Actions Combination Method for Reinforcement Learning

Marcelo J. Karanik
Universidad Tecnológica Nacional – Facultad Regional Resistencia, GISIA,
Resistencia, Argentina, 3500
marcelo@frre.utn.edu.ar

and

Sergio D. Gramajo
Universidad Tecnológica Nacional – Facultad Regional Resistencia, GISIA,
Resistencia, Argentina, 3500
sergiogramajo@gmail.com

Abstract

The software agents are programs that can perceive from their environment and they act to reach their design goals. In most cases the selected agent architecture determines its behaviour in response to different problem states. However, there are some problem domains in which it is desirable that the agent learns a good action execution policy by interacting with its environment. This kind of learning is called Reinforcement Learning (RL) and is useful in the process control area. Given a problem state, the agent selects the adequate action to do and receives an immediate reward. Then it actualizes its estimations about every action and, after a certain period of time, the agent learns which the best action to execute is. Most RL algorithms execute simple actions even if two o more can be executed. This work involves the use of RL algorithms to find an optimal policy in a gridworld problem and proposes a mechanism to combine actions of different types.

Keywords: Reinforcement Learning, Actions Combination, SARSA, Optimal Policy

Resumen

Los agentes de software son programas que pueden percibir de su entorno y actuar para alcanzar sus objetivos de diseño. En la mayoría de los casos la selección de una arquitectura de agente en particular determina el comportamiento como respuesta a los diferentes estados del problema. Sin embargo, existen algunos dominios de problema en los cuales es deseable que el agente aprenda una buena política de ejecución de acciones mediante la interacción con su entorno. Esta clase de aprendizaje se conoce coma Aprendizaje por Refuerzo (o Reinforcement Learning) y es muy utilizado en el área de control de procesos. Dado un estado del problema, el agente selecciona la acción adecuada y recibe una recompensa inmediatamente. Luego, actualiza sus estimaciones acerca de cada acción y, después de cierto tiempo, aprende cual es la mejor acción a ejecutar. La mayoría de los algoritmos de aprendizaje por refuerzo ejecutan acciones simples aunque varias estén en condiciones de ser utilizadas. En este trabajo se utiliza aprendizaje por refuerzo para encontrar una política óptima en un problema de grid y propone también un mecanismo para combinar acciones de distinto tipo.

Palabras Claves: Aprendizaje por Refuerzo, Combinación de Acciones, SARSA, Política Óptima.

1. Introduction

By using Reinforcement Learning (RL) techniques, an agent interacts with its environment to achieve a goal. In RL, the agent attempts to reach the objective based on learning by trial-and-error [8], [14]. The agent must learn what and how to do to optimally achieve its goal through mapping situations to actions [8]. To reach the optimal solution, it is necessary to maximize a numerical reward signal. The agent must discover by itself what action to take so as to maximize the reward signal. The action that is taken by the agent affects their status and not only does it have immediate reward but also through subsequent actions [8], [14].

There are two basic characteristics of RL, trial-and-error and RL delayed reward. The agent must be able to learn from delayed reinforcement in a long sequence of actions, receiving insignificant reinforcement at the beginning of interaction, and finally arrive at the state with high reward [14], [15].

The goal of RL is to program agents that learn by reward and punishment (negative reward), being needless to specify how the task is made [8].

This kind of learning performed by RL is unsupervised because the agent learns by himself and it is not necessary to include input-output pairs provided by an external expert, such as Artificial Neural Network (ANN) methods [8]. In unsupervised learning the agent is not told what action to take to achieve the best rewards over time. Here, it is

necessary for the agent to acquire useful experience about the possible system states, actions, transitions between states and rewards to operate optimally and so achieve the goal. To do this, the agent must exploit what is already known to obtain a reward, but should also explore to make a better selection of actions in the future [14]. The problems with delayed reinforcement are well modeled with Markov Decision Processes [14]. Formally, the model consists of:

- S, a discrete set of environment states.
- A, a discrete set of agent actions.
- A reward function $R: S\ x\ A\ \rightarrow\ \Re$ [S] or set of scalar reinforcement signal {0,1} or real numbers.
- A state transition function $R: S\ x\ A\ \rightarrow\ \pi(S)$, where a member of $\pi(S)$ is the probability distribution over the set S. It maps states to probabilities, i.e. $T(s, a, s)$ is a probability of making a transition from the state s to s' applying an action.

