Bayesian programming and reinforcement learning in stochastic multiagent systems

Matheus Vieira Portela*, Guilherme Novaes Ramos†
University of Brasília
*matheus.portela@aluno.unb.br, †gnramos@unb.br

***Abstract*—Intelligent agents act based on sensor measurements in order to fulfill their goals. When the environment is dynamic, such as a multiagent system, agents must adapt its actions selection processes to reflect the ever changing system state since behaviors that previously were considered the best choice may becomes sub-optimal. The problem is even greater when stochasticity is taken into account, since the environment true state is unknown to the agents. This work proposes a learning algorithm for stochastic multiagent systems, in which Bayesian programming is used for state estimation and Q-learning with function approximation provides learning capabilities so as agents can select the appropriate behaviors. An experimental setup to evaluate the effectiveness of this approach using electronic games is described, as well as the preliminary results.**

***Index Terms*—Artificial Intelligence, Multiagent System, Bayesian Programming, Reinforcement Learning, Q-learning, Function Approximation, Predator-pursuit**

I. Introduction

A major interest in Artificial Intelligence (AI) is creating intelligent agents that can perform real-world tasks. To this end, Bayesian inference (BI) has shown successful results even in highly noisy environments [1] and Bayesian Programming (BP) provides a generic approach for modeling and decision-taking based on BI [2].

An agent can be defined as an entity capable of acting and sensing in an environment in order to maximize a performance measurement [3]. A Multiagent System (MAS) is inherently dynamic, and agents are expected to adapt their actions based on previous experiences accumulated by interaction [4]. A method suited to MASs is Reinforcement Learning (RL), in which agents learn from interaction which actions yield higher numeric rewards [5]. A control framework using *behaviors*, pre-programmed sequences of actions, can be used by agents, sometime even accelerating the learning process [6].

This work approaches the problem of creating agents that can learn to select the best behaviors in cooperative MAS where stochasticity is present, and is organized as follows. Section II presents previous work with points in common with the work here developed. Section III summarizes the Bayesian programming theory, and Section IV discusses reinforcement learning. Section V describes the algorithm proposed in this paper using these. Finally, Section VI explains the validation experiments that will be conducted to verify the algorithm's performance.

II. Related Work

During the past few decades, several methodologies have been proposed to create computational systems that deal with uncertainties in order to solve real-world problems. Using Probabilistic Graphical Models, computers can reach meaningful conclusions in stochastic environments by representing the system with probabilistic models and taking into account the most probable cases [1]. Another generic framework, Bayesian Programming incorporates uncertainty in the design of intelligent agents, besides providing efficient computation by using conditional independence assumptions [7]. It must be noticed that most of these approaches were initially developed for a single agent system and do not take into account MAS specificities and challenges.

Work in MAS were stimulated by the modern cloud computing infrastructure created by distributing processing across multiple machines connected by networks [8]. State-of-the-art MAS algorithms consider computers to be individuals capable of sensing and acting autonomously [3], but not necessarily learning agents. Multiagent learning has been studied considering particularities for several scenarios, containing or not elements such as competition, communication, and heterogeneity [9]. Work has been done to investigate differences in independent and cooperative learning [10]. Further studies consider learning in behavior-based systems [11] and usage of game-theory for reinforcement learning in competitive MAS [12].

An specific case of MAS research, predator-pursuit problems have been studied under the light of two techniques: reinforcement learning [9] and evolutionary computing [13][14].

III. Bayesian Programming

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 [2].

A. Bayesian Programming Fundamentals

The fundamental element in BP is the *logical proposition*: a hypothesis a that can be either true or false. 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 [7]. 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 $P(a|\pi)$ [2].

The concept of *discrete variables* is also important in BP: 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, such as conjunction (1), normalization (2), and marginalization (3), calculate unknown propositions and variables probabilities based on other known probabilities [2].

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

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

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

B. 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 [2]. 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) [2]. Usually, it is difficult to directly calculate the *description*, requiring the system designer to apply conditional independence hypotheses in order to reduce its complexity [7].

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

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) [7].

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

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.

IV. Reinforcement Learning

In RL problems, an agent can sense its surroundings and act to modify the state of the environment [5]. 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 [5]. 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 [5].

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 [5]. 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 [5].

