Combining Learning Algorithms: An Approach to Markov Decision Processes

Richardson Ribeiro[1(✉)], Fábio Favarim[1], Marco A. C. Barbosa[1], Alessandro L. Koerich[2], and Fabrício Enembreck[2]

[1] Graduate Program in Computer Engineering,
Federal Technological University of Paraná, Pato Branco, Paraná, Brazil
{richardsonr, favarim, marcocb}@utfpr.edu.br
[2] Post-Graduate Program in Computer Science,
Pontificial Catholical University of Paraná, Curitiba, Paraná, Brazil
{koerich, enembreck}@ppgia.pucpr.br

Abstract. In this paper we present a technique for estimating policies which combines instance-based learning and reinforcement learning algorithms in Markovian environments. This approach has been developed for speeding up the convergence of adaptive intelligent agents that using reinforcement learning algorithms. Speeding up the learning of an intelligent agent is a complex task since the choice of inadequate updating techniques may cause delays in the learning process or even induce an unexpected acceleration that causes the agent to converge to a non-satisfactory policy. Experimental results in real-world scenarios have shown that the proposed technique is able to speed up the convergence of the agents while achieving optimal policies, overcoming problems of classical reinforcement learning approaches.

Keywords: Reinforcement learning · Dynamic environments · Adaptive agents

1 Introduction

Markov Decision Processes (MDP) are a popular framework for sequential decision-making for single agents, when agents' actions have stochastic effect on the environment state and need to learn how to execute sequential actions. Adaptive intelligent agents emerge as an alternative to cope with several complex problems including control, optimization, planning, manufacturing and so on. A particular case is an environment where events and changes in policy may occur continuously (i.e., dynamic environment). A way of addressing such a problem is to use Reinforcement Learning (RL) algorithms, which are often used to explore a very large space of policies in an unknown environment by trial and error. It has been shown that RL algorithms, such as the Q-Learning algorithm [1], converge to optimal policies when a large number of trials are carried out in stationary environments [2].

J. Cordeiro et al. (Eds.): ICEIS 2012, LNBIP 141, pp. 172–188, 2013.
DOI: 10.1007/978-3-642-40654-6_11,

Several works using RL algorithms and adaptive agents in different applications can be found in the literature [3–9]. However, one of the main drawbacks of RL algorithms is the rate of convergence which can be too slow for many real-world problems, e.g. traffic environments, sensor networks, supply chain management and so forth. In such problems, there is no guarantee that RL algorithms will converge, since they were originally developed and applied to static problems, where the objective function is unchanged over time. However, there are few real-world problems that are static, i.e. problems in which changes in priorities for resources do not occur, goals do not change, or where there are tasks that are no longer needed. Where changes are needed through time, the environment is dynamic.

In such environments, several approaches for achieving rapid convergence to an optimal policy have been proposed in recent years [10–13]. They are based mainly on the exploration of the state-action space, leading to a long learning process and requiring great computational effort.

To improve convergence rate, we have developed an instance-based reinforcement learning algorithm coupled with conventional exploration strategies such as the ε-greedy [14]. The algorithm is better able to estimate rewards, and to generate new action policies, than conventional RL algorithms. An action policy is a function mapping states to actions by estimating a probability that a state s' can be reached after taking action a in state s.

In MDP, algorithms attempt to compute a policy such that the expected long-term reward is maximized by interacting with an environment [2]. The approach updates into state-action space the rewards of unsatisfactory policies generated by the RL algorithm. States with similar features are given similar rewards; rewards are anticipated and the number of iterations in the Q-learning algorithm is decreased.

In this paper we show that, even in partially-known and dynamic environments, it is possible to achieve a policy close to the optimal very quickly. To measure the quality of our approach we use a stationary policy computed previously, comparing the return from our algorithm with that from the stationary policy, as in [15].

This article is organized as follows: Section 2 introduces the RL principles and the usage of heuristics to discover action policies. The technique proposed for dynamic environments is presented in Sect. 3 where we also discuss the Q-Learning algorithm and the k-Nearest Neighbor (k-NN) algorithm. Section 4 gives experimental results obtained using the proposed technique. In the final section, some conclusions are stated and some perspectives for future work are discussed.

2 Background and Notation

Many real-life problems such as games [16,17], robotics [18], traffic light control [19,20] or air traffic [21,22], occur in dynamic environments. Agents that interact in this kind of environment need techniques to help them, e.g., to reach some

goal, to solve a problem or to improve performance. However because individual circumstances are so diverse, it is difficult to propose a generic approach (heuristics) that can be used to deal with every kind of problem. Environment is the world in which an agent operates.

A dynamic environment consists of changing surroundings in which the agent navigates. It changes over time independent of agent actions. Thus, unlike the static case, the agent must adapt to new situations and overcome possibly unpredictable obstacles [23,24]. Traditional planning systems have presented problems when dealing with dynamic environments. In particular, issues such as truth maintenance in the agent's symbolic world model, and replanning in response to changes in the environment, must be addressed.

Predicting the behavior (i.e., actions) of an adaptive agent in dynamic environments is a complex task. The actions chosen by the agent are often unexpected, which makes it difficult to choose a good technique (or heuristic) to improve agent performance. A heuristic can be defined as a method that improves the efficiency in searching a problem solution, adding knowledge about the problem to an algorithm.

