Genetic Algorithms in Repeated Matrix Games: The Effects of Algorithmic Modifications and Human Input with Various Associates

Yomna M. Hassan and Jacob W. Crandall

***Abstract*—In many real-world systems, multiple independent entities (or agents) repeatedly interact. Such repeated interactions, in which agents may or may not share the same preferences over outcomes, provide opportunities for the agents to adapt to each other to become more successful. Successful agents must be able to reason and learn given the dynamic behavior of others. This is challenging for artificial agents since the non-stationarity of the environment does not lend itself well to straight-forward application of traditional machine learning methods. In this paper, we study the performance of genetic algorithms (GAs) in repeated matrix games (RMGs) played against other learning agents. In so doing, we consider how particular variations in the GA affect its performance. Our results show the potential of using GAs to learn and adapt in RMGs, and highlight important characteristics of successful GAs in these games. However, the GAs we consider do not always perform effectively in RMGs. Thus, we also discuss and analyze how human input could potentially be used to improve their performance in RMGs.**

I. Introduction

In social networks, new-agent power systems, online advertising, and many other applications, computers (*agents*) must repeatedly interact with other intelligent computing resources that may not have the same goals. Given these repeated interactions, each artificial agent has the opportunity (and incentive) to learn and adapt to the behavior of other agents in order to maximize its own payoffs. Given the non-stationarity of the environment, straight-forward applications of traditional machine learning techniques are not justified.

While many different kinds of multi-agent learning algorithms have been proposed for such situations [1], [2], [3], this paper investigates the effectiveness of genetic algorithms (GAs) for learning in the presence of other learners. GAs have become a common optimization search technique, especially for large, but stationary, search spaces [4]. For problems with high computational requirements, methods have also been developed to use GAs in a distributed fashion [5], [6]. In such approaches, multiple agents cooperate to solve a problem in which all agents have the same goal. However, other multi-agent systems in which GAs could be applied require agents to learn successful behavior when other agents do not necessarily share the same goals and preferences. In such circumstances, a GA must be able to learn in the presence of other learning agents, including having the ability to quickly adapt to sudden changes in associates' behaviors.

Y. M. Hassan is with Intel Middle East Mobile Innovation Center (MEMIC), Egypt. E-mail: yomnax.hassan@intel.com.

J. W. Crandall is with the Masdar Institute of Science and Technology, Abu Dhabi, UAE. E-mail: jcrandall@masdar.ac.ae.

The behavior and effectiveness of GAs in this latter domain are less studied. Previous work has developed GAs for dynamic environments (e.g., [7]). However, these methods either do not consider the existence of other heterogeneous learning entities in the system, or they are designed to learn only under certain identified constraints [8], [9].

In this paper, we analyze the behavior and performance of GAs in repeated matrix games (RMGs) played against other learning algorithms. In so doing, we evaluate how algorithmic variations affect the agent's learning rate and long-term proficiency. We also discuss and evaluate the potential of interactive genetic algorithms (IGAs), or GAs that leverage human input, to enhance the performance of GAs in RMGs.

The paper proceeds as follows. In Section II, we provide background information and review previous related work. We then define a GA for RMGs and propose four variations on this basic algorithm in Section III. In Section IV, we evaluate these algorithms in a set of 2-player RMGs played against several different learning algorithms. While these algorithms perform well in many circumstances, they have a number of shortcomings. Thus, in Section V, we discuss and evaluate how human input can potentially be used to enhance a GA's performance in RMGs. Finally, we discuss conclusions and future work in Section VI.

II. Background and Related Work

In this section, we first define RMGs and introduce necessary notation. We then review related work on using genetic algorithms in dynamic environments and, more specifically, in RMGs.

A. Repeated Matrix Games

A matrix game M, also called a normal-form game, is defined as a tuple $M = (I, A, R)$, where I is the set of n players (or agents), $A = A_1 \times A_2 \times \cdots \times A_n$ is the set of joint actions (where A_i is the set of actions available to agent $i \in I$), and $R : A \rightarrow \mathbb{R}^n$ is a function that specifies a reward or payoff to each agent given the joint action. For simplicity, we assume that $m = |A_i| = |A_j|$ for all $i, j \in I$. In a matrix game, each agent $i \in I$ simultaneously chooses an action $a_i \in A_i$ without knowing the actions selected by the other agents. The resulting joint action $\mathbf{a} = (a_1, a_2, \ldots, a_n)$

produces a payoff vector $\mathbf{r} = (r_1, r_2, \ldots, r_n)$, where r_i is the payoff to agent i.

A repeated matrix game (RMG) is a matrix game that is played repeatedly by the same set of agents. It consists of a set of *episodes*, or time steps, where an episode is a single play (joint action) of the game. Let $r_i(\mathbf{a})$ denote the payoff to player i when the joint action $\mathbf{a} \in A$ is played. Additionally, let a_i^t be the joint action played by agent i in time step t, and let r_i^t denote the payoff obtained by player i in time step t. We assume that an agent observes the actions taken by the other agents, but not their rewards. That is, once the joint action $\mathbf{a}^t = (a_1^t, a_2^t, \ldots, a_n^t)$ is played, agent i observes a_j^t for all $j \in I$, but not r_j^t.

