State estimation and reinforcement learning for behavior selection in stochastic multiagent systems

Matheus Vieira Portela[1]
Faculty of Technology
University of Brasília

Guilherme Novaes Ramos[2]
Department of Computer Science
University of Brasília

Abstract

Intelligent agents can act based on sensor measurements in order to fulfill their goals. In dynamic systems, agents must adapt its behavior selection processes to reflect the changing system state since behaviors that previously were considered the best choice may become sub-optimal. Multiple agents that co-exist in the environment is one example of such a dynamic system. The problem is even greater when stochastic systems are considered, since the states the agents are actually in are unknown. This work proposes a learning algorithm for stochastic multiagent systems, in which Bayesian programming is used for state estimation and Q-learning provides learning capabilities to the agents. An experimental setup using electronic games is described to evaluate the effectiveness of this approach.

Keywords: Artificial Intelligence, Multiagent System, Bayesian Programming, Reinforcement Learning, Q-learning, Predator-pursuit

Author's Contact:

[1] matheus.portela@aluno.unb.br
[2] gnramos@unb.br

1 Introduction

Digital games provide a test bed for experimentation and study of Artificial Intelligence (AI), and there has been a growing interest in applying AI in several problems present in them [Lucas 2008]. One of the more interesting and difficult uses is controlling non-playable characters (NPCs), whose behavior should be interesting, believable, and adapt to the player [Lucas 2009]. Such games also present a well defined environment in which complex situations can be simulated for developing solutions for actual problems [Caldeira et al. 2013].

Uncertainty is inherent to real-world applications, most aspects of the environment are not directly measurable and measurements are often noisy. Therefore, descriptions of a state of the environment require some stochastic approach to handle such uncertainties [Koller and Friedman 2009]. Among current stochastic methods, by using Bayesian inference to reach reasonable conclusions in highly noisy environments, Bayesian methods find successful applications in both academic and industrial problems [Koller and Friedman 2009]. In this scenario, Bayesian programming (BP) provides a generic approach for modeling and decision-taking based solely in Bayesian inference [Lebeltel et al. 2004].

An agent is defined as an entity capable of acting and sensing in an environment in order to maximize a performance measurement [Weiss 1999]. In cooperative multiagent systems (cooperative MAS), multiple agents coexist in the same environment and an agent that improves its performance positively affects the performance of other agents in the same environment. Therefore, multiple agents can reach their goals concurrently [Russell and Norvig 2010].

When dealing with cooperative MAS, which are dynamic environments, an agent is expected to adapt its actions based on previous experiences accumulated by interacting with the environment [Russell and Norvig 2010]. The development of learning machines is a major challenge in contemporary AI research, difficult enough to justify the creation of a whole academic area: machine learning (ML). Within this are reinforcement learning (RL), where agents must learn from interaction which actions yield higher numeric rewards, found extensive usage in MAS [Sutton and Barto 1998].

Typically, agents are controlled by periodically selecting the action that will be executed [Martinson et al. 2001]. A control framework using behaviors, pre-programmed sequences of actions, can also be used so as agents can perform complex tasks. In some cases, using behaviors can even accelerate learning processes [Martinson et al. 2001].

Usually, learning algorithms are validated by evaluating their performance on problems well-known by the scientific community. In cooperative MAS, the predator-pursuit problem is such a standard test. Traditionally, an environment is set up with four predators and one prey where the prey must avoid being captured and predators must capture it [Stone and Veloso 2000]. Eletronic games provide an excellent platform for simulating such conditions, and Pac-Man, one of the most famous games in history [Kent 2001], has its gameplay defined exactly as this type of real-world problem.

This work approaches the problem of creating agents that can learn to select the best behaviors in cooperative MAS where stochasticity is present, such as eletronic games, robotics scenarios, and others. This paper is organized as follows. Section 2 presents previous work with points in common with the work here developed. Section 3 summarizes the Bayesian programming theory, and Section 4 discusses reinforcement learning. Section 5 describes the algorithm proposed in this paper using these. Finally, Section 6 explains the validation experiments that will be conducted to verify the algorithm's performance.

