Massively Parallel Model of Evolutionary Game Dynamics

Amanda Peters Randles*, David Rand†, Christopher Lee*,
Greg Morrisett* , Jayanta Sircar*, Martin Nowak† and Hanspeter Pfister*
*School of Engineering and Applied Sciences
Harvard University, Cambridge, Massachusetts 02138
Contact Email: apeters@fas.harvard.edu
†Program for Evolutionary Dynamics
Harvard University, Cambridge, Massachusetts 02138

***Abstract*—To study the emergence of cooperative behavior, we have developed a scalable parallel framework for evolutionary game dynamics. Our goal is to enable domain scientists to model the interaction between self-interested agents and to draw conclusions about successful game strategies in large groups of these agents. An important aspect is the amount of history, or memory steps, that each agent can keep. With six memory steps are taken into account, the strategy space spans 2^{4096} potential strategies, requiring large populations of agents. We present the first scalable game dynamics algorithm that models up to six memory steps for populations consisting of up to 10^{18} agents. Our algorithm exhibits excellent weak scaling and 82% strong scaling efficiency on up to 262,144 processors of BlueGene/P (Jugene). Our framework marks an important step in the study of game dynamics with potential applications in fields ranging from biology to economics and sociology.**

***Index Terms*—volutionary dynamics, parallel computing, game theory, multi-agent model, heterogeneous computingvolutionary dynamics, parallel computing, game theory, multi-agent model, heterogeneous computinge**

I. Introduction

Computational approaches are transforming science and could also transform economics given the difficulty in applying analytical solutions to complex environments. Economists are looking to model interactions seen in business using methods that describe the interactions between individuals in competition with one another. One such example is the large-scale modeling of adaptive agents that act as traders in the electronic marketplace [1] [2]. This has the potential of creating trading models that are both more efficient and more profitable. Game theory is commonly employed and exploited to describe and predict the interactions between agents in a multi-agent system. It is a general method of studying decision making and characterizing the strategic stability of potential outcomes that has been used extensively to model behaviors such as market trades, petty theft, or nuclear war [3], [4], [5], [6].

Through game theory, it is possible to measure the emergence of altruistic behaviors that can be of benefit to both interacting parties. Despite short-term penalty to one or more of the individuals, altruism arises through the expectation of reciprocity under rational conditions. Cooperation is common throughout the natural world, and forms the basis of human society. Consistent with this observation, innumerable behavioral experiments have demonstrated the human capacity for cooperative behavior [7], [8], [9], [10]. These models are being used to gain a greater understanding of not only why cooperative behaviors can emerge in society, but what factors can increase its likelihood. Understanding why and how altruism emerges in a population is vital for predicting how agents will interact, whether these agents are companies, nation states, or biological pathogens.

The common thread among these fields is the desire to model the interaction between self-interested agents and draw conclusions about groups of these agents. Each agent behaves following a prescribed set of moves, or a *strategy*. In this work, we focus on situations in which agents will engage in repeated play against other agents. Each interaction between two individuals in a game is referred to as a *round*. The strategy can be as simple as always cooperating with the opponent, or it can be defined as a response to the opponent's previous play. For example, a common strategy referred to as *Tit-For-Tat (TFT)* prescribes play as always taking the same action that the opponent did in the previous round: If the opponent previously cooperated, the agent will cooperate, but if the opponent defected, the agent will, in turn, defect. Basing the current move on the actions taken in the previous n rounds is referred to as a *memory-n* strategy.

Taking into account memory-n strategies has been shown to be important. This is due to the fact that these strategies can yield both a higher likelihood of cooperation from their opponent and also be more robust to factors such as error in game play [11], [3]. However, the added complexity leads to exponential growth of the strategy space, making exact game-theoretic modeling and classical analysis impossible [12]. One commonly applied method is to use computer simulation and sampling [4]. However, previous empirical models for memory-n strategies have been hindered by computational demands of large state space, population size, and duration of the simulations. Currently, the most common models use memory-one strategies [11] [13].

In this work, we developed a large-scale parallel simulation method for investigating strategies with up to memory-six game play in a large population of agents. We divided the problem into two levels of parallelism: subset of agents divided by groups of strategies, and concurrent game play of agents within each strategy group. This structure enables optimized communication patterns and memory usage by handling individual data locally and population snapshots globally. Our implementation provides the framework to allow a new scale of research in evolutionary game dynamics. Our simulations can provide new insights into the emergence of cooperative behavior and play a key role in assessing the importance of factors such as history of previous game play. The main contributions of our work is a new massively parallel framework to enable – for the first time – large-scale modeling of up to memory-six behavioral strategies in populations consisting of up to 10^{18} agents. We implemented our scalable multi-agent framework on the IBM Blue Gene architecture and show results that achieve linear weak-scaling and 82% strong scaling parallel efficiency on up to 262,144

processors.

II. Related Work

Game theory simulations form the basis of research in a variety of fields [14], [15], [16], [17]. Axelrod and Hammond laid groundwork for the use of computational models when they studied social dynamics with a simple evolutionary model [18]. Larger-scale models for studies in social dynamics have since been developed by Macal et al. [19] and in a variety of fields ranging from urban planning [20], to facility evacuation models [21], to trade networks under varying market conditions [1] [22], to tissue formulation [23]. All of these methods use only up to memory-one strategies.

There have been a few attempts to handle the complexity that arises from larger memory strategies. One method has been to use search profiles for strategy exploration in empirical games. By establishing a search algorithm to intelligently focus on strategies that are more likely to be strong, the problem space can be limited [24] [4]. Other studies have limited the strategy space by focusing on smaller populations. For example, Golbeck examined the traits of memory-three strategies by focusing on 20 strategies at a time and analyzing common traits [3]. Brunauer et al. [12] studied strategies up to memory-six for a population of fifty agents and analyzed twenty games. They demonstrated that taking into account more memory steps would likely lead to more cooperative strategies. By leveraging large-scale parallel computing we build on this work to produce a method that enables both the simulation of a large number of agents and up to memory-six strategies.