A *strategy* of player i, denoted π_i, is a probability distribution over the action set A_i for each possible joint action history h_t. Thus, the task of a learning algorithm in a RMG is to learn to a strategy π_i such that its average per-episode payoff over all time, denoted $\bar{r_i}$, is maximized. Formally, $\bar{r_i}$ is given by

$$\bar{r_i} = \frac{1}{T} \sum_{t=1}^{T} r_i^t. \tag{1}$$

Here, $1 \leq T \leq \infty$ is the number of episodes in the game. Since r_i^t is dependent on the actions of all agents and since other agents are also seeking to learn a strategy that will maximize their future rewards, a successful learning algorithm must consider not only the impact of its actions on its immediate rewards, but also how its strategy will influence the behavior of other agents in future episodes.

B. GAs in Repeated Matrix Games

Many machine learning algorithms have been designed for RMGs, particularly over the last several decades. These algorithms include reinforcement learning algorithms [10], [11], [12], [13], [3], belief-based and opponent modeling algorithms [14], [15], and expert algorithms [16], [17], [18]. However, GAs have not been as thoroughly evaluated in the context of RMGs played against other learning algorithms.

GAs are adaptive search algorithms that can be used for many purposes. They are based upon the principles of evolution and natural selection. GAs are adept at searching large, non-linear search spaces where it is not possible to solve the problem using traditional iterative search techniques [19]. GAs possess the ability to efficiently determine near optimal solutions in reasonable time frames by simulating biological evolution. Specifically, GAs simulate selection, mutation, and crossover.

A number of researchers have addressed using GAs in dynamic environments. However, these algorithms either (1) require heavy training, (2) are limited to environments in which all agents share the same goals [7], or (3) have been shown to work only under certain constraints [8]. Despite these limitations, this body of literature has identified techniques that have high potential for RMGs. In particular, this work shows that GAs for dynamic environments should increase the diversity within the population when change is detected within the environment. We consider this technique in the context of RMGs in this paper.

Despite the challenges of utilizing GAs in dynamic environments, GAs have been studied in the context of the repeated Prisoner's Dilemma, a well-studied RMG. For example, Axelrod used GAs in order to identify evolutionarily robust strategies in the prisoner's dilemma [20]. Additionally, in work more similar in nature to ours, Buntai studied the ability of a GA to learn a strategy in the repeated Prisoners' Dilemma played against static strategies [9]. Buntai's work provides initial analysis of how GAs may perform in RMGs. However, in this paper, we consider the ability of GAs to learn a successful strategy in RMGs played against associates that also learn (i.e., the environment is dynamic).

GAs have also been used in a simple formulation of a buyer-seller dilemma [21]. In this work, each chromosome in the GA's population implemented a particular mixed strategy. The performance of this GA was then compared to various reinforcement learning algorithms. The reinforcement learning algorithms were shown to adapt faster to feedback from the environment than the GA. However, the GA was shown to exhibit less variation, and, hence, better convergence properties. The authors of this work also raised questions of how human input could be used to enhance the performance of GAs [21]. We address this consideration in the context of RMGs in Section V.

III. Genetic Algorithms

In this section, we describe a set of GAs for RMGs. First, we describe the structure of a *basic* GA. We then propose four variations. In Section IV, we evaluate these algorithms in RMGs played against several different learning algorithms.

A. Basic Algorithm

In a typical GA, a population or pool of chromosomes is initially generated, usually randomly. The fitness of each chromosome is then computed, in our case by having each chromosome play the game for a certain number of episodes of the RMG. Based on these fitnesses, the chromosome population is then adapted via selection, mutation, and crossover. This new population is then evaluated, and the process repeats until the game ends. To fully specify this algorithm, we specify the chromosome structure implemented by our GA as well as parameters related to the population update.

1) Chromosome Structure: Central to a genetic algorithm's success is the structure of its chromosomes. In our implementation, each chromosome encodes a particular strategy. Here, a strategy defines the agent's action given the last ω joint actions taken by the agents in the game. Given an n-player, m-action RMG, there are $(nm)^\omega$ possible joint-action histories of length ω. Thus, each chromosome consists of $(nm)^\omega$ bits of information (or numbers), each of which specifies the agent's action for a particular joint-action history.

Let $h_t = (\mathbf{a}^{t-\omega}, \ldots, \mathbf{a}^{t-1})$ denote the last ω joint actions played before time t, and let h_t^i be the joint action $\mathbf{a}^{t-i}$ converted to a base 10 number. For example, in a 2-agent, 2-action game, the joint action $(0, 1)$ can be represented as the base 10 number 2, the joint action $(1, 0)$ can be converted to the base 10 number 3, and $(1, 1)$ can be converted to the

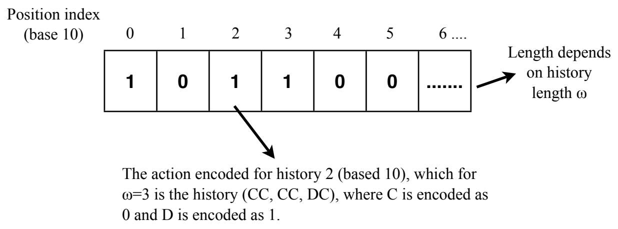

Position index (base 10)
0 1 2 3 4 5 6 ....
1 0 1 1 0 0 ....
Length depends on history length ω
The action encoded for history 2 (based 10), which for ω=3 is the history (CC, CC, DC), where C is encoded as 0 and D is encoded as 1.

