Probabilistic Model Checking Multi-agent Behaviors in Dispersion Games Using Counter Abstraction

Jianye Hao[1], Songzheng Song[2], Yang Liu[2], Jun Sun[3],
Lin Gui[2], Jin Song Dong[2], and Ho-fung Leung[1]

[1] The Chinese University of Hong Kong
{jyhao,lhf}@cse.cuhk.edu.hk
[2] National University of Singapore
{songsongzheng,tslliuya,lin.gui,dongjs.comp}@nus.edu.sg
[3] Singapore University of Technology and Design
sunjun@sutd.edu.sg

Abstract. Accurate analysis of the stochastic dynamics of multi-agent system is important but challenging. Probabilistic model checking, a formal technique for analysing a system which exhibits stochastic behaviors, can be a natural solution to analyse multi-agent systems. In this paper, we investigate this problem in the context of dispersion games focusing on two strategies: *basic simple strategy* (BSS) and *extended simple strategies* (ESS). We model the system using discrete-time Markov chain (DTMC) and reduce the state space of the models by applying counter abstraction technique. Two important properties of the system are considered: convergence and convergence rate. We show that these kinds of properties can be automatically analysed and verified using probabilistic model checking techniques. Better understanding of the dynamics of the strategies can be obtained compared with empirical evaluations in previous work. Through the analysis, we are able to demonstrate that probabilistic model checking technique is applicable, and indeed useful for automatic analysis and verification of multi-agent dynamics.

1 Introduction

Multi-agent learning is an important research area which has been applied in a wide range of practical domains [15, 28]. Because of the coexistence of multiple learners, a multi-agent system usually exhibits stochastic and unpredictable behaviors, which can be quite complex and difficult to analyse. To have a better understanding of the system's dynamics and further optimize the system's performance, an accurate analysis of the system's behavior beforehand becomes particularly important.

Most of the existing work on analysing such systems is based on extensive simulations [22, 6], which is the most convenient approach to take. The disadvantage of this approach is that the simulation results are usually inaccurate and also some important properties of the system (e.g., convergence) cannot

be directly proved [22]. Another line of research is to analyse the system's behavior theoretically through the construction of a mathematical model of the system [25, 9, 24]. This approach has its merits in that it can give a better understanding of the system's dynamics than simulation-based approach and also the properties of the system can be proved directly. The downside is that the proof construction is in general quite tedious and usually requires a good deal of ingenuity. Moreover, in some cases, the system may be too complex such that it is impossible to construct an accurate mathematical model.

To tackle the problems in these existing approaches, we propose using probabilistic model checking techniques [2] to analyse the behavior of a multi-learner system. Probabilistic model checking is a formal verification technique for analysing a system exhibiting stochastic behaviors, which makes it naturally suitable for the analysis of multi-learner systems. In probabilistic model checking, a probabilistic model (e.g., Markov decision process (MDP)) of the system's behavior is constructed first, and then a quantitative analysis of the model is performed, by applying a combination of exhaustive search and numerical solution methods. This approach is different from both simulation and mathematical analysis techniques. It is not only automatic as simulation technique does, but also provides exact rather than approximated analysis results, since it takes all possible behaviors that the system may exhibit into consideration.

To make the discussion concrete, in this paper, we focus on an important type of scenario modeled as dispersion games [22]. Dispersion games are the generalization of anti-coordination games to an arbitrary number of players and actions. This class of games has received wide attentions and they have been applied to model a variety of practical applications, e.g., load balancing problems [28], and niche selection in economics such as Santa Fe bar problem [4] and minority games [7]. We focus on two novel strategies designed for dispersion games: *basic simple strategy* (BSS) and *extended simple strategy* (ESS). Previous work [22, 1] has investigated the performance of both strategies through extensive simulations and shown the convergence to an Maximal Dispersion Outcome (MDO). However, only preliminary analytical results have been provided for the analysis of both strategies, and it is particularly difficult to give very accurate analytical results.

In this work, we investigate how probabilistic model checking can be used to analyse the behaviors of the agents under the two strategies (BSS and ESS) in the context of dispersion games in an accurate and automatic way. The dynamics of the agents under both strategies in dispersion games are modeled as discrete-time Markov chains (DTMCs). Since the agents always adopt the same strategy and thus exhibit similar behaviors, we propose to adopt process counter abstraction technique to reduce the state space of the model. Process counter abstraction is a special kind of symmetry reduction where the properties to be proved are irrelevant with the process identifiers. We prove that the probabilistic verification based on the abstract DTMC model is still guaranteed to be sound and complete. We focus on checking two important properties of the system: convergence and convergence rate. We are able to automatically prove that

the outcome is guaranteed to converge to an MDO when the agents adopt BSS while the convergence property is lost in ESS. For ESS, with probabilistic model checking we can also obtain the exact probability that the outcome deviates from an MDO by checking the corresponding property. For the property of convergence rate, the exact average number of rounds for the outcome to converge to an MDO is automatically obtained. To show the effectiveness of the process counter abstraction technique we propose, we also compare the state space and verification time cost with the case without using abstraction. Overall, through the analysis we are able to show that probabilistic model checking technique is applicable, and indeed useful in analysing the properties of a multi-agent system and providing additional insights into the system.

