2013 IEEE 27th International Symposium on Parallel & Distributed Processing

Massively Parallel Model of Extended Memory Use In Evolutionary Game Dynamics

Amanda Peters Randles*, David G. Rand†, Christopher Lee*, Greg Morrisett*,
Jayanta Sircar*, Martin A. 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. This is a critical computational tool enabling large-scale agent simulation research. An important aspect is the amount of history, or memory steps, that each agent can keep. When six memory steps are taken into account, the strategy space spans 2^{4096} potential strategies, requiring large populations of agents. We introduce a multi-level decomposition method that allows us to exploit both multi-node and thread-level parallel scaling while minimizing communication overhead. We present the results of a production run modeling up to six memory steps for populations consisting of up to 10^{18} agents, making this study one of the largest yet undertaken. The high rate of mutation within the population results in a non-trivial parallel implementation. The strong and weak scaling studies provide insight into parallel scalability and programmability trade-offs for large-scale simulations, while exhibiting near perfect weak and strong scaling on 16,384 tasks on Blue Gene/Q. We further show 99% weak scaling up to 294,912 processors 82% strong scaling efficiency up to 262,144 processors of Blue Gene/P. Our framework marks an important step in the study of game dynamics with potential applications in fields ranging from biology to economics and sociology.**

***Keywords*-game theory; evolutionary dynamics; multicore optimization;**

I. Introduction

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 [1], [2], [3], [4].

Through game theory, it is possible to measure the emergence of altruistic behaviors that can be of benefit to both interacting parties. Cooperation is common throughout the natural world, and forms the basis of human society. Despite short-term penalty to one or more of the individuals, altruism arises through the expectation of reciprocity under rational conditions. Consistent with this observation, innumerable behavioral experiments have demonstrated the power of repeated interactions for cooperative promoting behavior (c.f. [5], [6], [7], [8]. 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 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 [9], [1]. However, the added complexity leads to exponential growth of the strategy space, making exact game-theoretic modeling and classical analysis impossible [10]. One commonly applied method is to use computer simulation and sampling [2]. 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 [9] [11]. In light of recent experimental evidence that humans in fact use strategies with longer memories [8], it is important to develop methods for simulating more complex strategies.

Modeling the impact of history on the evolution of cooperation through an evolving population of interacting agents presents a strong computational challenge. Not only does the problem space increase drastically with the number of strategies and population size, the strategies held by the

1530-2075/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPS.2013.102

agents themselves are constantly changing over the course of generations, resulting in a constantly shifting population definition. The strategy space is adequately sampled by allowing agents to adapt new strategies either through *learning* or random mutations. The increase in size of the strategy space correlates to a subsequent increase in the number of generations simulated. To enable proper interaction, an updated view of the full population at each time step must be maintained on a per-node-basis leading to a strong interdependence and subsequent communication across the population after any mutation, potentially leading to a large communication overhead.

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 show that memory-six is the highest-level strategy that can be modeled on current supercomputing platforms due to memory restrictions. It is important to both reduce the overall time to solution for these population models as well as enable the simulation of much larger pools of strategies. To this end, we define our metric of success to be both the strong and weak scaling efficiency of the code.

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 contribution of our work is the development of a highly efficient parallel framework enabling large-scale modeling of up to memory-six behavioral strategies in populations consisting of up to 10^{18} agents. Secondly, the abstraction of Strategy Sets enables a hybrid implementation to exploit two levels of parallelism: the subset of agents divided by groups of strategies, and concurrent game play of agents within each strategy group through a flat MPI/OpenMP model. This structure enables optimized communication patterns and memory usage by handling individual data locally and population changes globally. Our tuned code exhibits near perfect weak and strong scaling up to 16,384 MPI ranks on the IBM Blue Gene/Q supercomputer. We have further demonstrated the ability to perform at larger scale by demonstrating 99% efficient weak scaling up to 262,144 processors on the IBM Blue Gene/P supercomputer and 82% strong scaling parallel efficiency up to 262,144 processors.

