Expert Systems with Applications xxx (2011) xxx–xxx

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy

Behrooz Masoumi [a,*], M.R. Meybodi [b]

[a] Department of Computer Engineering and Information Technology, Islamic Azad University, Qazvin Branch, Qazvin, Iran
[b] Soft Computing Laboratory, Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran

ARTICLE INFO

Index Terms:
Entropy
Learning automata
Markov games
Multi agent systems
Stigmergy

ABSTRACT

Learning automata (LA) were recently shown to be valuable tools for designing Multi-Agent Reinforcement Learning algorithms and are able to control the stochastic games. In this paper, the concepts of stigmergy and entropy are imported into learning automata based multi-agent systems with the purpose of providing a simple framework for interaction and coordination in multi-agent systems and speeding up the learning process. The multi-agent system considered in this paper is designed to find optimal policies in Markov games. We consider several dummy agents that walk around in the states of the environment, make local learning automaton active, and bring information so that the involved learning automaton can update their local state. The entropy of the probability vector for the learning automata of the next state is used to determine reward or penalty for the actions of learning automata. The experimental results have shown that in terms of the speed of reaching the optimal policy, the proposed algorithm has better learning performance than other learning algorithms.

1. Introduction

A multi-agent system (MAS) is comprised of a collection of autonomous and intelligent agents that interact with each other in an environment to optimize a performance measure (Vlassis, 2007). Multi-agent systems are applied in a wide variety of domains including distributed control, resource management, robotic teams, collaborative decision support systems, data mining, and are useful in the modeling, analysis and design of systems where control is distributed among several autonomous decision makers (Goel, Kumar, & Srinivasan, 2008; Parunak, 2000; Vlassis, 2007).

There are several models proposed in the literatures for multi-agent systems based on Markov models. One of these models is Stochastic games (also called Markov Games – MGs) (Osborne & Rubinstein, 1994). The Markov game view of MAS is a sequence of games that have to be played by multiple players, with each game belonging to a different state of the system. Markov games are extensions of Markov Decision Process (MDP) to multiple agents, and are used as a suitable framework for Multi Agent Reinforcement Learning (MARL) (Littman, 1994). MARL deals with the problem of learning optimal behavior within a multi-agent dynamic environment when the environmental dynamic and the algorithms employed by the other agents are initially unknown. In a Markov game, actions are the result of joint action selection of all agents, while *rewards* and the state transitions depend on these joint actions (Claus & Boutilier, 1998; Stone & Veloso, 2000). Moreover, each agent has its own private reward function. As a special case, when only *one state* is assumed, the Markov game is actually known as a repeated *normal form game* in game theory. In addition, when only *one agent* is assumed, the Markov game is known as MDP. Markov games are categorized based on the agents' rewards into cooperative and non-cooperative games. Non-cooperative games may be classified as competitive games and general-sum games (Littman, 1994).

Stigmergy and entropy are two important concepts that can be used to enhance the performance multi-agent systems (Nowe & Verbeeck, 2002; Valckenears, Brussel, Kollingbaum, & Bochmann, 2001; Vrancx, Verbeeck, & Nowé, 2007; Zhuang, 2005). The stigmergy concept provides a relatively simple framework for agent communication and coordination, which is defined as a class of mechanisms that mediate animal-to-animal interactions through the environment (Theraulaz & Bonabeau, 1999). The idea behind stigmergy is that individual agents coordinate their actions by modifying the environment locally rather than through direct interaction. The changed environmental situation caused by one animal, will stimulate others to perform certain actions. Entropy is a significant concept in the thermodynamics, representing the degree of disorder in a thermodynamic system that is played an important role in various fields of computer science, such as coding theory, learning, compression, and others (Lieb & Yngvason, 1999; Zhuang & Chen, 2008).

* Corresponding author. Tel./fax: +98 281 3675782.
E-mail addresses: masoumi@Qiau.ac.ir (B. Masoumi), mmeybodi@aut.ac.ir (M.R. Meybodi).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.12.152

Currently, learning automata (LA) are among the valuable tools to design Reinforcement Learning algorithms and multi-agent systems (Khojasteh & Meybodi, 2007; Nowe, Verbeeck, & Peeters, 2006; Vrancx, Verbeeck, & Nowé, 2008). Due to certain specifications such as structure simplicity, little need for information and feedback from the environment, LAs are very useful in multi-agent systems. In this paper, the concepts of stigmergy and entropy are imported into learning automata based multi-agent systems with the purpose of providing a simple framework for interaction and coordination in multi-agent systems and speeding the learning process. The multi-agent system considered in this paper is designed to find optimal policies in general Markov games. In the proposed method in addition to real agents, we consider several dummy agents that walk around in the states of the environment. In each state of the environment, for each real agent, a learning automaton is placed which controls the agents' behavior for the state transition so that the best solution to maximize the expected reward is found. The selected actions of learning automata in each state determine the next state of the agents. The reward or penalty for the selected action of LAs in each state is determined based on the immediate reward received by agent and the entropy of the probability vector for the learning automata of the next state. The entropy information is used to improve the selection of the policy taken at each transition leading us to reach optimal policy in Markov games. To evaluate the proposed method, it has been applied to two grid games, which are examples of Markov games. The results of extensive computer simulations have shown that in terms of the speed of reaching the optimal policy, the proposed algorithm has better learning performance than other learning algorithms such as NASH-Q algorithm, Nash Bargaining algorithm and Abtahi's algorithm.

The rest of this paper is organized as follows. Section 2 gives an overview on related works reported on solution of Markov games. Section 3 gives an overview on related works reported on Markov decision process and Markov games solution. In Section 4, two concepts of the learning automata and the entropy is studied. Definitions of stigmergy and stigmergetic algorithms in multi-agent system are discussed in Section 5. The proposed algorithm is given in Section 6. Section 7 presents the computer experiments and Section 8 concludes the paper.

2. Related work

Several different MARL algorithms have been proposed to find optimal policies in Markov games including competitive, fully cooperative, and general ones (Busniu, Babuska, & Schutter, 2008). Most of these algorithms seek to find the equilibrium policy. Littman suggested *"the minimax solution"* for zero-sum (or competitive) games (Littman, 1994). Hu and Wellman extended this algorithm for solving general-sum (non-cooperative) games. They proposed an algorithm called as Nash-Q, which, under restrictive conditions, converges to Nash equilibrium policy (Hu & Wellman, 2003). Using linear programming, Greenwald et al. presented a Correlated Equilibrium-based algorithm that was capable to calculate correlated equilibrium points (Greenwald & Hall, 2003). Meiping Song et al. proposed an algorithm called Pareto-Q for cooperative general-sum games, with the Pareto Optimum allowing social conventions to benefit the convergence (Song, Bai, & Chen, 2007; Song, Gu, & Zhang, 2005). Qio et al. used Nash Bargaining solution for general MGs (Qio, Szidarovszky, Rozenblit, & Yong, 2006). In Wang and Sandholm (2002), team Markov Games are also approximated as a sequence of intermediate games. In these games, all agents get the same reward function and the agents should learn to select the same optimal equilibrium strategy. The authors present optimal adaptive learning and prove convergence to a Nash equilibrium of the limiting game. However, agents know the joint actions played, and they all receive the same reward. Thus, they are able to build up the game structure. A detailed study of different algorithms can be found in (Busniu et al., 2008).

Many Algorithms based on learning automata (LA) have developed for learning optimal strategies in Markov games. In the past few years, a significant part of the research has focused on comprehending and solving single-stage multi-agent problems, modeled as a *normal form* game (Verbeeck, Nowé, Parent, & Tuyls, 2007) and multi-stage games modeled as Markov games (Verbeeck, Nowé, Peeters, & Tuyls, 2005). Wheeler et al. have shown that a set of decentralized learning automata is able to control a finite Markov Chain with unknown transition probabilities and rewards (Wheeler & Narendra, 1986). In Sastry, Phansalkar, and Thathachar (1994) it is shown that a team of learning automata involved in a general N-person stochastic game converges to Nash equilibrium if each of the team members makes use of a linear learning algorithm called L_{R-I} algorithms. Nowe and Verbeeck (2002) first introduced the use of interconnected learning automata as a model for stigmergetic communication in multi-agent system to solve MMDPs. Vrancx et al. extends this model for Markov games (Vrancx et al., 2007). Masoumi, Meybodi, and Jafarpour (2008) proposed a method based on learning automata for finding optimal policy in general-sum stochastic games in which, the entropy of probability vector for learning automata is used as reward or penalty in each step. Vrancx et al. (2008) showed that a network of independent LA is able to reach equilibrium strategies in Markov games with some ergodic assumptions. Masoumi and Meybodi extended this algorithm with the intermediate rewards to accelerate learning convergence (Masoumi & Meybodi, 2009). Abtahi and Meybodi (2008) proposed two new algorithms based on learning automata that can be effectively used to solve MMDPs and to find the optimal policy.

3. Markov decision process and Markov games

Markov decision process is formally defined as follows (Puterman, 1994; Watkins & Dayan, 1992):

Definition 1. A Markov decision process (MDP) is a tuple (S, A, R, T) where S is a finite state space; A is the space of actions the agent can take; $R : S \times A \rightarrow \Re$ is a payoff function (R(s, a) is the expected payoff for taking action a in state s); and $T: S \times A \times S \rightarrow [0, 1]$ is a transition function (T(s, a, s') is the probability of ending in state s', given that action a is taken in state s).

In a Markov decision process, an agent's objective is to find a strategy (policy) $\pi: S \rightarrow A$ to maximize the sum of discounted expected rewards,

$$v(S,\pi) = \sum_{t=0}^{\infty} \gamma^t E(r_t|\pi, s_0 = s) \quad (1)$$

where, s is a particular state, s_0 indicates the initial state, r_t is the reward at time t, and $\gamma \in [0, 1)$ is the discount factor. In Markov decision problems, the task of the control system is to bring the environment into a goal state by a state-transfer sequence. The cost to reach the goal state, for instance the number of decision step, is determined by the state transfer probability, which is determined by both the environment's stochastic property and the control strategy. Reinforcement learning makes a search in the strategy space to make the initial control strategy converge to an optimal one.