The remainder of the paper is organized as follows. Section 2 gives an overview of related work. In Section 3, we first review the dispersion games and the strategies, and then present how to reduce the state space using counter abstraction technique and implement the model. In Section 4, we perform an extensive analysis of two important properties of the system based on the models we previously construct. Lastly conclusion and future work are given in Section 5.

2 Related Work

Ballarini et al. [3] apply probabilistic model checking to automatically analyse the uncertainty existing in a two-agent negotiation game. In the negotiation game, there exist one seller and one buyer bargaining over a single item, and both players exhibit probabilistic behaviors based on the opponent's previous behavior. They model the dynamics of the two-player system as a discrete-time Markov chain (DTMC). They mainly illustrate how to use the probabilistic model checker PRISM [12] to automatically analyse the probability that the players reach an agreement within each round of the game. This property is specified in probabilistic computation tree logic (PCTL) [10]. Their work is similar to ours in that both work apply the probabilistic model checking technique to automatically analyse the dynamics of a multi-agent system in a game-like scenario. However, we study a complex scenario involving an arbitrary number of players and actions, and we propose using the abstraction technique to reduce the model's state space.

Tadjouddine et al. [23] investigate the problem of automatically verifying game-theoretical property of strategy-proofness for auction protocols. They consider the case of Vickrey auction protocol and check the property of strategy-proofness using the model checker SPIN [8]. To solve the state space explosion problem, they apply two types of abstraction approaches to solve it, i.e., program slicing technique and abstract interpretation. Program slicing is a technique to remove portion of codes in the model which is irrelevant with respect to the property checked. The basic idea behind abstract interpretation is to map the original strategy domain onto an abstract and less complex domain, and then perform model checking on the abstract model. By using these two abstraction methods, the authors show that strategy-proofness of Vickrey auction can

be automatically verified in SPIN for any number of players. However, in their work, there does not involve any probabilistic element within the protocol and the agents' behaviors, while the system we consider exhibits highly stochastic behaviors.

Bordini et al. [5] review the problem of verifying multi-agent system implemented in language AgentSpeak using model checking techniques. They aim at automatically verifying whether certain specifications are satisfied using existing model checkers. For this purpose, the original multi-agent system implemented in a BDI language AgentSpeak [17] need to be transformed into the formal language supported by current model checkers first. They introduce a variant of language AgentSpeak, AgentSpeak(F), which can be automatically transformed into Promela, the model specification language of SPIN [8]. They also describe another approach based on the translation of the system in AgentSpeak into a system in Java, which then can be checked by another model checker JPF [27]. Additionally, they adopt a simplified form of BDI logic to specify the properties to be checked, which can be transformed into LTL, supported by previous model checkers. With the combination of these two techniques, the properties of a multi-agent system implemented in AgentSpeak can be automatically checked with existing model checkers. There also exists other similar work [26] that transforms other agent-based languages such as Mable [26] into Promela and use SPIN to perform model checking. However, in our work, the model is implemented directly in the modeling language supported by the model checker PAT, which avoids the additional language transformation cost. Besides, probabilistic property checking, which is important in analysing multi-agent system dynamics, is not supported in their work.

3 Modeling Multi-agent Learning Dynamics in Dispersion Games

We consider the multi-agent learning problem in the context of *dispersion games* (DGs) [22]. Dispersion games are the generalization of anti-coordination games to arbitrary number of players and actions. This class of games is particularly important in that it can be applied to model a variety of practical applications including load balancing problems [28] and niche selection in economics [4, 7]. Two novel strategies designed for dispersion games are studied here: *basic simple strategy* (BSS) and *extended simple strategy* (ESS). Next we first give the detailed description of dispersion games and the strategies. Following that, we will present how to model the dynamics of these strategies and further reduce the state space using counter abstraction techniques.

3.1 Dispersion Games and Strategies Definition

In the following, we assume that the readers are familiar with game theory notations used in [14]. Dispersion games (DGs) [22] generalize the anti-coordination games by allowing arbitrary number of players and actions. In this class of games,

the agents prefer the outcomes in which their action choices are as dispersed as possible over all possible actions. Formally a N-player dispersion game is a tuple $\langle N, (A_i), (u_i) \rangle$ where

- $N = \{1, 2, \ldots, n\}$ is the set of agents.
- A_i is the set of actions available to agent i.
- u_i is the utility function of each agent i, where $u_i(O)$ corresponds to the payoff agent i receives when the outcome O is achieved.

We assume that all agents have the same set of actions, that is, $A_1 = A_2 = \ldots = A_n$, and also the game is both agent symmetric and action symmetric. That is, each agent's utility over a particular outcome is only determined by the number of agents choosing the same action as itself.