II. Related Work

Game theory simulations form the basis of research in a variety of fields (c.f. [12], [13], [14], [15]). Axelrod and Hammond laid groundwork for the use of computational models when they studied social dynamics with a simple evolutionary model [16]. Larger-scale models for studies in social dynamics have since been developed by Macal et al. [17] and in a variety of fields ranging from urban planning [18], to facility evacuation models [19], to trade networks under varying market conditions [20] [21], to tissue formulation [22]. All of these methods use at most memory-one strategies.

There have been a few attempts to handle the complexity that arises from larger memory strategies. One method is to 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 [23] [2]. 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 [1]. Brunauer et al. [10] 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 [4], [9], introduced by Flood and Dresher's work at the RAND Corporation in the 1950's and formulated by Tucker [24] [10]. 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:

	Opponent	
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 becomes an unbeatable strategy if $f(T) > f(R) > f(P) > f(S)$ [25].

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 [4]. 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 has been shown in many behavioral experiments ([26]). Numerous explanations have been suggested to resolve this apparent contradiction [27]. A possible 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 [3]. In such a repeated game, the two agents face each other in many different *rounds*. Summing the fitness achieved in each round of game-play assesses each agents fitness. 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) [25].

C. Population Model

It should be stressed that in a pairwise interaction with any agent with any given strategy, an agent exhibiting the TFT strategy will not do better than its opponent. TFT can, however, elicit cooperation from agents who would otherwise defect [11]. To study the evolutionary role a strategy can have, one can either simulate whether or not a homogenous population of a given strategy will resist invasion by mutant strategies over time or start with a heterogeneous population and subject the agents to methods of selection and mutation. In this work, we will focus on the latter in an effort to establish a framework for teasing out strategies from a wide array of possibilities. As will be discussed in the following sections, the number of potential strategies for the IPD is so large that brute force modeling of all interactions of all potential strategies is not feasible. To this end, we adapt a heterogeneous population model with high rates of selection and mutation rates to sample the strategy space.

D. 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 moves are chosen 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.

E. 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 memory steps of the model define the size of the state space. 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:

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.

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[10]. 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 4096 states.

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 $2^{numStates}$ and $numStates = 2^{2memSteps}$. 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	2^4
2	2^{16}
3	2^{64}
4	2^{1024}
5	2^{2048}
6	2^{4096}

Table IV
NUMBER OF PURE STRATEGIES FOR DIFFERENT MEMORY STEPS

F. Errors

The applicability of these strategies in a 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 [9]. WSLS is a memory-one strategy. It considers both the player's own move in the previous round as well as the opponent's 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" [3]. 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. Our multi-tiered parallel method minimized the time necessary to simulate each generation and subsequently enabled us 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. In this paper, we introduce the idea of distinct SSets. We define SSets as groups of agents that are assigned the same strategy. As will be discussed in the following sections, this enables a tiered parallel scheme whose hierarchy is depicted in Figure 1.

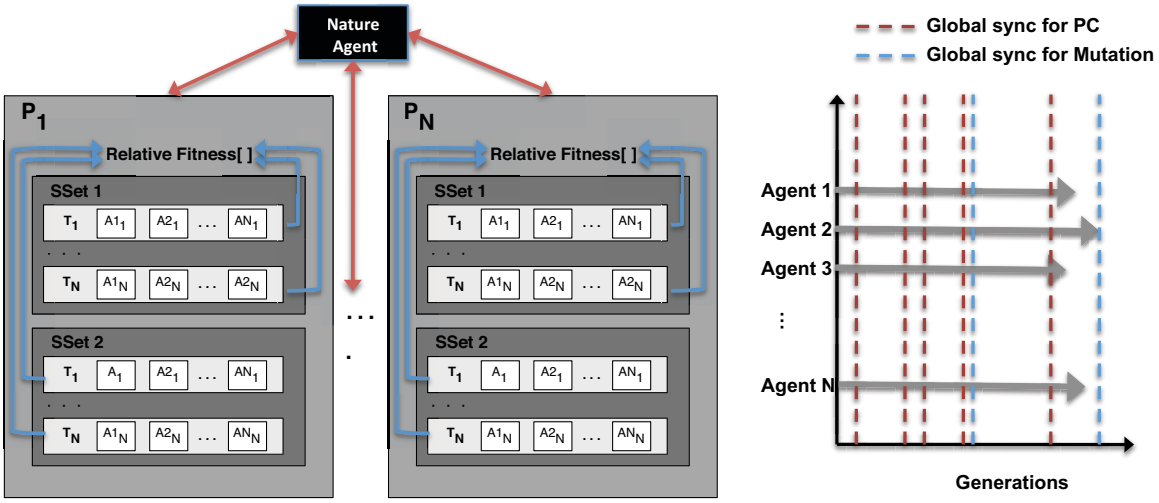
We separated the algorithm into two main components: The interactions of the Agents within each SSet define the two-player *game dynamics*, while the interactions between the Nature Agent and the SSets define the *population dynamics*.