Markov games are a generalization of MDPs to multiple agents and can be used as a framework for investigating multi-agent learning. In the general case (general-sum games), each player would have a separate payoffs. A standard formal definition follows (Thusijsman, 1992):

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

Definition 2. A stochastic game (Markov game) is a tuple $(n, S, A_{1..n}, T, R_{1..n})$, where n is the number of agents, s is a set of states, A_i is the set of actions available to agent i (and A is the joint action space $A_1 \times A_2 \times \ldots \times A_n$), T is a transition function $S \times A \times S \rightarrow [0, 1]$), and r is a reward function for the i_{th} agent $S \times A \rightarrow \Re$.

In a discounted Markov game, the objective of each player is to maximize the discounted sum of rewards, with discount factor γ [0,1). Let π^i be the strategy of player i. For a given initial state s, player i tries to maximize:

$$v(S, \pi^1, \pi^2 \ldots \pi^n) = \sum_{t=0}^{\infty} \gamma^t E(r_t | \pi^1, \pi^2 \ldots \pi^n, S_0 = S) \quad (2)$$

Markov games are categorized based on the agents' rewards into cooperative and non-cooperative games. Non-cooperative games may be classified as competitive games and general-sum games. Strictly competitive games, or zero-sum games, are two-player games where one player's reward is always the negative of the others'. General-sum games are ones where the reward sum is not restricted to zero or any constant, and allow the agents' rewards to be arbitrarily related. However, in fully cooperative games, or team games, rewards are always positively related. In a fully cooperative MG (or team MG) called a multi-agent MDP (or MMDP), all agents share the same reward function. Nevertheless, in general MG (or general-sum MG) there is no constraint on the sum of the agents' rewards and the agents should learn to find and agree on the same optimal policy. However, in a general Markov game, an equilibrium point is sought; i.e. a situation in which no agent alone can change its policy to improve its reward when all other agents keep their policy fixed. The baseline solution concept for general-sum games is the Nash equilibrium (Nash, 1951). Nash *equilibrium* is a strategy profile from which none of the players has any incentive to deviate. The strategies that constitute Nash equilibrium can be stationary strategies. In (Fink, 1964) it is showed that every discounted stochastic game possesses at least, one Nash equilibrium in stationary strategies.

A template for multi-agent Q-learning is presented in (Hu & Wellman, 2003), while an alternative view on Markov Games is taken, i.e. the game can be seen as a sequence of local normal form games. The algorithms takes as input an equilibrium selection function which compute the value function V, given matrix-vector $\vec{Q} = (Q_1 \ldots Q_n)$. If computation of value function V is done according to Eq. (3) the algorithm is called *Nash-Q algorithm* (Hu & Wellman, 2003). Nash-Q values are Q-functions over joint actions ($\vec{a}$) If computation of value function V is done according to Eq. (4), the algorithm is called *Nash-Bargaining algorithm* (Qio et al., 2006).

$$\begin{aligned} &V_i(s) \in NASH_i(Q_1(s, \vec{a}) \ldots Q_n(s, \vec{a})) \\ &NASH_i(Q_i(s)) = \pi^1(s) \ldots \pi^n(s) \cdot Q_i(s) \end{aligned} \quad (3)$$

$$\begin{aligned} &V_i(s) \in NB_i(Q_1(s, \vec{a}) \ldots Q_n(s, \vec{a})) \\ &NB_i(Q_i(s)) = \mathrm{Max}_{\vec{a}}(Q_1(s, \vec{a}) \times \ldots Q_n(s, \vec{a})) \end{aligned} \quad (4)$$

3.1. The Abtahi's et al. algorithm

Abtahi algorithm is proposed for finding optimal policy in multi-agent MDPs (MMDPs) (Abtahi & Meybodi, 2008). In this algorithm, the environment is modeled as a directed graph. The nodes of this graph represent the states and directed edges between nodes represent the actions that result in transition from one state to another. Each node of the graph is then equipped with a learning automaton. The actions of each learning automaton are the outgoing edges of corresponding node. Agents move on this graph and in each state, they get help from corresponding learning automaton to choose a desirable action and move to next state. The probability vector of learning automata included in agent's path will be updated based on the cost of the path traversed by the agent from the initial state to the goal state. When agent j reaches the goal state, the cost of path π_i ($CPAgent(j, \pi_i)$), taken by j is calculated using $t_i(\pi_i)/R_G$, where R_G is the reward of reaching the goal state, and t_i is the time elapsed since starting from start state until reaching the goal state using path π_i. Then the probability vectors of all learning automata along the path are updated according to the learning algorithm. This process will be repeated several times for each agent and gradually, the path taken by each agent will converge to the optimal path or a pre-defined condition will be satisfied. Abtahi's algorithm is illustrated in Fig. 1.

4. Learning automata, learning automata games and entropy

Learning Automaton is a machine that can perform a finite number of actions (Narendra & Thathachar, 1989). Each selected action is evaluated against a probabilistic environment and the result of this evaluation is given to the automata as a positive or negative signal. In choosing its next action, the automata take influence from this signal. The ultimate goal is to make the automata learn to choose the best action from among its possible actions. The best action is one that maximizes the possibility of receiving reward from environment. Fig. 2 illustrates a learning automaton's work in interaction with its environment. Environment can be shown by a triple $E = (\alpha, \beta, c)$ in which $\alpha \equiv \{\alpha_1, \alpha_2 \ldots \alpha_r\}$ is the set of inputs, $\beta \equiv \{\beta_1, \beta_2 \ldots \beta_m\}$ is the set of outputs and $c \equiv \{c_1, c_2 \ldots c_r\}$ is the set of possibilities for being penalized. If β is a set with two members, then environment is of type P. in such an environment $\beta = 1$ is considered as a penalty while $\beta = 0$ would be a reward. In an environment of type Q, $\beta(n)$ can take any discrete values in the interval [0, 1] and in an environment of type S, $\beta(n)$ is a random variable in the interval [0, 1]. c_i is the probability of α_i having an undesirable payoff. In a stationary environment c_i values remain unchanged but in a non-stationary environment these values can change. Learning automata can be classified into two categories: fixed structure learning automata and variable structure learning automata (VSLA) (Narendra & Thathachar, 1989). Variable structure learning automata can be shown by a tuple $\{\alpha, \beta, p, T\}$ where, $\alpha = \{\alpha_1, \alpha_2 \ldots \alpha_r\}$ is the set of actions of the automaton, $\beta = \{\beta_1, \beta_2 \ldots \beta_m\}$ is its set of inputs, $p = \{p_1 \ldots p_r\}$ is probability vector for selection of each action and $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ is the learning algorithm. The following algorithm that is based on Eqs. (5) and (6) is an example of a linear learning algorithm.

Suppose action α_i is selected in n_{th} stage, then

$$\begin{aligned} &p_i(n+1) = p_i(n) + a[1 - p_i(n)] \\ &p_j(n+1) = (1-a)p_j(n) \quad \forall j \quad j \neq i \end{aligned} \quad (5)$$

if favorable response is received from environment, and

$$\begin{aligned} &p_i(n+1) = (1-b)p_i(n) \\ &p_j(n+1) = (b/r-1) + (1-b)p_j(n) \quad \forall j \quad j \neq i \end{aligned} \quad (6)$$

If unfavorable response is received from environment.

In Eqs. (5) and (6), a and b are reward and penalty parameters, respectively. For $a = b$ learning algorithm is called $L_{R\text{-}P}$, for $b \ll a$ it is called $L_{R\text{-}\varepsilon P}$, and for $b = 0$, it is called $L_{R\text{-}I}$. Let a VSLA operate in an *S-Model* environment. A general linear schema for updating action probabilities when action i is performed is given by:

$$\begin{aligned} &p_i(n+1) = p_i(n) + a(1 - \beta_i(n))(1 - p_i(n)) - b\beta_i(n)p_i(n) \\ &p_j(n+1) = p_j(n) - a(1 - \beta_i(n))p_j(n) + b\beta_i(n)\left[\tfrac{1}{r-1} - p_j(n)\right] \\ &\qquad \forall j \quad j \neq i \end{aligned} \quad (7)$$

where, r is the number of possible actions, a and b are reward and penalty parameters, respectively. For more information on learning

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

The Abtahi et al. Algorithm

```
Input : A directed graph corresponding to the problem is created; states and actions are mapped on
        nodes and edges respectively, a, b : learning parameter, M : total training Time.
Initialize :
 for every agents j do
      T(j) =0;
 end for
 for i:= 1 to M do
  s= StartState;
  repeat
      Action1 = agent1.action();
      Action2 = agent2.action();
      NewState = ChooseNewState(s, Action1, Action2) ;
    if NewState is goal then
       for each agent i do
        CPAgent(i,πi)= ti(πi)/RG
       end for
       for each agent j do
         if CPAgent(j,πi) < T(j), then path πi is rewarded
         else path πi is penalized
        end if
        T(j)←T(j) +(CPAgent(j,πi )-T(j)) / i
       end for
    end if
     s ←newState,
  until agent j reaches the goal state
 end for
```

Fig. 1. The Abtahi et al. algorithm.

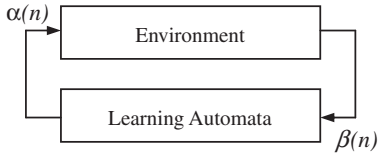

Fig. 2. The interaction between learning automata and environment.

automata, the reader may refer to (Thathachar & Sastry, 2002; Thathachar & Sastry, 2004).

4.1. Learning automata games

Automata games were introduced to see if automata could be interconnected in useful ways so as to exhibit group behavior that is attractive for either modeling or controlling complex system (Thathachar & Sastry, 2004). A play $a(t) = (a_1(t). . .a_n(t))$ of n automata is a set of strategies chosen by the automata at stage t, such that $a_j(t)$ is an element of the action set of the jth automaton. Correspondingly the outcome is now also a vector $\beta(t) = (\beta_1(t). . .\beta_n(t))$. At every time-step, all automata update their probability distributions based on the responses of the environment. Each automaton participating in the game operates without information concerning the number of other participants, their strategies, actions or payoffs. In zero-sum games, the $L_{R\text{-}I}$ scheme converges to the equilibrium point if it exists in pure strategies, while the $L_{R\varepsilon P}$ scheme can arbitrarily close approach a mixed equilibrium (Lakshmivarahan & Narendra, 1981). In general non zero-sum games it is shown that when the automata use a $L_{R\text{-}I}$ scheme and the game is such that a unique pure equilibrium point exists, convergence is guaranteed (Sastry et al., 1994). In cases where the game matrix has more than one pure equilibrium, which equilibrium is found depends on the initial conditions.