A. Q-learning

Among available RL algorithms, Q-learning is a popular off-policy, model-free reinforcement learning algorithm used to control agents by iteratively estimating the values $Q(s_t,a_t)$ of state-action pairs based on the last received reward [5]. However, the classical Q-learning cannot be applied in continuous environments due to an infinite number of state-action pairs. In this situation, it is usual 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 parameters vector $\vec{\theta}(s)$ 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 (6) [15].

$$Q(s_t,a_t) = \vec{\theta}^T \vec{\phi}(s_t) = \sum_{i=1}^{N} \theta_i \phi_i(s_t) \quad (6)$$

Learning occurs by updating the parameter vector according to some rule [15]. Usually, gradient descent is used [15] as shown in (7) where $\delta = r_t + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_{t-1})$ and $\nabla_{\vec{\theta}} Q(s_t,a_t) = \vec{\phi}(s_t)$.

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \nabla_{\vec{\theta}} Q(s_t,a_t) \quad (7)$$

V. Proposed Algorithm

This work presents an algorithm that 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, thus the following description assumes the index 1 to be a reference to the learning agent itself.

A. 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 agent), the

measurements variable Z (composed by the measured distances Z_{d_i} and directions Z_{θ_i} to every other agent), and its action U. All these variables change throughout time, these transitions being denoted by the suffix $X^{0:t} = X^0 \wedge X^1 \wedge ... \wedge X^t$.

The *description* of this Bayesian program, therefore, is:

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) \tag{8}$$

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

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) = P(S^t \wedge Z^t \wedge U^t | S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} \wedge \pi) \cdot P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} | \pi) \tag{9}$$

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 [16]. (9) is then simplified to:

$$P(S^{0:t} \wedge Z^{0:t} \wedge U^{0:t} | \pi) = P(S^t \wedge Z^t \wedge U^t | S^{t-1} \wedge Z^{t-1} \wedge U^{t-1} \pi) \cdot P(S^{0:t-1} \wedge Z^{0:t-1} \wedge U^{0:t-1} | \pi) \tag{10}$$

The *description* (10) can be re-written to (11) 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 $P(S^j \wedge Z^j \wedge U^j | S^{j-1} \wedge Z^{j-1} \wedge U^{j-1} \pi)$.

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

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 [7].

The transition distribution can be further 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 depends on the current state S^t, and the action U^t is assumed to be independent from all variables since it is chosen by the action selection algorithm.

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

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

$$P(S^t | S^{t-1} \wedge U^{t-1} \wedge \pi) = P(S^t_{d_i} | S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi) \cdot P(S^t_{\theta_i} | S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi) \tag{13}$$

$$P(Z^t | S^t \wedge \pi) = P(Z^t_{d_i} | S^t_{d_i} \wedge \pi) \cdot P(Z^t_{\theta_i} | S^t_{\theta_i} \wedge \pi) \tag{14}$$

To complete the *description*, it is necessary to define the probabilistic forms of $P(S^t_{d_i} | S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi)$, $P(S^t_{\theta_i} | S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi)$, $P(Z^t_{d_i} | S^t_{d_i} \wedge \pi)$, $P(Z^t_{\theta_i} | S^t_{\theta_i} \wedge \pi)$, 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, (15) is the expected distance to the i-th agent, after executing U^{t-1}, and (16) is the expected direction.

$$d = \sqrt{(x_i - x)^2 + (y_i - y)^2} \tag{15}$$

$$\theta = \frac{y_i - y}{x_i - x} \tag{16}$$

$P(S^t_{d_i} | S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi)$ 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.

$$P(S^t_{d_i} | S^{t-1}_{d_i} \wedge U^{t-1} \wedge \pi) = \mathcal{G}(S^{t-1}_{d_i}, U^{t-1}), \mu = d, \sigma \tag{17}$$

$$P(S^t_{\theta_i} | S^{t-1}_{\theta_i} \wedge U^{t-1} \wedge \pi) = \mathcal{G}(S^{t-1}_{\theta_i} \wedge U^{t-1}), \mu = \theta, \sigma \tag{18}$$

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.