Fig. 1. Example chromosome used in our implementation.

base 10 number 4. Then, to determine the action specified by a chromosome given the joint-action history h_t, we simply look up the number in position A_p in the chromosome, which is given by:

$$A_p = \sum_{i=1}^{\omega} m^{2(i-1)} \times base10(\mathbf{a}^{t-i}), \tag{2}$$

where m is the number of actions available to each agent, and $base10(\mathbf{a})$ is the base 10 number represented by the joint action $\mathbf{a}$. Small values of ω make the chromosome smaller (smaller search space), but limit the agent's modeling capability. In this paper, we use $\omega = 3$ as has been done in past work [22].

For example, consider Figure 1, which shows the first six bits of a chromosome for a 2-player, 2-action prisoners' dilemma. To determine the action encoded by this chromosome for the joint-action history $h_t = (CC, CC, DC)$, we encode each joint action of h_t in binary $(00, 00, 10)$, and use Eq. (2) to determine that $A_p = 2$. The number in position $A_p = 2$ (the first position is $A_p = 0$) in the chromosome is 1. Thus, this chromosome says that, given the joint-action history (CC, CC, DC), the agent should play action D.

2) Population Update: In our algorithm, we used standard forms of selection, mutation, and crossover as has been used in previous research [22], [23]. We use elitism selection in our algorithm, with elitism rate identified as $P_e = 0.3$. After selecting the elite chromosomes, we generate the remainder of the population through crossover and mutation process. That is, we set the probability of mutation to be $P_m = 0.01$ and the probability of crossover to be $P_c = 0.5$, with the crossover point selected randomly.

B. Algorithmic Variations

Branke [7] identified several successful modifications for GAs in dynamic (but cooperative) environments. Similarly, we found three variations to have important impact in RMGs: (1) fitness propagation over generations, (2) stopping conditions, and (3) a dynamic mutation rate. We discuss each in turn.

1) Fitness Propagation Over Generations: Normally, the new population of chromosomes is generated based on the fitness acquired in the latest generation. In RMGs played with learning associates, this causes the population to be generated in a myopic manner that can lead local, but not global, maxima. To avoid this, we can use the fitness of a chromosome as its average fitness in all generations in which it was present in the population. This average fitness can then be used to generate the new population.

TABLE I

Change in Fitness	$\mathbf{P_m}$	$\mathbf{P_e}$
Increased or equal	0.01	0.5
Decreased by $< 20\%$	0.05	0.5
Decreased 20–40%	0.1	0.3
Decreased by $> 40\%$	0.2	0.2

(a) Dynamic mutation and elitism rates

Name	**Fitness Prop.**	**Stopping Condition**	**Dyn. mutation rate**
Basic			
w/ Prop.	√		
w/ Stopping		√	
Dynamic			√
Dynamic w/ Stopping		√	√

(b) Five algorithmic variations

2) Stopping Condition: A dynamic environment can potentially cause a GA to continually shift its population over time, sometimes in a myopic manner. This problem can potentially be mitigated via an appropriate stopping condition. Once a stopping condition is met, learning ceases; the population remains fixed. We found it useful to stop evolving the population when the average fitness of the population remained stable for three straight generations.

3) Dynamic Mutation Rate: In dynamic environments, it is common to dynamically set the GA's parameters, such as the mutation and elitism rates, based on the chromosome's performance. If the fitness increases from one generation to another, we used a smaller mutation rate. If the average fitness of the population is deteriorating, we increased the mutation rate in attempt to generate a more fit population. This is similar to the win-or-learn-fast principle popular in the multi-agent learning literature [13]. Our GA variants that implemented dynamic mutation rates used the mutation and elitism rates specified in Table I(a).

IV. Evaluation

We evaluated five variations on our GA (Table I(b)) in RMGs played against other learning algorithms. Each of the variants incorporated a different combination of the basic algorithm and modifications discussed in the previous subsection. In this section, we first describe the experiment used to evaluate the algorithms, after which we describe and analyze the results of this experiment.

A. Experimental Setup

We paired each of our algorithms against three earning algorithms in four different RMGs. We describe these RMGs and learning algorithms in this subsection.