2 Related Work

Probabilistic graphical models have been used as a generic framework to allow automated reasoning in real-world applications. By representing the system with models and considering the most probable cases, computers can reach meaningful conclusions even though not all variables are explicit. This framework is currently being used in medical diagnosis, market analysis, natural language processing, communication systems, among others [Koller and Friedman 2009].

Bayesian programming is a generic framework that incorporates uncertainty in the design of intelligent agents, besides providing efficient computation by using conditional independence assumptions [Koike 2005]. Bayesian programming was used previously in CAD systems, environment mapping, and driving assistance systems [Lebeltel et al. 2004].

Modern computing infrastructures are developed by distributing processing across multiple machines, often in different geographic locations, being connected by networks [Shoham and Leyton-Brown 2008]. State-of-the-art multiagent system algorithms consider computers to be individuals capable of sensing and acting autonomously. This approach, extensively used in cloud computing, creates robust and computationally powerful systems that are able to adapt and process millions of requests per second [Weiss 1999].

Multiagent learning has been studied considering particularities for several scenarios, containing or not elements such as competition, communication, and heterogeneity [Stone and Veloso 2000]. Work has been done to investigate differences in independent and cooperative learning [Tan 1993]. Further studies consider learning in behavior-based systems [Matarić 2001] and usage of game-theory for reinforcement learning in competitive MAS [Littman 1994].

In predator-pursuit research, two techniques are usually employed in order to develop intelligent agents: reinforcement learning [Stone and Veloso 2000] and evolutionary computing [Haynes and Sen 1996][Jim and Giles 2000].

3 Bayesian Programming

Decision taking processes that take into account environmental uncertainties have been studied by AI researchers in the last few decades, initially with Bayesian networks and probabilistic graphical models. Since then, Bayesian approaches are replacing traditional methods with more robust algorithms that allows robots to deal with stochasticity in an efficient way [Thrun et al. 2005].

Bayesian programming is a framework to develop intelligent systems using Bayesian inference only. The mathematic formalism developed for BP is generic enough to reinterpret, under its light, classical probabilistic techniques such as Bayesian networks, Bayesian filters, Markov hidden models, Kalman filter, and particle filter [Lebeltel et al. 2004].

3.1 Bayesian Programming Foundations

The fundamental element in BP is the *logical proposition*: a hypothesis a that can be either true or false. By applying logical operators, such as conjunction, disjunction, and negation, one can create new propositions based on existing ones. Probability values are attributed to propositions to deal with uncertainties. For instance, $P(a) = 0.9$ means the proposition a has 90% chance of being true [Koike 2005]. It is important to notice that all propositions depend on the designer's previous knowledge of the system, represented by π. Therefore, proposition probabilities are always conditioned on π, denoted by $P(a|\pi)$ [Lebeltel et al. 2004].

Another important concept in BP is *discrete variables*: a set X of mutually exclusive and exhaustive propositions, i.e., $x_i \wedge x_j$ is false for $i \neq j$ and at least one proposition in X is true. The probability of X is defined as the conjunction probability for all of its propositions.

Bayesian inference rules enable the calculation of unknown probabilities of some propositions and variables based on known probabilities of other propositions or variables. The conjunction rule, presented in (1), calculates the probability of a conjunction of propositions. The normalization rule, as in (2), defines the relationship between the probabilities of a proposition and its negation. Finally, the marginalisation rule (see (3)) derives probabilities of discrete variables based on conditional probabilities [Lebeltel et al. 2004].

$$P(x \wedge y|\pi) = P(x|\pi) \cdot P(y|x \wedge \pi) = P(y|\pi) \cdot P(x|y \wedge \pi) \quad (1)$$

$$P(x|\pi) + P(\neg x|\pi) = 1 \quad (2)$$

$$\sum_X P(X \wedge Y|\pi) = P(Y|\pi) \quad (3)$$

3.2 Bayesian Program Elements

Using discrete variables, a Bayesian program is a mathematical procedure to specify a family of probability distributions in order to control agents to execute complex tasks [Lebeltel et al. 2004]. Any Bayesian program contains two parts: a description and a question. The *description* is the joint probability distribution of all pertinent variables $\{X_1, X_2, \ldots, X_n\}$ using previous knowledge (π) and experimental data (δ) such as presented in (4) [Lebeltel et al. 2004].

$$P(X_1 \wedge X_2 \wedge \ldots \wedge X_n|\delta \wedge \pi) \quad (4)$$

Usually, it is difficult to calculate a *description*, requiring the system designer to apply conditional independence hypotheses in order to reduce its complexity [Koike 2005].