Before discussing related work, we introduce the MDP which is used to describe our domain. A MDP is a tuple $(S, A, \partial^a_{s,s'}, R^a_{s,s'}, \gamma)$ where S is a discrete set of environment states that can be composed by a sequence of state variables $< x_1, x_2, ..., x_y >$. An episode is a sequence of actions $a \in A$ that leads the agent from a state s to s'. $\partial^a_{s,s'}$ is a function defining the probability that the agent arrives in state s' when an action a is applied in state s. Similarly, $R^a_{s,s'}$ is the reward received whenever the transition $\partial^a_{s,s'}$ occurs and $\gamma \in \{0...1\}$ is a discount rate parameter.

A RL agent must learn a policy $Q : S \rightarrow A$ that maximizes its expected cumulative reward [1], where $Q(s, a)$ is the probability of selecting action a from state s. Such a policy, denoted as Q^*, must satisfy Bellman's equation [14] for each state $s \in S$ (Eq. 1).

$$Q(s, a) \leftarrow R(s, a) + \gamma \sum_{s'} \partial(s, a, s') \times \max Q(s', a) \tag{1}$$

where γ weights the value of future rewards and $Q(s, a)$ is the expected cumulative reward given for executing an action a in state s. To reach an optimal policy (Q^*), a RL algorithm must iteratively explore the space $S \times A$ updating the cumulative rewards and storing such values in a table $\hat{Q}$.

In the Q-learning algorithm proposed by Watkins [1], the task of an agent is to learn a mapping from environment states to actions so as to maximize a numerical reward signal. The algorithm approaches convergence to Q^* by applying an update rule (Eqs. (2), (3)) after a time step t:

$$v \leftarrow \gamma \max Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t) \tag{2}$$

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha[R(s_t, a_t) + v] \tag{3}$$

where V is the utility value to perform an action a in state s and $\alpha \in \{0, 1\}$ is the learning rate.

In dynamic environments such as traffic jams, it is helpful to use strategies like ε-greedy exploration [14] where the agent selects an action with the greatest Q value with probability $1 - \varepsilon$. In some Q-Learning experiments, we have found that the agent does not always converge during training (see Sect. 4). To overcome this problem we have used a well known Q-Learning property: actions can be chosen using an exploration strategy. A very common strategy is random exploration, where an action is randomly chosen with probability ε and the state transition is given by Eq. 4.

$$Q(s) = \begin{cases} \max Q(s, a), & \text{if} \quad q > \varepsilon \\ a_{random}, & \text{otherwise} \end{cases} \tag{4}$$

where q is a random value with uniform probability in $[0, 1]$ and $\varepsilon \in [0, 1]$ is a parameter that defines the exploration trade-off. The greater the value of ε, the smaller is the probability of a random choice, and a_{random} is a random action selected among the possible actions in state s.

Several authors have shown that matching some techniques with heuristics can improve the performance of agents, and that traditional techniques, such as ε-greedy, yield interesting results [10,11,25]. Bianchi [11] proposed a new class of algorithms aimed at speeding up the learning of good action policies. An RL algorithm uses a heuristic function to force the agent to choose actions during the learning process. The technique is used only for choosing the action to be taken, while not affecting the operation of the algorithm or modifying its properties. Bianchi et al. [11] also proposed an automatic method based on heuristic propagation to compute a policy in runtime from the structure extracted from the domain. The technique propagates from its goal-state, the correct actions which would lead to that state. Drummond [25] proposes a method that accelerates RL by transferring parts of previously learned solutions to a new problem, exploiting the results of prior learning to speed up the process. The method is called *reusing policies*, where the heuristics based on the cases are used as the state transition rules to choose the a_t action, taken in the s_t state. Then, ε-greedy random exploration strategies are used to estimate the T transition probability functions and the R reward.

Butz [26] proposes the combination of an online model learner with a state value learner in a MDP. The model learner learns a predictive model that approximates the state transition function of the MDP in a compact, generalized form. State values are evaluated by means of the evolving predictive model representation. In combination, the actual choice of action depends on anticipating state values given by the predictive model. It is shown that this combination can be applied to increase further the learning of an optimal policy.

Koenig and Simmons [27] proposed a method that stores the reinforcements received during the learning of an agent in a separated structure. This structure has a table storing the reinforcements received in each state and a threshold is used to generate a map of the positions where the agent received more reinforcements. Therefore, the tables that keep the positive and negative reinforcements

contain information for defining the heuristics, together with information about the actions to be carried out.

Bianchi et al. [28] improved action selection for online policy learning in robotic scenarios combining RL algorithms with heuristic functions. The heuristics can be used to select appropriate actions, so as to guide exploration of the state-action space during the learning process, which can be directed towards useful regions of the state-action space, improving the learner behavior, even at initial stages of the learning process.

In this paper we propose going further in the use of exploration strategies to achieve a policy closer to the Q^*. To do this we have used policy estimation techniques based on an instance learning, such as the k-Nearest Neighbors (k-NN) algorithm. We have observed that is possible to reuse previous states, eliminating the need of a prior heuristic.

3 *k*-NR: Instance-Based Reinforcement Learning Approach

In RL, learning takes place through a direct interaction of the algorithm with the agent and the environment. Unfortunately, the convergence of the RL algorithms can only be reached after an exhaustive exploration of the state-action space, which usually converges very slowly. However, the convergence of the RL algorithm may be accelerated through the use of strategies dedicated to guiding the search in the state-action space.

The proposed approach, named k-Nearest Reinforcement (k-NR), has been developed from the observation that algorithms based on different learning paradigms may be complementary to discover action policies [29]. The information gathered during the learning process of an agent with the Q-Learning algorithm is the input for the k-NR. The reward values are calculated with an instance-based learning algorithm. This algorithm is able to accumulate the learned values until a suitable action policy is reached.