A. Game Dynamics

Agents that play the same game strategy are grouped into SSets. The fitness score for an 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 in all SSets is called the *population*. Throughout each generation, the overall population size remains constant.

All agents in an SSet are assigned the same strategy. Naively, the setup would entail all agents enacting games against all other strategies held in the population. In this work, we take advantage of the fact that for deterministic strategies this would lead to redundant work within the strategy set. We can reduce the number of games played by each agent within the SSet by assigning 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 each SSet can handle the computation completed in a single generation locally. For mixed strategies, a similar method can be employed, however, longer generation counts may be required to ensure proper coverage.

Traditionally, the strategies being represented in a population would be assigned to an individual agent. This agent would simulate the interaction with all other strategies in the population in a serial manner and then handle the mutation and selections steps at the end of each round. The algorithm



(a) Diagram of Layout

(b) Communication Layout

Figure 1. Figure 1(a) shows the relationship between the agents and SSets per processor. In this example, there are two SSets per processor. Within each SSet, there is a set of agents split across separate threads. All of these threads update a local array containing the relative fitness of each SSet on the processor. Figure 1(b) depicts the communication breakdown over time. A grey arrow shows each agents trajectory over the course of the generations. At random generations, the agents are all interrupted by the Nature Agent's random global broadcast of SSets for pairwise comparison (PC) or mutation. This is followed by point-to-point communication with processors containing selected SSets and finally a broadcast to update all agent's of the updated population state, resulting in a non-trivial communication pattern.

would be parallelized by having each processor handle a separate agent, thus maxing out the number of processors used at the number of agents in the population. The introduction of the SSets expands the parallelism another level by assigning many agents to one strategy and enabling the more fine-grained game-level parallelism.

B. Population Dynamics

The population is evolved through both pairwise learning and mutation [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 [13]. If the two SSets have different strategies, we use the Fermi function from statistical physics to define p similar to previous work (c.f. [13] [14] [31] [32]), shown in Eqn. 1.

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

The payoffs are denoted 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.

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, each agent's current_view will be the opposite 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:

```
Set up global states
Receive SSet _strat
Determine opponents to play based on MPI rank
while t < max_generations
   for all opponents
      Call IPD(SSet_Strat, Opp_strat)
   collect relative_fitness for ranks in SSet
   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 fitness and alerting all SSets to any alteration, but also controls the rate of mutation 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 are 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, 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 1. 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

The algorithm is mapped to the Blue Gene architecture such that one processor is assigned as the Nature Agent and all other processors are assigned to SSets. 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 fitness of each SSet is 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 a 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. This reduces communication overhead by removing the need for the Nature Agent to broadcast specific strategy assignments to each individual agent.

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 individual game, making them ideal for thread-level parallelism. A hybrid-model, OpenMP/MPI, is used to allow the code to exploit the on-node hardware support for shared memory. The computation for each SSet is further split across all threads in a processor as shown in the following adaptation to the non-Nature Agent processor's pseudo code:

```
Set up global states
Receive SSet_strat
Determine opponents to play based on MPI rank

while t < max_generations
  for all SSets for this processor
    myfit=0
    #pragma omp parallel for shared(myStrat)
        private(Opp,myfit)
    for all opponents
        myfit += Call IPD(myStrat, Opp)