The second part of a Bayesian program, the *question*, is a probability distribution that is calculated using the description. First, it is necessary to split the description variables into three sets: S, representing the variables whose probabilities are calculated, K, as the variables that are observable, and U, as the unknown variables. Then, the question is mathematically described by (5) [Koike 2005].

$$P(S|K \wedge \delta \wedge \pi) = \frac{\sum_U P(S \wedge K \wedge U|\delta \wedge \pi)}{\sum_{U,S} P(S \wedge K \wedge U|\delta \wedge \pi} \quad (5)$$

For instance, a probabilistic question that estimates the agent state S based on its measurements Z and actions U is defined as $P(S|Z \wedge U \wedge \pi)$. Afterwards, this estimation can feed other parts of the intelligent system, such as action selection.

4 Reinforcement Learning

In RL problems, an agent can sense its surroundings and act to modify the state of the environment [Sutton and Barto 1998]. However, the agent does not know which action yields the best performance in the current state. The only feedback available is a numeric value, the reward r, given by the environment after the execution of an action [Sutton and Barto 1998]. Therefore, the agent must learn, from previous experiences, which actions maximize future rewards.

Based on this idea, a value function $V(s)$ returns the amount of reward an agent expects to receive starting from the current state and is used for action selection. Nevertheless, this function is not directly observable and needs to be estimated on all received rewards in the agent's history [Sutton and Barto 1998].

The agent's policy π maps environment states to actions and may be implemented as either state-action tables, simple functions, or complex search processes [Sutton and Barto 1998]. During policy design, it is important to consider the trade-off between exploitation and exploration. The former selects the best estimated action, improving performance. The latter uses sub-optimal actions to collect information that may lead the agent to receive higher rewards in the long run [Sutton and Barto 1998].

4.1 Q-learning

Among available RL algorithms, Q-learning is an off-policy, model-free reinforcement learning algorithm used to control agents by iteratively estimating the values of state-action pairs based on the last received reward [Sutton and Barto 1998]. (6) and (7) present the correction equations, where γ is the discount factor, α is the learning factor, and ϵ is the estimation error.

$$\epsilon \leftarrow \Big[r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1})\Big] \quad (6)$$

$$Q(s_{t-1}, a_{t-1}) \leftarrow Q(s_{t-1}, a_{t-1}) + \alpha\epsilon \quad (7)$$

However, (6) and (7) cannot be applied in continuous environments due to an infinite number of state-action pairs. In this situation, a better alternative is to apply function approximation methods to estimate $Q(s_t, a_t)$. In linear approximation, features from the current state are selected, composing a feature column-vector $\vec{\phi}(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_n(s)]^T$, where $0 \leq \phi_i(s, b) \leq 1$ indicates the probability of the i-th feature. A vector of parameters $\vec{\theta}(s)$ of same dimension is defined, where θ_i indicates the relevance of the feature $\phi_i(s)$. $Q(s_t, a_t)$ is then estimated by (8) [Irodova and Sloan 2005].

$$Q(s_t, a_t) = \vec{\theta}^T \vec{\phi}(s_t) = \sum_{i=1}^{N} \theta_i \phi_i(s_t) \quad (8)$$

Learning occurs by updating the parameter vector according to some rule [Irodova and Sloan 2005]. Usually, gradient descent is used [Irodova and Sloan 2005] as shown in (9) where $\delta = r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1})$ and $\nabla_{\vec{\theta}} Q(s_t, a_t) = \vec{\phi}(s_t)$.