To analyze the convergence of the agent with the k-NR algorithm, we assume a generative model governing the optimal policy. With such a model it is possible to evaluate the learning table generated by the Q-Learning algorithm. To do this, an agent is inserted into a partially known environment with the following features:

1. Q-Learning algorithm: learning rate (α), discount factor (γ) and reward (r);
2. Environment E: the environment consists of a state space where there is an initial state ($s_{initial}$), a goal state (s_{goal}) and a set of actions A=$\{\uparrow,\downarrow,\rightarrow,\leftarrow\}$, where $\uparrow$, $\downarrow$, $\rightarrow$, $\leftarrow$ mean respectively *east*, *south*, *north* and *west* (Fig. 1).

A state s is an ordered pair (x, y) with positional coordinates on the axis X and Y respectively. In other words, the set of states S represents a discrete city map. A status function $st : S \rightarrow ST$ maps states and traffic situations where $ST = \{(-0.1, -0.2, -0.3, -0.4, -1.0, 1.0)\}$, where $-0.1, -0.2, -0.3, -0.4, -1.0$

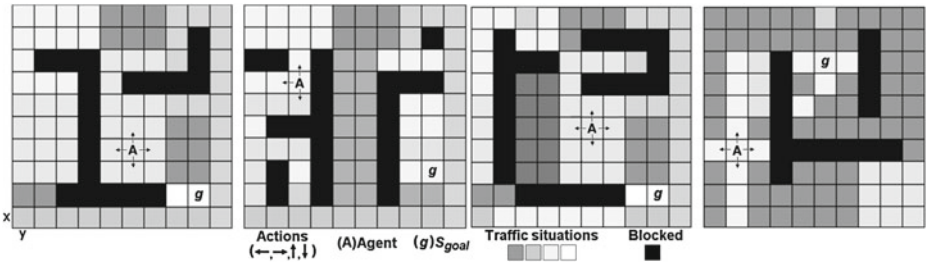

Fig. 1. Environment: A single agent is placed at random positions in the grid, having a visual field depth of 1.

and 1.0 mean respectively *free*, *low jam*, *jam* or *unknown*, *high jam*, *blocked*, and *goal*. After each move (transition) from state s to s' the agent knows whether its action is positive or negative through the rewards attributed by the environment. Thus, the reward for a transition $\partial^a_{s,s'}$ is $st(s')$ and Eqs. (2), (3) is used as update function. In other words, the agent will know if its action has been positive if, having found itself in a state with traffic jam, its action has led to a state where the traffic jam is less severe. However, if the action leads the agent to a more congested status then it receives a negative reward.

The pseudocode to estimate the values for the learning parameters for the Q-Learning using the k-NR is presented in Algorithm 1. The following definitions parameters are used in such an algorithm:

- a set $S = \{s_1, \ldots, s_m\}$ of states;
- an instant discrete steps $step = 1, 2, 3, ..., n$;
- a time window T_x that represents the learning time (cycle(x) of steps);
- a set $A = \{a_1, \ldots, a_m\}$ of actions, where each action is executable in a step n;
- a status function $st : S \rightarrow ST$ where $ST = \{-1, -0.4, -0.3, -0.2, -0.1\}$;
- learning parameters: α=0.2 and γ=0.9;
- a learning table $QT : (S \times A) \rightarrow \mathbb{R}$ used to store the accumulative rewards calculated with the Q-Learning algorithm;
- a learning table $kT : (S \times A) \rightarrow \mathbb{R}$ used to store the reward values estimated with the k-NN;
- $\#changes$ is the number of changes in the environment.

3.1 *k*-NN and *k*-NR

The instance-based learning paradigm determines the hypothesis directly from training instances. Thus, the k-NN algorithm saves training instances in the memory as points in an n-dimensional space, defined by the n attributes which describe them [30,31]. When a new instance must be classified, the most frequent class among the k nearest neighbors is chosen. In this paper the k-NN algorithm is used to generate intermediate policies which speed up the convergence of RL algorithms. Such an algorithm receives as input a set of instances

Algorithm 1: Policy estimation with k-NR.

Require: Learning Table: $QT(s, a)$;
$kT(s, a)$; S=$\{s_1, ..., s_m\}$; A=$\{a_1, ..., a_m\}$
st: S → ST;
Time window T_x;
Environment E;
Ensure:
1. **for all** $s \epsilon$ S **do**
2. **for all** $a \epsilon$ A **do**
3. $QT(s, a) \leftarrow 0$; $kT(s, a) \leftarrow 0$;
4. **end for**
5. **end for**
6. **while** not stop_condition() **do**
7. **CHOOSE** $s \epsilon$ S, $a \epsilon$ A
8. **Update rule:**
9. $Q_{t+1}(s_t,a_t) \leftarrow Q_t(s_t,a_t)+\alpha[\mathrm{R}(s_t,a_t)+v]$ where,
10. $v \leftarrow \gamma \max Q_t(s_{t+1},a_{t+1})-Q_t(s_t, a_t)$
11. step ← step + 1;
12. **if** step $< T_x$ **then**
13. **GOTO**{8};
14. **end if**
15. **if** changes are supposed to occur **then**
16. **for** I ← 1 to #changes **do**
17. **Choose** $s \epsilon$ S
18. $st(s) \leftarrow$ a new status $st \epsilon$ ST;
19. **end for**
20. **Otherwise** continue()
21. **end if**
22. k-NR(T_x, s, a); // *Algorithm 2*
23. **for** $s \epsilon$ S **do**
24. **for** $a \epsilon$ A **do**
25. $QT(s, a) \leftarrow kT(s, a)$
26. **end for**
27. **end for**
28. **end while**
29. **return** (.,.);

generated from an action policy during the learning stage of the Q-Learning. For each environment state, four instances are generated (one for each action) and they represent the values learned by the agent. Table 1 shows some examples of training instances. Each training instance has the following attributes:

1. attributes for the representation of the state in the way of the expected rewards for the actions: north (N), south (S), east (E) and west (W);
2. an action and;
3. reward for this action.