TABLE II
MATRIX GAMES CONSIDERED IN THIS PAPER.

	a	b
A	4, 4	0, 0
B	0, 0	2, 2

(a) Cooperation Game

	c	d
C	3, 3	0, 5
D	5, 0	1, 1

(b) Prisoners' Dilemma

	a	b
A	6, 6	4, 7
B	7, 4	2, 2

(c) Chicken

	a	b	c
A	0, 0	0, 1	1, 0
B	1, 0	0, 0	0, 1
C	0, 1	1, 0	0, 0

(d) Shapley's Game

1) Set of Games: We evaluated the performance of our GAs in four different RMGs: the Prisoners' Dilemma, Cooperation Game, Chicken, and Shapley's game. The payoff matrices representing the score given to the opponents when specific pairs of actions are performed within these games are shown in Table II. In each matrix, one player selects the row, while the other player selects a column. Thus, each cell of the payoff matrix represents an outcome of a joint action, with the row player's payoff specified first, followed by the column player's payoff. We selected games with expected various outcomes to visualize unexpected behaviour from both the GA variations and their associates in order to be able to generalize our results.

Cooperation Game is a common-interest game in which both agents always receive the same payoff. The other three games are games of conflicting interests. The players are neither in full cooperation nor full competition with each other. Thus, they must learn a profitable compromise if they are to achieve high payoffs.

2) Set of Associates: Previous work studying the performance of GAs in RMGs has focused on situations in which associates do not learn, but rather encode static strategies [9]. In this paper, we focus on situations in which associates also learn. We paired each GA with three different learning algorithms as opponents: itself (self play), Q-learning [24], and GIGA-WoLF [25]. These three associates were chosen to represent different popular genres of learning algorithms the agent could face, and the performance of these associates have been studied extensively in previous research [2].

GIGA-WoLF is a gradient ascent learning algorithm that exhibits the property of no-regret [25]. It adapts its learning rate according to its perceived success. It is an apt competitor. However, it often converges to myopic solutions in RMGs due since it does not form a strategy over past joint actions.

Q-learning is a model-free reinforcement learning algorithm that has been studied repeatedly in RMGs. Our implementation of Q-learning has a discount factor of $\gamma = 0.95$, encodes state as the last joint action played by the agents, and uses an ε-greedy exploration policy in which $\varepsilon = \frac{1.0}{10.0+(t/1000.0)}$.

Each repeated game consisted of 100,000 episodes. The GAs divided the game into 100 generations, with each generation consisting of 1,000 episodes. We ran 20 different trials for each pairing of algorithms in each game.

B. Results

Figure 2 shows the final distribution of payoffs after 100,000 episodes for all pairings in each of the four matrix games. We make several observations. First, by and large, all of the GA variants performed quite well against GIGA-WoLF. For example, ideal behavior against GIGA-WoLF is to defect in the Prisoners' Dilemma (since GIGA-WoLF learns to always defect), play the $(4, 4)$ solution in the Cooperation Game, and to play B in Chicken (GIGA-WoLF will then learn to play a). The GA variants that employ dynamic mutation and elitism rates (with and without the stopping condition) learned all of these outcomes.

Second, the GAs scored higher in the Prisoners' Dilemma against Q-learning than against GIGA-WoLF, since Q-learning does not always learn to defect in this game. Nevertheless, our GAs' performance in this game did not reach mutual cooperation, which would have resulted in a payoff of 3, which is possible against this Q-learning algorithm. However, our GAs' did not perform as well in the Cooperation Game or in Chicken against Q-learning. There is no clear trend indicating which GA variant is best against Q-learning.

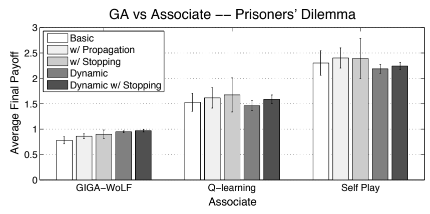
In self play, the GA variant employing a dynamic mutation rate with a stopping condition often performed the best, though the difference was not always statistically signficant. This variant was able to achieve average payoffs around 2.25 in the Prisoners' Dilemma, compared favorably with the other GA variants in both Chicken and the Cooperation Game, and obtained an average payoff of about 0.4 (better than random) in Shapley's Game. While adequate, these performance levels are below those of the Nash bargaining solution in each game [26]. Since a successful learning algorithm should be able to obtain the value of the Nash bargaining solution in self play [3], these results show that the learning behavior of each of these GAs is not ideal.

Figure 3 compares the five GA variants averaged over all four games and all three associates with respect to the average difference in final payoffs and average standard deviation (per game and associate). These results show that the Dyn-Stop performed statistically better than Basic and the GA with Propagation ($p < 0.001$). Additionally, the dynamic GAs had the lowest standard deviation, statistically lower than the GA with a stopping condition ($p = 0.002$).

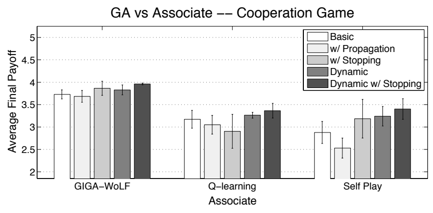
Overall, a GA employing a dynamic mutation rate and a stopping condition appears to perform the best overall in these RMGs against these opponents. However, this GA variant did not always perform as well as possible. This points to the need to enhance the GA learning process for RMGs. In the remainder of this paper, we focus on the potential of using human input to enhance this learning process.