4.2. Entropy in learning process

Entropy is a significant concept in the thermodynamics, representing the degree of disorder in a thermodynamic system (Lieb & Yngvason, 1999). Shannon has introduced this concept into the information theory, by the name of *"information entropy"*. Entropy, in its basic, indicates a measure of uncertainty rather than a measure of information. More specifically, the information entropy is a case of the entropy of random variables defined as follows (Costa, Goldberger, & Peng, 2002; Dianhu, Shaohui, & Xiaojun, 1997):

$$H(x) = -\sum_{x\in\chi} P(x)\log(P(x)) \tag{8}$$

where, X represents a random variable with set of values χ and probability mass function $P(x)$. Entropy is always a positive value and can change bases freely as $H_b(X) = \log_b(a)\cdot H_a(X)$. Entropy measures the uncertainty inherent in the distribution of a random variable. The entropy of random variables has problem-dependent meanings in different applications. In this paper, entropy is introduced into reinforcement learning as a measure of learning process in discrete state space in which each state s_i is assigned a set of decision probability values. For each state s_i there is a corresponding random variable X_i representing the probability distribution of decision probability values in the state. In the learning process, the entropy of X_i represents the uncertainty of the decision under state s_i. In any particular state s_i a local entropy which represents the uncertainty of the decision in that state is defined as follows (Zhuang, 2005):

$$H_{local}(s_i) = -\sum_{j=1}^{N} P_j(s_i)\log(P_j(s_i)) \tag{9}$$

where, N is the number of the elements in the action set and $P_j(s_i)$ is the probability to select the j-th action under state s_i. The local entropy can only represent the uncertainty of the decision for a single state. According to all states' local entropy and to overall evaluate the uncertainty of the whole system, the *general entropy* concept is introduced as the average of all the entropies (Zhuang & Chen, 2008). In reinforcement learning, the process of optimization increases the probability values of selecting optimal actions, i.e. the initial random strategy will converge to a specific optimal strategy. Hence, the uncertainty of both the strategy and the general entropy

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

will decrease. Therefore, the entropy can represent the degree of convergence in the learning process.

5. Stigmergetic algorithms in multi-agent system

In this section, we describe the basic elements of stigmergetic model in multi-agent system. The term of stigmergy which is introduced for the first time by entomologist Paul Grass'e, is defined as a class of mechanisms that mediate animal-to-animal interactions through the environment (Grasśe, 1959; Theraulaz & Bonabeau, 1995). This concept provides a relatively simple framework for agent communication and the coordination of multi agent systems. Several approaches have been proposed to apply stigmergy to multi-agent systems. A frequently used method is to let agents communicate by using artificial pheromones, which is called *sign-based* stigmergy (Kollingbaum, 2001). Agents can observe and change the local pheromone values that guide action selection. Ant Colony Optimization is an example of this type of algorithm (Dorigo & Stützle, 2004). Other algorithms are based on behavior of termite and wasp nest building (Theraulaz & Bonabeau, 1995) or ant brood sorting (Holland & Melhuish, 1999). In these systems, an individual's actions (e.g. building, depositing dirt) modify the local environment and cause reactions of other workers (i.e. moving brood, building more, etc.). In most of the stigmergetic algorithms, a set of common elements can be isolated (Vrancx et al., 2007):

- The environment is subdivided in a number of discrete locations, which agents can visit.
- Each location contains a local state that visiting agent can access and update.
- Agents can perform actions in the locations that they visit. Based on the local state, an agent determines the probability of an action.
- An interconnection scheme between locations is defined, allowing agents to travel between locations.

One of the simplest stigmergetic algorithms is ACO that is proposed to study the basic properties of ant colony optimization algorithms (Dorigo & Stützle, 2004). The first algorithm was aiming to search for an optimal path in a graph; based on the behavior of ants seeking a path between their colony and a source of food. The goal of the ACO algorithm is to find the minimum-cost path between two nodes in a weighted graph. Each graphs' edge, connecting node with node, has a variable associated with it. This variable represents the amount of pheromone on the edge, while moving ants add pheromone to the edges on their way. They move according to a probabilistic decision policy based on the amount of pheromone trail they "smell" on the graph's edges. The algorithm uses a colony of very simple ant-like agents. These agents travel through the graph starting from the source node, until they reach the destination (or goal) node. In all nodes, a pheromone value is associated with the outgoing edges. When an agent arrives in the node, it reads these values and uses them to assign a probability to each edge. This probability is used to choose an edge to follow to the next node. When all agents reach the goal state, they return to the source node and the pheromone value on each edge (ij) i.e. τ_{ij} is updated using the following formula:

$$\tau_{ij} \leftarrow \rho\tau_{ij} + \Delta\tau_{ij} \tag{10}$$

The pheromone update described above consists of two parts. First the pheromones on the edges are multiplied with a factor $\rho \epsilon [0, 1]$. This simulates pheromone evaporation and prevents the pheromones from increasing without bound. After evaporation, a new amount of pheromone $\Delta\tau$ is added. The paths found by the ants determine the amount of new pheromones. Edges that are used in lower cost paths receive more pheromones than those used in high cost paths. By using this system, the ant agents are able to coordinate their behavior until all agents follow the same shortest path. Each agent builds up its own path, and the feedback it receives is based only on the cost of this path. The paths followed by other agents do not influence this feedback.

In complex problems, the agents can have different goals and directly influence each other's reward. Examples of these algorithms can be found in *multi-pheromone* algorithms. These systems use not one, but several pheromone gradients to guide agents. One such system was proposed in (Nowe, Verbeeck, & Vrancx, 2004; Panait & Luke, 2004) to let different colonies of agents find disjoint paths in a network.

5.1. Using learning automata as a model for stigmergetic interactions

The first use of learning automata as a model for stigmergetic communication was introduced in (Nowe & Verbeeck, 2002). One limitation of this approach is that all agents must share the same goal, to avoid that the LAs are updated using conflicting responses. The idea behind this model is to move decision making from the agents to the local environment states so that each local state contains one LA. When an agent visits a location, it activates the LA that lodges in that location. This automaton then decides the action the agent should take in that location. Transition to the next location triggers learning automata from that location to become active and take some action. Agents themselves can be viewed as dummy mobile agents, which walk around in the graph of interconnected locations, make local LA active, and bring information so that the LA involved can update their local state. The learning automaton LA_i in state i is activated, but will not be informed about the immediate reward $r_{ij}(a_k)$ yielded from its action a_k in transition from state i to j. Instead, when state i is visited again, LA_i receives two parts of data: the cumulative reward from the beginning of the process up to then and the current global time. Using these data, LA_i calculates the reward received since the last visit of state i and the corresponding elapsed global time. Then, the input to LA_i is calculated using the following equation:

$$\beta^i(t_i + 1) = \frac{\rho^i(t_i + 1)}{\eta^i(t_i + 1)} \tag{11}$$

where, $\rho^i(t_i + 1)$ is the cumulative total reward generated for action a_k in state i and $\eta^i(t_i + 1)$ the cumulative total time elapsed. When comparing the pheromone update rule given in Eq. (10) with the update scheme of the LA model, see Eq. (7), the shared attributes are obvious. In fact, the trail update rule of Eq. (10) is actually a L_{R-P} update. The pheromone trail in ACO is updated with an amount that depends on the total length or cost of the path (Vrancx et al., 2007). So depending on the quality of the visited path, the pheromone trail is rewarded or penalized. In fact, the dummy agents play the role of the ants here. The LA model described above is based on the LA algorithm introduced by Wheeler and Narendra to solve Markov Decision Problems (Wheeler & Narendra, 1986). The only difference is that here multiple dummy agents co-exist, so that multiple automata can be activated at the same time. Vrancx et. al extend the model to allow agents to have different goals (Vrancx et al., 2007). Therefore, in the extended model, each location contains one or more learning automata, corresponding to different goals that the agents try to achieve. When an agent visits a location, it activates the learning automaton corresponding to its goal. For each agent, one LA in its current location is active at each time step and the transition to the next location triggers an automaton from that location to become active and take some action.

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

6. The proposed method

In this section, a learning automata based multi-agent systems in which the concepts of stigmergy and entropy are imported is proposed. The proposed multi-agent systems will be used to find optimal policies in MG. In a MG, the selected actions are the results of the joint action of all agents while rewards and state transition depend on these joint actions. In the proposed method, it is assumed that the environment consists of a set of discrete states $S = \{s_1, s_2 \ldots s_m\}$. For each state s_i in the environment of the game and for each *real* agent such as *k*, *k: 1…n* (can be corresponding to different goals that agents are trying to achieve), an *S-model* variable structure learning automaton LA_k^i is placed that tries to learn the optimal action probabilities in those states. The number of its adjacent states (neighbors) determines the number of actions of each learning automaton in each state and every joint action corresponds to a transition to an adjacent state. The state with the highest payoff value is the goal state. *Real* agents start from "*start state*" and move toward "*end* (*or goal*) *state*". The selected actions of the learning automata in each state determine the next state of the agents. In proposed method, it is assumed that in addition to *real* agents, a number of dummy mobile agents (maximum as $(m-1)*(n)$) can be viewed that walk around in the graph of interconnected states and make local *LA* (corresponding to each *real agent*) active and bring information so that the LA involved can update their local state.

