2008

A New Method for Conflict Resoluton Based on Multi-Agent Reinforcement Learning Algorithms

Donghua Li
lidonghua@nuaa.edu.cn

Ju Jiang
jiangju@nuaa.edu.cn

Huajun Gong
ghj301@nuaa.edu.cn

Jianye Liu
liyac@nuaa.edu.cn

Bin Jiang
binjiang@nuaa.ecu.cn

A New Method for Conflict Resolution Based on Multi-agent Reinforcement Learning Algorithms

Donghua Li
Affiliation
lidonghua@nuaa.edu.cn

Ju Jiang
Affiliation
jiangju@nuaa.edu.cn

Huajun Gong
Affiliation
ghj301@nuaa.edu.cn

Jianye Liu
Affiliation
liyac@nuaa.edu.cn

Bin Jiang
Affiliation
binjiang@nuaa.ecu.cn

ABSTRACT

Conflict resolution is a research topic for game theory (GT) and conflict analysis. A decision support system (DSS) is very helpful for conflict decision making. Reinforcement learning (RL) is an efficient method to learn knowledge by agents themselves. Although successful applications of RL have been reported in single-agent domain, a lot of work should be done in the case of multi-agent domain. Nash Q-learning is a famous learning algorithm for multi-agent RL. Based on the Nash Q-learning, a novel DSS: multi-agent RL based DSS (MRLDSS) is proposed in this paper and is tested by using several typical examples of conflict resolution. Experimental results show that the proposed architecture and algorithm can solve conflict resolution problems correctly and efficiently.

Keywords

Decision support system, reinforcement learning, multi-agent, game theory, conflict analysis.

INTRODUCTION

The research purpose of this paper is to apply multi-agent RL technique to establish a DSS for conflict resolution. DSS is firstly introduced by Scott Marton and Garry in the beginning of 1970's (Marakas, 1993). A DSS is a new synthesized form that consists of an electronic data process (EDP) and a manage information system (MIS). A DSS is applied to support structured, semi-structured or unstructured decision and allows users' participation.

DSS has gone through several development periods. Now, IDSS (Intelligent DSS) is a new focus in DSS research (Wu and Chen, 2007). IDSS is the combination of artificial intelligence (AI) and DSS. With the expert system (ES) technology, IDSS can make more fully use of human knowledge or intelligent knowledge, i.e., description knowledge about decision, decision processing knowledge and reasoning knowledge to make decision correctly and efficiently. According to the architecture of IDSS, it can be divided into: ES-based IDSS, ML-based IDSS, Agent-based IDSS, and so on.

Game Theory, which specializes in conflict analysis, was invented by John von Neumann and Oskar Morgenstern. Since the World War II, it has bloomed and formed many conflict resolution concepts (Fang, Hiple, and Kilgour, 1993). The concepts of Nash stability and general metarationality (GMR) are two of the most important ones and will be used in this paper.

Reinforcement learning (RL) describes a method of mapping the set of states to the set of actions for obtaining the maximal long-term reward. The maximal long-term reward trend is formed through rewarding the expected results and punishing the unexpected ones (Kazuo, 2000).

The rest of the paper is organized as follows. Section 2 gives the basic concepts of RL and Game Theory (GT). The architecture of the proposed MRLDSS and the description of the suggested algorithm are provided in Section 3. Section 4 presents the experimental procedures and results. Some conclusions drawn from the experimental results are summarized in Section 5.

BACKGROUND

Single-agent Q-learning

Single-agent Q-learning is a widely used RL algorithm proposed by Watkins (Sutton, and Barto, 1998; Watkins and Dayan, 1992).

Definition 1: An RL problem is a quadruplet <*S, A, r, T*>, where *S* is a discrete state space, *A* is a discrete action space, $r: S \times A \rightarrow Re$ is a reward function defined below for an agent, and $T: S \times A \rightarrow \triangle(S)$ is the transition function, where $\triangle(S)$ is the set of probability distributions over state space *S*.

