JCS&T Vol. 10 No. 1 April 2010

Using Combination of Actions in Reinforcement Learning

Marcelo J. Karanik
marcelo@frre.utn.edu.ar
Sergio D. Gramajo
sergiogramajo@gmail.com

Artificial Intelligence Group
National Technological University
Resistencia (3500), Chaco - Argentina

ABSTRACT

Software agents are programs that can observe their environment and act in an attempt to reach their design goals. In most cases the selection of particular agent architecture determines the behaviour in response to the 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 and it 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 estimations about every action are updated and, after a certain period of time, the agent learns which the best action to be executed is. Most reinforcement learning algorithms perform simple actions while two or more are capable of being used. 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, SARSA, Optimal Policy, Action Combination.

1. INTRODUCTION

By using Reinforcement Learning (RL) techniques, an agent interacts with its environment to achieve a goal. Agent attempts to reach the objective based on learning by trial-and-error [7] [13], then what to do and how to do it to optimally achieve its goal through mapping situations to actions must be learnt[8]. To reach the optimal solution, it is necessary to maximize a numerical reward signal. The agent must discover by itself what action should be taken in order to maximize the reward signal. It affects agent status and not only does it have immediate reward but also has it through subsequent actions [7], [13].

There are two RL basic characteristics; 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 [2], [13].

The goal of RL is to program agents that learn by reward and punishment (negative reward), being how the task is performed needless to specify [7].

This kind of learning performed by RL is unsupervised because the agent learns by itself and it is not necessary to include input-output pairs provided by an external expert, such as Artificial Neural Network (ANN) methods [7]. 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 [13]. The problems with delayed reinforcement are well modeled with Markov Decision Processes [13]. 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 a 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 a.

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 at two different times may result in a different next state and/or different reinforcement values [7]. 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. The reinforcement learning paradigm described has been successfully implemented for many well-defined problems such as games theory [3], [8]; robotics [5], [9]; scheduling [1], [6], [12], [17]; telecommunications [4], [14], elevators controls [11], etc.

RL algorithms find an optimal policy to fulfill actions, and sometimes, several of them can be fulfilled in a state. In this paper an alternative to combine actions in a gridworld problem is presented. RL SARSA algorithm and the actions combination method are described in section 2. In section 3 the problem description is made. In section 4 simulations and results are showed, and finally, in section 5, a discussion about some topics and future work are presented.

2. ACTION COMBINATION METHOD

Given the model characteristics described in section 1, 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 this case the agent has estimated values of every possible action to perform for each state:

$$Q(s, a) \qquad (1)$$

where:
$s \in S$ is the environment state;
$a \in A$ is the selected action.

The estimated values actualization is made by using:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)] \qquad (2)$$

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.

This updating is made by using SARSA algorithm, whose behavior is showed in Algorithm 1 [13]. In line 1 $Q(s,a)$ matrix is initialized with five, which is called optimistic initialization. This method makes the stabilization of the system take longer but increases the probabilistic that the actions performed are the best.

In line 3 the initialization of s is made at random, then an action a is selected from s using Softmax policy [13]. Next action is selected using Gibbs distribution:

$$p(select\ action\ a) = \frac{e^{\frac{Q_t(s,a)}{\tau}}}{\sum_{b=1}^{n} e^{\frac{Q_t(s,b)}{\tau}}} \qquad (3)$$

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

Thus, Softmax policy in the exploration is based on the values of $Q(s,a)$, favoring the actions with the highest values of Q because it has more probability of being selected. Softmax ensures the selection of the 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.

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 five.
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 current 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 speed and direction driving a car. For action 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 it is shown that the action combination method is performed after updating $Q(s,a)$ matrix and new state assignation (lines 9, 10), with low probability ϑ. In line 11 two actions from basic set of actions are selected by using Softmax algorithm.