III. Background

In this section we summarize the key components of evolutionary game dynamics relevant to the development of our framework.

A. Prisoner's Dilemma

A fundamental question in game theory is: When will two interacting agents cooperate? To provide an intuition, let us first focus on the most basic scenario: Two agents interacting once with no previous history (i.e., memory-zero). This is referred to as the Prisoner's Dilemma (PD) and is the leading model for the evolution of altruistic behavior in populations of selfish agents [6], [11], introduced by Flood and Dresher's work at the RAND Corporation in the 1950's and formulated by Tucker [25] [12]. In this paper, we focus on the two-person PD. The two agents can choose to either cooperate (C) or defect (D). The four potential outcomes and their corresponding payoff are shown in Table I:

	Oppenent	
Agent	**C**	**D**
C	RR	ST
D	TS	PP

TABLE I
THE PRISONER'S DILEMMA PAYOFF MATRIX.

The payoff labels stand for ***Reward R***, ***Sucker S***, ***Temptation T***, and ***Punishment P***. In this work we use the following payoff or *fitness* values: $f[R,S,T,P] = [3,0,4,1]$. If both agents cooperate (C), they each receive a reward (3). If both agents defect (D), both will get punished (P) with a lower fitness (1). If one of them defects (D), the other is the sucker (S) with the lowest payoff (0), while the defector is rewarded (T) with the highest fitness (4). Herein lies the problem: While it is collectively better for the players to each cooperate, defection has become an unbeatable strategy if $f(T) > f(R) > f(P) > f(S)$ [26].

The study of moves that result in cooperation where one individual incurs a cost in order to benefit its opponent has been a topic of particular focus in evolutionary game theory [6]. It might be expected that natural selection would slowly weed out cooperators in favor of defection since it becomes the best play in a Prisoner's Dilemma game. However, cooperation is shown to exist throughout nature and forms the basis of human society as have been shown in many behavioral experiments [7] [8] [10] [9]. Numerous explanations have been suggested to resolve this apparent contradiction [27]. A solution begins to become apparent as we look at the much more compelling problem of the repeated Prisoner's Dilemma or Iterated Prisoner's Dilemma (IPD).

B. Iterated Prisoner's Dilemma

It has been suggested that cooperative behaviors may be exhibited because in real-life situations opponents have an expectation of playing an opponent again in the future [5]. In such a repeated game, the two agents face each other in many different *rounds*. Each agent's fitness is assessed by summing the fitness achieved in each round of game-play. The goal of each agent is to accumulate the highest fitness possible. Each agent can use historical information to determine their best move and to maximize their fitness. The rules that determine an agent's next move are referred to as a *strategy*.

A simple strategy known as *Tit-For-Tat (TFT)* is based on the opponent's previous move. TFT introduces the idea of direct reciprocity [28]: When interactions are repeated, cooperation can be favored by natural selection. An agent following TFT begins by cooperating, and then subsequently copies the behavior of the opponent in the previous round. This can lead to the emergence of sustained cooperation from both agents. Axelrod [29] hosted multiple competitions in which researchers would submit computer games to play a game of repeated Prisoner's Dilemma in a round robin tournament. Each program would play five games against all other players and at the end the scores would be tallied. TFT kept emerging as the winner (e.g. obtained the highest overall fitness) [26].

C. Strategy Types

There are two types of strategies used in game theoretical models to describe an agent's action plan: Pure and mixed. *Pure strategies* are those in which the chosen move is taken 100% of the time in response to a specific state. *Mixed strategies* are those in which the chosen move is with a certain probability. For example, for a memory-one TFT strategy, the agent following a pure strategy would always cooperate in a round following mutual cooperation. A mixed strategy, however, would define the move by saying that 90% of the time the agent should cooperate following a round of mutual cooperation, but 10% of the time that agent should defect. By enabling probabilistic strategies such as this, we widen the strategy space even further.

D. Strategy Space Size

Here we are defining a *state* as the different game situations given by the binary decisions (C or D) of the players in the past n-rounds. The size of the state space is defined by the memory steps of the model. For example, in strategies like TFT only the opponent's previous move is taken into account. In this instance, each player would decide the next move based on this one bit of information with $2^2 = 4$ potential states shown in Table II:

With each memory increase, another full prior round is taken into account. The number of potential states is $2^{2n} = 4^n$ distinct states for memory-n strategies[12]. In this work, we introduce a method that allows the simulation of a large population of agents for up to memory-six strategies, or up to 2^{4096} states.

State	Agent	Opponent
1	C	C
2	C	D
3	D	C
4	D	D

TABLE II
POTENTIAL GAME STATES FOR A MEMORY-ONE TFT STRATEGY.

As mentioned previously, a strategy defines the move an agent will take in a given round of a game given the current state of the game. For example, if we were looking at a memory-one system, each agent would look at both his previous move as well as his opponent's previous move, determine which state the game is in, and pick the next move based on the agent's strategy. All potential strategies for a memory-one mode are shown in Table III.

Strategy	State1	State2	State3	State4
1	C	C	C	C
2	D	C	C	C
3	C	D	C	C
4	C	C	D	C
5	C	C	C	D
6	D	D	C	C
7	D	C	D	C
8	D	C	C	D
9	C	D	D	C
10	C	D	C	D
11	C	C	D	D
12	D	D	D	C
13	D	C	D	D
14	D	D	C	D
15	C	D	D	D
16	D	D	D	D

TABLE III
ALL POTENTIAL MEMORY-ONE STRATEGIES.

For each given state, there are two possible moves that can be defined by the strategy. This means that the number of potential pure strategies for $numStates$ number of states is $2^{numStates}$. As we increase the number of memory steps in the model, the number of potential pure strategies increases dramatically as shown in Table IV.