When the agents are interacting with one another in DG-like environments, the most desirable outcomes would be the case that all agents' action choices are as dispersed as possible, from both individual agent's and the overall system's perspectives. This kind of outcomes is called maximal dispersion outcomes (MDOs) [22]. Formally, an MDO can be defined as follows.

Definition 1. *Given a dispersion game, an outcome* $O = \{a_1, \ldots, a_i, \ldots, a_n\}$ *is* ***maximal dispersion outcome*** *iff for each agent* $i \in N$ *and each outcome* $O' = \{a_1, \ldots, a'_i, \ldots, a_n\}$ *such that* $a_i \neq a'_i$*, we have* $n^O_{a_i} \leq n^{O'}_{a'_i}$*. Here* $n^O_{a_i}$ *and* $n^{O'}_{a'_i}$ *are the number of agents choosing action* a_i *and* a'_i *under outcome* O *and* O' *respectively.*

The strategies we consider here are *basic simple strategy* (BSS) and *extended simple strategy* (ESS). *Basic simple strategy* is a novel strategy for agents to make decisions in repeated dispersion games proposed by Alpern [1]. This strategy is specifically designed for the case when the number of agents n is equal to the number of actions $k (k = |A_i|)$. According to BSS, initially each agent i chooses a random action. If no other agent chooses the same action, agent i will still choose the same action next round. If there exist other agents choosing the same action, agent i will randomly choose an action from the set $A' = \{a' \in A_i \mid n^O_{a'} \neq 1\}$ of actions in the next round. Note that this strategy only requires that the agents know which actions are chosen by only one agent in previous round.

Another strategy we consider is *extended simple strategy*, which extends BSS for the general case when $n \neq k$. In each round t, each agent i chooses the same action a_i as previous round if $n^{O_t}_{a_i} \leq \lfloor n/k \rfloor$, where $n^{O_t}_{a_i}$ is the number of agents choosing a_i in round t. Otherwise, agent i chooses action a_i with probability $\frac{n/k}{n^{O_t}_{a_i}}$ and with probability $1 - \frac{n/k}{n^{O_t}_{a_i}}$ randomly chooses an action over the action set $\{a' \in A_i \mid n^{O_t}_{a'} < \lceil n/k \rceil\}$.

Unlike BSS, ESS does not assign equal probability to those actions that are not chosen by only one agent. For example, consider the case when there are 4 agents and the action set $A_1 = A_2 = \ldots = A_4 = \{a_1, a_2, a_3, a_4\}$, and the outcome in the current round t is $O_t = \{a_1, a_1, a_2, a_2\}$. In ESS, the agents choosing action a_1 in current round t will choose action a_1 with probability 0.5

and either action a_3 or a_4 with probability 0.25 in round $t+1$. In contrast, the agents will randomly select one action to perform according to strategy BSS.

3.2 Modeling BSS and ESS in Dispersion Games Using Counter Abstraction Technique

For both BSS and ESS, in each round, the agents simultaneously choose their actions in a probabilistic manner based on the outcome of the previous round. The natural way of modeling the agents' dynamics in dispersion games is to represent each agent's learning dynamics as a process. The overall system exhibits highly stochastic behaviors and non-determinism because of the coexistence of multiple probabilistic learners. However, since each agent makes its decision independently each round, the concurrent behaviors among agents can be equivalently modeled as a series of sequential behaviors. In this way, the non-determinism in the system is eliminated and thus the system can be naturally modeled as a discrete-time Markov chain (DTMC).

Definition 2. *A discrete-time Markov chain is a tuple* $\mathcal{M} = (S, P, l_{init}, AP, L)$, *where*

- S *is a countable, non-empty set of global states,*
- $P : S \times S \to [0, 1]$ *is the transition probability function such that for all states* s*:* $\sum_{s' \in S} P(s, s') = 1$,
- l_{init} *is the initial distribution such that* $\sum_{s \in S} l_{int}(s) = 1$, *and*
- AP *is a set of atomic propositions and* $L : S \to 2^{AP}$ *a labeling function.*

Each agent (process) i has its own local states $s_i \in A_i$, i.e., its strategy choice, and each global state $s = (v, \langle s_1, \ldots, s_n \rangle) = (v, O_i)$, which is the combination of the valuations of the global variables v [4] and the local states of all agents (or the game outcome O_i). The transition relation P reflects the joint transition probability between different global states, which depends on the specific probabilistic strategy adopted by the agents in the system and can be easily calculated based on the strategy specifications in Section 3.1. Thus the DTMC which models the dynamics of the system can be automatically constructed and is uniquely determined. However, the major problem is that the state space of the model can be arbitrarily large and thus hinder the efficiency of analysing the model, due to the explosion of the combination of all agent processes' local states. For example, consider the case of n agents and k actions, without taking the global variables into consideration, the number of possible combinations of all agent processes' local states is k^n, which grows rapidly as the values of n and k increase.