Algorithm 2 shows that the instances are computed to a new table, denoted as kT, which stores the values generated by the k-NR with the k-NN. Such values represent the sum of the rewards received with the interaction with the environment. Rewards are computed using Eq. 6 which calculates the similarity between two training instances $\boldsymbol{s_i}$ and $\boldsymbol{s_m}$.

$$f(\boldsymbol{s_i}, \boldsymbol{s_m}) = \frac{\sum_{x=1}^{x}(s_{i_x} \times s_{m_x})}{\sum_{x=1}^{x} s_{i_x}^2 \times \sum_{x=1}^{x} s_{m_x}^2} \tag{6}$$

Table 1. Training instances.

State (x,y)	Reward (N)	Reward (S)	Reward (E)	Reward (W)	Action Chosen	Reward Action
(2,3)	−0.875	−0.967	0.382	−0.615	(N)	−0.875
(2,3)	−0.875	−0.968	0.382	−0.615	(S)	−0.968
(2,3)	−0.875	−0.968	0.382	−0.615	(W)	0.382
(2,3)	−0.875	−0.968	0.382	−0.615	(E)	−0.615
(1,2)	−0.144	1.655	−0.933	0.350	(N)	−0.144
...	...	...	...	...	...	...

Algorithm 2: k-NR(T_x, s, a).

1. **for all** s ϵ S and s $\neq s_{goal}$ **do**
2. costQT $\leftarrow$ cost(s, s_{goal}, QT)
3. costQ^* $\leftarrow$ cost(s, s_{goal}, Q^*)
4. **if** $costQT_s \neq costQ*_s$ **then**
5. $$kT(s,a) \leftarrow \frac{\sum_{i=1}^{k} HQ_i(.,.)}{k} \quad (5)$$
6. **end if**
7. **end for**
8. **return** $(kT(s,a))$

The cost function (Eq. 7) calculates the cost for an episode (path from a current state s to the state s_{goal} based on the current policy).

$$cost(s, s_{goal}) = \sum_{s \in S}^{s_{goal}} 0.1 + \sum_{s \in S}^{s_{goal}} st(s) \quad (7)$$

Equation 5 used in Algorithm 2 shows how the k-NN algorithm can be used to generate the arrangements of training instances: here, $kT(s,a)$ is the estimated reward value for a given state s and action a, k is the number of nearest neighbors, and $HQ_i(.,.)$ is the i-th existing nearest neighbor in the set of training instances generated from $QT(s,a)$.

Using the k-NR, the values learned by the Q-Learning are stored in the kT table. This contains the best values generated by the Q-Learning and the values that have been estimated by the k-NR.

We have evaluated different ways of generating the arrangements of instances for the k-NN algorithm with the aim of finding the best training sets. First, we used the full arrangement of instances generated throughout runtime. Second, instances generated inside n time windows were selected, where $A_{[T(n)]}$ denotes an arrangement of $T(n)$ windows. In this core, each window generates a new arrangement and previously instances are discarded. We have also evaluated the efficiency rate considering only the arrangement given by the last window $A_{[T(last)]}$. Finally, we have evaluated the efficiency rate of the agent using the last

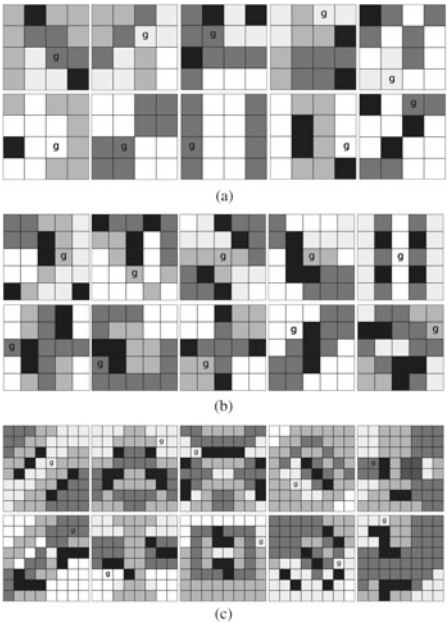

Fig. 2. Simulated environments: (a) 16-state, (b) 25-state, (c) 64-state.

arrangement calculated by the k-NN algorithm - $A_{[T(last),T(k-NN)]}$. The results on these different configurations for generating instances are shown in Sect. 4.

4 Experimental Results

In this section we present the main results obtained from using the k-NR and Q-Learning algorithms. The experiments were carried out in dynamic environments with three different sizes as shown in Fig. 2: 16 (4×4), 25 (5×5) and 64 (8×8) states. Note that a number of states S can generate a long solution space, in which the number of possible policy is $|A||^s|$.

For each environment, ten different configurations were arbitrarily generated to simulate real-world scenarios. The learning process was repeated twenty times for each environment configuration to evaluate the variations that can arise from the agent's actions which are autonomous and stochastic. The results presented in this section for each environment size (16, 25, and 64) are therefore average values over twenty runs. The results do not improve significantly when more scenarios are used (≈2.15 %). The efficiency of the k-NR and Q-Learning algorithms (Y axis in figures) takes into account the number of successful outcomes of a policy in a cycle of steps. We evaluated the agent's behavior in two situations:

1. ♯percent of changes (10, 20, 30) in environment for a window T_x=100. In 64-state environments the changes were inserted after each 1000 steps (T_x=1000)

because in dynamic environments such large environments require many steps to reach a good intermediary policy.
2. ♯percent of changes in environment after the agent finds its best action policy. In this case, we use the full arrangement of learning instances $A_{[T_x]}$, because it gave the best results.