   #pragma omp atomic
   fit[SSet] += myfit

   receive PC_comparison
   if (mySSet == PC_comparison[0])
         return relative_fitness
   if (mySSet == PC_comparison[1])
  return relative_fitness
   receive SSet_strat
```

This allows the agent-agent games within the SSet to be played out in parallel by the individual threads on the processor. The relative fitness is calculated and stored locally to the agent for every SSet handled by that processor. Further optimization is obtained by handling the mutation and learning global calls only once per processor instead of once per SSet.

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 and all subsequent alerts of the SSets selected for pairwise comparison, alerts of selected agents for random mutations, and global strategy updates. The amount of communication varies based on the mutation and pairwise comparison rates. In this paper, the rates result in population updates every few generations in which all processors must synchronize.

Once the Nature Node has broadcast the selected SSet, processors 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 fitness levels to the Nature Agent, while further collective communication is used to broadcast information such as the resulting global strategy update. The amount of global synchronization for the mutations and comparisons leads to a non-embarrassingly parallel problem. There is strong interaction within the SSets via non-blocking Send and Receives, but the larger barrier to overcome is the regular need for global communication whenever there is a mutation. Global communication required is on the order of microseconds and so, if not implemented correctly, would not permit scaling. An example is demonstrated in Figure 1(b).

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

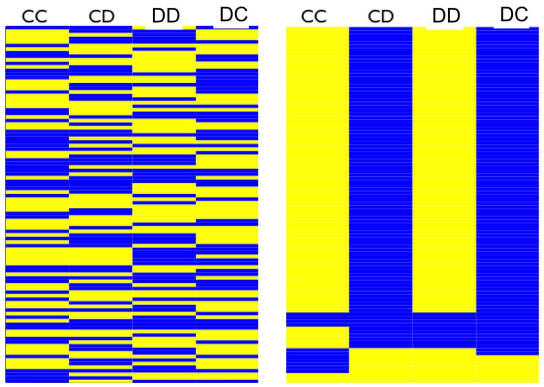
The maximum number of rounds for a generation of Iterated Prisoner's Dilemma was set to 200, similar to Smith and Price's mathematical model [33]. Strategy evolution across the population was controlled by a pairwise comparison rate of 10%. Random mutation of the strategy associated with a SSet was set 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. This was the largest memory step model that could fit into memory on both the IBM Blue Gene/Q platform.

VI. Results

We conducted both a series of validation and small-scaling analysis as well as a large-scaling study using predominantly the IBM Blue Gene/Q platform. Each node consists of 16 cores with 4 potential threads per core and a 204.8 GFlop/s peak performance per node. Memory per node is expanded to 16Gb [34]. For point-to-point communication, 5D torus is introduced to handle point-to-point communication at 32 GB/s [35]. The compiler used in these studies is the IBM XL/C. These studies are discussed in the following sections.

A. Validation Studies

To validate our framework, 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. [9]. We used pure strategies 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.

Figure 2. View of strategies represented in the population.

Figure 2 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. [9].

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 16-128 nodes, or up to 2,048 processors, of the IBM Blue Gene/Q supercomputer.

1) Optimization: In order to assess the true impact of these two factors on the performance of the code, we first ensured that we were using a fully optimized version of the code. For this study, 4096 SSets were modeled memory step one, 100 maximum generations, 200 rounds per game, and all on 256 processors. Figure 3 shows the impact

each optimization level had on the runtime as measured in wall clock seconds as well as the average time spent in communication.

The first level of optimization shows the impact of switching to non-blocking communication when the Nature Agent receives the relative fitness scores from processors containing SSets involved in the pairwise learning process. This change only reduces the average communication time by a small factor as the bulk of the communication is spent in global broadcasts.