To address this problem, we adopt counter abstraction technique [16, 20] to reduce the state space. If a system is composed of a large number of behaviorally similar processes, we can abstract its state space by grouping the processes based

[4] Here the global variables refer to all variables defined in the model apart from the local variables $(s_1, \ldots, s_n)$ storing the action choices for each agent.

on which local state they reside in. For example, suppose there are 3 behaviorally similar processes residing in a system. Instead of saying "process 1 is in state s, process 2 is in state t and process 3 is in state s", we simply say "two processes are in state s and one process is in state t". In this way, the state space can be reduced by exploiting a powerful state space symmetry. Since the agents always adopt the same strategy (either BSS or ESS) in dispersion games, this abstraction technique can be naturally applied here. Specifically we only need to consider how many agents choose each action in each possible outcome, since there is no need for us to distinguish the identities of agents. Accordingly, for those outcomes in which the number of agents choosing each action is the same but only the identities of the agents choosing the same action are different, they belong to two different global states but now they can be merged as the same one. For example, considering a dispersion game with 5 agents and 3 actions and two possible global states $s = (v, \langle a_1, a_1, a_2, a_2, a_3\rangle)$ and $s' = (v, \langle a_1, a_2, a_2, a_3, a_1\rangle)$. We only need to keep track of the number of agents choosing each action, i.e., we have $f(a_1) = 2, f(a_2) = 2, f(a_3) = 1$, where $f(a)$ records the number of agents choosing action a, and thus the two original global states are reduced to a single one (v, f).

Definition 3. *Given a global state $s = (v, \langle s_1, \ldots, s_n\rangle)$, its corresponding abstract global state s^A is a pair (v, f) where v is the valuation of the global variables and $f : A \to \mathbb{N}$ is a total function such that $f(a) = m$ if and only if m agents choose action $a \in A$. Here A is the set of actions of all agents, and $A = A_i, \forall 1 \leq i \leq n$.*

Accordingly, given a DTMC, by applying counter abstraction, its corresponding abstract DTMC can be defined as follows.

Definition 4. *An abstract discrete-time Markov chain is a tuple $\mathcal{M}^{\mathcal{A}} = (S^A, P^A, l^A_{init}, AP^A, L^A)$, where*

- *S^A is a countable, non-empty set of abstract global states,*
- *$P^A : S^A \times S^A \to [0, 1]$ is the transition probability function such that for all states $s : \sum_{s' \in S^A} P^A(s, s') = 1$,*
- *l^A_{init} is the initial distribution such that $\sum_{s \in S^A} l^A_{int}(s) = 1$, and*
- *AP^A is a set of atomic proposition and $L^A : S^A \to 2^{AP^A}$ a labeling function.*

For each abstract state $s^A_i \in \mathcal{M}^{\mathcal{A}}$, it may correspond to a set of original states in $\mathcal{M}$, which is denoted as $maps(s^A_i)$. The transition probability between two abstract states s^A_i, s^A_j is constructed as follows, $P^A(s^A_i, s^A_j) = \sum_{s_j \in maps(s^A_j)} P(s_i, s_j), s_i \in maps(s^A_i)$. Note that we only need to calculate $P^A(s^A_i, s^A_j)$ based on any single state s_i which can be mapped to the abstract state s^A_i, since each original state s_i mapping to s^A_i is symmetric to each other. The set of atomic propositions AP^A corresponds to those atomic propositions irrelevant with process identifiers, which are preserved after abstraction.

In this way, for the same case of n agents and k actions, if $k = n$, the number of possible combinations of all action processes' local states is reduced

from n^n to $\binom{2n-1}{n-1}$. If k is a small constant value with n varying, the maximum number of possible combinations of all action processes' local states is always smaller than n^k, which is polynomial in the number of agents n; while in the original case without abstraction, it is exponential in the number of agents n. This reduction is of significant value since it is usually the case that $k \ll n$ is true in practical DG-like scenarios such as load balancing problems. Besides, essentially no information is lost during the abstraction and the model is still accurate.

Theorem 1. *Given a DTMC $\mathcal{M}$ and a property ϕ[5], if property ϕ is irrelevant with process identifiers, then the probabilities that property ϕ is satisfied in the original model $\mathcal{M}$ and the abstract one are always the same, i.e., $\mathcal{P}r(\mathcal{M} \models \phi) = \mathcal{P}r(\mathcal{M}^{\mathcal{A}} \models \phi)$.*

Proof. For simplicity, we assume there is only one initial state in the system. In the following, we prove this theorem by proving that all paths in M, represented by L, can be separated into different groups L_1, L_2,... satisfying that 1) $L_i \cap L_j = \phi$ and 2) $L_1 \cup L_2 \cup \cdots = L$, meanwhile 3) for every L_i there exists a unique path l_i^A in M^A that they have the same probability of reaching target states, where ϕ is true.