Memory Steps	Number of Strategies
1	16
2	65536
3	$1.84 * 10^{19}$
4	$1.16 * 10^{77}$
5	2^{2048}
6	2^{4096}

TABLE IV
NUMBER OF PURE STRATEGIES FOR DIFFERENT MEMORY STEPS

E. Errors

The applicability of these strategies in real world scenario becomes much more complex due to the presence of *errors* in moves. An error occurs with a certain probability, and leads a player to make the opposite move than the one defined by its strategy. This would be fatal for the TFT strategy, as any accidental play of defection would shift the pair into a continuously repeated play of defection from both sides.

The more complicated strategy of *Win-Stay Lose-Shift (WSLS)* has been shown to outperform TFT in the presence of errors [11]. WSLS is a memory-one strategy. It considers both the players own move in the previous round as well as the opponents move and is essentially defined as "cooperate on the first move, and on subsequent moves switch strategies if you were punished on the previous move" [5]. Exploration of strategies that look further into the past, however, have been limited by the size of the strategy space (i.e. number of possible strategies) and the corresponding number of generations required to adequately sample the strategy space that could be analyzed. As the strategy space increases, the discrete number of generations required to show emergent behavior increases drastically. By minimizing the time it takes to complete each generation through our multi-tiered parallel method, we are able to model strategies such as WSLS for up to 10^7 generations in a matter of minutes.

IV. ALGORITHM

Our approach considers three major entities: Agents, Strategy Sets (SSets), and a Nature Agent. This hierarchy is depicted in Figure IV. We separated the algorithm into two main components: The interactions of the Agents within each SSet define the two-player *game dynamics*, while the interactions between the Nature Agent and the SSets define the *population dynamics*.

A. Game Dynamics

Agents that play the same game strategy are grouped into SSets. The fitness scores for a SSet is defined as the sum of the fitness scores of all agents in the set. One game iteration, or a *generation*, involves several rounds (in our case 200) of all agents in an SSet engaging in a pure or mixed strategy of two-player Iterated Prisoner's Dilemma. The collection of all agents is called the *population*. Throughout each generation, the overall population size remains constant.

Opponents are selected among SSets such that each SSet plays all other strategies in the population. Each agent in a SSet is assigned a subset of SSets to play against. For example, assume there are s different SSets and a agents per SSet. In each generation, each agent is assigned $\frac{s}{a}$ opposing SSets to play against. Note that the computation completed in a single generation can be handled locally by each SSet.

B. Population Dynamics

The population is evolved through a pairwise learning and mutation phase [30]. The learning phase involves the *pairwise comparison (PC)* of two SSet strategies that were selected at random by the Nature Agent. One SSet is designated the 'teacher' and the other is the 'learner'. If upon comparing their associated fitness scores the teacher has a greater fitness than the learner, the learner (and all agents in the SSet) will adopt the teacher's strategy in the next generation with probability p [15]. If the two SSets have different strategies, we use the Fermi function from statistical physics to define p simlar to previous work [15] [16] [31] [32]:

$$p = \frac{1}{1 + e^{-\beta(\pi_T - \pi_L)}} \tag{1}$$

The payoffs are defined by the variables π_T and π_L for the teacher and learner, respectively, while β is the intensity of selection. A small β leads to almost random strategy selection, while large values of β the rate of selecting the strategy with the higher relative fitness increases. As β approaches infinity, the better strategy will always be adopted.

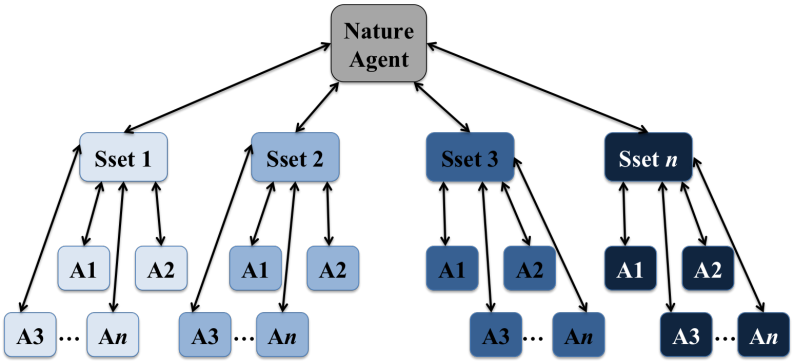


Fig. 1. Tree Diagram of Node Layout

To introduce potentially new strategies into the population, there is also a *mutation* phase. A new strategy is randomly generated and assigned to a random SSet at a rate of μ. After each generation, if any of the SSets strategies changed due to pairwise comparison or mutation, the SSets update their strategy sets, which only requires minimal communication.

C. Agents

The basic building block of the model is an ensemble of autonomous selfish agents. Each agent is given a strategy that it will use in an Iterated Prisoner's Dilemma game against a finite set of other agents. For each memory-n model there are a finite number of potential states that are defined by the global array *states*. Each state is given a unique ID. This is shown in Table V for the example of a WSLS memory-one strategy. The strategy is described as the set of moves for all potential current states, making the full column correspond to the strategy of WSLS.

State of Previous Round	Current State	Strategy
0	00	0
1	01	1
2	11	0
3	10	1

TABLE V
POTENTIAL WSLS STATES FOR MEMORY-ONE GAMES.

