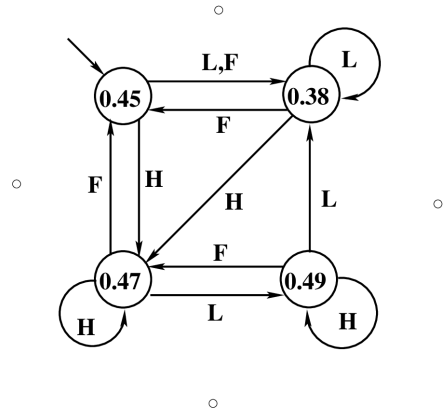

Fig. 2. An example of a four-state automata of the sort used in this study.

numbers $\bmod s$ to obtain the next-state transitions for play outcomes H, L, and F respectively. The fourth number is divided by 2^{13} to obtain the state label, which serves as the bid and is in the interval $[0, 1)$.

When updating a population, pairs of agents reproduce to replace other agents. This reproduction uses two-point crossover of the vector of integers. Mutation selects a number of loci and replaces the integers there with new ones generated uniformly at random. A parameter of the algorithm, the maximum number of mutations (MNM), controls the distribution of number of loci mutated. The number of loci mutated is chosen among all integers in the range $[1, \text{MNM}]$ uniformly at random.

4) Function Stacks: Function stacks are a form of linear genetic programming similar to Cartesian genetic programming [14]. A function stack is an array of nodes, each of which consists of an operation, two argument specifiers, and two ephemeral real constants. The output of a function stack is the value of node 0. When a node is evaluated, its argument specifiers are resolved and then the operation of the node is applied to the values of the arguments to yield the value of the node. The available operations are given in Table I. The protected division returns x/y unless x/y is not a number (NaN), in which case the operation returns x.

TABLE I
THE OPERATIONS AVAILABLE TO THE FUNCTION STACKS. PROTECTED DIVISION IS EXPLAIN IN THE TEXT.

Operation	Description
add	Add the arguments, $x + y$.
sub	Subtract the arguments, $x - y$.
mul	Multiply the arguments, $x \cdot y$.
max	Find the maximum, $\max(x, y)$.
min	Find the minimum, $\min(x, y)$.
amn	Arithmetic mean, $(x + y)/2$.
gmn	Geometric mean, $\sqrt{x \cdot y}$.
div	Protected division, x/y.
tws	Twist of arguments, $xy/(x^2 + y^2)$.

These operations were chosen as a simple collection of operations enabling basic arithmetic and various averaging operations that can form the foundation for effective strategies for an iterated divide the dollar game.

The argument specifiers for the function stacks include ephemeral constants in the range $[-1, 1)$, the output of earlier nodes in the stack, rendering the function stack a directed acyclic graph for purposes of information flow, and the input variables, which consist of earlier plays by the agents.

$$f(x) = \frac{e^{x-F}}{1 + e^{x-F}} \tag{2}$$

The agents function more efficiently if they can only produce valid divide-the-dollar bids — in this case, values in the interval $(0, 1)$. The genetic programming used can produce values in a much wider range. For that reason, a squashing function, given by Equation 2, was used to coerce the output into the correct range. This function generates a sigmoid curve with range $(0, 1)$. The constant F shifts the curve.

Similarly to the ANN agents, each function stack agent maintains a history vector of size m, denoting the bids from the last few rounds of play. These bids do not necessarily reflect the scores obtained as it is possible the bids were not in the coordination set C.

B. Other Agents

We also use a random agent that picks its bids uniformly in the range [0,1] and three constant agents that bid 0.4, 0.45, and 0.5 every time. The preceeding list comprises 45 agent specifications that are indexed in Table II. The index numbers are used in figures.

TABLE II
INDEX NUMBERS FOR EXPERIMENTS PERFORMED. M AND H ARE MEMORY AND HIDDEN NEURONS FOR ARTIFICIAL NETS. ST IS THE NUMBER OF STATES IN A MOORE MACHINE. N IS THE NUMBER OF NODES FOR A FUNCTION STACK WHILE F IS ITS SHIFT. C IS THE VALUE OF A CONSTANT AGENT.