Assume $l_1^A = \{s_0^A, s_1^A, ...s_n^A\}$ is a path in M^A and s_n is a target state, then the probability of this path satisfying the property ϕ is $P^A(s_0^A, s_1^A) \times P^A(s_1^A, s_2^A) \times \cdots \times P^A(s_{n-1}^A, s_n^A)$. Now in M, L_1 can be built as picking all paths $\{s_0, s_1, ...s_n\}$ in L satisfying $s_i \in maps(s_i^A)$. Assume $maps(s_i^A) = \{s_i^0, s_i^1, ...s_i^{m_i}\}$ and $P(s_{i-1}^0, s_i^{k_i}) = p_{i-1}^{k_i}$. Then the sum of probability of L_1 is

$$\begin{aligned}\sum\{p(l)|l \in L_1\} &= \textstyle\sum_{k_0 \in [0,m_0], k_1 \in [0,m_1], \cdots, k_n \in [0,m_n]} (\prod_{i \in [0,n-1]} P(s_i^{k_i}, s_{i+1}^{k_{i+1}})) \\ &= \textstyle\sum_{k_0 \in [0,m_0], k_1 \in [0,m_1], \cdots, k_n \in [0,m_n]} (\prod_{i \in [0,n-1]} P(s_i^0, s_{i+1}^{k_{i+1}})) \\ &= \textstyle\sum_{k_0 \in [0,m_0], k_1 \in [0,m_1], \cdots, k_n \in [0,m_n]} (\prod_{i \in [0,n-1]} p_i^{k_{i+1}}) \\ &= \textstyle\prod_{i \in [0,n-1]} (\sum_{k_i \in [0,m_i]} p_i^{k_{i+1}}) = \prod_{i \in [0,n-1]} P^A(s_i^A, s_{i+1}^A) = p^A(l_1^A)\end{aligned}$$

Therefore, L_1 and l_1^A have the same probability of reaching $maps(s_n^A)$ and s_n^A. Thus we get that for each path in M^A, there is a group of paths in M which have the same probability of reaching target states. Suppose there is a path l in M that is not grouped in any L_i, then there must be a state on l which does not have a representative state in M^A, which is conflict with the definition of M^A, therefore 3) is true. Because L_i and L_j correspond to different paths in M^A, then 1) true, otherwise $l_i^A = l_j^A$ which is impossible. Since no path in M does not belong to any L_i, then 2) is true. Therefore, the theorem holds.

Following previous analysis, we model each action instead of each agent as a process in model implementation. Each action process's behavior is determined by the stochastic behaviors of all agents previously choosing it. The current local state of each action (process) is represented by the number of agents currently

[5] ϕ can be any formally defined property such as LTL or CTL formula.

choosing it, which will be updated accordingly based on the stochastic behaviors of the relevant agents. If there is a new agent choosing action a_i, then the variable recording the local state of action a_i will be increased by 1. Each global state of the system is determined by the local states of all the action processes (the game outcome) together with all global variables. Fig. 1 shows the behaviors of the model for ESS with $|A_i| = 2$ and any number of agents. In this model, two processes, Action 1 and Action 2, are executing in parallel, and are also synchronized at the end of each round. For each action process i, its current local state is represented by the number n_i of agents choosing it in the current round. The execution path of each process is determined by its current local state and the behaviors of the agents choosing it. Specifically, each process i repeatedly checks whether there is any agent that takes action i in current round but has not made its next round decision yet. If yes, the process proceeds by allowing this agent to make its decision in the way as specified by ESS and makes update accordingly; if not, the process waits, updates its local state and starts the next round after the other process also finishes this round. The behaviors of the model for BSS are similar and we omit it here.

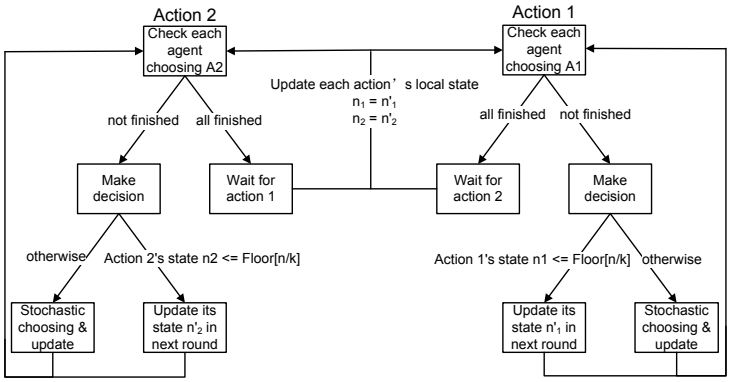

Fig. 1. Finite state automaton of the model of ESS with $|A_i| = 2$

Besides, by using two sets of variables to record the local states of the action processes (e.g, n_1 and n_1' in Fig. 1), the behavior of each process does not have any side effect on the behaviors of other processes. Therefore the updating of each action process can be performed in a sequential way without involving any non-determinism, and thus it is sufficient to model the system as DTMC instead of MDP as previously mentioned. We implemented both models in PAT [19, 21], which is an extensible model checking framework supporting simulation and verification of concurrent, real-time and stochastic systems as well as other domain-specific systems. The concrete PAT models for both strategies are omitted here and the full versions can be found at [11].

4 Analysis and Verifications

In this section, we present how we can analyse the performance of both strategies BSS and ESS with probabilistic model checking. We mainly concentrate on the following two properties: convergence and convergence rate.