Each state defines the potential combinations of plays made by the two players in the previous round. If in the previous round both the agent and opponent cooperated (played a '0'), the current state would be defined as state 0 and the agent would proceed by cooperating (i.e., playing a '0'). The agents receive payoffs as defined in Table I. Each agent engages in the following game play:

```
FUNCTION IPD(myStrat, oppStrat)
global states;
play0=1
play1=1
for (i < nMemSteps)
   current_view[i][0] = 0
   current_view[i][1] = 0
end
while (r < maxRounds)
    current_state=find_state(curr_view,0)
    play0 = myStrat[current_state]
    current_state=find_state(curr_view,1)
    play1 = oppStrat[state]
    current_view[nMemSteps] = 1
    calc_fitness(play0,play1)
return fitness
```

Each game is a two-player Iterated Prisoner's Dilemma in which `play0` indicates the agent's play in that round and `play1` refers to the opponents play. The first play of each agent is arbitrarily set to 0. For instances in which multiple memory steps are being modeled, each agent needs to maintain a `current_view`, which is the agent's perspective of the state of the game. In a memory-one game, `current_view` would be set to the previous plays made by each agent. During each round, the agent determines the current `state` by searching the list of defined potential states for a match to the `current_view` . For each playing pair of agent, each agent's `current_view` will be the oppostie of its opponent. Given the current `state`, the agent uses its assigned strategy to determine the next play. After each round, the agent calculates its fitness by adding the fitness score received in that round to the total fitness achieved for that IPD game. At the end of the game, the total fitness score is returned to the SSet.

D. Strategy Sets (SSets)

A Strategy Set consists of a group of agents that are all playing the same strategy. All the agents in a SSet keep the same strategy throughout the simulation, even after all mutations and evolutionary learning processes. The agents are responsible for playing the iterated two-player game. The opponents are selected so that within each SSet, the agents have played all potential games against all the other strategies in the population. Each SSet follows these steps:

```
global states
Receive SSet _strat
Determine opponents to play
while t < max_generations
   for all opponents
      Call IPD(SSet_Strat, Opp_strat)
   collect relative_fitness
   receive PC_comparison
   if (mySSet == PC_comparison[0])
                return relative_fitness
   if (mySSet == PC_comparison[1])
        return relative_fitness
   receive SSet_strat
```

During the initialization stage, the potential `states` are globally defined based on the number memory steps being modeled. Each SSet then receives an array of `SSet_strat` defining the strategy IDs assigned to all SSets in the population. Each agent in the SSet in turn determines both its assigned strategy and which opposing strategies it will handle. Within each SSet, the

two-player IPD games between all strategies assigned to SSets must be played. Based on the number of agents assigned to each SSet, opponent strategies are assigned to each agent. The simulation proceeds until the maximum number of generations has been reached. Within each generation, agents call `IPD()` for each assigned opponent. The `relative_fitness` of a given strategy is computed by summing the relative fitness of all agents in the SSet. This is the value used for the pairwise comparison process by the Nature Agent. Each SSet is alerted to the randomly selected SSets for pairwise comparison and, if selected, return their `relative_fitness` to the Nature Agent. When the Nature Agent finishes the learning and the mutation process, the SSets receive an updated `SSet_strat`.

E. Nature Agent

The Nature Agent not only acts as a master, keeping track of the strategy assigned to each SSet and associated fitnesses and alerting all SSets to any alteration, but also controls the rate of mutations and determines which agents are impacted both by mutations and pairwise comparisons. The algorithm followed by the Nature Agent is presented below:

```
Split agents into SSets
Initialize strat_space[SSet]
while t < max_generations
   if (rand < PC_rate)
      SSets_for_pc[0] = rand
      SSets_for_pc[1] = rand
      send SSets_for_pc to all
      receive fitness from selected SSets
      if (fitness_teacher > fitness_learner)
        if (rand > p)
           strat1=strat(SSets_for_pc[0])
           strat2=strat(SSets_for_pc[1])
           SSet[strat1] = SSet[strat2];
        end
      end
      if (rand > mu)
           SSet[strat1] = gen_new_strat();
      end
      update all SSets
```

The Nature Agent splits all agents into SSets and initializes all SSets with a global view of the strategy space `strat_space`. At random generations until a maximum number of generations is reached, the Nature Agent determines when the evolutionary processes will occur. It initiates pairwise comparison learning at a set rate `PC_rate`. The fitness scores are retrieved for two randomly selected SSets. One is designated as the 'teacher' and the other the 'learner'. For a set probability `p`, the learner will adopt the strategy of the teacher if the teacher's fitness score is higher with a probability p as defined in Equation IV-B. For certain generations, an entirely new strategy will be assigned to a randomly selected SSet. This is controlled by the rate `mu` and enables the strategy space to be expanded to explore a larger domain. The Nature Agent then propagates any strategy changes to all SSets.

V. Implementation

We implemented our model in C and MPI. Originally developed for the IBM Blue Gene/L, this code has also been optimized for the IBM Blue Gene/P supercomputer. The Blue Gene/P has a quad SMP processor chip per node with 2 GB of memory per node and utilizes a system-on-a-chip design with 850 MHz PowerPC 440 cores. It has a three-dimensional torus interconnect with auxiliary networks for global communications, I/O, and management. The kernel is a lightweight, Unix-like OS per node for minimal system overhead [33].

The algorithm is mapped to the Blue Gene architecture such that one node is assigned as the Nature Agent and all other nodes handle individual agents. Depending on the population size, each node can handle multiple agents from multiple SSets. Within each generation, each node will iterate over all SSets and agents it is assigned and execute the game play for each. All game play is handled locally and relative fitnesses are maintained on each node. The Nature Agent communicates with all nodes and initiates both the evolutionary pairwise learning process and random mutations. The Nature Agent also handles all file I/O to record the global variables across generations. Memory is used mainly to store the local view of the strategy space at each SSet. It is minimized by having the Nature Agent act as the records keeper maintaining record of strategies assigned to SSets throughout the generations as well as global fitness data. On the individual agent level, only strategies currently held by other SSets at the given generation are kept in memory. Furthermore, as each node can access global state information, we are able to leverage the system size and processor rank data to allow each node to calculate its position within an SSet and its subsequent opponent strategies individually.

A. Local Interaction: Game Dynamics

Each agent within an SSet is responsible for engaging in the Iterated Prisoner's Dilemma game with a set of opponents representing strategies from other SSets. Within each SSet, the fitness of the assigned strategy must be measured against the fitness of the strategies of all other SSets, or in other words, all possible opponent strategies must be modeled. Each agent determines the subset of opponents to play based on its relative rank in its SSet. These agent-agent games are handled locally with no communication to other agents through the duration of the game. The relative fitness is calculated and stored locally to the agent. The game dynamics is easily parallelized across nodes and does not require any communication.