For the one-step Q-learning algorithm, it updates its state-action values as

$$Q_k(s_t, a_t) = (1-\alpha)Q_{k-1}(s_t, a_t) + \alpha[r_t + \gamma \max_{a \in A} Q_{k-1}(s_{t+1}, a)]. \quad (1)$$

In the above equation, $Q_k(s_t, a_t)$ is the state-action value, called Q-value or value function, for action a_t in state s_t after the k^{th} updating. The initial Q-values, $Q_0(s_t, a_t)$, $s_t \in S$, $a_t \in A$, can be set randomly. r_t is the instant reward given to the agent after it takes the action a_t and transfers to the new state, s_{t+1}. Step-size or learning rate, α, is a changeable parameter between 0 and 1 and is decreased during learning process. The factor of discount, γ, measures how the immediate rewards are weighted against the future rewards. In addition, this discount factor ensures that the returns from all the visited states are finite.

Multi-agent Q-learning

Littman extended the single-agent Q-learning to the case of multi-agent environment for zero-sum game and develop the multi-agent Q-learning algorithm, Minimax-Q (Littman, 1994). Based on the method, Hu and Wellman developed the Nash Q-learning algorithm. In Nash Q-learning, Hu and Wellman defined "optimal Q-values as Q-values received in a Nash equilibrium (Hu and Wellman, 2003)". The goal of Nash Q-learning is to find the Nash Q-values of different learning agents.

Definition 2: An n-player RL problem is a set <*S*, $A^1 \cdots A^n$, $r^1 \cdots r^n$, *T*>, where *S* is the state space, A^i is the action space of agent *i*, $r^i: S \times A^1 \times \cdots \times A^n \rightarrow Re$ is the reward function for agent *i*, and *T* is the state transition function. In this paper, *T=1*.

In this situation, the Nash Q-values function of agent *i* is defined as follow

$$Q_k^{i,*}(s_t, a^{joint}) = r^i + \gamma \sum_{s_{t+1} \in S} v^i(s_{t+1}, \pi^{1,*}, \cdots, \pi^{n,*}), \quad (2)$$

where $(\pi^{1,*}, \cdots, \pi^{n,*})$ is the optimal joint policy in a Nash equilibrium, and $V^i(s_{t+1}, \pi^{1,*}, \cdots, \pi^{n,*})$ is the total discounted reward of agent *i* over infinite periods starting from state s_{t+1} when all agents follow the equilibrium strategies. a^{joint} is the joint action, and r^i is the reward of agent *i* $(i=1, \cdots, n)$ when all the agents take joint action a^{joint} and divert to the new state, s_{t+1}.

Nash Q-learning updates the Q-values as

$$Q_k^i(s_t, a^{joint}) = (1-\alpha)Q_{k-1}^i(s_t, a^{joint}) + \alpha[r_t^i + \gamma NashQ_{k-1}^i(s_t, a^{joint})]. \quad (3)$$

Where, $NashQ_{k-1}^i(s_{t+1})$ is the Q-value of agent *i* in state s_{t+1} for the selected Nash equilibrium. The main difference between Nash Q-learning and single-agent Q-learning is how to use the Q-values of the next state to update the Q-values of the current state. For single-agent Q-learning method, the algorithm selects the optimal Q-value of the next state; for Nash Q-learning algorithm, it adopts the Q-values in Nash equilibrium (Hu and Wellman, 2003).

Equilibrium States of Conflict Analysis

The environment to which the proposed multi-agent RL algorithm is applied is an n-DM RL problem defined by Definition 2. Individual stability and associated equilibrium are introduced within the framework of the graph model for conflict resolution (Fang et al., 1993). First consider the definitions for various types of movements among states as controlled by the DMs. In particular, $R_i(s)$ is the set of reachable states, which DM i or agent i can reach in one step, or unilateral movement, from state s. $R_i^+(s)$ is the set of states, referred to as unilateral improvements (UIs), that belongs to $R_i(s)$ and their payoffs are larger than that for state s for DM i. Let $P_i(s)$ be the payoff of DM i for state s. Take the chicken game as an example, the state space and payoff are shown in Table 5.