The agent's job consists of finding a policy π, mapping states to actions to maximize the reward of long-term reinforcement. In general, the environment is non deterministic, that is, taking the same action in the same state on two different times may result in a different next state and / or different reinforcement values [8]. It is assumed that the environment is stationary, that is, the probabilities of making a transition state or receiving a specific reinforcement signal do not change over time. There are also non-stationary environments to build the theoretical system of learning, but they are not focused on this paper. Specially, Artificial Intelligence makes search algorithms where the trajectory from initial state to goal problem through states graphic is achieved. The reinforcement learning paradigm described has been successfully implemented for many well-defined problems such as games theory [2], [6]; robotics [1], [13]; scheduling [9], [10], [11], [16]; telecommunications [3], [12], elevators controls [4], etc.
RL algorithms find an optimal policy to execute actions, and sometimes, several ones can be executed in a state. In this paper an alternative to combine actions in a gridworld problem is presented. In section 2 the problem description is made. RL SARSA algorithm and actions combination method are described in section 3. In section 4 simulations and results are showed, and finally, in section 5, a discussion about some topics and future work are presented.

2. Problem Description

To test the actions combination algorithm a gridworld problem is used. The problem consists of a 10x10 grid divided into two parts. Each one which represents an inclined surface as showed in Figure 1. The agent (A) must go from start (S) to goal (G) across the grid without falling outside. The inclined surfaces involve additional trouble for the agent, because it must vary the velocity for ascendant and descendant cases. The state set is specified through the following variables:

$$S = \{xPos, yPos, actDir, actVel\} \quad (1)$$

where:
xPos is the grid row subindex (1≤xPos≤10);
yPos is the grid column subindex (1≤yPos≤10);
actDir is the agent's direction (with actDir={up,left,down,right});
actVel is the agent's velocity (with actVel={stop,low,medium,high}).

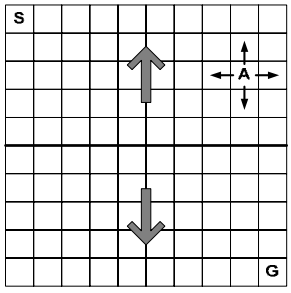

Figure 1. Gridworld Environment.

The agent can use the following six actions:

$$A = \{mDir, turnLeft, turnRight, mVel, Accel, Deacel\} \quad (2)$$

where:
$mDir$ is to maintain direction (this causes a reduction in velocity);
$turnLeft$ is turned to the left (this causes a reduction in velocity);
$turnRight$ is turned to the right (this causes a reduction in velocity);
$mVel$ is to maintain velocity (this makes the agent turn to the left);
$Accel$ is to increment velocity (this makes the agent turn to the left);
$Deacel$ is to decrement velocity (this makes the agent turn to the left).

The environment rules describe the agent movements under the following conditions:

a) for horizontal movements (left or right),
   (1) if actVel is stop then agent advances 0 positions;
   (2) if actVel is low then agent advances 1 position;
   (3) if actVel is medium then agent advances 2 positions;
   (4) if actVel is high then agent advances 3 positions;
b) for vertical movements (up or down in coincidence with inclination)
   (1) if $actVel$ is $stop$ then agent advances 0 positions;
   (2) if $actVel$ is low then agent advances 1 position;
   (3) if $actVel$ is $medium$ then agent advances 3 positions;
   (4) if $actVel$ is $high$ then agent advances 4 positions;
c) for vertical movements (up or down in opposite with inclination)
   (1) if actVel is stop then agent advances 0 positions;
   (2) if actVel is low then agent advances 1 position;
   (3) if actVel is medium then agent advances 1 positions;
   (4) if actVel is high then agent advances 2 positions;

The agent selects and executes an action (in time step t) and receives the immediate reward r_t given to:

a) If agent falls out of the grid then $r_t = -5$;
b) If agent reaches the goal state (G) then $r_t = +20$;
c) $r_t = -1$ in other case (this negative reward is used to find a fast solution in time steps).

Considering the agent position, direction and velocity, this problem has 1600 distinct states; for that it is necessary to use an efficient policy in order to select actions correctly.