The changes were simulated considering real traffic conditions such as: different levels of traffic jams, partial blocking and free traffic for vehicle flowing. We also allowed for the possibility that unpredictable factors may change traffic behavior, such as accidents, route changing or roadway policy, collisions in traffic lights or intersections, and so on. Changes in the environment were made as follows: for every T_x window, the status of a number of positions is altered random. Equation 8 calculates the number of altered states (♯changes) in T_x.

$$\sharp changes_{(T_x)} = \left(\frac{\sharp states}{100} \right) \times \sharp percent \tag{8}$$

Figure 3 shows the initial experiments with Q-Learning. It is seen that, even with a low change rate in the environment, the agent has trouble converging without the support of exploration strategies. This is because the agent may find a partial policy (a policy found in a cycle of steps before the environment is changed) in response to a certain set of states. To solve this problem, we used the Q-Learning together with the ε-greedy strategy, which allows the agent to explore states with low rewards. With such a strategy, the agent starts to re-explore the states that underwent changes in their status. Figure 4 shows the convergence of the Q-Learning using the ε-greedy strategy in several dynamic environments. More details of the ε-greedy strategy in others scenarios are given in [15]. It can be seen from this experiment that the presence of changed states in an action policy may decrease the agent's convergence significantly. Thus, the reward values that would lead the agent to states with positive rewards can cause the agent to search over states with negative rewards, causing errors. In the next experiments we therefore introduce the k-NR.

4.1 *k*-NR Evaluation

We used the k-NR to optimize the performance of Q-Learning. The technique was applied only to the environment states where changes occur. Thus, the agent modifies its learning and converges more rapidly to a good action policy. Figure 5 shows how this modification in the heuristic affects convergence of Q-Learning. It is seen that Q-Learning converges slowly without the k-NR (Fig. 4). However using k-NR, the agent rapidly converges to a good policy, because it uses reward values that were not altered when the environment was changed.

It is seen that the proposed approach may accelerate convergence of the RL algorithms, while decreasing the noise rate during the learning process. Moreover, in dynamic environments the aim is to find alternatives which decrease the number of steps that the agent takes until it starts to converge again. The k-NR

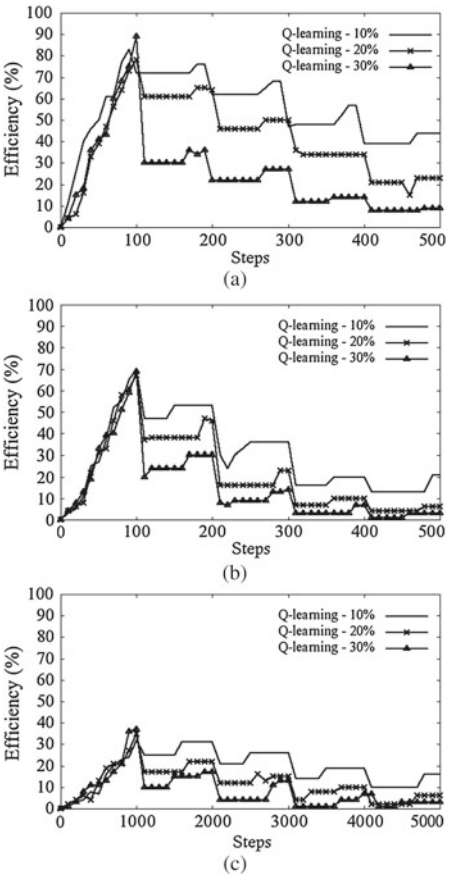

Fig. 3. Performance of the Q-Learning algorithm: (a) 16-state environment, (b) 25-state environment, (c) 64-state environment.

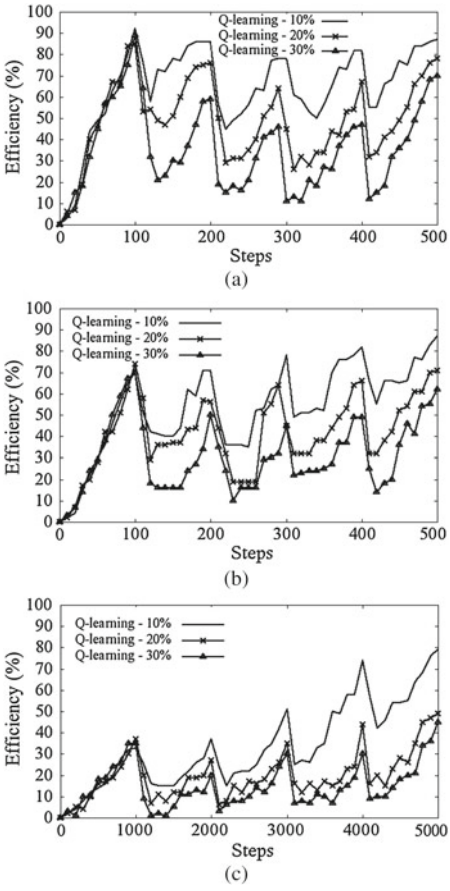

Fig. 4. Performance of the Q-Learning algorithm with ϵ-greedy in a: (a) 16-state environment, (b) 25-state environment, (c) 64-state environment.

algorithm causes the agent to find new action policies, for the states that have had their status altered by the reward values of unaltered neighbor states. In some situations, the agent may continue to convergeeven after a change of the environment. This happens because some states have poor reward values (values that are either too high or too low) as a consequence of too few visits, or too many. Therefore, such states must be altered by giving them more appropriate reward values.