Finally, we were able to fine-tune the performance by adding intrinsics instructions such as the fused multiply-add. The bulk of the time is spent in the game-play between the individual agents, and while much of this code depends on if-statements, the fitness calculation was hand-coded to use the built-in fpadd instruction.

Throughout all of the optimizations, the average communication time is minimized. This is due to the fact that the reduction in the game play time is helping reduce the communication imbalance imposed by having only a few nodes involved in each pairwise comparison decision, but all nodes involved in the global updates.

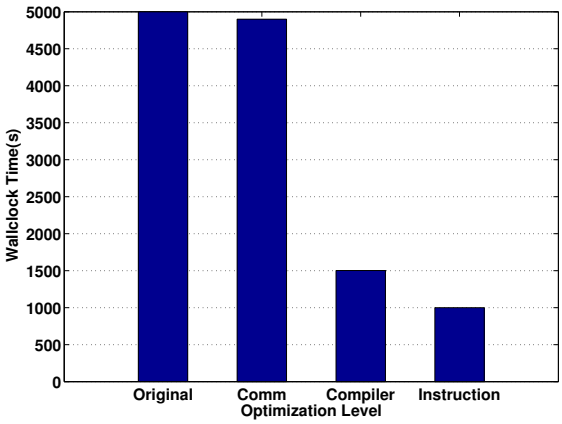


Figure 3. Optimization levels and their subsequent impact on the overall runtime.

2) Population Size: We next studied the role population plays on our method's scalability as a function of population size. 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 as 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 4, demonstrating the impact that population size, dictated by the number of SSets, has on the parallel efficiency. When each processor handles fewer 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.

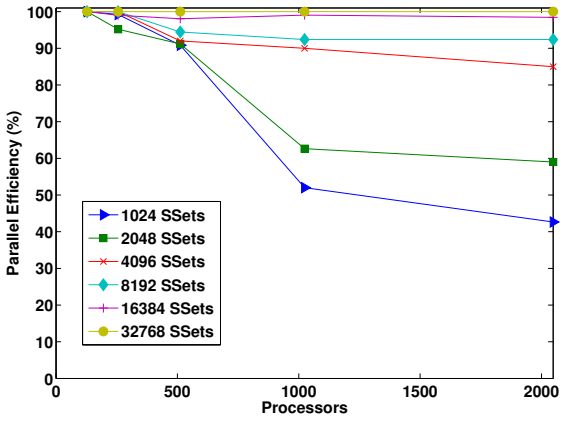


Figure 4. Strong scaling as the number of SSets is increased.

3) Ratio of SSet to Processor Count: We also investigated the number of SSets per processor that is ideal. As shown previously, for Blue Gene/P work we optimized for SSets per processor maxing out the processor count by running the partitions in virtual node mode. In the strong scaling results from Figure 4, the parallel efficiency notably drops off. This occurs when the SSet per processor drops below 1 as shown in Table VI. This becomes more important as we analyze the large-scaling studies in the following section.

R	0.5	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0
P.E.	50	55	99.7	99.7	99.9	99.9	99.9	100	100

Table VI
SSET PER PROCESSOR. AS THE NUMBER OF SSETS HANDLED BY EACH PROCESSOR LOWERS, THE PARALLEL EFFICIENCY DECREASES. R IS DEFINED AS THE RATIO OF SSETS TO PROCESSORS.

4) Memory Step Size: We then looked at the impact that increasing the number of memory-steps had on strong scaling and overall runtime. We found that increase in memory step size had a significant impact on the overall runtime of the simulation but had little effect ($< 2\%$) on the parallel efficiency as long as the processors were fully saturated with SSets. We are defining parallel efficiency as the percent of ideal speedup achieved for each processor count. Figure 5 shows the breakdown of the runtime in seconds for time spent in computation and communication at varying memory step sizes. These were for simulations of 2048 SSets for 20 generations with a PC rate of .1 and were completed on 2048 processors of Blue Gene/P. 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

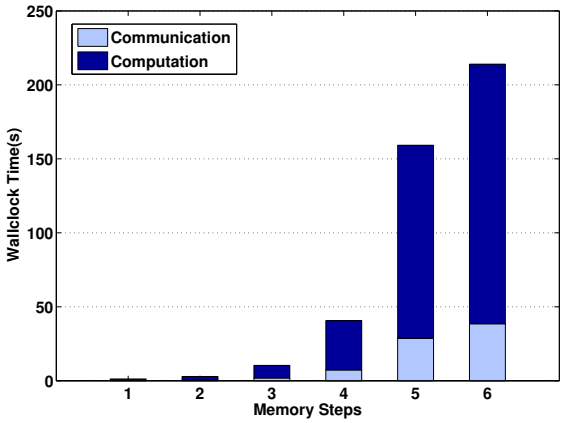
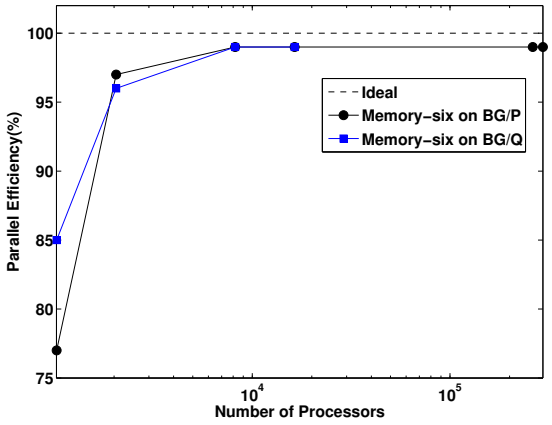
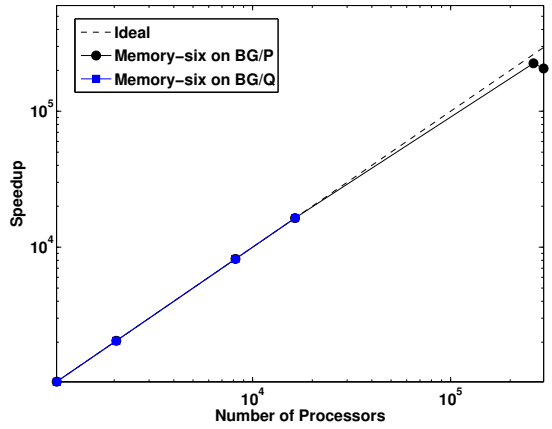


Figure 5. Run time analysis for varying memory steps.



(a) Weak Scaling up to 294,912 processors.



(b) Strong Scaling up to 262,144 processors.

Figure 6. Scalability results for memory-six models up to 294,912 processors. We demonstrate near perfect weak scaling results at scale for both BG/P and BG/Q. The code exhibits perfect strong scaling through 16,384 processors on BG/P and BG/Q. A degradation in performance is seen at 262,144 and above due to the fact that SSets are being split at suboptimal levels for the extended BG/P run.

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.

C. Large-scale Studies