At first, all of the learning automata in all states choose their actions with equal probabilities. Reward is received when the action taken by an automaton leads to a goal state. Otherwise, the entropy of the probability vector for the learning automata of the next state and immediate reward will be used to determine reward or penalty for them. Entropy of the action probability vector measures the degree of uncertainty that the next state of learning automaton encounters when choosing an action. High value of entropy indicates that learning automaton has no information about where the goal state is and chooses its action randomly (exploration). On the contrary, the low value of entropy indicates that the learning automaton has useful information about where the goal state is and, therefore, chooses its actions in such a way that it leads the agent to the goal state with higher probability (exploitation). If $p = \{p_1, p_2 \ldots p_r\}$ is the action probability vector of a learning automaton with *r* actions, then the entropy of this probability vector for each state *s* and agent *i* will be computed according to the Eq. (12). $p_j^i(S)$ denotes the probability vector of actions for LA_i^S.

$$H_i(s) = \sum_{j=1}^{r} p_j^i(s) \mathrm{Log}\left(p_j^i(s)\right) \tag{12}$$

Entropy has its maximum value when all the actions have equal probabilities of selection and has value zero (its minimum) when the action probability vector is a unit vector. In order to be able to use entropy as a reinforcement signal for *S-Model* variable structure learning automata, the entropy needs to be rescaled in the range of [0, 1]. Suppose that agent *i* is in state *s* and its learning automaton, that is LA_i^S, leads the agent to state *s'*. In this case, reinforcement signal is determined using Eq. (13). β_i^S is then used to update the probability vector of LA_i^S according to the Eq. (7).

$$\beta_i^s = \begin{cases} 0 & \text{if } s' = \text{EndState (reward is max)} \\ r(i) * H_i(s') / (\mathrm{Max}(H_i(s'))^K & \text{else} \end{cases} \tag{13}$$

where, $r(i)$ is immediate reward for agent *i*, which is further normalized to ensure that it lies between zero and one. *Max* (H_i (s')) is maximum entropy in state *s'* for agent *i* defined by Eq. (14) and *K* is a parameter of the method which controls the degree of exploration/exploitation in the environment. The value of parameter *K* should be carefully selected by trial and error in conjunction with the application and requirements of the problem. High values of *K* result in low values of β, implying that the learning automata get rewards most of the time. This means more exploration in the environment. On the contrary, lower values of *K* yield higher values of β. This causes the learning automata to be penalized more. This greater penalization means that even desirable states do not receive due rewards. In the proposed method, learning process stops when a *real* agent reaches its goal or when a fixed number of iterations are passed. The proposed algorithm is presented in Fig. 3.

$$\mathrm{Max}(H_i(s)) = \sum_{j=1}^{r(s)} \frac{1}{r(i)} \mathrm{Log}\left(\frac{1}{r(i)}\right) = \mathrm{Log}_2^{r(s)} \tag{14}$$

7. Experiments

The Markov game we consider is a sequential grid-world game, which is proposed by Hu and Wellman (2003). This game has different varieties all of which are two-player non-cooperative (or general-sum) games. To evaluate the performance of the proposed method they are tested on two different two-player general sum games. In the rest of this section, we first describe these two grid games and then report the results of experimentations.

7.1. Grid game environment as a Markov game

Two types of Grid Game are illustrated in Fig. 4. The first one, i.e. GridGame1 (GG1) is a version of Chicken game that has several states and one goal. The second one, i.e. GridGame2 (GG2) is a multiple state coordination game with two agents and two goals. In GG1, two agents start from a corner of the page and try to reach goal with the least possible number of moves. The two grid-world games can be easily modeled as Markov games. Players' actions are defined as four actions in four different directions, namely Up, Down, Left, Right. State space set is defined as $S = \{s \mid s = (l_1, l_2)\}$, in which each state $s = (l_1, l_2)$ indicates the coordinates of agents 1 and 2. Agents cannot take the same coordinates at the same time. In other words, if both agents try to move to the same square, both of their moves will fail. If agents move to two different non-goal positions, both receive zero rewards and if one reaches the goal position, it receives 100 units of reward. However, if they collide with each other both receive one unit of punishment and stay in their previous position. In both games, the state transitions are deterministic, i.e. the next state is uniquely determined by the current state and the joint action of the agents. In this game, agents are assumed not to know the goal position and the other agent's reward functions. Agents choose their actions simultaneously and can only know about the previous moves of the other agents and their own current state.

A path in these games represents sequences of actions from the starting to the end position. In game terminology, such a path is called a policy or strategy. The shortest path, not interfering the path taken by the other agent, is called the optimal policy or *Nash path*. Fig. 5 illustrates the optimal solution to games GG1 and GG2. To find the optimal policy using the proposed method, one learning automaton is employed for each state (l_1, l_2) for each real agent. Each agent moves from one state to the new state based on the joint action provided that the two agents do not collide. Agents move simultaneously and both can view the new state, immediate rewards and the moves done by the other agent.

7.2. Experimental results and discussion

In order to evaluate the performance of the proposed method several experiments have been conducted whose results are re-

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

The proposed Algorithm (MG, a, b, K, M, N)

Inputs: a, b: reward and penalty parameters for each LA, **K**: exploration parameter, **M**: total training time, **N**: number of states contain dummy agents

Initialize : *In each state, a Learning Automaton of type S for each agent is placed. The set of actions of this LA is the set of permissible movements to other states.*

1 **for** all states s, agents *i* **do**
2 P(s, i) =1/number of permissible actions for agent *i*
3 **end for**
4 *Generate N random number r_i between 0, N-1 and add to Set dstate.* ***dstate*** *=\{ s_{r1}, s_{r2},... s_{rN} \} // states contain dummy agents .*
5 **for** episode =1 to M **do** //Main Loop
6 OldState = StartState; //random or fixed
7 **Repeat**
8 JointAction =∅;
9 **for** every ***real agent*** *i* **do**
10 Action = SelectAction (P(OldState, agent *i*)
11 LastAction (OldState,agent *i*)=Action ;
12 JointAction = JointAction ∪ action
13 **end for**
14 NewState= GetNextState (OldState, JointAction)
15 **for** every **real agent** *i* **do**
16 Reward (OldState, agent *i*)=Reward (OldState, agent i) +GetReward (OldState, jointAction)
17 NormalReward=Normalize (Reward (OldState, agent *i)*
18 **if NewSate** = GoalState **then**
19 Feedback(agent *i* ,OldState) = 0
20 **else Feedback** (agent *i*, OldState)= NormalReward* $H_{agenti}^{newState} / Max(H_i(NewState))^K$
21 **end if**
22 UpdateProbabilities (P(OldState, agent *i*), LastAction (OldState, agent *i*), Feedback (agent *i*, OldState))
23 **end for**
24 **for** each state **ps in dstate** except OldState ***do concurrently***
25 **for** every **dummy agent** *j* in ***ps*** **do**
26 Action = SelectAction (P (***ps***, agent *j*))
27 JointAction = JointAction ∪ action
28 LastAction (***ps***,agent *i*)=Action ;
29 **end for**
30 NewPS= GetNextState (***ps***, JointAction)
31 **for** every ***dummy agent j*** **do**
32 Reward (***ps***, agent ***j***)=Reward (***ps*** ,agent ***j***) + GetReward (***ps***, jointAction)
33 NormalReward= Normalize (Reward (***ps***, agent *i)*
34 **if** NewPS = GoalState **then**
35 Feedback(agent ***j*** , *ps*) = 0
36 **else** Feedback(agent ***j***, *ps*)= NormalReward * $H_{agentj}^{NewPS} / Max(H_j(newPS))^K$
37 **endif**
38 UpdateProbabilities (P(***ps***, agent ***j***), LastAction(*ps*,agent ***j***), Feedback (agent ***j***, ***ps***))
39 **end for**
40 **end for**
41 OldState = NewState
42 **Until** the goal state for *real agent* is reached or a fixed number of iterations are passed
43 **end for** episode

Fig. 3. The proposed algorithm.

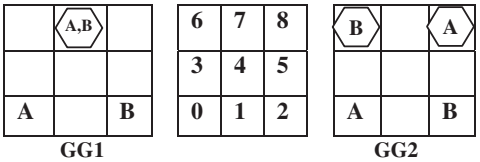

Fig. 4. Two types of grid world game and illustration of game coordinates.

ported below. The experiments are conducted for two cases: online learning and offline learning. In experiment 1, in order to study the role of entropy and stigmergy on the performance of the proposed algorithm, we compare the proposed algorithm with three other algorithms: (i) basic algorithm: an algorithm which uses learning automata without the help of dummy agents and entropy to search for the optimal policy, (ii) Algorithm 1: obtained by importing only the concept of stigmergy into the basic algorithm via using dummy agents in the basic algorithm, and (iii) Algorithm 2: obtained by importing only the concept of entropy into the basic algorithm. In other experiments, we only study the proposed algorithm. In the remaining part of this section, we first briefly explain the basic algorithm, Algorithm 1, and Algorithm 2 and then present the results of experimentations.

7.2.1. The basic algorithm

In the basic algorithm, for each state s_i ($1 \leqslant i \leqslant m$, *m: number of states*), in the environment of the game and for each agent *k*, an S-model variable structure learning automaton LA_k^i is defined. Automaton LA_k^i tries to learn the optimal action of agent *k in state* s_i. The number of actions of automaton LA_k^i , which is assigned to agent *k*, is the number of states adjacent to state *i*. The selected actions of the learning automata in each state determine the next state of the agents that is every joint action corresponds to a transition to an adjacent state. At first, all of the learning automata in all states choose their actions with equal probabilities. The action taken by an automaton is rewarded when that action leads to a goal state. Otherwise, the immediate reward will be used to reward or penalize the action. The process of choosing actions by automata

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

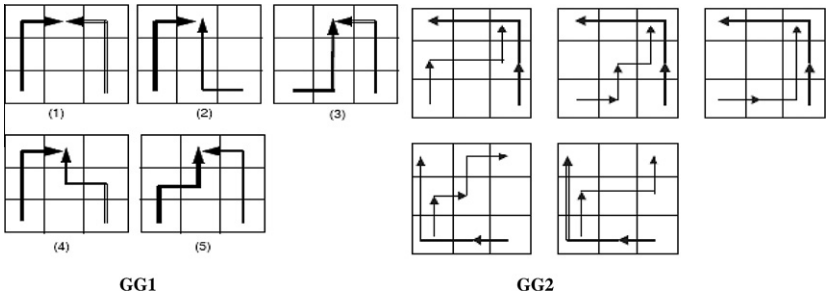

Fig. 5. Optimal solutions for two grid world games.

and then rewarding or penalizing them are repeated until at least one of the agents reaches its goal or when a fixed number of iterations are passed. The basic algorithm is given in Fig. 6. Algorithm 1 which is presented in Fig. 7 is obtained by importing only the concept of stigmergy into the basic algorithm via using dummy agents in the basic algorithm (as done in the proposed algorithm). Algorithm 2 is the basic algorithm in which the concept of entropy has been used (as used in the proposed algorithm). Algorithm 2 is obtained from Algorithm 1 by omitting lines 24–40 from Algorithm 1.