To observe the behavior of the agent in other situations, changes were introduced into the environment only after the agent finds its near-optimal policy (a policy is optimal when the agent knows the best actions). The aim is to analyze the agent's performance when an optimal or near-optimal policy has been

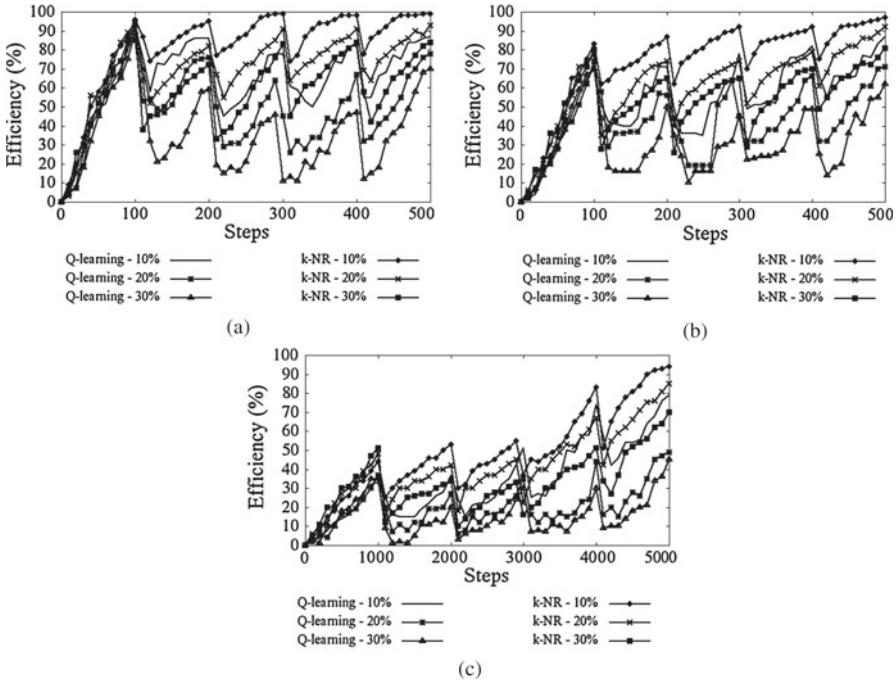

Fig. 5. Performance of the K-NR and Q-Learning algorithms in a: (a) 16-state environment, (b) 25-state environment, (c) 64-state environment.

discovered, and to observe the agent's capacity then to adapt itself to a modified environment.

Enembreck et al. [32] have shown that this is a good way to observe the behavior of an adaptive agent. We have analyzed the agent's adaptation with the k-NR and Q-Learning algorithms. The Q-Learning presents a period of divergence (after some changes were generated), usually a decreasing performance (Fig. 6). However, after a reasonable number of steps, it is seen that there is again convergence to a better policy, as happens when learning begins and performance improves. The decreasing performance occurs because Q-Learning needs to re-explore all the state space, re-visiting states with low rewards to accumulate better values for the future. The ε-greedy strategy helps the agent by introducing random actions so that local maxima are avoided. For example, a *blocked* state that changed to *low jam* must have negative rewards and would no longer be visited.

We used the k-NR algorithm with heuristic to optimize agent performance with the methodology discussed in Sect. 2, which uses instance-based learning in an attempt to solve the problem described in the previous subsection. The heuristic has been applied only to the environment states where changes occur.

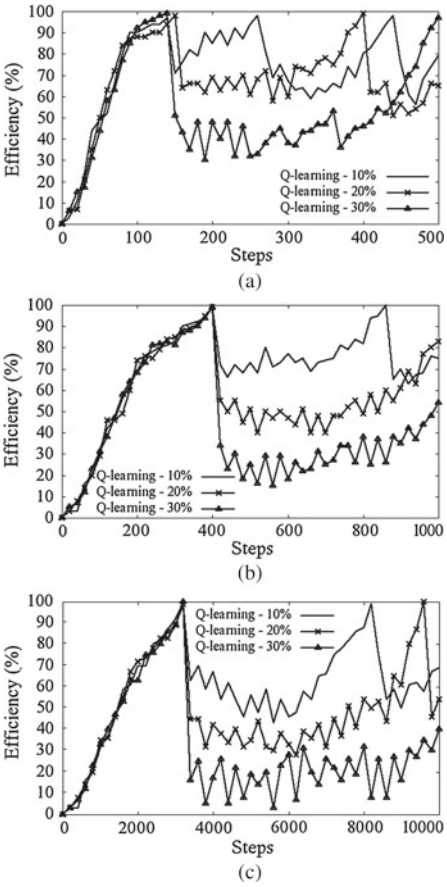

Fig. 6. Agent adaptation using the Q-Learning in a: (a) 16-state environment, (b) 25-state environment, (c) 64-state environment.