V. ADDING HUMAN INPUT

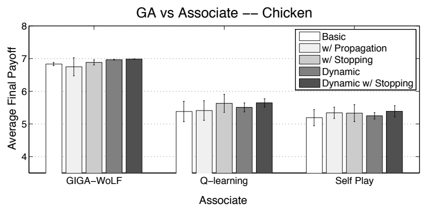
Given the potential and shortcoming of GAs in RMGs, we consider in the remainder of this paper how human input could potentially be obtained and used to increase the performance of GAs. Specifically, we discuss how a few intermittent demonstrations of behavior by a human can be used to potentially improve the performance of GAs.



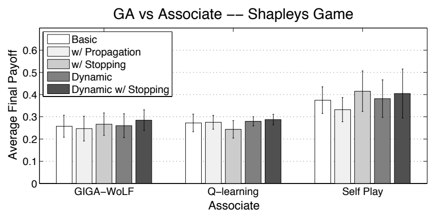
(a) Prisoners' Dilemma



(b) Cooperation Game

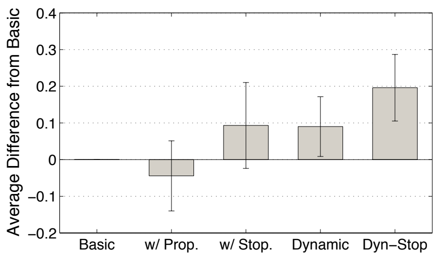


(c) Chicken

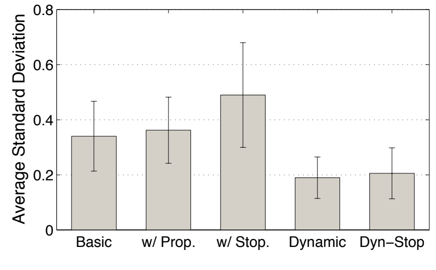


(d) Shapley's Game

Fig. 2. Comparison of various GAs in the four games against each associate. Error bars show a 95% confidence interval on the mean.



(a)



(b)

Fig. 3. Comparison of the (a) difference in mean and (2) standard deviation over all games and associates.

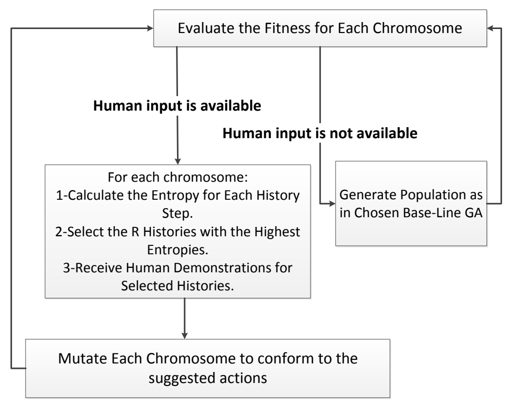

Evaluate the Fitness for Each Chromosome
Human input is available
Human input is not available
For each chromosome:
1-Calculate the Entropy for Each History Step.
2-Select the R Histories with the Highest Entropies.
3-Receive Human Demonstrations for Selected Histories.
Generate Population as in Chosen Base-Line GA
Mutate Each Chromosome to conform to the suggested actions

Fig. 4. Overview of our interactive GA (IGA) for adapting the population using a limited amount of human input.

A. Background

Previous work has investigated how human input can be used to enhance the performance of GAs. Human input can enhance the performance of GAs in various ways. The most common method is to use human input to help evaluate the fitness of chromosomes in the population. Applications of such methods include domains where it is hard or impossible to design an effective fitness function, including evaluating how images, music, and artistic designs fit a user's aesthetic preferences. This can be done by either having the human rank available solutions [27] or directly assigning the fitness function value to the available policies in the population. Human input has also been used in the mutation stage, where the human first selects the best policy from his or her point of view, and suggests a mutation operation to enhance its performance [28]. Finally, human input has also been used to set the parameters of a GA at the beginning of a board game, including the number of generations and the mutation rate [29].

B. Algorithm

In this paper, we focus on using human input to improve GAs in RMGs using the interactive GA (IGA) overviewed in Figure 4. In this algorithm, the human provides a few select demonstrations of successful behavior to the agent every few generations. Specifically, the human provides demonstrations of successful behavior for a few carefully selected histories of play. These demonstrations are used to generate the population of chromosomes in the subsequent generation.

1) Selecting Human Input: In our algorithm, the fitness of each chromosome in the population is evaluated in a generation as before. If the human is not available, the normal sequence of selection, crossover, and mutation is used to evolve the population for the next generation. However, if the human is available for input after the chromosomes' fitnesses are evaluated, demonstrations of human input are used to evolve the population.

Given that humans will typically only be able to provide a limited amount of input, the demonstrations sought by the algorithm from the human should be chosen carefully. Our algorithm does this by selecting R joint-action histories, and asking the human to provide demonstrations of successful behavior for each of these R histories. To select which demonstrations will be most useful, our algorithm computes the indecisiveness of the population for each joint-action history. Our metric of indecisiveness is a form of entropy, with chromosomes having higher fitness given more weight in this calculation.