7.2.2. On line learning performance

In online learning, an agent is evaluated based on rewards obtained while it learns. For all experiments, each reported value is obtained by averaging over 200 runs and it is assumed that at the beginning of an episode, each real agent starts from position (0 2). In experiments 2 through 8, the number of dummy agent is set to a maximum which is $(m-1)n$. The following list gives the purpose of conducting each of the experiments reported in the remaining part of this section.

Experiment 1: to study the role of the entropy and stigmergy in the proposed method.
Experiment 2: to study the impact of the number of dummy agents on the amount of reward received by agent 1 during an episode.
Experiment 3: to study the impact of learning parameter a on the amount of reward received by agent 1 during an episode.
Experiment 4: to study the impact of exploration parameter *K* on the amount of reward received during an episode.
Experiment 5 to study the impact of learning parameter a on the number of agent collisions during an episode.
Experiment 6: to study the impact of exploration parameter *K* on the number of collisions during an episode.
Experiment 7: to study the relationship between the entropy of action probability vector and the amount of reward received.
Experiment 8: to study the changes in the general entropy during playing the game.
Experiment 9: to study the role of parameter *K* in balancing exploration with exploitation.
Experiment 10: to compare the proposed algorithm with three existing algorithms in terms of the amount of reward received.

7.2.2.1. Experiment 1. In this experiment, we study the role of the entropy and stigmergy in the proposed algorithm. For this purpose, we compare the proposed algorithm with three algorithms: (i) basic algorithm: (ii) Algorithm 1 and (iii) Algorithm 2 in terms of the total reward received by agent 1 per episode. The learning parameter *a*: is set to 0.05 while parameter *K* in Eq. (13) is set to one.

The basic algorithm (Γ, a, b, M)

```
    Inputs: Markov Game Γ, reward and penalty parameters for each LA a, b, total training time M
    Initialize
1   for all states s, agents i
2     Initialize P(s,i) =1/number of permissible actions for agent i
3   end for
4   for episode =1 to M do
5     OldState = StartState;    //random or fixed
6    Repeat
7       JointAction =∅;
8      for each agent i do
9         Action = Select Action based on probability P (OldState, agent i)
10         LastAction (OldState,agent i)=Action ;
11         JointAction = JointAction ∪ Action
12      endfor
13     NewState = GetNextState (OldState, JointAction)
14     for every agent i do
15       Reward (OldState,agent i)=Reward(OldState, agent i) +GetReward(OldState, jointAction)
16       Ri= Normalize(Reward (OldState, agent i))
17           if  NewSate = End (or Goal) State then
18              Feedback(agent i, OldState) = 0
19           else
20              Feedback(agent i, OldState)= Ri;
21           endif
22        Update (P(OldState, agent i), LastAction(OldState, agent i), Feedback(agent i, OldState))
23      Endfor
24     OldState=NewState
25   Until the Goal state is reached or a fixed number of iterations are passed
26  endfor
```

Fig. 6. The basic algorithm.

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

Algorithm 1 (Γ, a, b, K, M, N)

Inputs: a, b: reward and penalty parameters for each LA, **K**: exploration parameter, **M**: total training time, **N** :number of states contain dummy agents
Initialize : *In each state, a Learning Automaton of type S for each agent is placed. The set of actions of this LA is the set of permissible movements to other states.*

```
1   for all states s, agents i do
2   P(s, i) =1/number of permissible actions for agent i
3   end for
4   Generate N random number r_i between 0, N-1 and add to Set dstate. dstate ={ s_r1, s_r2,... s_rN} // states contain
    dummy agents
5  for episode =1 to M do //Main Loop
6      OldState = StartState;   //random or fixed
7   Repeat
8      JointAction =∅;
9      for every real agent i do
10        Action = SelectAction (P(OldState, agent i )
11        LastAction (OldState,agent i)=Action ;
12        JointAction = JointAction ∪ action
13     end for
14    New State= GetNextState (OldState, JointAction)
15    for every real agent i do
16        Reward (OldState, agent i)=Reward (OldState, agent i) +GetReward (OldState, jointAction)
17        NormalReward=Normalize (Reward (OldState, agent i)
18        if NewSate = GoalState then
19          Feedback(agent i ,OldState) = 0
20        else Feedback (agent i, OldState)= NormalReward
21        end if
22        UpdateProbabilities ( P( OldState, agent i), LastAction (OldState, agent i), Feedback (agent i, OldState))
23     end for
24    for each state ps in dstate except OldState do concurrently
25     for every dummy agent j in ps do
26        Action = SelectAction (P (ps, agent j ))
27        JointAction = JointAction ∪ action
28        LastAction (ps,agent i)=Action ;
29     end for
30     NewPS= GetNextState (ps, JointAction)
31     for every dummy agent j do
32        Reward (ps, agent j)=Reward (ps ,agent j) + GetReward (ps, jointAction)
33        NormalReward= Normalize (Reward (ps, agent i)
34        if NewPS = GoalState then
35            Feedback(agent j , ps) = 0
36        else Feedback(agent j, ps)= NormalReward
37        endif
38      UpdateProbabilities (P(ps, agent j), LastAction(ps,agent j), Feedback (agent j, ps))
39     end for
40     end for
41   OldState = NewState
42  Until the goal state for real agent is reached or a fixed number of iterations are passed
43 end for //episode
```

Fig. 7. Algorithm 1.

Fig. 8 illustrates the results of this experiment for both games GG1 and GG2. As it is shown, introducing each of the concepts of entropy or stigmergy into the basic algorithm in which neither entropy nor stigmergy is taken into consideration improves the performance. Using Algorithm 2, which uses only entropy, produces better results in terms accuracy and the speed of learning as compared to Algorithm 1, which uses only stigmergy. The best results are achieved when the proposed algorithm is used in which both entropy and stigmergy.

7.2.2.2. Experiment 2. This experiment is conducted to study the impact of the number of dummy agents on the amount of reward received by agent 1 during an episode when the proposed algorithm is used. For this purpose, we plot the reward per episode for different number of dummy agents: {$(m-1)n$, $((m-1)/4)n$, $((m-1)/8)n$ *and zero*} where m is the number of states and n is the number of agents in the game. The learning parameter a: is set to 0.01 while parameter K in Eq. (13) is set to one. Fig. 9 illustrates the results of this experiment for both GG1 and GG2. From the result, it is evident that the number of dummy agent has a large effect on the performance of the proposed algorithm. Although the increase in the number of dummy agents leads to improvement of performance, time complexity increases relatively as well. When m is set to zero the proposed algorithm uses only the entropy concept.

7.2.2.3. Experiment 3. This experiment is conducted to study the impact of learning parameter a of L_{R-I} *learning* algorithm on the amount of reward received by agent 1 during an episode when the proposed algorithm is used. For this purpose, we plot the reward per episode for different values of learning parameter a: 0.005, 0.01, 0.05, 0.1 while parameter K in Eq. (13) is set to one and the number of *dummy agent* is maximum. Fig. 10 illustrates the results of this experiment for both GG1 and GG2. As the result illustrates clearly the choice of value for parameter a has a large effect on the performance of the proposed algorithm. In this experiment the best result is obtained for GG1and GG2 when a is set to 0.05.

7.2.2.4. Experiment 4. In this experiment, we study the impact of exploration parameter K on the amount of reward received by agent 1 during an episode when the proposed algorithm is used.

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

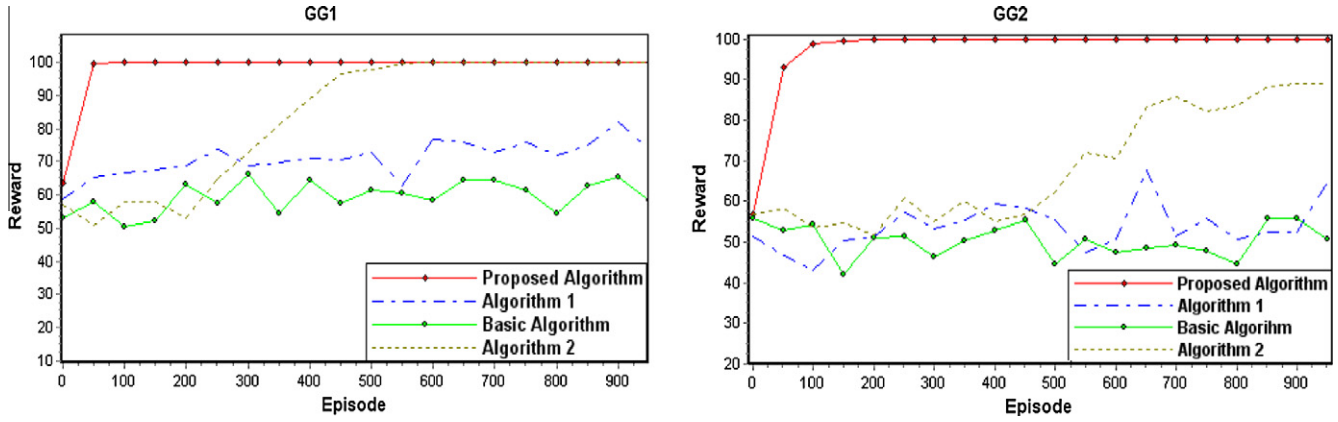

Fig. 8. The role of the entropy and stigmergy concept in GG1 and GG2.

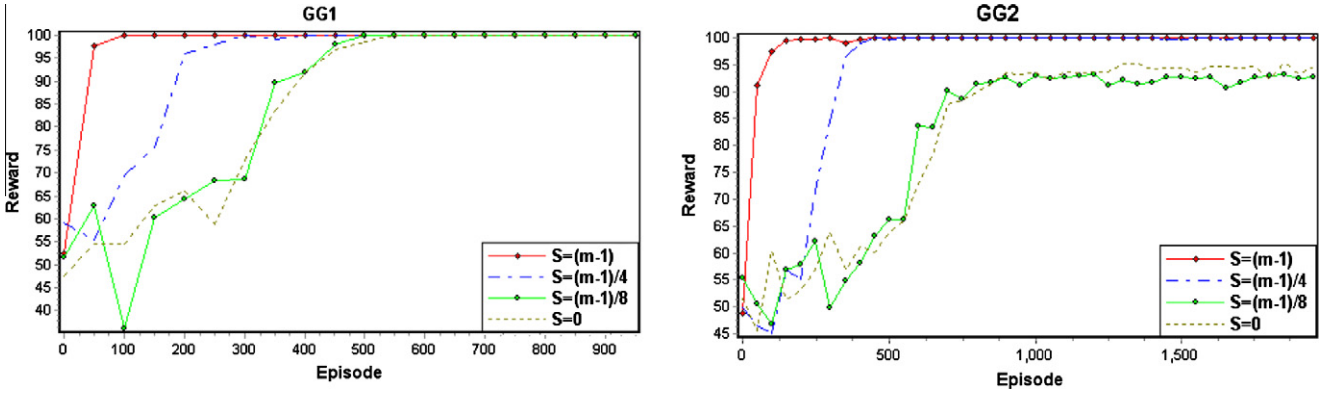

Fig. 9. The impact of the number of *dummy* agents on the amount of reward received by agent 1 in GG1 and GG2.