B. Global interaction: Population Dynamics

The population evolution consists of two main phases, the pairwise learning phase and the mutation phase. The pairwise learning phase involves the selection of two random SSets and comparing their associated fitness values. The node acting as the Nature Agent determines the generation that this occurs and randomly selects the two agents. We use global communication across the collective network to broadcast this pair selection to all agents. For actual data transfer we employ an `MPI_Bcast` call. Global communication over the collectives network is used anytime the Nature Agent needs to communicate with all SSets. This means that global broadcasting is used during the initial setup phase, alerting of the SSets selected for pairwise comparison, alerting of selected agents for random mutations, and global strategy updates.

Nodes containing agents then determine if they hold agents assigned to the selected SSet. If so, the agent's fitness is returned to the Nature Agent so that it can handle the pairwise comparison and subsequent learning process. Non-blocking point-to-point messages along the torus network are used to return the fitnesses to the Nature Agent, while further collective communication is used to broadcast information such as the resulting global strategy update. All nodes need to maintain an up to date view of the strategies assigned to all other SSets in order to determine all opponents strategies during game play.

When the Nature Agent determines the generation for a random mutation, a new strategy is randomly generated and assigned to a random SSet in the global view of SSet strategies maintained by the Nature Agent. This strategy along with the SSet identifier is then transmitted to all agents via an `MPI_Bcast` call.

C. Simulation Parameters

In our experiments, we used the following standard payoff matrix for the Iterated Prisoner's Dilemma at the agent level:

	C	D
C	**3,3**	**4,0**
D	**0,4**	**1,1**

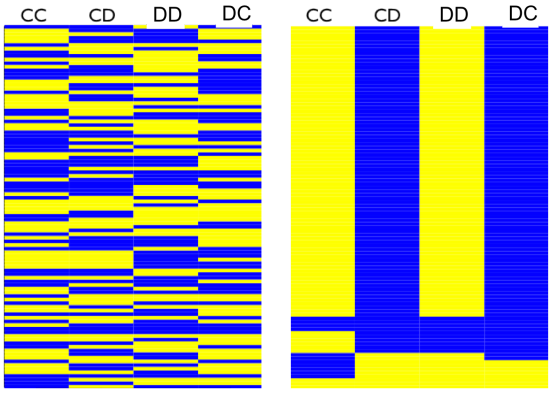
The maximum number of rounds for a generation of Iterated Prisoner's Dilemma was set to 200, similar to Smith and Price's mathematical model [34]. Strategy evolution across the population was controlled by a pairwise comparison rate of 10%. Random mutation of the strategy associated with a SSet was set to $\mu = 0.05$. The population size, number of SSets, and number of generations were varied to span several problem sizes. We successfully modeled memory-one through memory-six strategies. The number of agents per SSet was set to the number of total SSets so that each agent would handle one game per generation.

VI. Results

We conducted both a series of validation and small-scaling analysis as well as a large-scaling study. These studies are discussed in the following sections.

A. Validation Studies

To validate our framework and assess the code's performance on production runs, we performed various validation tests. For this work we used 2,048 processors of the IBM Blue Gene/L supercomputer[35]. In the first test, we modeled a population of 5,000 SSets and 20,000 agents for 10^7 generations looking only at memory-one strategies. We set up this simulation to mimic the work conducted in the original WSLS study by Nowak et al. [11]. We allowed the strategies to be probabilistic in nature and monitored the strategies assigned to each SSet over time. The results are shown in Figure 2.



(a) Population strategies at generation 0.

(b) Population strategies at final generation.

Fig. 2. View of strategies represented in the population. Each row represents a strategy held by a SSet, and each column represents a memory step. The combinations along the top show the potential states, and the colors indicate the move to make given each state. In the final population, the majority of the SSets have adopted the WSLS strategy of [0101] represented by the alternating blue and yellow colors.

It shows the overall state of the population, where each row represents a strategy by a SSet, and each column represents a memory step. Yellow indicates a cooperative move (C), and blue indicates the decision to defect (D). Figure 2(a) shows the initial status of the population of SSets. The strategies are randomly assigned to all SSets at the start of the simulation. Figure 2(b) shows the strategy distribution after 10^7 generations. The data has been clustered using Lloyd k-means clustering [36], allowing strategies that are more prevalent to be more easily identified. The results show that at the completion of the simulation, 85% of all SSets have adopted the strategy of [0101], which is WSLS. This is consistent with the results by Nowak et al. [11].

B. Small-scale Studies

The two limiting factors to previous computational models have been the memory step size and population sizes. To gain a better understanding of the role these factors play on parallel scalability and runtime, we undertook a series of small scaling studies using 2,048 processors of the IBM Blue Gene/L supercomputer.

1) Memory Step Size: **We first looked at the impact that increasing the number of memory-steps had on strong scaling. Table VI shows the overall runtime in seconds for the full simulation of 1,024 SSets as we increase from a memory-one to a memory-six strategy. For the purpose of these scaling tests, we limited the run to 1,000 generations and set the PC rate to 0.01. The matrix defining potential states increases drastically with the number of memory steps. Since it must be kept in local memory, and because the Blue Gene/L has only 512 MB of per-node memory, we had to limit our tests to memory-six.**

	Number of Processors				
Memory Steps	**128**	**256**	**512**	**1,024**	**2,048**
Memory-one	26.5	13.6	5.9	4.59	4.04
Memory-two	2,207	1,106	552	442	277
Memory-three	2,401	1,206	605	478	305
Memory-four	3,079	1,581	824	732	420
Memory-five	7,903	4,011	2,007	1,829	1,005
Memory-six	8,690	4,367	2,188	2,054	1,097

TABLE VI
TIME IN SECONDS FOR THE FULL SIMULATION OF 1,024 SSETS AS THE NUMBER OF MEMORY STEPS IS INCREASED.