Formally, we compute the entropy of action history h, denoted E_h, as follows:

$$E_h = \sum_{a \in A_i} -\log(P_h^a) \times P_h^a. \tag{3}$$

Here, P_h^a is the probability that the population will choose action a given the joint-action history h. Let C be the set of chromosomes in the population, and let $C_h^a \subseteq C$ be the set of chromosomes that choose action a given the history h. Also, let f_c be the fitness of chromosome c, and let $F_T = \sum_{c \in C} f_c$ be the *total fitness* of the chromosomes in the population. Then, P_h^a is estimated by:

$$P_h^a = \sum_{c \in C_h^a} \left(\frac{f_c}{F_T}\right)^2 \tag{4}$$

Once E_h is computed for each h, our algorithm selects the R histories with the highest entropy E_h. The system then asks the human to determine which action should be played given each of these R histories. We refer to these demonstrations as the *suggested actions*.

2) Mutating Chromosomes Given Human Input: Given the suggested actions for the selected joint-action histories, our algorithm generates a new population of chromosomes. This population is created by mutating each chromosome in the population to conform to the suggested actions, and then adding the resulting chromosome to the new population. Chromosomes are then ranked in the population according to the percentage of bits that match the human input.

C. Results

We now evaluate the potential effectiveness of the IGA described in the previous subsection. These results are intended to identify potential improvements that can be obtained by using human demonstrations. Thus, rather than use actual human input, we simulated human demonstrations in order to more easily evaluate the potential effects of human input on the performance of GAs in RMGs. We leave implementation with actual people to future work.

In analyzing the impact of human input on the effectiveness of GAs in RMGs, we wish to learn two things. First, we wish to determine the extent that human input, when used in the prescribed manner, can improve the effectiveness of the GA in RMGs played against other learning algorithms. Second, we wish to determine how less effective (or *uninformed*) demonstrations affect the algorithm's performance. While humans may sometimes be well-informed about successful strategies, they may not be (for many potential reasons).

TABLE III
INFORMED AND UNINFORMED DEMONSTRATION STRATEGIES FOR EACH GAME.

Game	**Human Input**	
	Informed	**Uninformed**
Prisoner's Dilemma	Tit-for-Tat	Inverse Tit-For-Tat
Cooperation Game	Play A or a	Play B or b
Chicken	Play B or b	Play A or a
Shapley's Game	Play randomly	Play a pure strategy

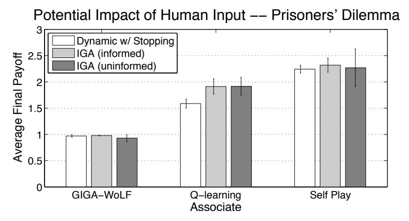
To pursue these two research agendas, we simulated two forms of human demonstrations. First, we simulated informed humans who demonstrate actions that tend to lead to high payoffs. Second, we simulated uninformed human demonstrations who demonstrate actions that tend to lead to low payoffs. Both of these simulations were carried out via hard-coded strategies, the strategy of choice at a certain point of time is played versus the opponent. These strategies for the four matrix games described previously are summarized in Table III. Informed strategies included Tit-for-tat in the Prisoners' Dilemma, the action corresponding to the pareto efficient solution in the Cooperation Game, the *bully* strategy in Chicken, and the maximin strategy in Shapley's Game. Uninformed strategies included the inverse of Tit-for-Tat in the Prisoners' Dilemma, the action with less potential in both Chicken and the Cooperation Game, and a constant deterministic strategy in Shapley's Game.

We paired the IGA with both informed (simulated) human input and uninformed (simulated) human input against GIGA-WoLF, Q-learning, and itself in the four games used in the previous section. Since the GA that used a dynamic mutation rate and a stopping condition performed the best in the previous section, we use it as a benchmark to evaluate the performance of our IGA in this section.

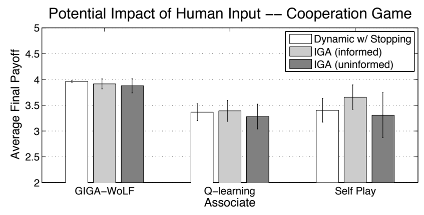
In each case, the IGA received four demonstrations (i.e., $R = 4$) from the human every fourth generations. Given that there were 100 total generations in each trial, this resulted in 100 demonstrations throughout a repeated game. Given a game with 100,000 moves, we deem this to be a relatively small number of demonstrations.

1) Potential of Human Input: Figure 5 compares the performance of four GAs in consideration in each of the games against each of the opponents. We make several observations about these results, beginning with a discussion of the impact of informed human demonstrations on the algorithms' performance. First, informed human input had little impact on the performance of the GA against GIGA-WoLF. As we observed in the previous section, GA is able to learn an effective strategy against GIGA-WoLF in each game.