Definition 3: Nash stability: state $k \in S$ is Nash stable for DM *i* ($i \in N$), *iff* (if and only if) $R_i^+(k) = \phi$. If *k* is Nash stable for all DMs, it is the Nash equilibrium state (Fang et al., 1993).

In the case of Nash stable, player *i* expects that player *j* will stay at any state it transfers to, i.e., player i believes that any state that it transfers will be the final state. As an example of Nash stability, consider state 3 from DM 1's viewpoint in chicken game. If DM 1 unilaterally moves (UMs) from state 3 to state 1, his payoff changes from $P_1(3)=2$ to $P_1(1)=1$. Hence, state 3 is Nash stable for DM 1 since he has no UI from state 3. Because state 3 is also Nash stable for DM 2, it constitutes a Nash equilibrium.

Definition 4: General Metarational (GMR): state $k \in S$ is GMR for DM *i* ($i \in N$), *iff* for every $k_1 \in R_i^+(k)$ there exists at least one $k_2 \in R_j(k_1)$ with $P_i(k_2) \leqslant P_i(k)$.

Thus, under GMR, DM *i* expects that DM *j* will respond by hurting DM *i* if it is possible for DM *j* to do so. Therefore, *k* is stable *iff* DM *j* can hurt DM *i* if DM *i* takes any UI. Here, in the chicken game, for instance, DM 1 has an UI from state 4 to state 2 because $P_1(4)=3$ and $P_1(2)=4$. However, DM 2 has an UM from state 2 to state 1 which is less preferred by DM 1 to state 4 ($P_1(1) < P_1(4)$). Hence, state 4 is GMR for DM 1. Since state 4 is also GMR for DM 2, this state is a GMR equilibrium.

ARCHITECTURE OF MULTI-AGENT REINFORCEMENT LEARNING BASED DECISION SUPPORT SYSTEM

Instruction about the Architecture

The proposed architecture of multi-agent RL based DSS (MRLDSS) is shown in Figure 1. This architecture consists of 4 modules: user interface module, model-matching module, intelligent learning module, and model-extract module. The user interface module is used to exchange information between users and MRLDSS. The model-matching module is used to process the information provided by users and tries to determine whether there is a matched model in the existing model-base. If the input problem can match with one of the existing models, the existing model is used to make the decision for the given problem directly. Otherwise, pass the given problem to the intelligent learning module (it is the multi-agent RL part in the Figure 1), and then the intelligent learning module learns the optimal decision policy for the problem. If the learned decision policy is stable and reliable, a new model is obtained and added to the model-base through the model-extract module.

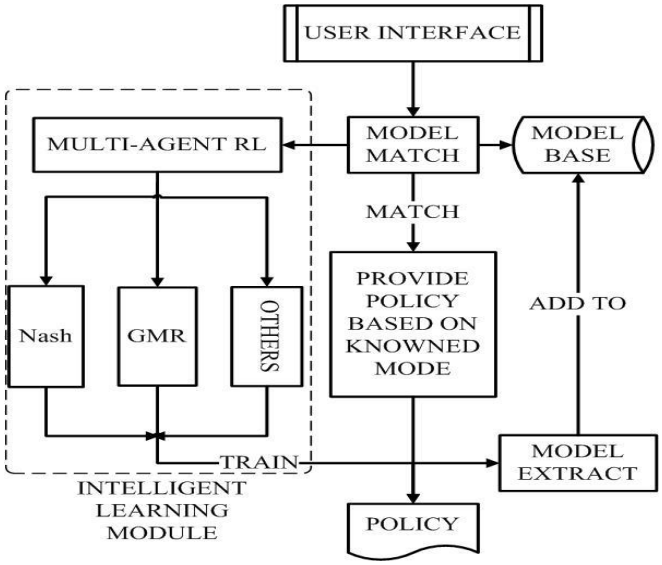

Figure 1. DSS Architecture

Algorithm Description

In Hu and Wellman's paper (Hu and Wellman, 2003), they verified the Nash Q-learning with a grid-world game. However, a decision making problem of GT is different from a grid-world game. We should develop a new algorithm to solve decision making problems based on Nash Q-learning. Nash Q-learning learns the Nash equilibrium of the current Q-values. The goal

of the proposed algorithm is to search the equilibrium states defined by GT. However, the equilibrium states have nothing to do with the Q-values. Therefore, in this algorithm, the equilibrium Q-values are used to update the Q-values.