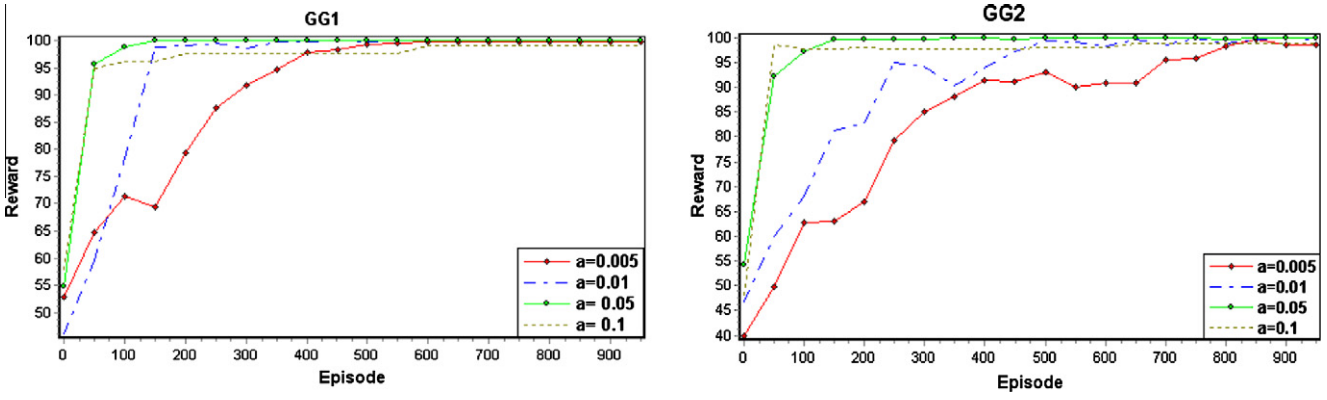

Fig. 10. The impact of learning rate on the reward received by agent 1 in GG1 and GG2.

As with previous experiment, the reward received per episode for different values of exploration parameter K is presented. We ran algorithm for different values of parameter K: 1, 3, 5 and 10 for both GG1 and GG2. Parameter a is chosen in such a way that the best result be obtained. Fig. 11 shows the results of this experiment. According to the results, the proposed algorithm on both games performs better when K has a small value.

7.2.2.5. Experiment 5. This experiment is conducted to study the impact of learning parameter a of L_{R-I} learning algorithm on the number of agent collisions during an episode. To do this we plot the number of collisions obtained by averaging over 200 runs during an episode for different values of learning parameter a: 0.005, 0.01, 0.05, 0.1 while parameter K is set to 1. Fig. 12a and b illustrate the results of this experiment for both GG1 and GG2. The results show that the choice of value for parameter a has a large effect on the number of agent collision during the process of learning. As the figures show the best result is obtained for both GG1 and GG2 when $a = 0.05$.

7.2.2.6. Experiment 6. In this experiment, the impact of exploration parameter K on the number of collisions during an episode is studied. For this purpose, as with previous experiment, the number of collisions obtained per episode for different values of exploration parameter K is presented. We ran algorithm for different values of parameter K: 1, 3, 5 and 10 for both GG1 and GG2. Parameter

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

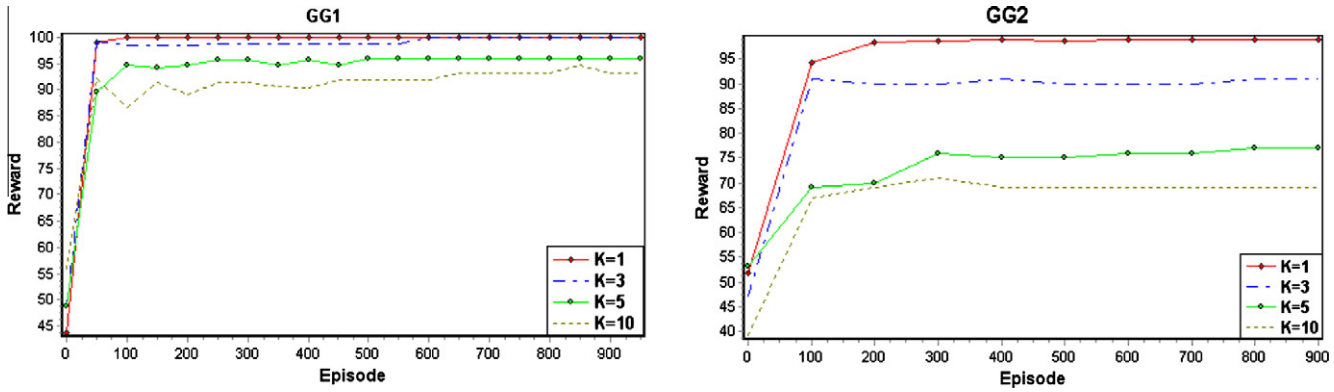

Fig. 11. The impact of parameter K on the performance of the proposed algorithm.

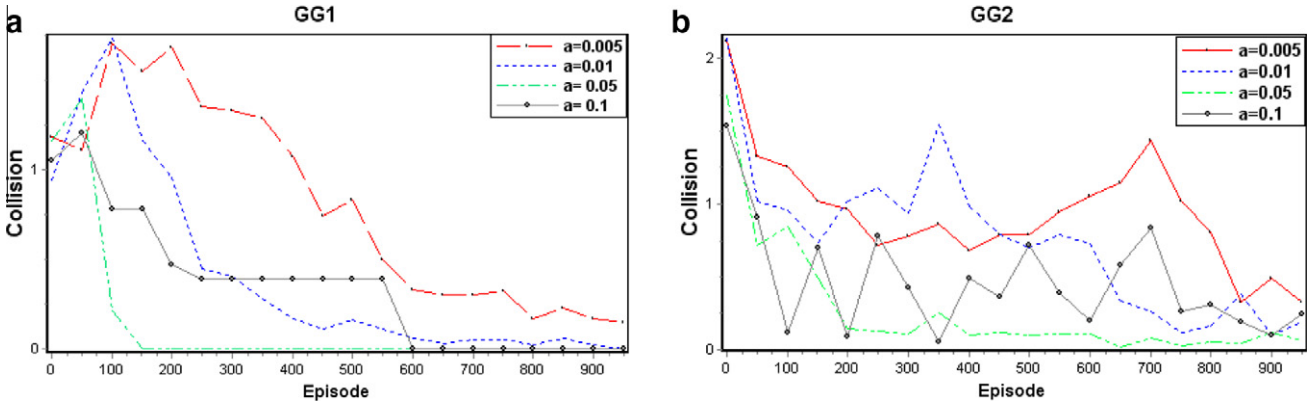

Fig. 12. The impact of learning rate on the number of collisions during an episode in GG1 and GG2.

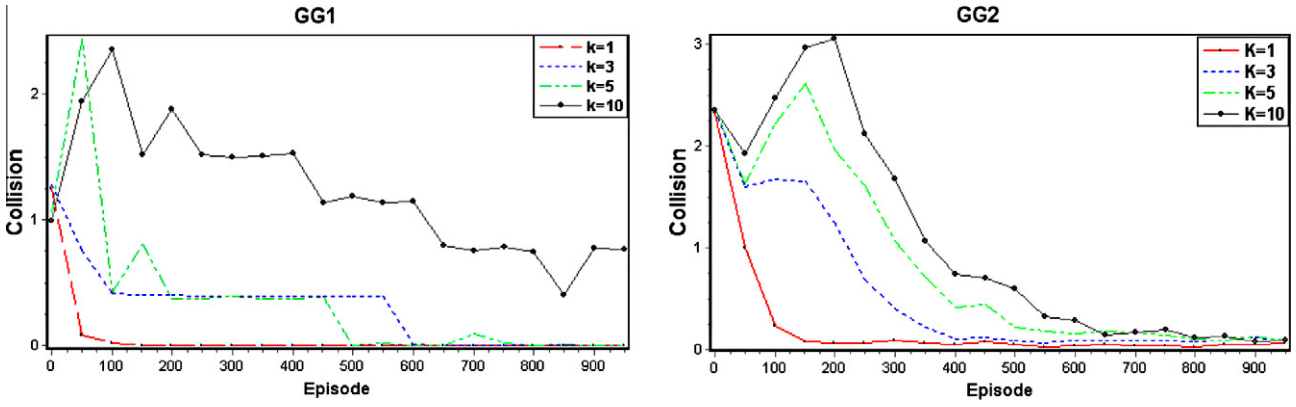

Fig. 13. The impact of learning rate on the number of collisions during an episode in GG1 and GG2.

a is chosen in such a way that the best result be obtained. As the Fig. 13 show the minimum number of collisions is obtained for both GG1 and GG2 when $K = 1$.

7.2.2.7. Experiment 7. In this experiment whose results are given in Fig. 14 we study the relationship between the entropy of action probability vector of learning automaton of agent 1 in state (0 2) and the amount of reward received by agent 1 during an episode for both GG1 and GG2. For this experiment parameters a and K are chosen such that the best results be obtained. As it is shown as the entropy of the action probability vector of learning automaton (or the uncertainty of the strategy) decreases the amount of reward received increases as the process of learning proceeds.

7.2.2.8. Experiment 8. This experiment is conducted to study the changes in the general entropy during the game for the proposed algorithms. For this propose we plot general entropy per episode while parameter *a, K are* chosen in such a way that the best result be obtained. Fig. 15 summarizes the results of this experiment for both GG1 and GG2. The results show, as expected, the general entropy (or the uncertainty of the strategy) is decreasing during the learning process.