The results of strong scalability tests are reported in Figure 3. We vary the number of memory steps from one to six and use pure strategies, meaning the strategy space ranges from 16 to $2^{4,096}$ potential strategies.

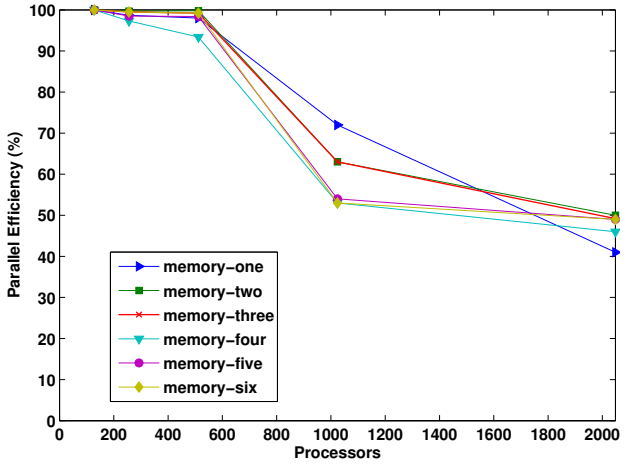


Fig. 3. Strong scaling graph for memory-one to memory-six strategies for 1,024 SSets. The addition of more memory steps has only a small impact on parallel efficiency.

As Figure 3 shows, the parallel efficiency does not change very much with increasing number of memory steps. We are defining

parallel efficiency as the percent of ideal speedup achieved for each processor count. For 1,024 SSets, each processor handles the agents of between $1/2$ to 8 full SSets. As the number of memory steps has little impact on the scaling efficiency, we focused further on its impact on overall runtime. These results are reported in Figure 4.

As the number of memory steps are increased, the overall runtime starts to increase dramatically. The fact that higher memory steps such as memory-six causes the number of potential strategies to increase exponentially does not mean that the runtime will increase similarly. Agents are able to determine their strategy and next move simply via a lookup based on the current state. The increase in runtime actually comes from identifying this state. During each round, each agent must determine the current state of the game by comparing it with its current view. As the number of memory steps increases, the size of the state description and subsequently the current view also increase, leading to larger single node run times.

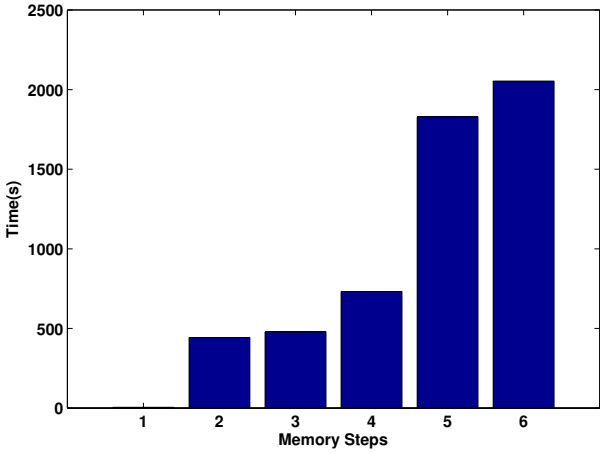


Fig. 4. Run time analysis for varying memory steps.

2) Population Size: **We performed scalability tests to assess the role population plays on our method's scalability. The overall runtimes for varying population sizes are presented in Table VII.**

	Number of Processors			
Nbr of SSets	**256**	**512**	**1024**	**2,048**
1,024	5.61	3.18	1.86	1.29
2,048	22.7	11.7	6.7	4.3
4,096	90.5	47.9	24.2	12.2
8,192	360	179.7	88.9	48.4
16,384	1,502	699	344	190
32,768	5,785	2,861	1,430	736

TABLE VII
TIME IN SECONDS FOR FULL RUNS AS THE NUMBER OF SSETS IS INCREASED.

The number of SSets greatly increases the overall runtime. This is mainly due to the fact that the number of games that need to be modeled grows with the square of the number of SSets because the agents belonging to each SSet must model the interaction with all strategies assigned to all other SSets.

The results of the strong scaling study are shown in Figure 5, demonstrating the impact that population size, dictated by the number of SSets, has on the parallel efficiency. When each processor handles less than 4,096 SSets, the computation per processor starts to be less than the communication overhead involved in the population dynamics component. This leads to a decrease in the overall parallel efficiency. As the population size grows, the impact of increasing the number of processors for the simulation increases.

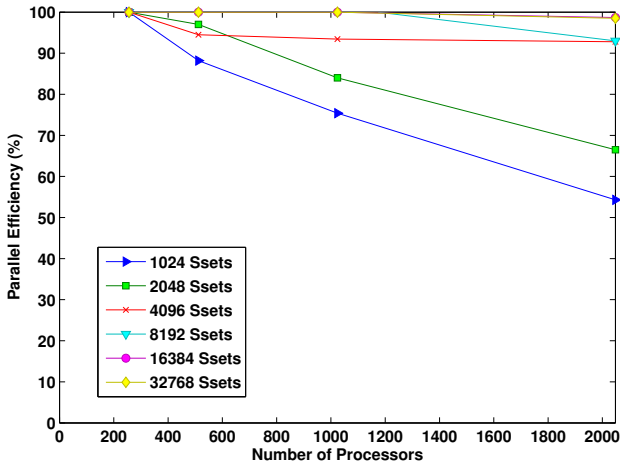


Fig. 5. Strong scaling as the number of SSets is increased. In this simulation, each SSet consists of one agent per potential opponent. That means the population size scales with the square of the number of SSets. This graph demonstrates the impact that population size has on parallel efficiency. When each processor handles less than 4,096 SSets, the computation per processor starts to be less than the overhead of the population dynamics. As the population size is increased, however, the impact of the number of processors on the overall parallel efficiency increases.