The Q-values function of equilibrium is the same as that of Nash Q-learning. But here, the $(\pi^{1,*}, \cdots, \pi^{n,*})$ is the joint policy to stay at equilibrium state. In the proposed algorithm, the Q-values are updated as follow.

$$Q_k^i(s_t, a^{joint}) = (1-\alpha)Q_{k-1}^i(s_t, a^{joint}) + \alpha[r_t^i + \gamma AEQ_{k-1}^i(s_{t+1})], \quad (4)$$

where $AEQ_{k-1}^i(s_{t+1})$ is the Q-value of state s_{t+1} at which every agent is assumed to tend to stay. For the joint action staying at state s_{t+1} is assumed, and the real joint action is unknown. So it is called AEQ (Assumed Equilibrium Q-value).

The reward function is based on the all agents' joint action and the current state. If an agent reaches its stable state, it will get a reward of its current payoff; else it will get a zero.

Two methods are used to select actions based on the Q-values. The first one is the simulated annealing (SA). In the beginning of learning, the agent does not have knowledge about the correct action selection, so, the Q-values cannot represent its real intention. SA selects actions in this situation to ensure fully exploring over all the state space. The second one is the greedy method. After learning enough times, the agent forms its policy. So selecting an action with the largest Q-value is reasonable.

To explain the proposed algorithm clearly, a simple example that satisfies Definition 2 with two agents and two feasible actions is used here. The state space and payoff of this example are shown in Table 1 (assumed the two agents have the same actios), and the Q-values are updated as follow:

$$\Delta Q^i(s, a^i, a^j) = \alpha[r^i + \gamma AEQ(s_{t+1})], \quad (5)$$

where $AEQ(s_{t+1}) = Q_i(s_{t+1}, a_s^i, a_s^j)$.

state	Agent 1		Agent 2	
	action	payoff	action	payoff
1	a_1	$P_1(1)$	a_1	$P_2(1)$
2	a_2	$P_1(2)$	a_2	$P_2(2)$
3	a_3	$P_1(3)$	a_3	$P_2(3)$
4	a_4	$P_1(4)$	a_4	$P_2(4)$

Table 1. States and Payoffs

The pseudo code of the proposed algorithm is provided in Table 2.

```
Initialize     // initial the necessary parameter
Q^1←φ, Q^2←φ, α←0.3, γ←1,select state randomly;
stepRecord←φ;   // record the convergent step number
stateRecord←φ;   // record convergent state every time
c_2←0;    // the number of continued convergent to the same s
loop 1  while k<N, loop continued
  c_1←0;    // the number of continued stay at the same s
  step←0;   //convergent step number
   loop 2  while c_1<m loop continued   //if c_1<m, it  converges
     a^1←ActionSelect, i=1,2 // select action
     lastState←currentState;
      currentState←new state s_{t+1};
     r^i_{Nash/GMR} ← reward, i = 1,2   //calculate reward
     Q^i←(1-α)Q^i+α[r^i+AEQ_i(s)], i=1,2;   //update Q-values
     if s_{t+1}==s_t
       c_1←c_1+1;
```

```
  else
    c1←0;
   step←step+1;
Return to loop 2;
stepRecord(k)←step;
stateRecord(k)←st
if stateRecord(k)== stateRecord(k-1)
  c2←c2+1;
else
  c2←0;
if c2 >n   //continued converged to a state more than n times
reset all
return to loop 1
```

Table 2. Pseudo Code of the Proposed Algorithm

Comparing of the three RL Algorithms

Table 3 lists the differences among the three algorithms mentioned above. The differences between the proposed algorithm and the others are summarized as follows.