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

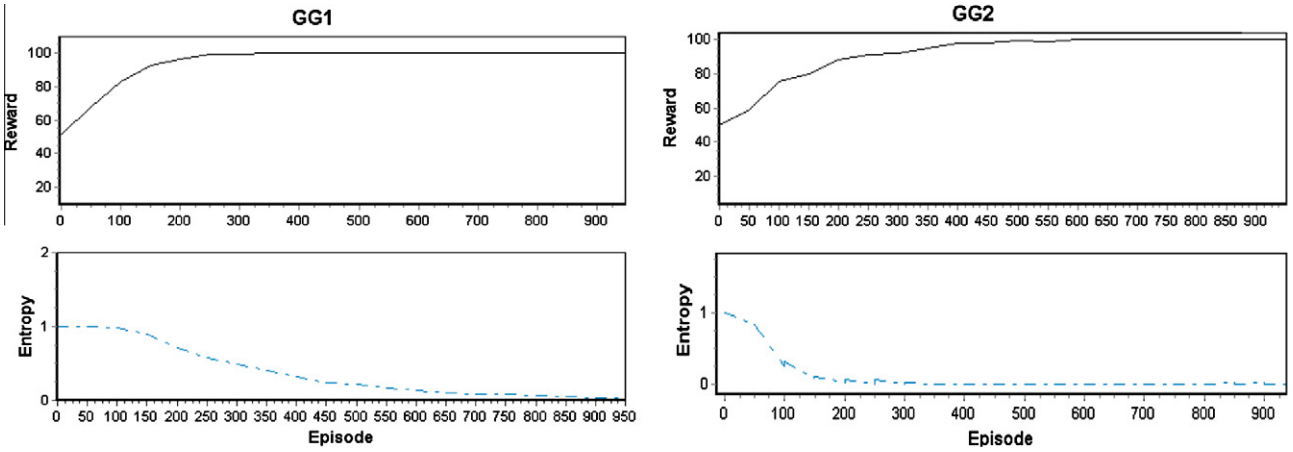

Fig. 14. Entropy of agent 1 in state (0 2) and the reward received by agent 1 for GG1 and GG2.

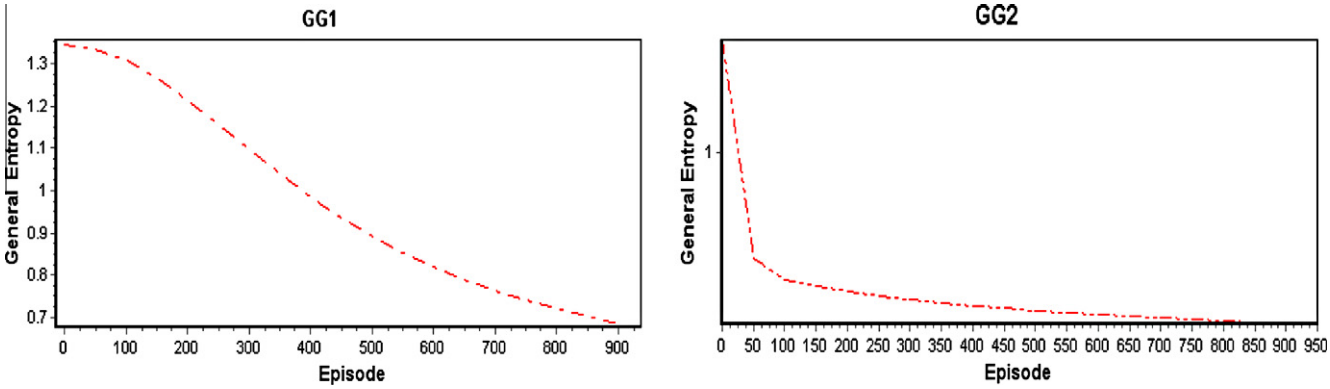

Fig. 15. General entropy for GG1 and GG2.

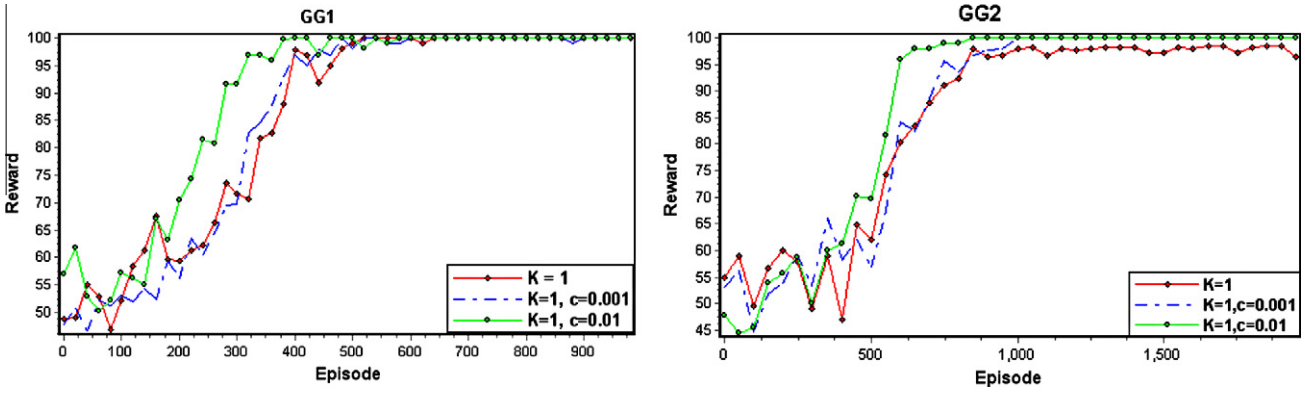

Fig. 16. Adaptatation of parameter *K*.

7.2.2.9. Experiment 9. This experiment is designed to study the role of parameter *K* in balancing exploration with exploitation to yield improved performance. At the beginning of the algorithm parameter *K* is selected to be small. As time passes, *K* is increased by a constant *c* causing the agents to perform more exploitation of their own knowledge. Fig. 16 shows the result of this experiment for both GG1 and GG2. As it is seen when the agents start by exploration and gradually shift to greater exploitation (by increasing the value of *K*) a performance improvement is achievable.

7.2.2.10. Experiment 10. This experiment is conducted to compare the proposed algorithm with three existing algorithms: (i) NASH-Q algorithm (Hu & Wellman, 2003) (ii) Nash Bargaining algorithm(Qio et al., 2006) and (iii) Abtahi's algorithm (Abtahi & Meybodi, 2008) in terms of the amount of reward received by agent 1 per episode. This experiment is conducted for both games GG1 and GG2. For Nash Bargaining and NASH-Q algorithms we set $\alpha = 0.1$ and $\gamma = 0.99$. For the proposed algorithm and Abtahi's algorithm we set $a = 0.05$, $b = 0$ and $K = 1$. Figs. 17 and 18 show the results obtained for this

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

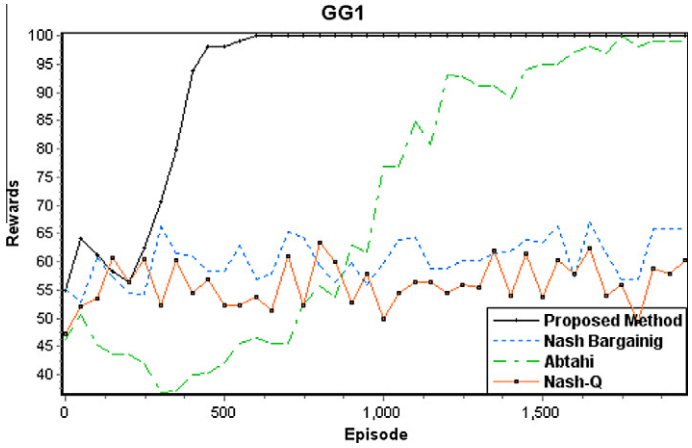

Fig. 17. Comparison of different methods in terms of the amount of received reward for GG1.

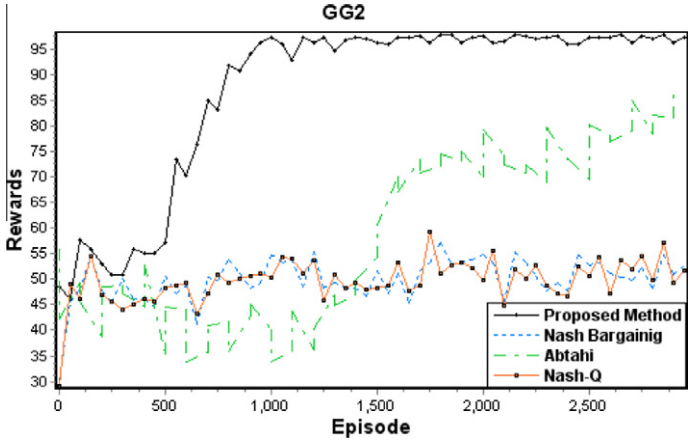

Fig. 18. Comparison of different methods in terms of the amount of received reward for GG2.

Table 1
Grid game 1.

Learning	Strategy	Percent that reach a Nash path (%)
Agent1	Agent2	
Single-agent Q-learning	Single-agent Q-learning	50
Nash-Q algorithm	Single-agent Q-learning	72
Nash-Q algorithm	Nash-Q algorithm	100
The proposed algorithm	The proposed algorithm	100

Table 2
Grid game 2.

Learning	Strategy	Percent that reach a Nash path (%)
Agent1	Agent2	
Single-agent Q-learning	Single-agent Q-learning	24
Nash-Q algorithm	Single-agent Q-learning	45
Nash-Q algorithm	Nash-Q algorithm	100
The proposed algorithm	The proposed algorithm	100

experiment. As it is shown, for both games, the proposed algorithm has higher performance and then comes Abtahi's algorithm.