#	Parameters	#	Parameters	#	Parameters
ANN		Function Stack		Function Stack (cont.)	
0	M6, H6	17	M1, N12, F0.0	29	M4, N12, F0.0
1	M6, H12	18	M1, N12, F0.5	30	M4, N12, F0.5
2	M6, H18	19	M1, N12, F1.0	31	M4, N12, F1.0
3	M8, H6	20	M1, N12, F1.5	32	M4, N12, F1.5
4	M8, H12	21	M1, N16, F0.0	33	M4, N16, F0.0
5	M8, H18	22	M1, N16, F0.5	34	M4, N16, F0.5
6	M10, H6	23	M1, N16, F1.0	35	M4, N16, F1.0
7	M10, H12	24	M1, N16, F1.5	36	M4, N16, F1.5
8	M10, H18	25	M1, N20, F0.0	37	M4, N20, F0.0
9	M12, H6	26	M1, N20, F0.5	38	M4, N20, F0.5
10	M12, H12	27	M1, N20, F1.0	39	M4, N20, F1.0
11	M12, H18	28	M1, N20, F1.5	40	M4, N20, F1.5
SimpleAgent				RandomAgent	
12	simple			41	random
Moore Machine				ConstantAgent	
13	St8			42	C0.4
14	St12			43	C0.45
15	St16			44	C0.5
16	St20				

C. Experimental Parameters

The Artificial Neural Network, Moore, and Function Stack agents had parameters that were modified and tested. They are as follows:

- Artificial Neural Networks (ANNs): memory $m \in \{6, 8, 10, 12\}$, hidden layer size $h \in \{6, 12, 18\}$
- Moore machines: number of states in $\{8, 12, 16, 20\}$
- Function stacks: memory $m \in \{1, 4\}$; number of nodes in $\{12, 16, 20\}$; shift $F \in \{0.0, 0.5, 1.0, 1.5\}$

For the ANNs and function stacks, the memory values represent input variables consisting of the recent history of play. A memory of 1 denotes that the gene remembers the opponent's last bid only. Larger memory parameters represent remembering the last bids of the agent and its opponent. For both ANNs and function stacks, all history values are initialized to 0.5 before they become available from play.

D. The Evolutionary Algorithm

Each representation variant listed in Section III-A was evolved with a population size of 36. Each of the 30 runs ran for 250 generations of the population; this value was chosen as it gave the population ample time to settle from the initial noise, without beginning to sample rare occurrences.

The algorithm uses fitness proportional selection on an elite consisting of the 24 highest scoring genes in each generation. This creates a moderate pro-fitness bias for selection. Each mating event consists of crossover between two different parents, who are chosen from the elite uniformly at random, without replacement. Each mating event overwrites two non-elite population members (hence there are $(36 - 24)/2 = 6$ mating events), and the offspring are subsequently each mutated. The use of a two-thirds elite follows earlier research in prisoner's dilemma and divide the dollar as a good trade off between stability, which arises from replacing relatively few agents, and exploration for new agent types which benefits from replacing more agents.

After selection occurs in each generation, a round-robin tournament of all 36 genes was conducted and the mean and best fitness as well as the coordination rate among all pairs of agents were saved. After the run, the parameters of the gene for the agent with the highest fitness were saved to be tested against agents evolved using other representations.

E. Experiments Performed

For each variant of each representation, 30 independent runs of the evolutionary algorithm were performed. The 30 best-of-run agents for each representation variant were saved for evaluation. Cross-representational matches then took place; all $1350 = 45 \times 30$ possible pairs of agents played 50 round matches and the results between each pair of representations were recorded.

For standardization, representations had to implement three functions:

- **Initialize.** The data stored by the agent is reset to prepare for a new game of divide the dollar.
- **Get bid.** The agent should provide a real number: the bid it intends to make on the next round.
- **Push bids.** The agent is given the results of the bid it and its opponent decided to make on the last round.

This interface allowed representations of different types to play against each other.

F. Analysis Techniques

In each run, the mean fitness and coordination rate over all matches was recorded after each generation. Let $m(g)$ and $c(g)$ denote the mean fitness and mean coordination rate on the g-th generation.