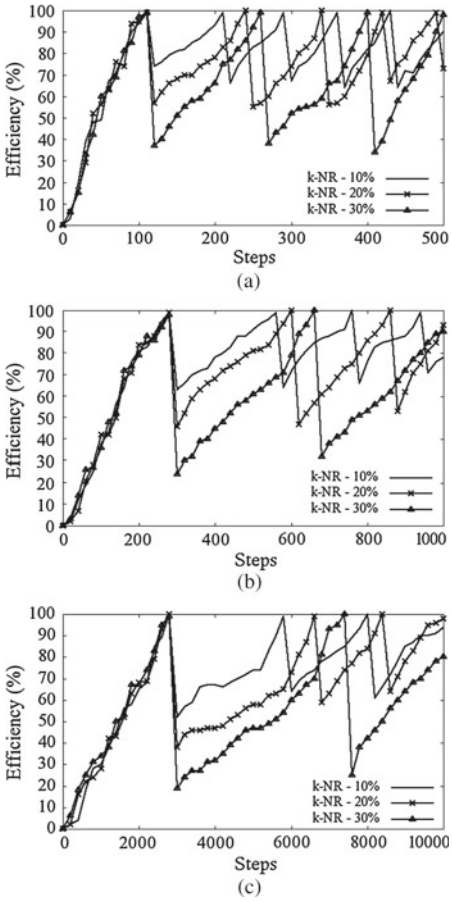

Fig. 7. Agent adaptation using the k-NR in a: (a) 16-state environment, (b) 25-state environment, (c) 64-state environment.

Thus, the heuristic usually caused the agent to modify its learning and converge more rapidly to a good action policy.

Figure 6 also shows that the Q-Learning does not show uniform convergence when compared with k-NR (Fig. 7). This occurs because the k-NR algorithm uses instance-based learning, giving superior performance and speeding up its convergence. The k-NR is able to accelerate the convergence because the states that have had their status altered were estimated from similar situations observed in the past, so that states with similar features have similar rewards.

Table 2 shows the number of steps needed for the agent to refind its best action policy. It is seen that k-NR performs better than standard Q-Learning. In 16-state environments, the agent finds its best policy of actions with 150 steps using the Q-Learning algorithm and 110 steps with k-NR. After changing the

Table 2. Number of steps needed for the agent to find its best policy after changes.

	Before changes		After changes					
			10 %		20 %		30 %	
♯ states	Q	k	Q	k	Q	k	Q	k
16	150	110	130	90	240	140	350	150
25	400	280	440	290	1130	320	2140	380
64	3,430	2,830	5,450	2,950	6,100	3,850	13,900	4,640

environment with 10 %, 20 % and 30 % the k-NR needed 47 % fewer steps on average before it once again finds a policy leading to convergence.

For 25-state environments the agent finds its best action policy in approximately 400 steps using Q-Learning, and in 280 steps with k-NR. In this environment, k-NR uses an average of 30 % fewer steps than Q-Learning, after alteration of the environment. For 64-state environments, the agent needed an average of 3,430 steps to find its best action policy with Q-Learning and 2,830 with k-NR. The k-NR used in average 18 % fewer steps than Q-Learning after environmental change. It is seen that k-NR is more robust in situations where the reward values vary unpredictably. This happens because the k-NN algorithm is less sensitive to noisy data.

5 Discussion and Conclusions

This paper has introduced a technique for speeding up convergence of a policy defined in dynamic environments. This is possible through the use of instance-based learning algorithms. Results obtained when the approach is used show that RL algorithms using instance-based learning can improve their performance in environments with configurations that change. From the experiments, it was concluded that the algorithm is robust in partially-known and complex dynamic environments, and can help to determine optimum actions. Combining algorithms from different paradigms is an interesting approach for the generation of good action policies. Experiments made with the k-NR algorithm show that although computational costs are higher, the results are encouraging because it is able to estimate values and find solutions that support the standard Q-Learning algorithm.

We also observed benefits related to other works using heuristic approaches. For instance, Bianchi et al. [11] proposes a heuristic for RL algorithms that show a significantly better performance (40 %) than the original algorithms. Pegoraro et al. [33] use a strategy that speeds up the convergence of the RL algorithms by 36 %, thus reducing the number of iterations compared with traditional RL algorithms. Although the results obtained with the new technique are satisfactory, additional experiments are needed to answer some questions raised. For example, a multi-agent architecture could be used to explore states placed further from the goal-state and in which the state rewards are smaller. Some of these

strategies are found in Ribeiro et al. [34,35]. We also intend to use more than one agent to analyze situations as: (i) sharing with other agents the learning of the best-performing one; (ii) sharing learning values among all the agents simultaneously; (iii) sharing learning values among the best agents only; (iv) sharing learning values only when the agent reaches the goal-state, in which its learning table would be unified with the tables of the others. Another possibility is to evaluate the algorithm in higher-dimension environments, that are also subject to greater variations. These possibilities will be explored in future research.