3. Actions Combination

Under the conditions described above, given a state s_t, the agent selects an action a_t, receives a reward r_{t+1} and finds itself in a new state s_{t+1}. In order to solve the described gridworld problem, SARSA [14] algorithm is used. In this case the agent has estimated values of every possible action to execute for each state:

$$Q(s, a) \tag{3}$$

where:
$s \in S$ is the environment state;
$a \in A$ is the selected action.
The estimated values actualization is made by using:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{4}$$

where:
$s' \in S$ is the next state;
$a' \in A$ is the selected action state s';
r is the obtained reward;
α is the learning rate;
γ is the discount factor.

The SARSA behavior is showed in Algorithm 1 [14]. In line 1 $Q(s, a)$ matrix is initialized with zeros because this allows improvement of the initial exploration of unknown states. In line 3 the initialization of s is made at random, then an action a from s is selected using ε-greedy policy. For every step of episode (lines 5-10), the selected action a is executed, the reward r and next state s' are observed and used to update the $Q(s, a)$ matrix.

Algorithm 1. SARSA Algorithm.

1. Initialize $Q(s, a)$ with zeros.
2. Repeat in each episode:
3. Initialize s
4. Choose a from s using policy derived from Q
5. Repeat for each step of episode:
6. Take action a, observe r, s'
7. Choose a' from s' using policy derived from Q
8. $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s, a) - Q(s, a)]$
9. $s \leftarrow s'; a \leftarrow a'$
10. Until s is terminal

By using SARSA algorithm, the agent selects just one action in actual state and, in many domains, this is sufficient to find an adequate action execution policy. Nevertheless, actions combination can help to find alternative solutions

provided by two or more actions at the same time. Normally, action combination constantly occurs in real life, for example when a person learns the relationship between velocity and direction driving a car. For combination, a simple method is proposed: start the learning process with basic actions and, incrementally, create new ones using actions which have better values of $Q(s,a)$ matrix. Then, new actions are added to the basic actions and they are available to be selected.
In Algorithm 2 the actions combination process can be observed. After updating $Q(s,a)$ matrix and new state assignation (lines 9, 10), the action combination with low probability ϑ is realized. In line 11 two actions from basic set of actions are selected by using softmax algorithm [14]. Softmax algorithm uses a Gibbs distribution:

$$p(select\ action\ a) = \frac{e^{Q_t(s,a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(s,b)/\tau}} \quad (5)$$

where:
n is the number of basic actions;
τ is the temperature parameter.

Softmax ensures to select best actions according to their $Q(s,a)$ value, and by varying the temperature parameter, it is possible to obtain a good balance between exploration and exploitation.
Finally, in lines 12 and 13 a new action is created and added to action set. These new actions are not basic, and they cannot be used to create new ones.

Algorithm 2. SARSA Algorithm with Actions Combination Process.
1. Initialize $Q(s,a)$ with zeros.
2. Repeat in each episode:
3. Initialize s
4. Choose a from s using policy derived from Q
5. Repeat for each step of episode:
6. Take action a, observe r, s'
7. Choose a' from s' using policy derived from Q
8. $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s,a) - Q(s,a)]$
9. $s \leftarrow s'; a \leftarrow a'$
10. Do action combination with ϑ probability:
11. Choose a_1 and a_2 from basic actions using softmax policy
12. Create new action na
13. Add na to Action set A
14. Until s is terminal

4. Simulations and Results.

In order to prove the actions combination process, the simulations were realized using the gridworld problem described in section 2. The main goal was to test the agent's efficiency for combining actions aiming to go from start position to goal position in the grid minimizing the time step quantity.
First, the SARSA algorithm was implemented and tested. Then, the actions combination process was incorporated with the following simulation parameters:

- episodes: 20000;
- maximum steps for episode: 1000;
- learning rate α: 0.3;
- discount factor γ: 0.9;
- action combination probability ϑ: 0.05;
- temperature parameter τ: $20 - (0.01999 \times actual\ step)$;
- action selection policy derived from Q: ε-greedy with $\varepsilon = 0.1$.