We consider the average value of $m(g)$ and $c(g)$ from generations $g \in \{50, 51, \ldots, 250\}$, which gives a measure of the performance of each representation variant, which are listed in Section III-C. The early generations (0 to 49) are not considered to allow the population time to settle from initial genetic randomness. The tournament includes representations, the random and constant agents, that were not evolved.

Call an agent that chooses its bid uniformly at random from the interval $[0, 1]$ a *RandomAgent*. Consider a match between two RandomAgents, where the first RandomAgent bid x and the second bid y. Note that on rounds where the two RandomAgents coordinate, (x, y) is a point chosen uniformly at random in the right triangle with vertices $(0, 0), (1, 0), (0, 1)$. Thus, the expected total payout for both players is

$$2 \int_0^1 \int_0^{1-x} (x + y)\, dy\, dx = \frac{1}{3},$$

where the factor of 2 comes from the fact that the right triangle has area $1/2$. Thus, the expected payout for each RandomAgent is $1/6$. Moreover, clearly the expected fraction of coordinations between two agents, where at least one of them is a RandomAgent, is at most $1/2$. These values were experimentally found to be accurate. These values for the random and constant agents provide a baseline for our comparisons.

The results between all pairs of versions of representations tested were used to calculate the performance of each representation versus all other representations. For each pair of different representations (R, S), the widths of the 95% confidence intervals on the agents mean score were recorded. We say that one version or a representation *dominates* another if it has a higher mean score and their confidence intervals do not overlap. If the confidence intervals do not overlap we say the representations have no clear dominance. Note that dominance is anti-symmetric and transitive as a relation; we declare by fiat that an instance of a representation trivially dominates itself, in order to make dominance a partial order on the instances of representations tested. We formalize this in Definition 2.

Definition 2: For any pair of instances of representations, if the 95% confidence intervals on their population mean scores in competition with one another fail to overlap, we say the instance with the higher mean score dominates the other. We also define an instance as dominating itself, to permit dominance to satisfy the definition of a partial order.

We examine the partial order determined by dominance as another way to evaluate the performance between representations. Figure 5 displays the Hasse diagram [10] of this partial order.

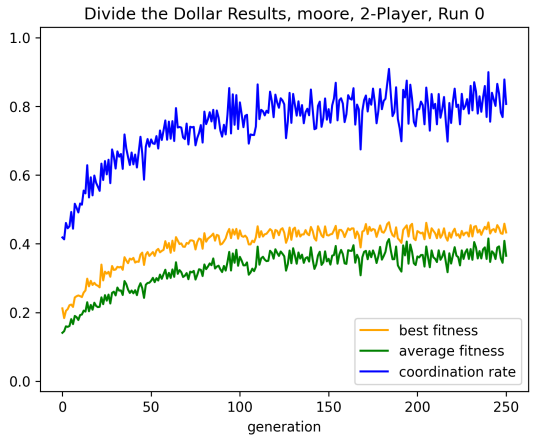

Fig. 3. One of the 30 runs with Moore machines with 20 states. The coordination rate fluctuated significantly from one generation to the next.

IV. Results and Discussion

For each run, the best fitness, population average fitness, and coordination rates were saved in each generation. Figure 3 shows the progression of one such run, demonstrating nominal evolutionary behavior for agent training.

A. Evolution of Representations

Figure 7 shows the range of coordination and fitness of the ANN, Moore, simple, and function stack representations from the evolutionary runs. Figures 8 and 9 break out the ANN and Moore agents for added clarity. The x- and y-coordinates show the average of the population mean fitness and the average of the coordination rates respectively, taken over generations 50 to 250 for all 30 runs of each representation variant. The data expressed by these coordinates are the normalized area under the blue and green curves of Figure 3 for all representations over all 30 runs. The crosses show the 95% confidence intervals for both dimensions.

In each evolutionary run, SimpleAgents evolved to have an aggression A close to 0. This is because significantly negative A are not beneficial to the agent as its bid will never increase, and very positive A yield collisions with other SimpleAgents.