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\nabla_{\vec{\theta}} Q(s_t, a_t) \quad (9)$$

5 Proposed Algorithm

This work presents an algorithm that to allows multiple agents to learn to select behaviors in stochastic MAS. It assumes all agents exist in a bi-dimensional environment where they can go either North, South, East, West, or remain in the same position. Moreover, agents have sensors to measure distances and directions in respect to each other.

The algorithm has three parts: environment state estimation, behavior selection, and action selection. Though N agents can exist in the environment, each one will use the same algorithm to learn, thus the following description assumes the index 1 to be a reference to the learning agent itself.

5.1 State Estimation

An agent must track the current environment state S, consisting of distances S_{d_i} and directions S_{θ_i} to every other gent. Another pertinent variable is the measurements variable Z, composed by the measured distances Z_{d_i} and directions Z_{θ_i} to every agent. Finally, the agent must know its action U. All these variables change throughout time, and these transitions are denoted by the suffix $X^{0:t} = X^0 \wedge X^1 \wedge \ldots \wedge X^t$.

The *description* of the Bayesian program, therefore, consists of:

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t}|\pi) \tag{10}$$

By applying the conjunction rule, the description is re-stated:

$$\begin{aligned} P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t}|\pi) = \\ P(S^t \wedge Z^t \wedge U^t|S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} \wedge \pi)\cdot \\ P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1}|\pi) \end{aligned} \tag{11}$$

This is not trivially computed since it depends on the variables' entire history. However, by considering the first-order Markov assumption, the probability of a variable at time t becomes independent of its history if the variable's value at time $t-1$ is known [Thrun et al. 2005]. Markov assumption can be applied to (11), which is then simplified to:

$$\begin{aligned} P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t}|\pi) = \\ P(S^t \wedge Z^t \wedge U^t|S^{t-1} \wedge Z^{t-1} \wedge U^{t-1}\pi)\cdot \\ P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1}|\pi) \end{aligned} \tag{12}$$

The *description* (12) can be re-written to (13) to highlight the fact that estimating the description is an iterative process that depends on two factors: an initial probability distribution $P(S^0 \wedge Z^0 \wedge U^0|\pi)$ and a transition probability distribution $P(S^j \wedge Z^j \wedge U^j|S^{j-1} \wedge Z^{j-1} \wedge U^{j-1}\pi)$.

$$\begin{aligned} P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t}|\pi) = \\ P(S^0 \wedge Z^0 \wedge U^0|\pi)\cdot \prod_{j=1}^{t} P(S^j \wedge Z^j \wedge U^j|S^{j-1} \wedge Z^{j-1} \wedge U^{j-1}\pi) \end{aligned} \tag{13}$$

The initial condition represents the system designer's knowledge on the state of the environment when the program starts executing. It is common to initialize it as a uniform distribution, representing that no knowledge is assumed [Koike 2005].

The transition distribution is simplified by assuming conditional independence. The state S^t depends only on the previous state S^{t-1} and the last executed action U^{t-1}. The measurement Z^t, in turn, depends only on the current state S^t. Finally, the action U^t is assumed to be independent from all variables since it is chosen by the action selection algorithm.

$$\begin{aligned} P(S^t \wedge Z^t \wedge U^t|S^{t-1} \wedge Z^{t-1} \wedge U^{t-1}\pi) = \\ P(S^t|S^{t-1} \wedge U^{t-1}\pi) \cdot P(Z^t|S^t\pi)P(U^t|\pi) \end{aligned} \tag{14}$$

The same conditional independence rationale is applied to the variables S_{d_i}, S_{θ_i}, Z_{d_i}, and Z_{θ_i}.

$$\begin{aligned} P(S^t|S^{t-1} \wedge U^{t-1} \wedge \pi) = P(S^t_{d_i}|S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi)\cdot \\ P(S^t_{\theta_i}|S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi) \end{aligned} \tag{15}$$

$$P(Z^t|S^t \wedge \pi) = P(Z^t_{d_i}|S^t_{d_i} \wedge \pi) \cdot P(Z^t_{\theta_i}|S^t_{\theta_i} \wedge \pi) \tag{16}$$

Finally, it is necessary to define the form of the probability distribution functions $P(S^t_{d_i}|S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi)$, $P(S^t_{\theta_i}|S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi)$, $P(Z^t_{d_i}|S^t_{d_i} \wedge \pi)$, $P(Z^t_{\theta_i}|S^t_{\theta_i} \wedge \pi)$, and $P(U^t|\pi)$.