To assess the performance of our algorithm on large systems, we supplemented our studies on the IBM Blue Gene/Q platform with work completed on the 294,912 Blue Gene/P system. This was due to the limited availability of larger Blue Gene/Q systems and shows the performance of the code at large-scales. Based on the results of the small scaling studies, we used memory-six strategies for all of these studies to enable at-scale production runs. For Blue Gene/P, the hybrid OpenMP/MPI implementation produced the same performance as running in virtual node mode to produce the maximum distributed MPI ranks, so all runs were completed with the flat-MPI version. For Blue Gene/Q, we saw a decrease in runtime when running with 32 tasks per node and 2 threads per task. The impact of the threads was minimal (reducing the time 2%), but enables the hierarchical parallelization that becomes necessary for small SSet/processor ratios. For all results reported below on Blue Gene/Q we used the hybrid method with this setup

For the weak scaling tests, the work load was held to 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 294,912 processors (72 racks of Blue Gene/P) are presented in Figure 6(a).

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 294,912 processors. This demonstrates the ability of our method to dramatically increase the potential population size while keeping the simulation time reasonable. On Blue Gene/Q, we show equivalent results up to 16,384 processors achieved by running 512 nodes with 32 tasks per node. We found this to be the optimal setup for maximal performance of our code on BG/Q.

The results of a strong scaling study are shown in Figure 6(b). 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. The degradation in performance at 262,144 processors comes

from the same issue we saw highlighted in Figure 4. Populations of the scale used for the 294,912 processor run of the weak scaling test would exceed the memory limits of the smaller partitions needed for strong scaling studies. The strong scaling tests were conducted with 32,768 strategies as that was the limit we could fit in memory for the small scale run on 1024 processors of BG/P. This means that for the 262,144 processor run, SSets are being split up resulting in one-half an SSet per processor, leading to a dip in parallel efficiency. For scaling runs on BG/Q, we used the same number of strategies and achieved perfect speedup to 16,384 processors.

D. Discussion

In this work, we introduce a method to explore evolutionary game dynamics on an unprecedented scale and explore the impact of application factors like memory steps and population sizes as well as optimizations such as the use of SSets and the hybrid programming model on massively parallel systems. The capability of the evolutionary game dynamics code is that it allows researchers to study the impact strategies have on an overall population. Our work explores novel optimization methods, including a multilevel decomposition of the SSets and individual agents. Results show that our strategy is essential for obtaining high performance at scale. As domain researchers need to both extend the number of generations studied as well as the size of the population, we focused on both the weak and strong scalability of the code. It is worth noting that while we were able to take advantage of vectorization in the fitness calculation, the core kernel is an iterative comparison game, causing the flop count to have 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

We have presented a highly scalable framework for modeling evolutionary game dynamics and the introduction of the hierarchical SSet abstraction. Our method enables domain scientists to study population dynamics at an unprecedented scale, spanning a larger population size and greater number memory steps. To evaluate our methodology at scale, we show results for memory-one through memory-six strategies for populations of up to 10^{18} agents, achieving near perfect weak scaling through 294,912 processors 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 hybrid implementation 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.