The implementation of SARSA algorithm initializes the $Q(s,a)$ matrix with zeros. The initial state is set randomly; this allows improvement of the exploration process and, consequently, the computation time decreases. The agent uses the six basic actions ($mDir, turnLeft, turnRight, mVel, Accel, Deacel$) to find a solution path, executes the selected action and the state is actualized. The actions combination process uses the same SARSA algorithm, but the six basic actions are used to create new ones. The obtained results are summarized in Tables 1 and 2, where the time step quantity is shown (GNR stands for main goal not reached).
As it can be observed in Table 1, at episode 10000 the goal is achieved between 15 and 19 steps. These values are maintained for the rest of episodes. From episode 14000 the goal is reached at all times. The same behavior is observed in Table 2, but the steps quantity is reduced to values between 9 and 12.

Table 1. Steps Quantity using SARSA Algorithm

	Episode #									
Iteration #	2000	4000	6000	8000	10000	12000	14000	16000	18000	20000
1	GNR	GNR	GNR	GNR	GNR	GNR	17	19	17	17
2	GNR	GNR	GNR	GNR	16	17	16	17	16	16
3	GNR	GNR	GNR	16	16	20	17	17	17	20
4	GNR	GNR	GNR	20	16	16	16	16	17	17
5	GNR	GNR	GNR	GNR	16	16	16	16	16	16
6	GNR	GNR	GNR	18	19	16	16	18	17	16
7	GNR	GNR	GNR	22	19	17	17	17	17	17
8	GNR	GNR	GNR	GNR	GNR	19	19	19	17	18
9	GNR	GNR	GNR	15	15	GNR	15	16	19	16
10	GNR	GNR	GNR	GNR	GNR	19	17	17	17	17

The existence of a variation in quantities of steps for different iterations can be showed in both Tables. This occurs because learning and exploration parameters are invariable in the implementations. This can solved decreasing their values when time steps increase.

Table 2. Steps Quantity using SARSA Algorithm with Actions Combination Process

	Episode #									
Iteration #	2000	4000	6000	8000	10000	12000	14000	16000	18000	20000
1	GNR	GNR	9	15	GNR	9	9	9	9	9
2	GNR	GNR	GNR	14	GNR	GNR	10	11	10	10
3	GNR	GNR	GNR	19	9	10	10	10	10	10
4	GNR	GNR	15	10	9	9	9	9	11	9
5	GNR	GNR	14	GNR	11	11	10	10	10	11
6	GNR	GNR	11	9	13	9	9	9	9	10
7	GNR	GNR	13	9	9	10	10	10	10	10
8	GNR	GNR	GNR	11	11	10	10	12	11	11
9	GNR	GNR	GNR	GNR	GNR	GNR	9	12	10	9
10	GNR	GNR	GNR	22	12	9	9	9	9	9

In actions combination alternative, the number of $Q(s,a)$ matrix entries grows significantly and the required computation time is considerably bigger. For this case, the algorithm generates 21 new actions plus 6 basic actions.

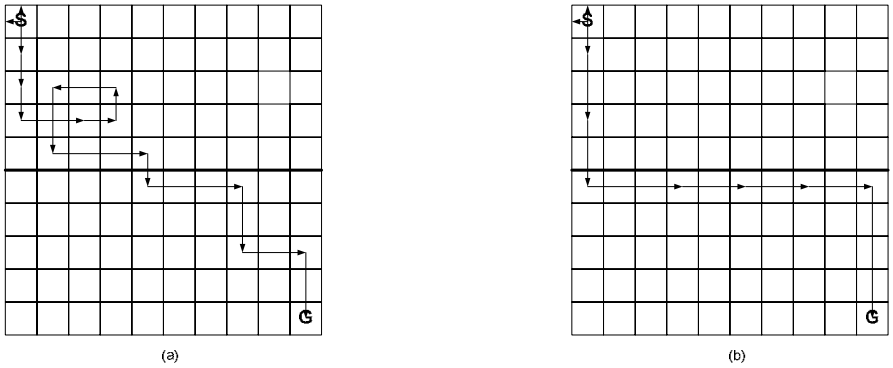

Figure 2. Paths Founded Using Basic Actions and Actions Combination.

In Figure 2 the behavior of SARSA (Figure 2 (a)) and SARSA with actions combination (Figure 2 (b)) at ending of learning process can be observed. In both cases, the agent finds a path from S to G. However, the difference is that the path found by actions combination alternative is more direct.