References

1. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**(3/4), 279–292 (1992)
2. Ribeiro, C.H.C.: A tutorial on reinforcement learning techniques. In: Proceedings of International Joint Conference on Neural Networks, Washington, USA, pp. 59–61 (1999)
3. Tesauro, G.: Temporal difference learning and td-gammon. Commun. ACM **38**(3), 58–68 (1995)
4. Taylor, M., Stone, P.: Using imagery to simplify perceptual abstraction in reinforcement learning agents. J. Mach. Learn. Res. (JMLR) **10**(1), 1633–1685 (2009)
5. Strehl, A.L., Li, L., Littman, M.L.: Reinforcement learning in finite mdps: Pac analysis. J. Mach. Learn. Res. (JMLR) **10**, 2413–2444 (2009)
6. Stula, M., Stipanicev, D., Bodrozic, L.: Intelligent modeling with agent-based fuzzy cognitive map. Int. J. Intell. Syst. **25**(24), 981–1004 (2010)
7. Walsh, T.J., Goschin, S., Littman, M.L.: Integrating sample-based planning and model-based reinforcement learning. In: Proceedings of 14th Conference on Artificial Intelligence (AAAI'10), vol. 1 (2010)
8. Zhang, C., Lesser, V., Abdallah, S.: Self-organization for cordinating decentralized reinforcement learning. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. AAMAS'10, International Foundation for Autonomous Agents and Multiagent Systems, pp. 739–746 (2010)
9. Wintermute, S.: Using imagery to simplify perceptual abstraction in reinforcement learning agents. In: Proceedings of 24th Conference on Artificial Intelligence (AAAI'10), Atlanta, Georgia, USA, pp. 1567–1573 (2010)
10. Price, B., Boutilier, C.: Accelerating reinforcement learning through implicit imitation. J. Artif. Intell. Res. **19**, 569–629 (2003)
11. Bianchi, R.A.C., Ribeiro, C.H.C., Costa, A.H.R.: Heuristicallyaccelerated Q–learning: A new approach to speed up reinforcementlearning. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS(LNAI), vol. 3171, pp. 245–254. Springer, Heidelberg (2004)
12. Comanici, G., Precup, D.: Optimal policy switching algorithms for reinforcement learning. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), pp. 709–714 (2010)
13. Banerjee, B., Kraemer, L.: Action discovery for reinforcement learning. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), pp. 585–1586 (2010)
14. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)

15. Ribeiro, R., Enembreck, F., Koerich, A.L.: A hybrid learningstrategy for discovery of policies of action. In: Sichman, J.S., Coelho, H., Rezende, S.O. (eds.) IBERAMIA-SBIA 2006. LNCS (LNAI),vol. 4140, pp. 268–277. Springer, Heidelberg (2006)
16. Jordan, P.R., Schvartzman, L.J., Wellman, M.P.: Strategy exploration in empirical games. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10),Toronto, Canada, vol. 1, pp. 1131–1138 (2010)
17. Amato, C., Shani, G.: High-level reinforcement learning in strategy games. In: Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), pp. 75–82 (2010)
18. Spaan, M.T.J., Melo, F.S.: Interaction-driven markov games for decentralized multiagent planning under uncertainty. In: Proceedings of 7th International Conference on AAMAS, Estoril, Portugal, pp. 525–532 (2008)
19. Mohammadian, M.: Multi-agents systems for intelligent control of traffic signals. In: Proceedings of International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce, Sydney, Australia, p. 270 (2006)
20. Le, T., Cai, C.: A new feature for approximate dynamic programming traffic light controller. In: Proceedings of 2th International Workshop on Computational Transportation Science (IWCTS'10), San Jose, CA, USA, pp. 29–34 (2010)
21. Sislak, D., Samek, J., Pechoucek, M.: Decentralized algorithms for collision avoidance in airspace. In: Proceedings of 7th International Conference on AAMAS, Estoril, Portugal, pp. 543–550 (2008)
22. Dimitrakiev, D., Nikolova, N., Tenekedjiev, K.: Simulation and discrete event optimization for automated decisions for in-queue flights. Int. J. Intell. Syst. **25**(28), 460–487 (2010)
23. Firby, R.J.: Adaptive execution in complex dynamic worlds. Ph.D. thesis, Yale University (1989)
24. Pelta, D., Cruz, C., Gonzlez, J.: A study on diversity and cooperation in a multiagent strategy for dynamic optimization problems. Int. J. Intell. Syst. **24**(18), 844–861 (2009)
25. Drummond, C.: Accelerating reinforcement learning by composing solutions of automatically identified subtask. J. Artif. Intell. Res. **16**, 59–104 (2002)
26. Butz, M.: State value learning with an anticipatory learning classifier system in a markov decision process. Technical report, Illinois Genetic Algorithms Laboratory (2002)
27. Koenig, S., Simmons, R.G.: The effect of representation and knowledge on goal-directed exploration with reinforcement learning algorithms. Mach. Learn. **22**(1/3), 227–250 (1996)
28. Bianchi, R.A.C., Ribeiro, C.H.C., Costa, A.H.R.: Accelerating autonomous learning by using heuristic selection of actions. J. Heuristics **14**, 135–168 (2008)
29. Kittler, J., Hatef, M., Duin, R.P.W., Matas, J.: On combining classifiers. IEEE Trans. Pattern Analysis Mach. Intell. **20**(3), 226–239 (1998)
30. Aha, D.W., Kibler, D., Albert, M.K.: Instance-based learning algorithms. Mach. Learn. **6**(1), 37–66 (1991)
31. Galvn, I., Valls, J., Garca, M., Isasi, P.: A lazy learning approach for building classification models. Int. J. Intell. Syst. **26**(8), 773–786 (2011)
32. Enembreck, F., Avila, B.C., Scalabrini, E.E., Barthes, J.P.A.: Learning drifting negotiations. Appl. Artif. Intell. **21**, 861–881 (2007)

33. Pegoraro, R., Costa, A.H.R., Ribeiro, C.H.C.: Experience generalization for multiagent reinforcement learning. In: Proceedings of XXI International Conference of the Chilean Computer Science Society, Punta Arenas, Chile, pp. 233–239 (2001)
34. Ribeiro, R., Borges, A.P., Enembreck, F.: Interaction models for multiagent reinforcement learning. In: Proceedings of International Conferences on Computational Intelligence for Modelling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce; and Innovation in Software Engineering, Vienna, Austria, pp. 464–469 (2008)
35. Ribeiro, R., Borges, A.P., Ronszcka, A.F., Scalabrin, E., Avila, B.C., Enembreck, F.: Combinando modelos de interao para melhorar a coordenao em sistemas multiagente. Revista de Informtica Terica e Aplicada **18**, 133–157 (2011)