1. The Q-table is different. Both the proposed algorithm and Nash Q-learning consider the opponent's actions, but the single-agent Q-learning does not.
2. The "optimal" Q-value is different. For updating the Q-values, single-agent Q-learning uses the largest Q-value that it can get in the next state; Nash Q-learning uses the Q-value that all agents select the joint action to Nash equilibrium in the next step; and the proposed algorithm uses the Q-value that all agent select joint action to stay at the next state.
3. The estimated opponent's Q-values are different. In Nash Q-learning, it is necessary to estimate opponents' Q-values for calculating the Nash equilibrium. However, neither the proposed algorithm nor the single-agent Q-learning needs.

	Single-agent Q-learning	Nash Q-learning	The proposed algorithm
Q-values	$Q(s, a)$	$Q(s, a^1, \cdots, a^n)$	$Q(s, a^1, \cdots, a^n)$
"optimal" Q-value	Max Q	Nash Q	AEQ
estimate opponent's Q-values	no	yes	no

Table 3. Comparing of the Three Algorithms

EXPERIMENTAL RESULTS

In this section, two famous examples of conflict resolution are used to evaluate the proposed algorithm.

Prisoners' Dilemma

In this example, there are two prisoners and each one has two actions to be selected: confession (C) and defence (D). Table 4 gives the state space and payoffs of the prisoners' dilemma problem. It also gives out the equilibrium state. Here, " √ " means yes, and "×" means no.

state	P1		P2		equilibrium	
	action	payoff	action	payoff	Nash	GMR
1	C	1.0	C	1.0	√	√
2	C	5.0	D	0.0	×	×
3	D	0.0	C	5.0	×	×
4	D	3.0	D	3.0	×	√

Table 4. States and Equilibrium of Prisoners' Dilemma

Chicken Game

In this game, two drivers are racing toward to each other and each driver can either swerve or not. Table 5 provides the state space and payoffs. Where, S means swerving and D means no.

state	D1		D2		equilibrium	
	action	payoff	action	payoff	Nash	GMR
1	D	1.0	D	1.0	×	×
2	D	4.0	S	2.0	√	√
3	S	2.0	D	4.0	√	√
4	S	3.0	S	3.0	×	√

Table 5. States and Equilibrium of Chicken Game

Experimental Results

In real games, players cannot change their decisions after they took them. One game ends after all players take their actions. For learning, all the games are assumed to be repeatable (Sandholm and Crites, 1996). The experimental results of the two examples are shown in Figures 1, 2, 3, and 4, respectively.