The SimpleAgents in each run evolved to be very similar. However, each of the 30 SimpleAgent representatives had quite different values of V and F, and so did not coordinate very well with each other. Hence, each run can be said to have a "culture" that yields good performance within the run, but not when compared against other SimpleAgents from different runs or other agents of different representations.

Figure 4 shows some typical patterns that were observed. The majority of runs had a "spike" where the coordination and fitness would increase rapidly within a small number of generations. It is important to note that the metric for evaluating individual described in III-F does not reflect the final scores of the genes.

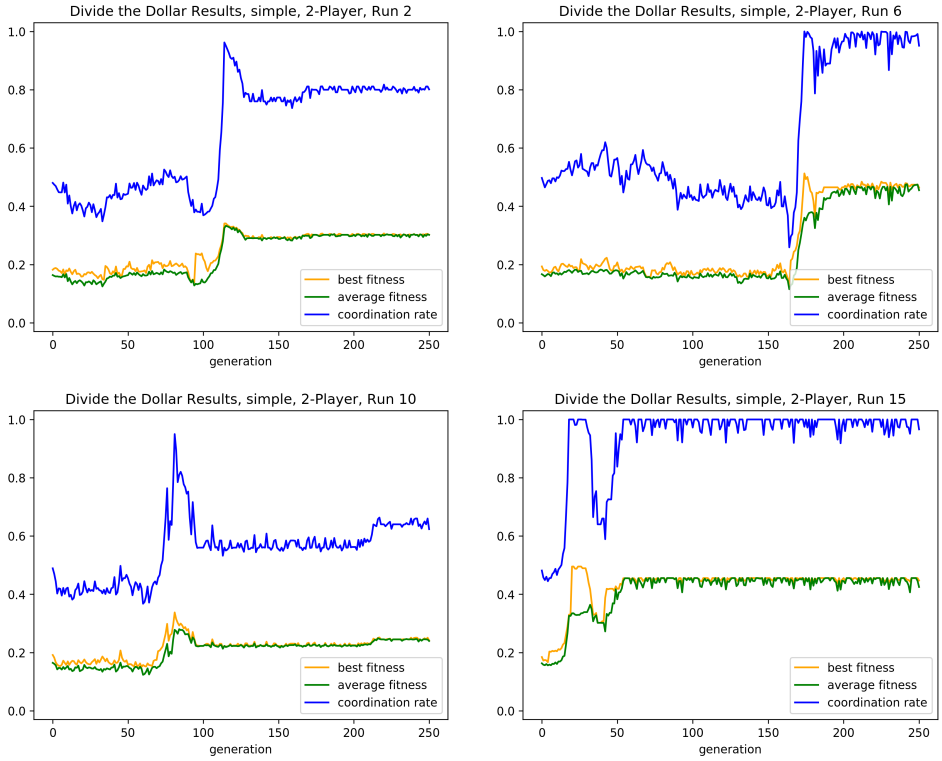

Fig. 4. Four runs from the evolution of SimpleAgents. Significant variability in the runs was observed as a result of the "cultures" that formed in each run. The scenario in the bottom left, where the coordination remained near 0.5 for a large portion of the 250 generations, occurred in a small number of runs; most runs exhibited a spike in fitness and coordination rate.

The Moore agents tended to perform worse with more states, coordinating less frequently and achieving worse average fitness. Figure 9 shows the correlation between resources and coordination.

For the artificial neural networks, a representation's performance was most influenced by the number of nodes in the hidden layer, and secondarily by the amount of memory. The representations using 18 hidden nodes achieved lower scores, on average, than representations using 12 hidden nodes, which in turn got lower scores than with 6 hidden nodes. Simpler neural agents learned divide the dollar better in a fixed amount of time.

Moreover, within the representations that used h hidden nodes for $h \in \{6, 12, 18\}$, the agents with less memory coordinated more. This result suggests that, while having more resources permits having a more intricate strategy, agents were not able to evolve to effectively make use of the increased information. However, ANNs with more nodes in each layer are able to simulate ANNs with fewer nodes layer. This illustrates that the duration of evolution required to achieve similar results, i.e. the rate of improvement, varies materially between agents with different amounts of resources.