7.2.3. Offline learning performance

In offline learning, training data for improving the learned function are collected in the training period, and performance is measured in the test period. The performance of the test period is measured by the reward an agent receives when both agents follow their learned strategies, starting from the initial positions at the lower corners. Since learning is internal to each agent, two agents might learn different sets of learning algorithm that yield different Nash equilibrium solutions. In this experiment, we compare the proposed algorithm with Nash-Q algorithm (Hu & Wellman, 2003) and single-agent Q-learning (Watkins & Dayan, 1992) in terms of the percentage of reaching a Nash path for both GG1 and GG2. A Single-Q agent uses the standard single-agent Q-learning method and learns about the Q-values for the state and its own actions, ignoring the actions of the other agents. In this experiment, 100 trials were run and calculated the fraction that reached the Nash path. Each trial runs for 5000 episodes. The experimental results are shown in Table 1 for GG1 and Table 2 for GG2. For example in GG1 when both agents employ single-agent Q-learning, they reach a Nash path only 50% of the time. This is because the single-agent learner never models the other agent's strategy. Note that when the both agent employ Nash-Q algorithm or both agent employ the proposed algorithm a Nash path is reached 100 percent of the time.

8. Conclusion

In this paper, a learning automata based multi-agent systems in which the concepts of stigmergy and entropy are used to enhance

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152

the performance of system and used to find optimal policies in Markov Games was proposed. The proposed algorithm by using the concepts of stigmergy and entropy tries to provide a simple framework for interaction and coordination in multi-agent systems and speeding up the learning process. Several experiments were conducted to investigate the learning performance of the proposed algorithm. The experimental results showed that the proposed algorithms have better learning performance in terms of the speed of reaching the optimal policy than other learning algorithms such as NASH-Q algorithm, Nash Bargaining algorithm and Abtahi's algorithm.

References

Abtahi, F., Meybodi, M. R. (2008). Solving multi-agent markov decision processes using learning automata. In: *The 6th international symposium on intelligent systems (SISY2008)* (pp. 1–6). Subotica, Serbia.

Busniu, L., Babuska, R., & Schutter, B. (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transaction on System, Man, Cybern, 38*, 156–171.

Claus, C., Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In: *The 15th national conference on artificial intelligence* (pp. 746–752). Wisconsin, USA.

Costa, M., Goldberger, A., & Peng, C. (2002). Multi-scale entropy analysis of complex physiologic time series. *Physical Review Letters, 89*, 1–4.

Dianhu, Z., Shaohui, F., & Xiaojun, D. (1997). Entropy – A measure of uncertainty of random variable. *Systems Engineering and Electronics, 11*, 1–3.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization (Bradford books)*. The MIT Press.

Fink, A. M. (1964). Equilibrium in a stochastic N-person game. *Journal of Science in Hiroshima University, Series A-I, 28*, 89–93.

Goel, A. K., Kumar, V., Srinivasan, S. (2008). Application of multi-agent system and agent coordination. In: *The second national conference mathematical techniques: emerging paradigms for electronics and IT industries (MATEIT-2008)* (pp. 328–337). New Delhi, India.

Grasśe, P. P. (1959). La Reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la theoriede la stigmergie: Essai d'interpŕetation du comportement des termites constructeurs. *Insectes Sociaux, 6*, 48–83.

Greenwald, A., Hall, K. (2003). Correlated Q-learning. In: *The twentieth international conference on machine learning (ICML 2003)* (pp. 242–249). California: AAAI Press.

Holland, O., & Melhuish, C. (1999). Stigmergy, self-organization, and sorting in collective robotics. *Artificial Life, 5*, 173–202.

Hu, J., & Wellman, M. P. (2003). Nash Q-learning for general-sum stochastic games. *Journal of Machine Learning Research, 4*, 1039–1069.

Khojasteh, M. R., & Meybodi, M. R. (2007). Evaluating learning automata as a model for cooperation in complex multi-agent domains. *RoboCup 2006: Robot Soccer World Cup* (Vol. 4434, pp. 410–417). Berlin: Springer.

Kollingbaum, M. (2001). *Stigmergic Control System Design*. Leuven: Working Paper, PMA, K.U.

Lakshmivarahan, S., & Narendra, K. (1981). Learning algorithms for two-person zero-sum stochastic games with incomplete information. *Mathematics of Operations Research, 6*, 379–386.

Lieb, E. H., & Yngvason, J. (1999). The physics and mathematics of the second law of thermodynamics. *Physics Report, 310*, 1–96.

Littman, M. (1994). Markov games as a framework for multi-agent reinforcement learning. In *11th international conference on machine learning* (pp. 157–163). San Francisco: Morgan Kaufmann.

Masoumi, B., Meybodi, M. R. (2009). Utilization of networks of learning automata for solving decision problems in decentralized multiagent systems. In: *17th Iranian conference on electrical engineering (ICEE2009)*. Tehran, Iran.

Masoumi, B., Meybodi, M. R., Jafarpour, B. (2008). Solving general sum stochastic games using learning automata. In: *The second joint congress on fuzzy and intelligent systems*. Tehran, Iran.

Narendra, K. S., & Thathachar, M. A. L. (1989). *Learning automata: An introduction*. Prentice-Hall Inc..

Nash, J. F. (1951). Non-cooperative games. *Annals of Mathematics, 54*, 286–295.

Nowe, A., Verbeeck, K., & Peeters, M. (2006). Learning automata as a basis for multi-agent reinforcement learning. *Learning and Adaption in Multi-Agent Systems* (Vol. 3898, pp. 71–85). Berlin: Springer.

Nowe, A., & Verbeeck, K. (2002). Colonies of learning automata. *IEEE Transactions on Systems, Man and Cybernetics, 32*, 772–780.

Nowe, A., Verbeeck, K., & Vrancx, P. (2004). Multi-type ant colony: The edge disjoint paths problem. *Ant colony, optimization and swarm intelligence* (Vol. 3172, pp. 202–213). Berlin: Springer.

Osborne, M. J., & Rubinstein, A. (1994). *A course in game theory*. Cambridge, MA: MIT Press.

Panait, L., Luke, S. (2004). A pheromone-based utility model for collaborative foraging, autonomous agents and multiagent systems. In: *The third international joint conference on AAMAS 2004* (pp. 36–43). New York, USA.

Parunak, H. V. D. (2000). A practitioners' review of industrial agent applications. *Autonomous Agents and Multi-Agent Systems (Springer, The Netherlands), 3*, 389–407.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. New York, USA: John Wiley and Sons.

Qio, H., Szidarovszky, F., Rozenblit, Yong, L. (2006). Multi-agent learning model with bargaining. In: *The 38th conference on winter simulation* (pp. 934–940). Monterey, California.

Sastry, P., Phansalkar, V., & Thathachar, M. A. L. (1994). Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Transactions on Systems, Man, and Cybernetics, 24*, 769–777.

Song, M., Bai, J., Chen, R. (2007). A new learning algorithm for cooperative agents in general-sum games. In: *The sixth international conference on machine learning and cybernetics* (pp. 50–54). Hong Kong.

Song, M., Gu, G., & Zhang, G. (2005). Pareto-Q learning algorithm for cooperative agents in general-sum games. *Multi-Agent Systems and Applications IV* (Vol. 3690, pp. 576–578). Berlin: Springer.

Stone, P., & Veloso, M. (2000). Multiagent systems: a survey from the machine learning perspective. *Autonomous Robots, 8*, 345–383.

Thathachar, M. A. L., & Sastry, P. S. (2002). Varieties of learning automata: An overview. *IEEE Transaction on Systems, Man, and Cybernetics-Part B: Cybernetics, 32*, 711–722.

Thathachar, M. A. L., & Sastry, P. S. (2004). *Networks of learning automata: Techniques for online stochastic optimization*. Kluwer Academic Publishers.

Theraulaz, G., & Bonabeau, E. (1995). Modelling the collective building of complex architectures in social insects with lattice swarms. *Journal of Theoretical Biology, 177*, 381–400.

Theraulaz, G., & Bonabeau, E. (1999). A brief history of stigmergy. *Artificial Life, 5*, 97–116.

Thusijsman, F. (1992). *Optimality and equilibria in stochastic games*. Amsterdam: Centrum voor Wiskunde en Informatica.

Valckenears, P., Brussel, H. V., Kollingbaum, M., & Bochmann, O. (2001). Multi-agent coordination and control using stigmergy applied to manufacturing control. In *Mutli-agents systems and applications* (pp. 317–334). New York: Springer-Verlag.

Verbeeck, K., Nowé, A., Parent, J., & Tuyls, K. (2007). Exploring selfish reinforcement learning in repeated games with stochastic rewards. *Journal of Autonomous Agents and Multi-Agent Systems, 14*, 239–269.

Verbeeck, K., Nowé, A., Peeters, M., & Tuyls, K. (2005). Multi-agent reinforcement learning in stochastic single and multi-stage games. *Adaptive Agents and Multi-Agent Systems III* (Vol. 3394, pp. 275–294). Berlin: Springer.

Vlassis, N. (2007). *A concise introduction to multiagent systems and distributed artificial intelligence*. Amesterdam: Morgan and Claypool Publishers.

Vrancx, P., Verbeeck, K., & Nowé, A. (2007). Analyzing stigmergetic algorithms through automata games. *Knowledge Discovery and Emergent Complexity in Bioinformatics* (Vol. 4366, pp. 145–156). Berlin: Springer.

Vrancx, P., Verbeeck, K., & Nowé, A. (2008). Decentralized learning in markov games. *IEEE Transactions on Systems, Man and Cybernetics (Part B: Cybernetics), 38*, 976–981.

Wang, X., & Sandholm, T. (2002). Reinforcement learning to play an optimal Nash equilibrium in team markov games. *Advances in Neural Information Processing Systems* (Vol. 15, pp. 1571–1578). MIT Press.

Watkins, J. C. H., & Dayan, P. (1992). *Q-learning machine learning, 3*, 279–292.

Wheeler, R. M., & Narendra, K. S. (1986). Decentralized learning in finite markov chains. *IEEE Transactions on Automatic Control, 31*, 519–526.

Zhuang, X. (2005). The strategy entropy of reinforcement learning for mobile robot navigation in complex environments. In: *The IEEE international conference on robotics and automation* (pp. 1742–1747). Barcelona, Spain.

Zhuang, X., Chen, Z. (2008). Strategy entropy as a measure of strategy convergence in reinforcement learning. In: *First international conference on intelligent networks and intelligent systems* (pp. 81–84). Wuhan, China.

Please cite this article in press as: Masoumi, B., & Meybodi, M. R. Speeding up learning automata based multi agent systems using the concepts of stigmergy and entropy. *Expert Systems with Applications* (2011), doi:10.1016/j.eswa.2010.12.152