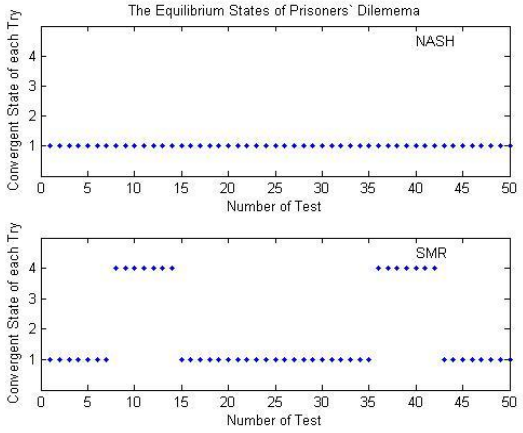

Figure 2. Equilibrium States of Prisoners' Dilemma

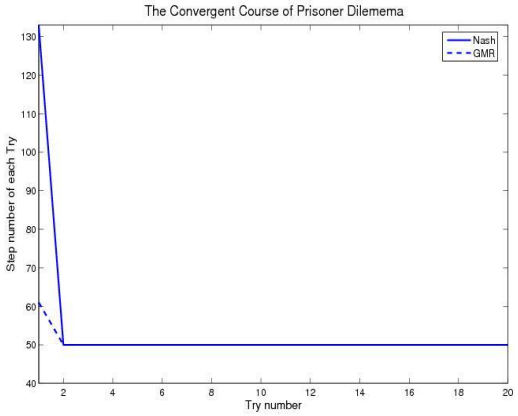

Figure 3. Learning Process of Prisoners' Dilemma

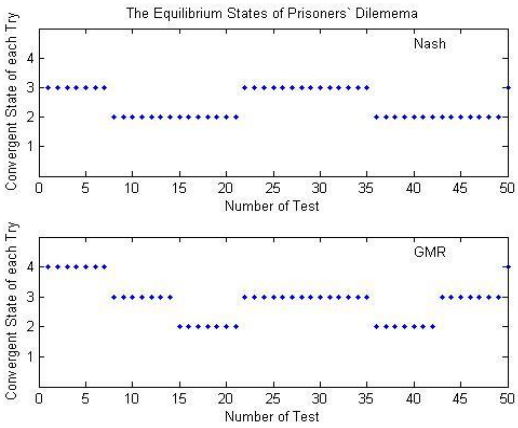

Figure 4. Equilibrium States of Chicken Game

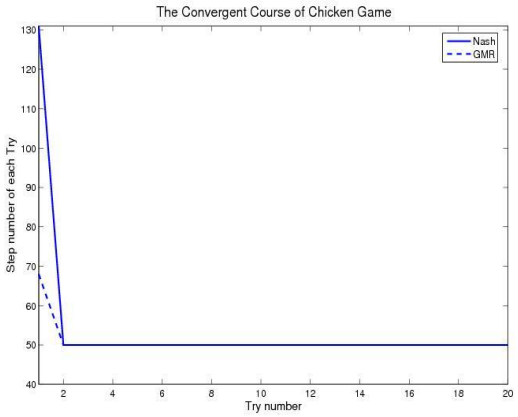

Figure 5. Learning Process of Chicken Game

Experimental Results Analysis

Figures 2 and 4 give out the stable states after 50 tries. These are the equilibrium states. The two figures show that the equilibrium states obtained with the proposed algorithm are the same as those provided by GT in Tables 4 and 5. These results evaluate the proposed algorithm. Figures 3 and 5 show the learning process of the proposed algorithm. Obviously, it learns fast.

The results above verify the feasibility of the proposed algorithm. After learning, all the equilibrium states can be obtained with the modified reward function. Besides, the proposed algorithm converges fast.

CONCLUSION

This paper suggests a novel method to solve conflict problems: searching for the equilibrium states with a proposed multi-agent RL algorithm. The experimental results show that the multi-agent RL method can be used to establish a DSS for conflict resolution. The proposed algorithm can search for the different equilibrium states for a given problem correctly and efficiently. Now the proposed algorithm can only evaluate the equilibrium states defined by the existing game theory. However, a powerful decision support system should provide users more choices and even let them define their own equilibrium states for different agents. The proposed algorithm has the potential and this is our future work.

REFERENCES

1. Fang, L. Hiple, K.W. and Kilgour, D.M. (1993) *Interactive Decision Making: The Graph Model for Conflict Resolution*, John Wiley and Sons Inc., New York.
2. Hu, J. and Wellman, M.P. (2003) Nash Q-learning for General-Sum Stochastic Games, *Journal of Machine Learning Research*, 1, 1039-1069.
3. Kazuo, M. (2000) Learning Scheduling Control Knowledge Through Reinforcements, *International Transactions in Operational Research*, 7, 2, 125-138.
4. Littman, M.L. (1994) Markov Games as a Framework for Multi-agent Reinforcement Learning, *Proc. 11th International Conf. on Machine Learning*, 157-163.
5. Marakas, G.M. (1993) *Decision Support Systems in the 21st Century*, Pearson Education.
6. Sage, A.P. (1991) *Decision Support Systems Engineering*, John Wiley and Sons Inc.
7. Sandholm, T.W. and Crites, R.H. (1996) Multiagent Reinforcement Learning in the Iterated Prisoners Dilemma, *Biosystems*, 37, 147-166.
8. Sutton, R.S. and Barto, A.G. (1998) *Reinforcement Learning: An Introduction*, MIT press.
9. Watkins, J.C.H. and Dayan, P. (1992) Q-learning, *Machine Learning*, 8, 279-292.
10. Wu, X. and Chen, Y. (2007) The Situation of Decision Support Systems Development and its Trends Analysis, *Annual Report about Intelligence Information*, 1, 6, 57-60.