Firstly, we are interested in checking the property of convergence of previous strategies, i.e., whether the agents adopting the strategy are guaranteed to converge to an MDO at last. This is an important property to analyse in dispersion games and in the literature of learning in games as well [22, 1]. If the answer is positive, it indicates that the strategy can let the agents to stabilize on the most efficient outcome eventually. Previous work [22] has empirically shown that the outcome converges to an MDO under both strategies. We automatically verify this property using probabilistic model checking and we gain better understandings than that only based on empirical evaluations. Secondly, we demonstrate an analysis of the average number of rounds that the agents using previous strategies take before converging to an MDO. This is an important metric in evaluating the learning efficiency of different learning strategies in terms of converging to an MDO.

Compared with the approximate results from simulation, the quantitative analysis results obtained through model checking are exact since it is based on exhaustive explorations of all possible paths of the system. We can gain better insights of the dynamics of both strategies through verifying the above properties using probabilistic model checking. For example, previous results based on simulation show that the agents always converge to an MDO under ESS, however, we find that the outcome may deviate depending on the relation between the number of agents and actions. The next two sections describe the analysis of these two important properties in details.

4.1 Convergence

We first consider the property whether the agents adopting the strategies BSS and ESS are guaranteed to converge to an MDO. To verify this property, we can check the probability that the system satisfies the LTL formula as follows:

$$\mathcal{P}r(System \models \Diamond\Box\ MDO); \tag{1}$$

where *System* models the overall system and *MDO* represents the state when the outcome is an MDO. The combination of $\Diamond$ and $\Box$ operators captures the meaning of convergence, i.e., the system will eventually reach the state MDO and will always be in that state thereafter. Through verifying property 1, the probability that the system converges to an MDO is obtained.

For the first strategy BSS, it is required that the number n of players is always equal to the number of actions $|A_i|$. Due to the state space explosion with the increase of problem size, we only consider three simple cases with $n = 3, 4$ and 5. By automatically verifying the previous property, the model checker PAT returns that the probability that the outcome converges to an MDO is 1 for all cases.

For the second strategy ESS, the number of players can be different from the number of actions. We consider the following cases here: the number of available actions $k = 2, 3, 4$, and the number of players n varies from 2 to 10. Table 1 shows the convergence probability $P_c(n, k)$ for all these cases by automatically verifying property 1.

Table 1. Probability of Convergence to an MDO of ESS

$P_c(n,k)$	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
k=2	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0
k=3	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
k=4	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0

From Table 1, we can see that the strategy ESS cannot guarantee that the outcome will always converge to an MDO, and it depends on the relation between the number of agents and actions available to them. Specifically, for $k = 2, 3, 4$ and $n = 1, ..., 10$, the outcome is guaranteed to converge to an MDO when $k \geq n$ or $n \mod k = 0$, and the convergence property does not hold otherwise. Intuitively, the underlying reason is as follows: when $n \mod k \neq 0$ and $n > k$, for any MDO, there always exists an unstable action such that the agents choosing this action would always have certain probability to choose other actions next round. If all agents choose another action next round simultaneously, the resulting outcome will not be an MDO any more.

For the cases when the convergence property is lost in ESS, it is interesting and useful to consider the following property: if the outcome in round t is an MDO, what is the expected probability that the outcome in next round $t + 1$ is not an MDO? For a given strategy, if this average probability of deviation is very low, we can say that the outcome approximately converges to an MDO under this strategy. To achieve this goal, we can check the probability that the system satisfies the LTL formula as follows.

$$\mathcal{P}r(System \models \ !MDO \ \mathbf{U} \ MDO \ \mathbf{X(m)} \ !MDO); \tag{2}$$

where *System* models the overall system; *MDO* and *!MDO* represent the states when an MDO is achieved or not respectively. $\mathbf{X(m)}$ is a syntactic sugar reading as "next m-th", and $\mathbf{U}$ is a temporal logic operator which reads as "until". The formulae *MDO* $\mathbf{X(m)}$ *!MDO* means that the condition *!MDO* is satisfied for the m-th state after the state which satisfies the condition *MDO*. By letting m equal to the number of states within one round, the whole formulae represents the condition that the outcome will deviate from MDO next round whenever an MDO is reached [6]. By checking property 2, we can automatically obtain the exact probability that the above condition is satisfied. This property is verified

[6] Here m is manually determined based on the generated simulation traces. We later propose a modeling language PMA [18] specifically designed for modeling multi-agent system so that the transitions between rounds can be identified automatically.

for all previous cases that the convergence property is lost in ESS, and the results are shown in Table 2.

Table 2. Probability of Deviation after reaching an MDO

$P_c(n,k)$	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
k=2	0.0625	0.0	0.0699	0.0	0.0746	0.0	0.0686	0.0
k=3	0.0	0.0725	0.125	0.0	0.0912	0.1368	0.0	0.092
k=4	0.0	0.0	0.122	0.1524	0.1583	0.0	0.1374	0.1573

In Table 2, the cases with value 0 indicate that the outcome will always converge to an MDO, which is in accordance with previous results. For the rest of cases that the convergence property is lost, the exact probability that the outcome will deviate from an MDO is obtained. These results give us a better understanding and more accurate prediction of the dynamics of the agents' behaviors.