Table VIII shows the number of agents being handled per processor for given numbers of SSets. It is important to optimize the ratio of agents per processor to make the best use of larger processor counts. This means balancing between allocating so many agents per processor that the runtime become infeasible and having so few that communication overhead begins to overshadow computation time.

C. Large-scale Studies

To assess the performance of our algorithm on large systems, we conducted both weak and strong scaling experiments on an IBM Blue Gene/P supercomputer consisting of 294,912 processors. Based on the results of the small scaling studies, we used memory-six strategies and kept the work load at 4,096 SSets per processors. This resulted in a maximum population size of 1,073,741,824 SSets containing O(10^18) agents. The results of the weak scaling test for up to 262,144 processors (64 racks of Blue Gene/P) are presented in Figure ??.

We obtained near perfect results as the overall runtime for the simulations fluctuated by at most 1 second as we scale from 1,024 processors up to the full 262,144 processors. This demonstrates the ability of our method to dramatically increase the potential population size while keeping the simulation time reasonable.

The results of a strong scaling study are shown in Figure 7.

We achieved near-ideal results with 82% scaling efficiency exhibited at 262,144 processors. Due to system availability, this work was conducted with tests on 1,024, 2,048, 8,192, 16,384, and 262,144 processors. Through 16,384 processors, 99% linear scaling is maintained.

D. Discussion

In this paper, we focus on the scalability of the code to of Blue Gene/P (262,144 processors). We also successfully scaled the code to the full 72 racks (294,912 processors), however, we saw a 15%

	Number of Processors			
Nbr of SSets	**256**	**512**	**1,024**	**2,048**
1,024	4,096	2,048	16,384	2,048
2,048	16,384	8,192	262,144	32,768
4,096	65,536	32,768	4,194,304	524,288
8,192	262,144	131,072	67,108,864	8,388,608
16,384	1,048,576	524,288	1,073,741,824	134,217,728
32,768	4,194,304	2,097,152	17,179,869,184	2,147,483,648

TABLE VIII
NUMBER OF AGENTS PER PROCESSOR.

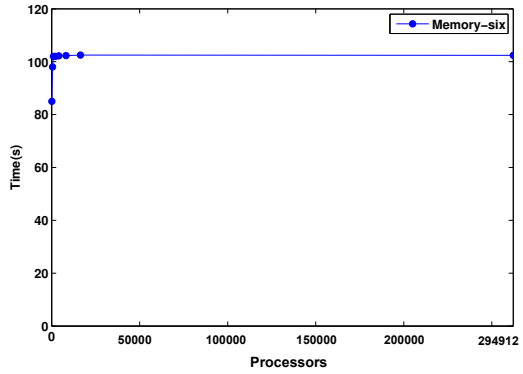


Fig. 6. Weak scaling analysis for 4,096 SSets per processor.

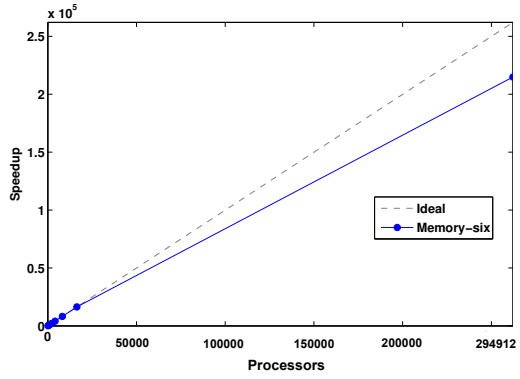


Fig. 7. Strong scaling for large systems.

degradation in efficiency. As with many applications, having a partition size that is not a power of two negatively impacts the mapping of our algorithm to the hardware topology [37].

The decrease in strong scaling at the full 64 racks is likely due to the low ratio of SSets to processors. However, this was necessary in order to allow the simulations on smaller processor counts finish in reasonable amounts of time. With extended runtime allowed for the small studies, larger SSet per node ratios could be used to further increase performance.

We also chose to focus on the scalability of the algorithm instead of the more traditional measure of floating point operations. As the core kernel is an iterative comparison game, the flop count has little meaning. For this application, the contribution gained from the application of high-performance-computing is the ability to model systems at an unprecedented scale.

E. Conclusions and Future Work

We have presented a highly-scalable framework for modeling evolutionary game dynamics. Our method enables domain scientists to study population dynamics at an unprecedented scale, spanning a larger population size and greater number memory steps. We show results for memory-one through memory-six strategies for populations of up to 10^{18} agents, achieving near perfect weak scaling and strong scaling of 82% efficiency on 262,144 processors. The excellent weak scaling of our approach enables us to grow the population size dramatically while keeping simulation time reasonable. Our framework will not only allow researchers to assess the role memory plays in game dynamics, but also to determine if there are more complex strategies that lead to the emergence of cooperation between agents. This has the potential to widely broaden the scope of game theory simulation and the fields in which it is used, particularly for large-scale economic models.

In future work we plan to investigate custom mappings to help the performance for non-powers-of-2 partition sizes. We also plan to implement our method on heterogeneous GPU-CPU clusters to exploit the fine-grained parallelism of agent simulations on massively-parallel processors.

VII. ACKNOWLEDGMENTS

This work was partially supported by NSF's GRFP and the Department of Energy's Computational Science Graduate Fellowship. We wish to thank A. Edelman (MIT), Bob Walkup (IBM), D. Parkes (Harvard University) and E. Randles (Boston University) for their helpful comments on this research and paper. We also would like to thank the staff at the CyberInfrastructure Lab at Harvard University for their support and providing us access to the Blue Gene/L installation. This research used resources at the Juelich Supercomputing Center. This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] D. Cliff and J. Bruten, "Animat market-trading interactions as collective social adaptive behavior," *Adaptive Behavior*, vol. 7, no. 3-4, p. 385, 1999.

[2] D. Cliff, "Explorations in evolutionary design of online auction market mechanisms," *Electronic Commerce Research and Applications*, vol. 2, no. 2, pp. 162–175, 2003.