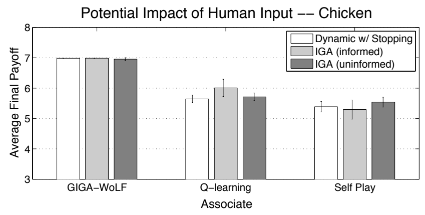
Second, Figure 5 shows that a limited number of informed human demonstrations could have a substantial (and statistically significant $p < 0.001$) impact on the performance of our GA against Q-learning in the Prisoners' Dilemma. While the GA without human input received an average reward of about 1.5, IGA with informed human input typically achieved a payoff near 2.0. A similar trend is seen in the case of Chicken. These result demonstrates the important impact that



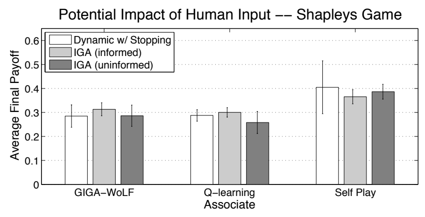
(a) Prisoners' Dilemma



(b) Cooperation Game



(c) Chicken



(d) Shapley's Game

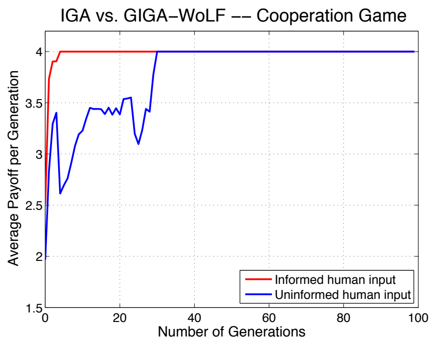
Fig. 5. Comparison of IGA and GA in each game against each associate. Error bars show a 95% confidence interval on the mean.

key interactions between a GA and a person can have on the algorithm's performance.

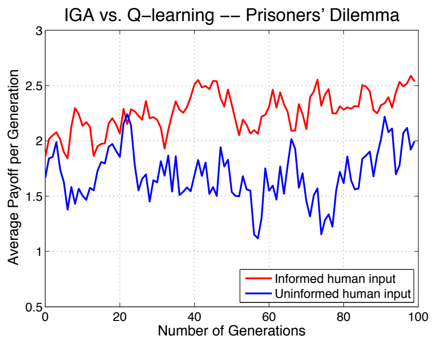
However, informed human demonstrations did not significantly increase the performance of a GA in self play. However, in the case of Shapley's Game and Cooperation Game, informed human input did served to reduce the variance in the algorithm's performance.

In summary, a limited number of carefully chosen human demonstrations can, in some circumstances, substantially boost a GA's performance in a RMG played against a learning opponent.

2) Impact of Demonstration Quality: We next discuss the impact of uninformed human input on the performance of GAs in RMGs played against other learning algorithms. In general, we would expect that uninformed human demonstrations would serve to decrease the effectiveness of the GA. Indeed, Figure 5 does show a decrease in performance in some circumstances. However, we note that, in many other circumstances, the GA was still able to maintain similar performance levels to those obtained by our GA in the absence of human input.



(a)



(b)

Fig. 6. Example trial runs of (a) IGA vs. GIGA-WoLF in the Cooperation Game and (b) IGA vs. Q-learning in the Prisoners' Dilemma.

Two sample demonstrations of the impact of uninformed (or unacceptable) demonstrations on the IGA's performance over time are shown in Figure 6. In these figures, IGA with informed human input is compared to IGA with uninformed human input. Figure 6a shows the performance of IGA over time in the Cooperation Game against GIGA-WoLF. While informed human input leads to fast convergence to the pareto efficient solution, uninformed demonstrations lead to much slower convergence. However, eventually the algorithm is able to overcome the negative impact of these uninformed demonstrations so that it too reaches the pareto efficient solution. We attribute these results to the fact that the IGA, although relies initially on human feedback, ignores chromosomes that were heavily supported by an uninformed human over its learning course with the opponent's stability. We consider this an important achievement.

Figure 6b shows runs of IGA against Q-learning in the Prisoners' Dilemma. The figure shows that informed demonstrations serve to boost the performance of GA throughout the repeated game, resulting in superior performance over the same algorithm given uninformed demonstrations. This shows both the potential and importance of having users that are able to provide informed input to a GA in RMGs played against other learning algorithms.

VI. Conclusions and Future Work

In this paper, we evaluated the ability of multiple variations of genetic algorithms (GAs) to learn in repeated matrix games (RMGs) played against other learning algorithms. We also explored the potential value of using a limited amount of human input to enhance the performance of GAs in such environments. Our results show that GAs often learn successful strategies in RMGs played against other learning algorithms, particular against learning algorithms with strong convergence properties. We also observed that using a dynamic mutation rate serves to increase the performance of the GA in RMGs.

However, GAs do not always learn successful strategies in RMGs. One potential solution to overcome these deficiencies is to leverage human input. Our simulations showed that a small number of demonstrations of successful behavior by a human could substantially improve the performance of GAs in RMGs played against other learning algorithms in some circumstances. Uninformed human input on the other hand sometimes decreases the performance of a GA, though the IGA we considered was often able to mitigate these negative effects.

During the course of this research, we discovered a number of areas that deserve further investigation. These areas include (1) testing the effectiveness of interactive GAs in more complex environments and against more opponents, (2) developing techniques to better overcome the negative effects of uninformed human input, (3) studying the use of actual human interactions with a GA that is playing a RMG, and (4) the potential of using human interactions with other learning algorithms.