4.2 Convergence Rate

Next we consider analysing another important property of the strategies: the convergence rate. For any strategy that has been proven to (approximately) converge to desirable outcomes, the natural following up question would be how fast the convergence could be. Here we illustrate how we can analyse this property of both strategies using reward checking techniques.

To analyse the convergence rate of the strategy BSS, we calculate the average number of rounds it takes for the outcome to converge to an MDO. Since we have previously shown that the outcome will not deviate once an MDO is achieved in BSS, it is equal to check the average number of rounds it takes for the outcome to reach an MDO for the first time. For the strategy ESS, it does not always guarantee the convergence to an MDO. For those cases that the convergence property is lost, here we only check the average number of rounds it takes until an MDO is reached for the first time.

This kind of property can be analysed using reward checking. Here we use transition reward to calculate the average rewards from the initial state to an MDO. Specifically, we set the reward of finishing each round is 1, and increase the accumulated reward by one after each round. Using the iterative method in [2], we can calculate the average rounds (rewards) from the initial state to an MDO. The property can be expressed as follows.

$$\mathcal{R}(System \models \Diamond MDO); \tag{3}$$

where *System* models the system's dynamics and *MDO* represents the condition that an MDO is reached. Through checking the above property, the exact average number of rounds the system takes to converge to an MDO can be obtained.

Due to space constraint, we only provide the results for ESS in different cases, shown in Table 3. For those cases that ESS guarantees the agents to converge to MDO (denoted in black), we can see that the average number of rounds required to converge to MDO is gradually increased when the number of agents n becomes larger. This implies that the convergence rate is gradually decreased with the increasing of the number of agents n. The intuitive reason is that the overall system becomes more dynamic due to the increase in the number of stochastic agents, thus making it more difficult for the agents to coordinate their actions. Another interesting observation is that the average number of rounds before convergence are always locally maximal (i.e., larger than that for both cases of $n-1$ and $n+1$) for those cases when the condition $n \mod k = 0$ is satisfied, i.e., when the convergence property holds.

Table 3. Average Number of Rounds to Converge to (Reach) an MDO in ESS

$\bar{R}$(n,k)	n = 3	n = 4	n=5	n=6	n=7	n=8	n=9	n=10
k=2	1.33	**2.44**	1.55	**2.69**	1.70	**2.87**	1.81	**3.00**
k=3	**2.63**	1.48	2.11	**3.20**	1.81	2.45	**3.52**	2.04
k=4	2.15	**3.08**	1.58	2.15	2.90	**3.73**	2.04	2.59

By applying counter abstraction technique in our modeling process, both the state space and the verification time cost are greatly reduced. To show this, here we list the details of verification time cost and state space cost of checking the convergence probability of ESS strategy in Table 4.[7] We can see that our counter abstraction approach can significantly reduce the state space and the verification time, and more than half cases are not able to verify within a reasonable amount of time without abstraction.

5 Conclusion and Future Work

In this paper, we proposed a novel way of analysing a multi-agent system using probabilistic model checking techniques. We studied two specific strategies: *basic simple strategy* and *extended simple strategy* in dispersion games, and demonstrated how the dynamics of the agents under these strategies can be modeled and proposed using abstraction techniques to reduce the model's state space. Better insights of the dynamics of these strategies were obtained compared with previous empirical evaluations.

Analysing the dynamics of a multi-agent system using model checking techniques has its unique advantages compared with both simulation and mathematical analysis techniques. However, there are a number of issues left as future work. First, The size of the state space of the model usually grows exponentially

[7] Note that the unit of the verification time is second and "-" means that the verification takes more than 10 minutes.

Table 4. The Number of States and Verification Time for Checking the Convergence Probability of ESS with and without Abstraction

	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
k=2 (state (with abstraction))	42	99	158	284	400	622	814	1159	1446
k=2 (state (without abstraction))	62	532	3482	35758	242602	-	-	-	-
k=2 (time (with abstraction))	0.015	0.017	0.027	0.027	0.061	0.043	0.168	0.064	0.276
k=2 (time (without abstraction))	0.063	0.073	0.631	28.77	126.7	-	-	-	-
k=3 (state (with abstraction))	99	349	1155	2413	3598	8009	13323	17614	32105
k=3 (state (without abstraction))	352	1985	38729	-	-	-	-	-	-
k=3 (time (with abstraction))	0.021	0.035	0.065	0.117	0.421	0.457	0.556	5.376	1.492
k=3 (time (without abstraction))	0.093	0.196	35.51	-	-	-	-	-	-
k=4 (state (with abstraction))	512	1885	3588	14478	34668	72848	103108	257861	462256
k=4 (state (without abstraction))	386	6662	148898	-	-	-	-	-	-
k=4 (time (with abstraction))	0.049	0.069	0.266	1.171	1.938	3.683	37.09	20.77	27.951
k=4 (time (without abstraction))	0.127	0.709	32.58	-	-	-	-	-	-

with the number of processes, which is known as the infamous state explosion problem. Even though the abstraction technique we propose in this paper has significantly reduced the state space, it still cannot handle the cases when the model becomes too large. Thus more effective approaches are required to make this technique more scalable and applicable for analysing larger scenarios. One promising direction is to resort to statistical model checking [13] which can support efficient analysis of larger scenarios.