[3] J. Golbeck, "Evolving strategies for the prisoners dilemma," *Advances in Intelligent Systems, Fuzzy Systems, and Evolutionary Computation*, vol. 2002, p. 299, 2002.

[4] P. Jordan, Y. Vorobeychik, and M. Wellman, "Searching for approximate equilibria in empirical games," pp. 1063–1070, 2008.

[5] P. Hingston and G. Kendall, "Learning versus evolution in iterated prisoner's dilemma," vol. 1, pp. 364–372, 2004.

[6] R. Axelrod and W. Hamilton, "The evolution of cooperation," *Science*, vol. 211, no. 4489, p. 1390, 1981.

[7] I. Bohnet and B. Frey, "The sound of silence in prisoner's dilemma and dictator games," *Journal of Economic Behavior & Organization*, vol. 38, no. 1, pp. 43–57, 1999.

[8] E. Fehr and S. Gachter, "Altruistic punishment in humans," *Nature*, vol. 415, no. 6868, pp. 137–140, 2002.

[9] R. Frank, T. Gilovich, and D. Regan, "Does studying economics inhibit cooperation?" *The Journal of Economic Perspectives*, vol. 7, no. 2, pp. 159–171, 1993.

[10] D. Rand, A. Dreber, T. Ellingsen, D. Fudenberg, and M. Nowak, "Positive interactions promote public cooperation," *Science*, vol. 325, no. 5945, p. 1272, 2009.

[11] M. Nowak and K. Sigmund, "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game," *Nature*, vol. 364, no. 6432, pp. 56–58, 1993.

[12] R. Brunauer, A. L
"ocker, H. Mayer, G. Mitterlechner, and H. Payer, "Evolution of iterated prisoner's dilemma strategies with different history lengths in static and cultural environments," pp. 720–727, 2007.

[13] M. Nowak and K. Sigmund, "Tit for tat in heterogeneous populations," *Nature*, vol. 355, no. 6357, pp. 250–253, 1992.

[14] J. Riley, "Evolutionary equilibrium strategies," *Journal of Theoretical Biology*, vol. 76, no. 2, pp. 109–123, 1979.

[15] A. Traulsen, J. Pacheco, and M. Nowak, "Pairwise comparison and selection temperature in evolutionary game dynamics," *Journal of theoretical biology*, vol. 246, no. 3, pp. 522–529, 2007.

[16] D. Rand, H. Ohtsuki, and M. Nowak, "Direct reciprocity with costly punishment: generous tit-for-tat prevails," *Journal of theoretical biology*, vol. 256, no. 1, pp. 45–57, 2009.

[17] G. Fogel, P. Andrews, and D. Fogel, "On the instability of evolutionary stable strategies in small populations," *Ecological Modelling*, vol. 109, no. 3, pp. 283–294, 1998.

[18] R. Hammond and R. Axelrod, "The evolution of ethnocentrism," *Journal of Conflict Resolution*, vol. 50, no. 6, p. 926, 2006.

[19] C. Macal and M. North, "Agent-based modeling and simulation: Desktop abms," in *Simulation Conference, 2007 Winter*. IEEE, 2007, pp. 95–106.

[20] X. Li, X. Zhang, A. Yeh, and X. Liu, "Parallel cellular automata for large-scale urban simulation using load-balancing techniques," *International Journal of Geographical Information Science*, vol. 24, no. 6, pp. 803–820, 2010.

[21] C. Chu, "A computer model for selecting facility evacuation design using cellular automata," *Computer-Aided Civil and Infrastructure Engineering*, vol. 24, no. 8, pp. 608–622, 2009.

[22] D. McFadzean, D. Stewart, and L. Tesfatsion, "A computational laboratory for evolutionary trade networks," *Evolutionary Computation, IEEE Transactions on*, vol. 5, no. 5, pp. 546–560, 2001.

[23] S. Adra, S. Coakley, M. Kiran, and P. McMinn, "An agent-based software platform for modelling systems biology," *University of Sheffield Epitheliome Project: Technical Report: University of Sheffield*, 2008.

[24] P. Jordan, L. Schvartzman, and M. Wellman, "Strategy exploration in empirical games," pp. 1131–1138, 2010.

[25] R. Axelrod, *The complexity of cooperation*. Princeton university press Princeton, NJ, 1997, vol. 159.

[26] M. Nowak, "Prisoners of the dilemma," *Nature*, vol. 427, no. 6974, pp. 491–491, 2004.

[27] ——, "Five rules for the evolution of cooperation," *science*, vol. 314, no. 5805, p. 1560, 2006.

[28] R. Trivers, "The evolution of reciprocal altruism," *Quarterly review of biology*, pp. 35–57, 1971.

[29] R. Axelrod, *The Evolution of Cooperation*. New York, New York: Basic Books, 1984.

[30] W. Davis, http://www.spatialised-prisoners-dilemma.net/, 2010.

[31] L. Blume, "The statistical mechanics of strategic interaction," *Games and economic behavior*, vol. 5, no. 3, pp. 387–424, 1993.

[32] C. Hauert and G. Szabó, "Game theory and physics," *American journal of physics*, vol. 73, p. 405, 2005.

[33] I. Team, "Overview of the blue gene/p project," *IBM J. Res. Dev*, vol. 52, no. 1/2, 2008.

[34] J. Maynard Smith and G. Price, "The logic of animal conflict," *Nature*, vol. 246, no. 5427, pp. 15–18, 1973.

[35] A. Gara, M. Blumrich, D. Chen, G. Chiu, P. Coteus, M. Giampapa, R. Haring, P. Heidelberger, D. Hoenicke, G. Kopcsay *et al.*, "Overview of the blue gene/l system architecture," *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 195–212, 2005.

[36] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 881–892, 2002.

[37] P. Vezolle, http://www2.fz-juelich.de/jsc/datapool/page/3365/SC-BlueGene-Architecture.pdf.