Both the Moore machines and ANN agents achieved higher levels of coordination with fewer resources — the Moore machines that coordinated most frequently were the ones with 8 states, and similarly the ANN representation with the fewest hidden layer and input (memory) nodes reached the highest average coordination in evolution.

The function stacks all performed similarly, and evolved to be close to always coordinating; the main performance-differentiating factor was the shift F; larger values of F led to an inherently easier time coordinating, which resulted in a slightly better fitness in evolution.

All representations were able to evolve to achieve significantly higher payouts and coordination rates than a RandomAgent as discussed in Section III-F. In an earlier study on representational sensitivity in Iterated Prisoner's Dilemma [6], both Cooperative Neural Networks and Neutral Neural Networks were not able to reach a 50% probability of being better than random play by generation 250 of evolution. The different outcome here can likely be attributed to a much finer granularity of the game Divide the Dollar, as agents were required to bid a real number in the interval $[0, 1]$ rather than provide a binary choice of "cooperate" or "defect" — the granularity of all representations is significantly finer for Iterated Divide the Dollar playing agents.

B. *Cross-Representational Play*

Crossover of members must happen between two members of the same representation, and so each evolutionary run was conducted on one representation only. Thus, for some representation R, the evolutionary runs favour those that can perform well against agents of type R, but not necessarily of other types. Thus, the performance of R when playing

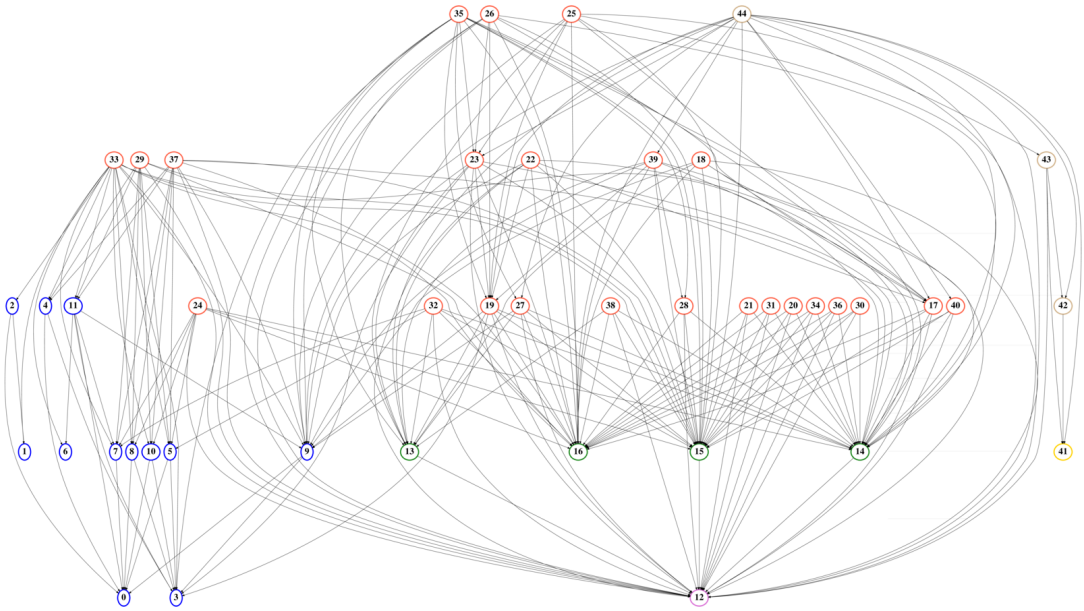

Fig. 5. Shown is a directed graph of competitive dominance. An arrow from node u to node v indicates that representation u dominates representation v. The experiments are keyed by their index numbers (see Table II), with ANN experiments shown in blue, Moore machines in green, function stacks in red, the simple agent in violet, the constant agents in brown, and the random number generator in gold, using the Graphviz dot layout engine.

against representations other than R reflects the robustness of R.

Figure 5 shows the partial order of dominance, explained in Definition 2. The width of this directed acyclic graph is a measure of the incomparability of representations. Notice that the agent representations are well grouped within the partial order, showing function stacks are superior to ANN and Moore agents. The Moore and ANN agents had no dominance over one another. The simple agent was dominated by almost everything else and the random agent performed badly.