Second, in this paper, we only focus on two specific strategies in the context of dispersion games under the assumption of agent and action symmetry, but we do believe that we have identified an alternative approach that has great potentials for automatic analysing multi-agent system dynamics. Thus another natural direction is to consider relaxing the symmetric assumption and also investigating other strategies in the multi-agent learning literature and how to automatically analyse them using model checking techniques.

Acknowledgements

The work presented in this paper was partially supported by a CUHK Research Committee Direct Grant for Research (Ref. No. EE09423).

References

1. Alpern, S.: Spatial dispersion as a dynamic coordination problem. Tech. rep., The London School of Economics" (2001)
2. Baier, C., Katoen, J.: Principles of Model Checking, chap. 10, pp. 866–883. The MIT Press (2008)
3. Ballarini, P., Fisher, M., Wooldridge, M.: Uncertain agents verification through probabilistic model-checking. In: SASEMAS'09. pp. 162–174. Lecture Notes in Computer Science (2009)

4. B.Arthur: Inductive reasoning and bounded rationality. American Economic Association Papers 84, 406–411 (1994)
5. Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M.: Verifying multi-agent programs by model checking. AAMAS 12, 239–256 (2006)
6. Claus, C., Boutilier, C.: The dynamics of reinforcement learning in cooperative multiagent systems. In: AAAI'98. pp. 746–752 (1998)
7. D.Challet, Y.Zhang: Emergence of cooperation and organization in an evolutionary game. Physica A 246, 407 (1994)
8. G.Holzmann: The spin model checker. TSE 23(5), 279–295 (1997)
9. Gomes, E.R., R.Kowalczyk: Dynamic analysis of multiagent -learning with e-greedy exploration. In: ICML'09 (2009)
10. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 102–111 (1994)
11. Hao, J.Y., Song, S.Z., Liu, Y., Sun, J., Dong, J.S., Leung, H.F.: Online Technical Report. `http://www.comp.nus.edu.sg/~pat/prima2012/`
12. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A Tool for Automatic Verification of Probabilistic Systems. In: TACAS'06. pp. 441–444 (2006)
13. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: RV'10. pp. 122–135 (2010)
14. Osborne, M.J., Rubinstein, A.: A Course in Game Theory. MIT Press, Cambridge (1994)
15. P. Stone, M.V.: Multiagent systems: A survey from a machine learning perspective. Autonomous Robots 8, 345–383 (2000)
16. Pnueli, A., Xu, J., Zuck, L.: Liveness with (0,1,∞)-counter abstraction. In: CAV'02. pp. 107–122 (2002)
17. Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In: MAAMAW'96. pp. 42–55 (1996)
18. Song, S.Z., Hao, J.Y., Liu, Y., Sun, J., Leung, H.F., Dong, J.S.: Analyzing multi-agent systems with probabilistic model checking approach. In: ICSE'12, NIER (2012)
19. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: Towards Flexible Verification under Fairness. In: CAV'09. pp. 709–714 (2009)
20. Sun, J., Liu, Y., Roychoudhury, A., Liu, S., Dong, J.S.: Fair model checking with process counter abstraction. In: FM'09. pp. 123–139 (2009)
21. Sun, J., Song, S., Liu, Y.: Model checking hierarchical probabilistic systems. In: ICFEM'10. pp. 388–403 (2010)
22. T. Grenager, R. Powers, Y.S.: Dispersion games: general definitions and some specific learning results. In: AAAI'02. pp. 398–403 (2002)
23. Tadjouddine, E.M., Guerin, F., Vasconcelos, W.: Abstraction for model checking game-theoretical properties of auction(short paper). In: AAMAS'08. pp. 1613–1616 (2008)
24. Tuyls, K., Verbeeck, K., Lenaerts, T.: A selection-mutation model for q-learning in multi-agent systems. In: AAMAS'03. pp. 693–700 (2003)
25. Vidal, J.M., Durfee, E.H.: Predicting the expected behavior of agents that learn about agents: The clri framework. AAMAS 6, 77–107 (2003)
26. Wooldridge, M., Fisher, M., Huget, M.P., Parsons, S.: Model checking multi-agent systems with mable. In: AAMAS'02. pp. 952–959 (2002)
27. W.Visser, Havelund, K., G.Brat, S.Park: Model checking programs. In: ASE'00. pp. 3–12 (2000)
28. Y.Azar, A.Z.Broder, A.R.Karlin, E.Upfa: Balanced allocations. SIAM Journal on Computing 29(1), 190–200 (2000)