Finally, in lines 12 and 13 a new action is created and added to action set from basic actions. 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 five.
2.  Repeat in each episode:
3.    Initialize $s$
4.    Choose $a$ from $s$ using derived from $Q$
5.    Repeat for each step of episode:
6.      Take action $a$, observe $r, s'$
7.      Choose $a'$ from $s'$ using 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 $\vartheta$ probability:
11.       Choose basic $a_1$ and $a_2$ using softmax policy
12.       Create new action $na$
13.       Add $na$ to Action set $A$
14.   Until $s$ is terminal
```

3. PROBLEM DESCRIPTION

To test the action combination algorithm, a simplification of Wumpus World problem [15] is used. It consists of an agent inside a network of rooms connected to each other. In some of these rooms is the Wumpus, a monster that devours all that enters there. There are also gaps, where the agent can fall. The agent's objective is to find a pile of gold, for which it must overcome obstacles. An example of this environment is showed in Figure 1. In addition, the agent can kill the Wumpus using the only arrow it has.

The problem consists of a $N \times M$ grid where $5 \le N \le 15$ and $5 \le M \le 15$. The agent must go from initial position set by the user to goal across the grid without falling into the gap or being killed by Wumpus.

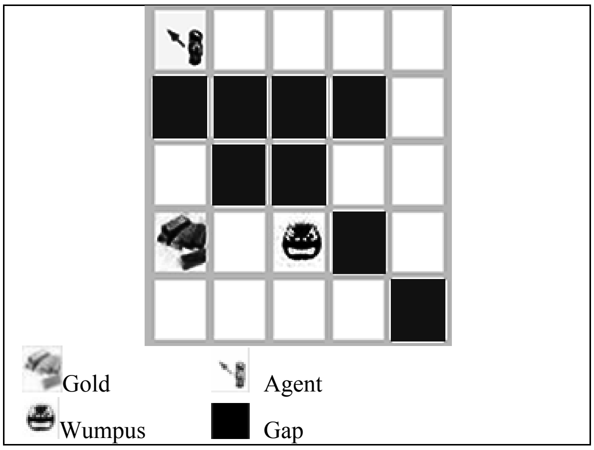


Figure 1. Gridworld Environment.

The state set is specified through the following variables:

$$S = \{xPos, yPos, actDir, agArr, stWum\} \qquad (4)$$

where:
$xPos$ is the grid row subindex ($1 \le xPos \le N$);
$yPos$ is the grid column subindex ($1 \le yPos \le M$);
$actDir$ is the agent's direction
$agArr$ indicates if the agent has an arrow
$stWum$ is the state of Wumpus (dead or alive)

The agent can use the following four actions:

$$A = \{turnLeft, turnRight, stepForward, shoot\} \quad (5)$$

where:
$turnLeft$ means the agent turns 45° to the left;
$turnRight$ means the agent turns 45° to the right;
$stepForward$ means that the agent moves forward
$shoot$ means the agent shoots the arrow.

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

a) If agent dies then r_t= -15;
b) If agent reaches the goal state, then r_t= 25;
c) r_t= -1 in other case (this negative reward is used to find a fast solution);
d) If agent kills the Wumpus then r_t= 22;
e) If agent shoots without arrow then r_t= -12;
f) If agent shoots whith arrow but does not kill the Wumpus then r_t= -9;

Considering the agent position, its direction, if the agent has an arrow and the status of the Wumpus, this problem can have many distinct states; for which it is necessary to use an efficient policy in order to select actions correctly.

4. SIMULATIONS AND RESULTS

In this section 3 environments of different sizes to obtain simulations results are presented, each of these with 10 episodes (the simulations was made using VBasic dot net™)[1]. These tests allow observing the benefits of SARSA Algorithm using simple actions and combined actions.
To do it, parameters with their defined values listed below were used:

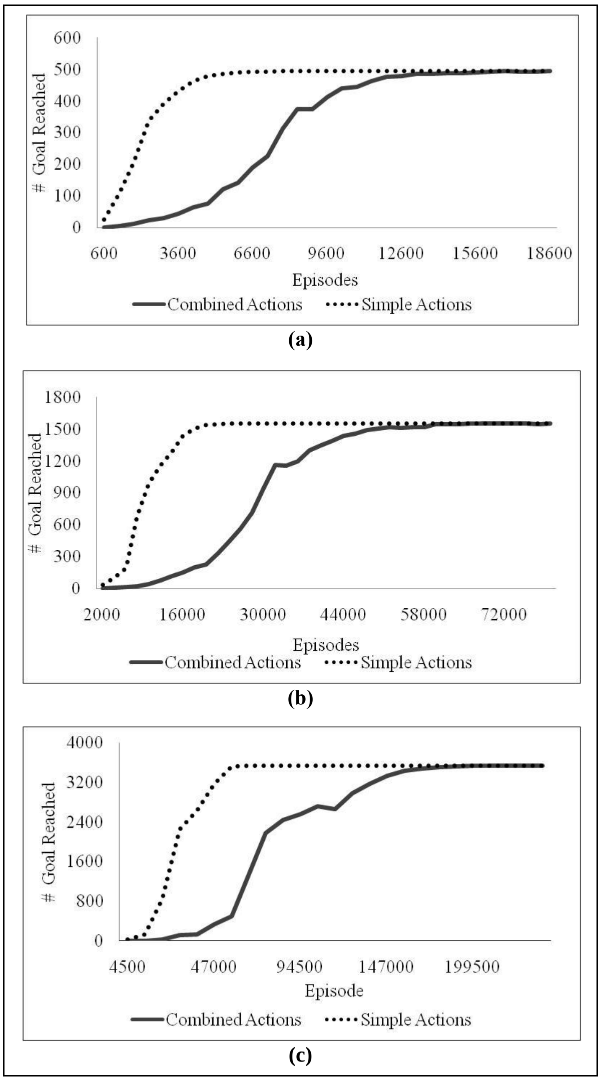
- learning rate α: 0.3;
- discount factor γ: 0.9;
- action combination probability ϑ : 0.005;
- temperature parameter τ: (η * 400), with η = 0.1 – 0.006 * Number_of_Episodes if η = 0 then τ =1;
- function valor: SOFTMAX;
- number of episodes: 10;
- initial fixed test position for three environments:
  Environment of 5x5 and 10x10: 0,0,6,1,1.
  Environment of 15x15: 14,14,2,1,1.

In table 1 the number of action required from initial states to reach the goal can be observed.
It is seen that the agent can learn faster with the use of simple actions, but the advantage of combined actions method is that fewer steps are required to reach the goal. This happens because when combined actions are implemented, the relationship of states-actions to explore on environment is four times greater than with simple actions.
In simulation a control break for each domain is performed, which is useful to verify if agent is able to reach the goal from the fixed test position. Thus, in Figure 2 learning curves are showed, i.e. the relationship between the amount of learning episodes and the number of states from which they can achieve the goal.

Table 1. Results of simple actions and combined actions

Grid Zize	Algorithm Type	Actions Used to Reach Goal	Number of Learning Episodes	
			Min	Max
5x5	Simple Actions	16	5940	7560
	Combined Actions	4	14040	19620
10x10	Simple Actions	20	24000	28800
	Combined Actions	7	67200	86400
15x15	Simple Actions	26	62400	63000
	Combined Actions	11	210000	241500

When training begins learning is slow because the agent explores the environment. Then there is an accelerated growth in the number of times the agent reaches the goal or ends state. And finally the agent knowledge is stabilized by exploiting the knowledge acquired.



(a)

(b)

(c)

Figure 2. Learning simple actions and combined actions.

Domains depicted in Figure 3(a), 3(b) and 3(c) show the number of states from where agent can reach the goal state, which are 495, 1551 and 3568 respectively. Gaps, Goal and Wumpus are not initial states and they reduce the numbers above.

[1] Programmers: Diana V. Cabrera and Leandro A. Varone (System Engineering undergraduate students)

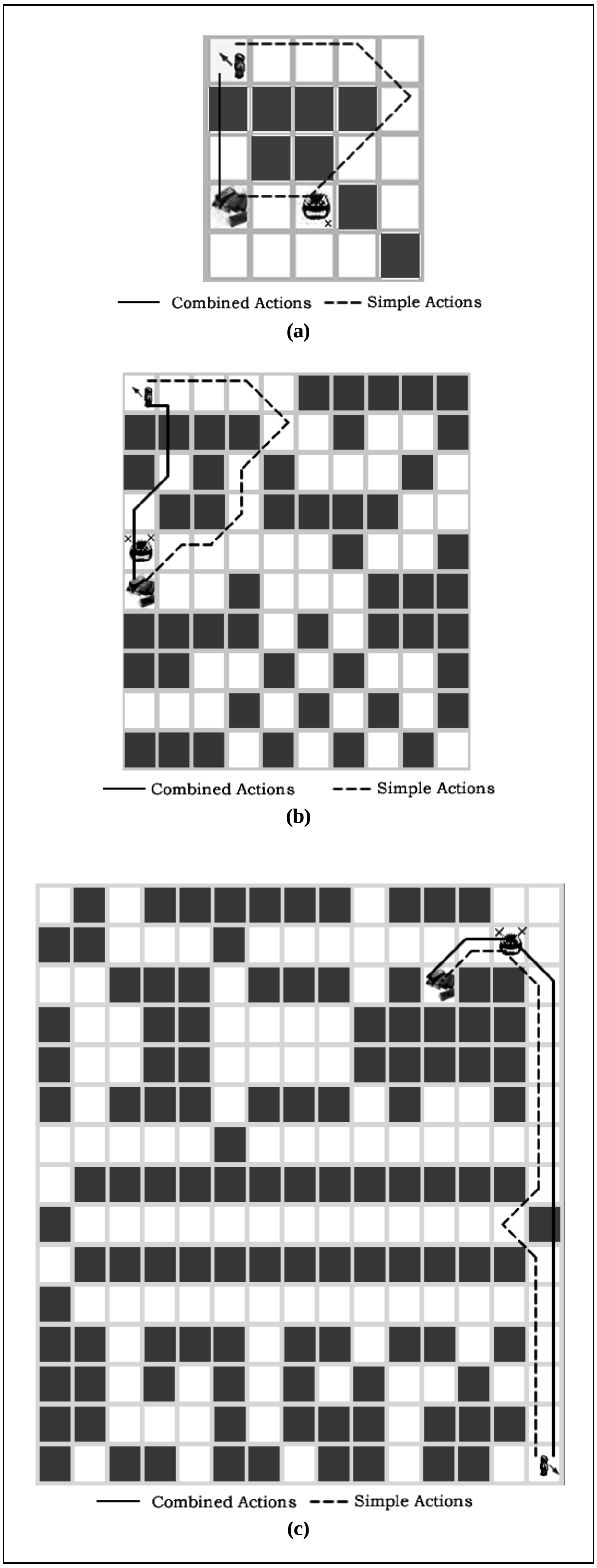


Figure 3. Optimal Paths to Reach Goal

This behavior is the same (for different sizes, configurations and types of actions) save for a variation in the number of episodes in which the system achieves a state of stabilization This number is proportionately related to the number of environment states. Thus, at the end of training, the agent has learned to reach goal state of optimal way, as shown in Figure 3 with gridworld domains of 5x5, 10x10 and 15x15, Figure 3(a), 3(b) and 3(c) respectively used.

In Figure 3(a) it can be observed that the agent combines the stepForwrd action twice in order to avoid falling into the gap (full line). This new combined action can be viewed as a jump. Also, by using this new action, the agent keeps away from Wumpus and does not need to kill it. The number of necessary actions to reach the goal decreases considerably using combination method.