For the bi-dimensional environment, the agent's coordinates (x, y) are estimated relative to S^{t-1} after executing the action U^{t-1}. Using the same procedure, it is possible to calculate the (x_i, y_i) relative to the i-th agent. Therefore, (17) is the expected distance to the i-th agent, after executing U^{t-1}, and (18) is the expected direction.

$$d = \sqrt{(x_i - x)^2 + (y_i - y)^2} \tag{17}$$

$$\theta = \frac{y_i - y}{x_i - x} \tag{18}$$

$P(S^t_{d_i}|S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi)$ is assumed to be a Gaussian distribution centered in d with an arbitrary standard deviation σ, which can be learned from experimental data δ. By analogy, $P(S^t_{\theta_i}|S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi)$ is centered around θ.

$$P(S^t_{d_i}|S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi) = \mathcal{G}(S^{t-1}_{d_i}, U^{t-1}), \mu = d, \sigma \tag{19}$$

$$P(S^t_{\theta_i}|S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi) = \mathcal{G}(S^{t-1}_{\theta_i} \wedge U^{t-1}), \mu = \theta, \sigma \tag{20}$$

Measurements, then, are Gaussian distributions around the estimated state for distance and direction. The standard deviations σ_{sd} and $\sigma_{s\theta}$ represent the quality of the distance and direction sensors.

$$P(Z^t_{d_i}|S^t_{d_i} \wedge \pi) = \mathcal{G}(S^t_{d_i}), \mu = S^t_{d_i}, \sigma = \sigma_{sd} \tag{21}$$

$$P(Z^t_{\theta_i}|S^t_{\theta_i} \wedge \pi) = \mathcal{G}(S^t_{\theta_i}), \mu = S^t_{\theta_i}, \sigma = \sigma_{s\theta} \tag{22}$$

The action is defined by the action selection algorithm, independent from the Bayesian program, and described by an uniform distribution.

$$P(U^t_i|\pi) = Uniform \tag{23}$$

Finally, the Bayesian program's *description* is finished and a probabilistic *question* can be stated. To accomplish this, the agent needs an estimation of the world's current state given its sensors' measurements and it's last action executed. Mathematically, (24) represents the state estimation and can be calculated using (13) and Bayesian inference rules.

$$P(S^t|Z^t \wedge U^{t-1} \wedge \pi) \tag{24}$$

5.2 Behavior selection

Typically, Q-learning is used to estimate state-action values and then select actions [Martinson et al. 2001]. In this work, however, Q-learning is used for behavior selection, hence, it will estimate the values of state-behavior pairs.

Since the agent exists in a bi-dimensional world, potentially a continuous environment, the function approximation, which uses a ϵ-greedy for exploration, is applied as:

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\nabla_{\vec{\theta}} Q(s_t, b_t) \tag{25}$$

$$\delta = r_t + \gamma \max_a Q(s_t, b) - Q(s_{t-1}, b_{t-1}) \tag{26}$$

$$\nabla_{\vec{\theta}} Q(s_t, b_t) = \vec{\phi}(s_t) \tag{27}$$

5.3 Action selection

Behaviors allow different actions to be selected in the same state. Therefore, it is required to define the rules that map state-behavior pairs to actions. This could be done by a Bayesian program, or even by sampling a probability distribution.

In this work, actions will be selected by procedural algorithms. For instance, a behavior can be programmed to return the action that moves the agent closer to another one according to the probability of the other agent's estimated state.

6 Experiments

The presented algorithm will be evaluated by computational simulations with multiagent electronic games, considering a predator-pursuit situation. The Pac-Man game simulator[1] provides the necessary features for this: a bi-dimensional environment, multiple agents with a clear task, well defined behaviors, and a direct reward value through obtained points. In contrast to the game's original goal of moving the Pac-Man through the map, collecting food and avoiding the ghosts, our experiments will evaluate if the proposed algorithm can make the ghosts learn the behaviors that lead to capturing the Pac-Man with minimal score, characterizing a cooperative MAS.