For these simulations, none of the combination heuristics were considered; therefore, all combinations of two basic actions were realized and tested during the learning process. This might be inadequate when the actions number is large and many combinations are not valid or do not produce some state change.

5. Discussion and Future Work.

The actions combination method presented in this work can be implemented in RL algorithms, and it is an interesting mechanism to find optimal solutions on control process problems based on known algorithms (for example SARSA).

Actions combination mechanism tends to discover if it is possible to use two o more actions in a particular state instead of one. This is an important issue as it is not necessary to program all possible combinations and restrictions on problem domain. However, since the number of combinations can grow exponentially, it is possible to implement an action clustering process in order to reduce the combinations to actions of different clusters.

The actions combination mechanism can be improved by using heuristics about particular domain or action-state values. Clearly, that implies further complexity in the implementation and, consequently, the computation time grows.

The actions combination heuristics and their improvement are focused on preceding actual work. Also, structural abstractions [7], multiple objective problems [5] and [7] are two topics under consideration with the intention of improving the algorithms implementation.

Acknowledgements

This work has been supported by the EIINRE571project (National Technological University, Argentina).

References

[1] Agre, P. A. and Chapman, D. What are plans for? *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back. Cambridge MA: MIT Press*. (1990), pp. 17-34.

[2] Bowling, M. and Veloso M.. An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning. *Technical report CMU-CS-00-165. Computer ScienceDepartment, Carnegie Mellon University.* (2000)

[3] Boyan, J.A., Littman, M.L. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. *Advances In Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA*, (1994), pp. 671-678.

[4] Crites, R. H. and Barto, A. G. Improving Elevator Performance Using Reinforcement Learning. *Advances in Neural Information Processing Systems 8, Conf., MIT Press, Cambridge, Mass.* (1996), pp.1017-1023.

[5] Dietterich, T.G. The MAXQ Method for Hierarchical Reinforcement Learning. *In Proceedings of the Fifteenth International Conference on Machine Learning.* (1998), pp. 118-126.

[6] Erev, I. and Roth, A. Predicting How People Play Games: Reinforcement Learning in Experimental Games with Unique Mixed Strategy Equilibria. *American Economic Review 8*, (1998), pp. 848-881.

[7] Fitch, R., Hengst, B., Suc, Dorian, Calbert, G., and Scholz, J. Structural Abstraction Experiments in Reinforcement Learning. *In Procceding Australian joint conference on artificial intelligence No18, Sydney, AUSTRALIE* vol. 3809, (2005), pp. 164-175.

[8] Kaelbling, L.P., Littman L.M. and Moore, A.W. Reinforcement Learning: a Survey. *Journal of Artificial Intelligence Research*, vol. 4, (May 1996), pp. 237–285.

[9] Kozierok , R. and Maes, P. A Learning Interface Agent for Scheduling Meetings. *In Proceeding 1st international conference on Intelligent user interfaces, Orlando, Florida, United States* (1993), pp.81-88.

[10] Lin, L. J. Self-Improving Reactive Agents Based on Reinforcement Learning, Planning, and Teaching. *Machine Learning,* Vol 8, (1992), pp. 293-321.

[11] McGovern, A., Moss, E. and Barto, A. G. Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts. *Machine Learning*, Vol 49 No 2-, (2002), pp 141–160.

[12] Peshkin, S. and Savova, V. Reinforcement Learning for Adaptive Routing. *In Proceedings of the International Joint Conference on Neural Networks (IJCNN),* (2002), pp. 1825-1830.

[13] Peters J., Vijayakumar S. and Schaal, S. Reinforcement Learning for Humanoid Robotics. *In Proceeding Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots, Karlsruhe, Germany,* (2003), pp. 2002.

[14] Sutton, R.S. and Barto, A.G. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, Massachusetts. 1998.

[15] Watkins, C.J. C. H. and Dayan, P. Q-Learning. *Machine Learning*, Vol 8 No. 4, (1992), pp. 279-292.

[16] Zhang, W. and Dietterich, T. G. A Reinforcement Learning Approach to Job-Shop Scheduling. *In Proceeding 1995 International Joint Conference on Artificial Intelligence, AAAI/MIT Press, Cambridge, MA.* (1995), pp. 1114-1120.