The same behavior can be observed in Figures 3(b) and 3(c). In Figure 3(b) for both paths the agent kills the Wumpus, but by using combined actions it modifies the trajectory in order to arrive at goal faster.

Figure 3(c) shows the same path using combined and simple actions. The main difference is the number of actions used to reach the goal. For example, in final steps, the agent combines turnLeft-stepForward and stepForward-shoot in order to kill Wumpus in just two actions. The suitability of decisions is possible given the fact that the knowledge acquired assigns best values to combined action.

5. CONCLUSIONS 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 to control process problems based on known algorithms (for example SARSA).

Action combination mechanism tends to discover if it is possible to use two o more actions in a particular state instead of one. For the Wumpus world problem, using action combination allows to observe that combining stepForward action twice, the agent discovers a new action that can be considered as a jump. This new action can be used to skip pits and the agent selects it only for those occasions. 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 action combination method explores extra actions that may not have been considered part of the solution. This is very important for solving problems requiring the application of several actions in one state because it is a simple mechanism to program.

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 [10], multiple objective problems [16] and [10] 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] A. McGovern, E. Moss and A. G. Barto, "Building a Basic Block Instruction Scheduler with Reinforcement Learning and Rollouts", Machine Learning, vol 49, No 2, 2002, pp 141–160.

[2] C. J. C. H. Watkins and P. Dayan, "Q-Learning", Machine Learning, vol 8 No. 4, 1992, pp. 279-292.

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

[4] J. A. Boyan and M. L. Littman, "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.

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

[6] L. J. Lin, "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning, and Teaching", Machine Learning, vol 8, 1992, pp. 293-321.

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

[8] M. Bowling and M. Veloso, "An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning", Technical report CMU-CS-00-165. Computer Science Department, Carnegie Mellon University, 2000.

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

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

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

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

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

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

[15] S. Russell and P. Norvig, Inteligencia Artificial: Un enfoque moderno. Naucalpan de Juárez Edo. Mexico: Prentice Hall, 1996.

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

[17] W. Zhang and T. G. Dietterich, "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.

Received: June 2008. Accepted: Nov. 2009