The experiments are designed in three phases, the first to evaluate if the algorithm works by analyzing if the Pac-man's performance improves as a result of learning the proper behaviors. The second part is to evaluate it in an actual cooperative MAS by verifying if the ghosts performance improves as a result of learning the proper behaviors. Should the ghosts be capable of learning, the game score is expected to be reduced compared to the scores generated by the baseline agents initially implemented in the simulator. Finally, we will compare a co-evolutionary approach, by having both Pac-Man and ghosts learning at the same time.

7 Conclusion

Eletronic games are an excellent test bed for developing Artificial Intelligence, and creating agents that learn is currently one of the most exciting and challenging areas of study. The problem becomes even harder when multiple learning agents co-exist in the same environment. In this case, the environment becomes stochastic since the agent does not know the other agent's actions in advance.

This work provides one step in the direction of developing an approach for multiple learning agents in bi-dimensional stochastic environments. A theoretical algorithm is proposed where Bayesian programming is used to deal with uncertainty, so agents seem more believable, and Q-learning with function approximation is used for changing the agents behavior, so it can provide a bigger challenge and adapt to new situations.

The proposed algorithm will be tested with the Pac-Man simulator but, due to its general approach, could be applied in any problem involving cooperative multiple agents in games or other fields.

[1]Available at: http://ai.berkeley.edu/multiagent.html

References

CALDEIRA, C., ARANHA, C., AND RAMOS, G. 2013. TORCS Training Interface: An auxiliary API for developing TORCS drivers. *XII Simpósio Brasileiro de Jogos e Entretenimento Digital*, 13.

HAYNES, T., AND SEN, S. 1996. Evolving behavioral strategies in predators and prey. In *Adaption and learning in multi-agent systems*. Springer, 113–126.

IRODOVA, M., AND SLOAN, R. H. 2005. Reinforcement learning and function approximation. In *FLAIRS Conference*, AAAI, 455–460.

JIM, K.-C., AND GILES, C. L. 2000. Talking helps: Evolving communicating agents for the predator-prey pursuit problem. *artificial life 6*, 3, 237–254.

KENT, S. L. 2001. *The ultimate history of video games: from Pong to Pokémon and beyond: the story behind the craze that touched our lives and changed the world*, 1st ed ed. Prima Pub, Roseville, Calif.

KOIKE, C. M. C. E. C. 2005. *Bayesian approach to action selection and attention focusing: an application in autonomous robot programming*. PhD thesis, Institut National Polytechnique de Grenoble.

KOLLER, D., AND FRIEDMAN, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

LEBELTEL, O., BESSIÈRE, P., DIARD, J., AND MAZER, E. 2004. Bayesian robot programming. *Autonomous Robots 16*, 1, 49–79.

LITTMAN, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, vol. 157, 157–163.

LUCAS, S. M. 2008. Computational intelligence and games: Challenges and opportunities. *International Journal of Automation and Computing 5*, 1, 45–57.

LUCAS, S. M. 2009. Computational intelligence and ai in games: A new ieee transactions. *Computational Intelligence and AI in Games, IEEE Transactions on 1*, 1 (March), 1–3.

MARTINSON, E., STOYTCHEV, A., AND ARKIN, R. C. 2001. Robot behavioral selection using q-learning.

MATARIĆ, M. J. 2001. Learning in behavior-based multi-robot systems: Policies, models, and other agents. *Cognitive Systems Research 2*, 1, 81–93.

RUSSELL, S. J., AND NORVIG, P. 2010. *Artificial intelligence: a modern approach*. Prentice Hall.

SHOHAM, Y., AND LEYTON-BROWN, K. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.

STONE, P., AND VELOSO, M. 2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots 8*, 3, 345–383.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement learning: an introduction*. MIT press Cambridge.

TAN, M. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, 330–337.

THRUN, S., BURGARD, W., AND FOX, D. 2005. *Probabilistic robotics*. MIT press.

WEISS, G. 1999. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press.