References

[1] Y. Shoham, R. Powers, and T. Grenager, "If multi-agent learning is the answer, what is the question?", *Artificial Intelligence*, vol. 171, no. 7, pp. 365–377, 2007.

[2] B. Bouzy and M. Metivier, "Multi-agent learning experiments in repeated matrix games", in *Proc. of the 27^{th} Intl. Conf. on Machine Learning*, 2010.

[3] J. W. Crandall and M. A. Goodrich, "Learning to compete, coordinate and cooperate in repeated general-sum games", *Machine Learning*, vol. 82(3), pp. 281–314, 2011.

[4] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning", *Journal of Artificial Intelligence Research*, vol. 11, pp. 241–276, 1999.

[5] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey", *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[6] C.F. Huang, S. Bieniawski, D.H. Wolpert, and C.E.M. Strauss, "A comparative study of probability collectives based multi-agent systems and genetic algorithms", in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, pp. 751–752.

[7] J. Branke, "Evolutionary approaches to dynamic optimization problems-updated survey", in *Proceedings of the GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*. 2001, pp. 27–30, Citeseer.

[8] A. Alharbi, W. Rand, and R. Riolo, "Understanding the semantics of the genetic algorithm in dynamic environments", in *Applications of Evolutionary Computing*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007.

[9] C. Buntai, "A game-theoretic comparison of genetic and model-based agents in learning strategic interactions", Master's thesis, University of Alabama in Huntsville, 2010.

[10] Michael L. Littman, "Markov games as a framework for multi-agent reinforcement learning", in *Proc. of the 11^{th} Intl. Conf. on Machine Learning*, 1994, pp. 157–163.

[11] Michael L. Littman, "Friend-or-foe: Q-learning in general-sum games", in *Proceedings of the 18^{th} International Conference on Machine Learning*, 2001, pp. 322–328.

[12] Junling Hu and Michael P. Wellman, "Multiagent reinforcement learning: Theoretical framework and an algorithm", in *Proc. of the 15^{th} Intl. Conf. on Machine Learning*, 1998, pp. 242–250.

[13] Michael Bowling and Manuela Veloso, "Multiagent learning using a variable learning rate", *Artificial Intelligence*, vol. 136(2), pp. 215–250, 2002.

[14] Drew Fudenberg and David K. Levine, *The Theory of Learning in Games*, The MIT Press, 1998.

[15] G. Tesauro, "Extending Q-learning to general adaptive multi-agent systems", in *Advances in Neural Information Processing Systems 16*. 2004, MIT Press.

[16] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: the adversarial multi-armed bandit problem", in *Proc. of the 36th Symp. on the Foundations of CS*. 1995, pp. 322–331, IEEE Computer Society Press.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multi-armed bandit problem", *Machine Learning*, vol. 47, pp. 235–256, 2002.

[18] Daniela de Farias and Nimrod Megiddo, "Exploration–exploitation tradeoffs for expert algorithms in reactive environments", in *Advances in Neural Information Processing Systems 17*, 2005, pp. 409–416.

[19] L. Waltman, N. van Eck, R. Dekker, and U. Kaymak, "Economic modeling using evolutionary algorithms: the effect of a binary encoding of strategies", *Journal of Evolutionary Economics*, 2010.

[20] R. Axelrod, *The Evolution of Cooperation: Revised Edition*, Basic Books, revised edition, 2006.

[21] R. Sikora and V. Sachdev, "Learning bidding strategies with autonomous agents in environments with unstable equilibrium", *Decision Support Systems*, vol. 46, 2008.

[22] R. Axelrod, *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*, Princeton University Press, 1st printing edition, 1997.

[23] A.J. Lockett, C.L. Chen, and R. Miikkulainen, "Evolving explicit opponent models in game playing", *GECCO*, 2007.

[24] C.J.C.H. Watkins and P. Dayan, "Q-learning", *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[25] M. Bowling, "Convergence and no-regret in multiagent learning", in *Advances in neural information processing systems 17: proceedings of the 2004 conference*. The MIT Press, 2005, p. 209.

[26] John F. Nash, "The bargaining problem", *Econometrica*, vol. 28, pp. 155–162, 1950.

[27] A. Singh, B. Minsker, and H. Takagi, "Interactive genetic algorithms for inverse groundwater modeling: issues with human fatigue and prediction models", *In Proceedings of the American Society of Civil Engineers (ASCE) Environmental and Water Resources Institute (EWRI) World Water and Environmental Resources Congress 2005 and Related Symposia*, 2005.

[28] G. Dozier, B. Carnahan, C. Seals, L. Kuntz, and S. Fu, "An interactive distributed evolutionary algorithm (idea) for design", *Proceedings of the IEEE International Conference on Systems, Man and Cybernatics*, vol. 1, 2005.

[29] J. Chaves-Gonzlez, N. Otero-Mateo, M. Vega-Rodrguez, J. Snchez-Prez, and J. Gmez-Pulido, "Game implementation: An interesting strategy to teach genetic algorithms", *Computers and Education*, pp. 205–223, 2007.