ACKNOWLEDGMENT

This work was supported by NSF's GRFP and the Department of Energy's Computational Science Graduate Fellowship, grant number DOE CSGF DE-FG02-97ER25308. We wish to thank A. Edelman (MIT), Bob Walkup (IBM), D. Parkes (Harvard University), E. Randles (Boston University), J.R. Hammond (ANL), and M. Schulz (LLNL) for their helpful comments on this research and paper. This research used resources at the Juelich Supercomputing Center, Lawrence Livermore National Lab, and 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] J. Golbeck, "Evolving strategies for the prisoner's dilemma," *Advances in Intelligent Systems, Fuzzy Systems, and Evolutionary Computation*, vol. 2002, p. 299, 2002.

[2] P. Jordan, Y. Vorobeychik, and M. Wellman, "Searching for approximate equilibria in empirical games," in *Proc. 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 2. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1063–1070.

[3] P. Hingston and G. Kendall, "Learning versus evolution in iterated prisoner's dilemma," in *Congress on Evolutionary Computation (CEC'04)*, vol. 1. IEEE, 2004, pp. 364–372.

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

[5] P. Bó, "Cooperation under the shadow of the future: experimental evidence from infinitely repeated games," *The American Economic Review*, vol. 95, no. 5, pp. 1591–1604, 2005.

[6] A. Dreber, D. Rand, D. Fudenberg, and M. Nowak, "Winners dont punish," *Nature*, vol. 452, no. 7185, pp. 348–351, 2008.

[7] P. Bo and G. Fréchette, "The evolution of cooperation in infinitely repeated games: Experimental evidence," *The American Economic Review*, vol. 101, no. 1, pp. 411–429, 2011.

[8] D. Fudenberg, D. Rand, and A. Dreber, "Slow to anger and fast to forget: leniency and forgiveness in an uncertain world," *American Economic Review*, vol. 102, pp. 720–479, 2012.

[9] 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.

[10] R. Brunauer *et al.*, "Evolution of iterated prisoner's dilemma strategies with different history lengths in static and cultural environments," in *Proc. ACM Symposium on Applied Computing*. ACM, 2007, pp. 720–727.

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

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

[13] 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.

[14] 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.

[15] 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.

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

[17] C. M. Macal and M. J. North, "Agent-based modeling and simulation: desktop ABMS," in *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*, ser. WSC '07. Piscataway, NJ, USA: IEEE Press, 2007, pp. 95–106. [Online]. Available: http://dl.acm.org/citation.cfm?id=1351542.1351564

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

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

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

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

[26] C. Camerer, *Behavioral game theory: Experiments in strategic interaction*. Princeton University Press, 2011.

[27] M. A. Nowak, "Five rules for the evolution of cooperation," *Science*, vol. 314, no. 5805, pp. 1560–1563, 2006. [Online]. Available: http://www.sciencemag.org/content/314/5805/1560.abstract

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

[34] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L.-T. Chiu, P. A. Boyle, N. H. Chist, and C. Kim, "The IBM Blue Gene/Q compute chip," *IEEE Micro*, vol. 32, pp. 48–60, 2012.

[35] D. Chen, N. Eisley, P. Heidelberger, R. Senger, Y. Sugaware, S. Kumar, V. Salapura, D. Satterfield, B. Burrow-Steinmacher, and J. Parker, "The IBM Blue Gene/Q interconnection network and message unit," in *Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. Washington, DC, USA: IEEE Computer Society, 2011.

[36] T. Kanungo *et al.*, "An efficient-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 881–892, 2002.