$$P(Z^t_{d_i} | S^t_{d_i} \wedge \pi) = \mathcal{G}(S^t_{d_i}), \mu = S^t_{d_i}, \sigma = \sigma_{sd} \tag{19}$$

$$P(Z^t_{\theta_i} | S^t_{\theta_i} \wedge \pi) = \mathcal{G}(S^t_{\theta_i}), \mu = S^t_{\theta_i}, \sigma = \sigma_{s\theta} \tag{20}$$

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

$$P(U^t_i | \pi) = Uniform \tag{21}$$

Finally, the Bayesian program's *description* is finished and a probabilistic *question* can be stated. In this work, the agent needs an estimation of the world's current state given its sensors' measurements and its last action executed. Mathematically, (22) represents the state estimation and can be calculated using (11) and Bayesian inference rules.

$$P(S^t | Z^t \wedge U^{t-1} \wedge \pi) \tag{22}$$

B. Behavior selection

Typically, Q-learning is used to estimate state-action values and then select actions [6]. 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:

$$\begin{gathered} \vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \nabla_{\vec{\theta}} Q(s_t, b_t) \\ \delta = r_t + \gamma \max_a Q(s_t, b) - Q(s_{t-1}, b_{t-1}) \\ \nabla_{\vec{\theta}} Q(s_t, b_t) = \vec{\phi}(s_t) \end{gathered} \tag{23}$$

C. 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 are 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.

VI. 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: a bi-dimensional environment, multiple agents with a clear task, well defined actions, and a direct reward value extracted from 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 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 two phases. The first part will evaluate the algorithm by analyzing whether 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 whether 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.

Some preliminary experiments indicated that, under some circumstances, Q-learning with function approximation might diverge, i.e., the estimated parameters tend to infinity. This situation, although rare, is mathematically sound [17]. Hence, it was necessary to normalize the parameters vector at each learning iteration [15].

These experiments also demonstrated the relevance of giving sufficient and correct information for the agent. Analyzing the Pac-Man agent, it could only win games by having information on ghosts and food distances, as well as Manhattan distance instead of Euclidian distance. Should these features not be available, the agent would not be able to reason how to fulfill its goal.

Finally, behavior algorithms are a critical part of agent, directly impacting its performance, since agents can only learn which behavior to execute, but not modify it on the fly.

VII. Conclusion

The development of learning agents is currently one of the most exciting and challenging areas of study in Artificial Intelligence. The problem becomes even harder when multiple learning agents co-exist in the same environment, since it becomes stochastic as an 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 become more robust, and Q-learning with function approximation is used for changing the agents behavior selection process, so it can provide a bigger challenge and adapt to new situations.

Eletronic games are an excellent test bed for developing AI methods, hence, the proposed algorithm is 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. Preliminary experiments reveal the critical parts of the system, as well as conditions for creating agents that perform well.

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

References

[1] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[2] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian robot programming," *Autonomous Robots*, vol. 16, no. 1, pp. 49–79, 2004.

[3] G. Weiss, *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 1999.

[4] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2010.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT press Cambridge, 1998.

[6] E. Martinson, A. Stoytchev, and R. C. Arkin, "Robot behavioral selection using q-learning," *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, 2001.

[7] C. M. C. E. C. Koike, "Bayesian approach to action selection and attention focusing: an application in autonomous robot programming," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2005.

[8] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[9] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Autonomous Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[10] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[11] M. J. Matarić, "Learning in behavior-based multi-robot systems: Policies, models, and other agents," *Cognitive Systems Research*, vol. 2, no. 1, pp. 81–93, 2001.

[12] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the eleventh international conference on machine learning*, vol. 157, 1994, pp. 157–163.

[13] T. Haynes and S. Sen, "Evolving behavioral strategies in predators and prey," in *Adaption and learning in multi-agent systems*. Springer, 1996, pp. 113–126.

[14] K.-C. Jim and C. L. Giles, "Talking helps: Evolving communicating agents for the predator-prey pursuit problem," *artificial life*, vol. 6, no. 3, pp. 237–254, 2000.

[15] M. Irodova and R. H. Sloan, "Reinforcement learning and function approximation," in *FLAIRS Conference*. AAAI, 2005, pp. 455–460.

[16] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[17] M. Fairbank and E. Alonso, "The divergence of reinforcement learning algorithms with value-iteration and function approximation," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*. IEEE, 2012, pp. 1–8.