The agents that were not evolved served as a baseline for comparison, as described in Section III-F. Against all representations from all four classes depicted in Figure 7, the ConstantAgent with constant 0.5 was able to achieve the highest overall score.

While it was noted that agents using fewer resources performed better during evolution, Figure 5 reveals that agents with higher memory are robust when playing against general opponents of varying representations. The representations numbered 2, 4, and 11 are ANN agents with hidden layer sizes 18, 12, and 18, respectively. Conversely, representations 0 and 3 are at the bottom of the digraph and have hidden layers of size 6, which is the smallest size that was tested. Similarly, function stack representations 25, 26, and 35 are not dominated by any other representation; these functions stacks used 20, 20, and 16 nodes, respectively.

The ConstantAgent with constant 0.5 performed the best as all representations evolved to bid slightly under 0.5 to coordinate with each other.

V. Conclusions

In this study, the representations with the highest fitness and coordination during evolution were the function stacks. The coordination rates varied moderately between the four classes of representations and a clear correlation between the amount of resources allotted to a representation and its coordination rate was observed. Compared to Iterated Prisoner's Dilemma, the complexity of Iterated Divide the Dollar means it is much more challenging to create robust agents that are able to perform well against an array of play styles.

This study demonstrates that representation and the amount of resources available are influential factors in the behavior of a divide the dollar playing agent, both performance-wise when playing against other agents and during evolutionary search.

Possible reasons for this variation are:

- **Sensitivity to random initialization.** Although our methods begin examining the population from generation 50 to allow for the early burn-in of random populations, exploitative agents may prevent the general population from increasing their fitnesses.
- **Sensitivity of the population to crossover and mutation.** In some representations with fewer parameters per gene, children have a high probability of having behaving similarly to the parents. In other cases, since the number of new children per generation is fixed, producing offspring that is easily exploitable or highly exploitative can cause instability in the population.

References

[1] D. Ashlock. GP-automata for dividing the dollar. In *Proceedings of the 1997 Genetic Programming Conference*, pages 18–26, Cambridge

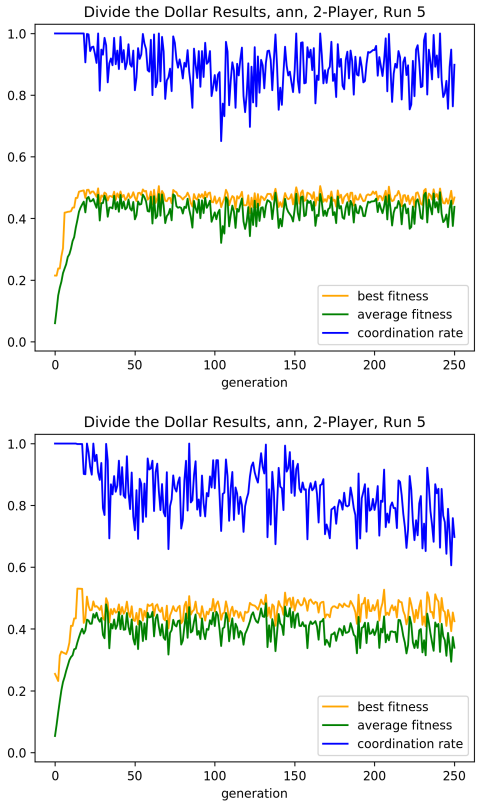

Fig. 6. Run 5 from the evolution of ANN agents with memory 6, hidden layer size 6 (top) and memory 12, hidden layer 18 (bottom). The results in Figure 8 can be explained by decrease in coordination, and thus also score, in ANN agents as agents further evolve, when agents have higher memory and hidden layers. This pattern was observed in the majority of runs in more resource-demanding ANN agents.

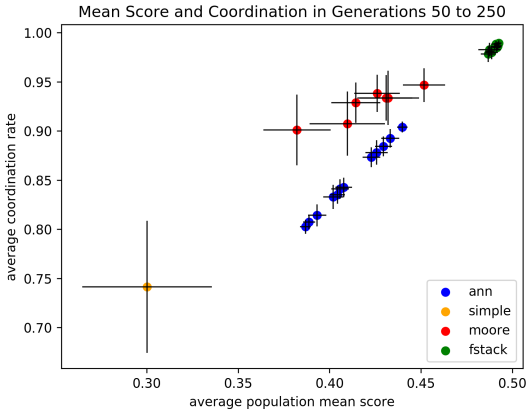

Fig. 7. Shown are 95% confidence intervals on mean fitness and coordination for each representation.

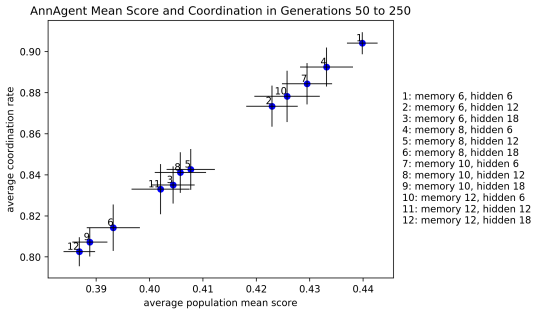

Fig. 8. This figure breaks out the ANN agents from Figure 7.

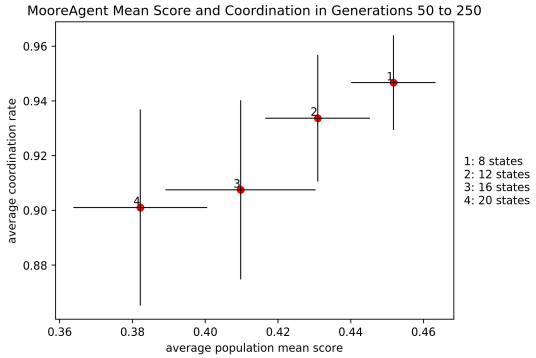

Fig. 9. This figure breaks out Moore agents from Figure 7.

MA, 1997. MIT Press.

[2] D. Ashlock, W. Ashlock, S. Samothrakis, S. Lucas, and C. Lee. From competition to cooperation: Co-evolution in a rewards continuum. In *Proceedings of the IEEE 2012 Conference on Computational Intelligence in Games*, pages 33–40, Piscataway, NJ, 2012. IEEE Press.

[3] D. Ashlock and G. Greenwood. Generalized divide the dollar. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 343–350, Piscataway, NJ, 2016. IEEE Press.

[4] D. Ashlock and G. Greenwood. Modeling undependable subsidies with three-player generalized divide the dollar. In *Proceedings of the IEEE 2017 Congress on Evolutionary Computation*, pages 1335–1342, Piscataway, NJ, 2017. IEEE Press.

[5] D. Ashlock, E.Y. Kim, and G. Greenwood. Characterizing scoring sets in generalized divide the dollar. In Press, IEEE Transaction on Games, 2021.

[6] D. Ashlock, E.Y. Kim, and N. Leahy. Understanding representational sensitivity in the iterated prisoner's dilemma with fingerprints. *Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, 36(4):464–475, 2006.

[7] D. Ashlock and C. Richter. The effect of splitting populations on bidding strategies. In *Proceedings of the 1997 Genetic Programming Conference*, pages 27–34, Cambridge MA, 1997. MIT Press.

[8] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[9] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming : An Introduction : On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, San Francisco, 1998.

[10] Garrett Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, RI, 1948.

[11] K. Foster and T. Wenseleers amd F. L. W. Ratnieks. Kin selection is the key to altruism. *Trends in Ecology and Evolution*, 21(2):57–60, 2000.

[12] G. Kendall, X. Yao, and S. Y. Chong. *The Iterated Prisoner's Dilemma, 20 years on*. World Scientific, 2007.

[13] E. Y. Kim and D. Ashlock. Changing resources available to game playing agents: Another relevant design factor in agent experiments. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(4):321 – 332, 2016.

[14] J.F. Miller and S. L. Smith. Redundancy and computational efficiency in cartesian genetic programming. *IEEE Transactions on Evolutionary Computation*, 10(2):167–174, 2006.

[15] J. Nash. Two-person cooperative games. *Econometrica*, 21(1):128–140, 1953.