Knowledge-Based Systems 32 (2012) 28–36

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Dyna-$\mathcal{H}$: A heuristic planning reinforcement learning algorithm applied to role-playing game strategy decision systems

Matilde Santos, José Antonio Martín H. *, Victoria López, Guillermo Botella

Computer Architectures and Automation, Complutense University of Madrid, Spain

ARTICLE INFO

Article history:
Available online 21 September 2011

Keywords:
Decision-making
Path-finding
Heuristic-search
A-star
Reinforcement-learning

ABSTRACT

In a role-playing game, finding optimal trajectories is one of the most important tasks. In fact, the strategy decision system becomes a key component of a game engine. Determining the way in which decisions are taken (e.g. online, batch or simulated) and the consumed resources in decision making (e.g. execution time, memory) will influence, to a major degree, the game performance. When classical search algorithms such as A^* can be used, they are the very first option. Nevertheless, such methods rely on precise and complete models of the search space so there are many interesting scenarios where its application is not possible, and hence, model free methods for sequential decision making under uncertainty are the best choice. In this paper, we propose a heuristic planning strategy to incorporate, into a Dyna agent, the ability of heuristic-search in path-finding. The proposed Dyna-$\mathcal{H}$ algorithm selects branches more likely to produce outcomes than other branches, just as A^* does. However, unlike A^*, it has the advantages of a model-free online reinforcement learning algorithm. We evaluate our proposed algorithm against the one-step Q-learning and Dyna-Q algorithms and found that the Dyna-$\mathcal{H}$, with its advantages, produced clearly superior results.

1. Introduction

Decision support systems (DSSs) are knowledge-based computer aided information systems that support business or any other organizational decision-making activities. DSSs help to make decisions, which may be rapidly changing and not easily specified in advance. The importance of making a good decision in any business is evident. In a dynamic environment, decision processes not only need to be well designed but they must adapt rapidly to changes in the environment. Existing work on decision making has centered on the concepts of rational and boundedly rational decision processes. Recent works include a third model of decision, based on the use of heuristics. In the last years, there has been an increasing interest in the issues of cost-sensitive learning and decision making, in a variety of applications, in order to maximize the total benefits over time. A number of approaches have been developed that are effective at optimizing cost-sensitive decisions [20,13], some of them based on a synergy between different intelligent techniques and other fields that together comprise what is called knowledge engineering [21].

In any decision making strategy, an agent seeks to achieve a goal, despite uncertainty about its environment. The agent's actions influence the future state of the environment, thereby affecting the options and possible alternatives at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may include foresight or planning [26].

Among all the decisions involved in computer-games, the most common is probably path-finding, i.e., looking for a good route or path for moving an entity from here to there. The entity can be a single person, a vehicle, or a combat unit; the genre can be an action game, a simulator, a role-playing game, or a strategy game. The main focus of this research is to compute collision-free shortest-paths as quickly as possible. Although path-finding is not trivial, there are some well-established, solid algorithms that have been applied, some of them more efficient than others [4,3].

In this paper we use, as the case study, the role-playing games (RPG) scenario, where the player selects a target point (t) from its current position and the entity (e) is automatically taken to t without interacting with the system, avoiding obstacles and optimizing the trajectory. This automatic process can be carried out by different approaches [17]. Most of the searching strategies proposed in the literature are included in the wide area of machine learning [2,23]. When classical search algorithms such as A^* can be used, they are the very first choice for computing optimal solutions. Nevertheless, these methods can be computationally demanding, especially for very large environments. For instance, A^* based

* Corresponding author. Address: Facultad de Informática, Universidad Complutense de Madrid, C. Prof. José García Santesmases, s/n., 28040 Madrid, Spain. Tel.: +34 91 394 7620; fax: +34 91 394 7510.

E-mail addresses: msantos@dacya.ucm.es (M. Santos), jamartinh@fdi.ucm.es (J.A. Martín H.), vlopez@dacya.ucm.es (V. López), gbotella@fdi.ucm.es (G. Botella).

0950-7051/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2011.09.008

algorithms usually demands quite high execution time since the decisions rely on a exhaustive planning strategy. Even more, such methods depend heavily on precise and complete models of the environment, e.g. the game arena. So, there are many interesting scenarios where they cannot be applied. Therefore, model free methods for sequential decision making problems under uncertainty are well suited to these cases since the incremental nature of its learning mechanisms and the direct action selection mechanism of its decision making procedures make it possible to use them in real-time applications.

Many other applications of these learning strategies can be found in the literature. Without being exhaustive, some recent paradigmatic examples can be cited. The airline ticket purchasing problem [11], where author uses different techniques to acquire a flight ticket at the lowest cost. MALADY: A Machine Learning-Based Autonomous Decision-Making System for Sensor Networks [18], where sensor networks are able to learn and make decisions in real time. Muse et al. [24] present a system for visual robotic docking using an omnidirectional camera coupled with the actor critic reinforcement learning (RL) algorithm. In this case, a network trained via reinforcement allows the robot to turn to and approach a table to pick an object. Janssens et al. [15] present an application of reinforcement learning (*Q*-learning) that simulates time and location information for a given sequence of travel activities. Even in a different field such as education we can find some interesting applications [14], e.g. in the process of learning pedagogical policies according to the students needs. Kaelbling et al. [16] and Busoniu et al. [7] have written surveys on reinforcement learning and its applications. A heuristic method can use searching trees. However, instead of generating all possible solution branches, a heuristic selects branches more likely to produce successful outcomes than other branches. It is selective at each decision point. This paper is an extension of a previous one on path-finding for RPGs [3].

In this article, we introduce a novel algorithm that includes a heuristic planning module (sampling from the worst trajectories) and a function $\mathcal{H}$ (the a priori knowledge injected to the system) that can contain any kind of information that express how good/bad is taking an action at a particular situation, for example, the Euclidean distance between a goal state and the current state. The proposed Dyna-$\mathcal{H}$ algorithm is based on the well-known Dyna architecture [27,26].

Grid world like environments treated as Markov sequential decision problems (MDPs) are used nowadays in many research works to evaluate and show the behavior of standard algorithms against new proposed ones. The results obtained in this test cases are easily generalizable to other problems, such as robot navigation, and, in general, any sequential decision problem. In this particular case, to an informed (i.e. knowledge-based) sequential decision problem. The proposed method, the one-step *Q*-learning and Dyna-*Q* algorithms have been applied to the same problem and compared in terms of learning rate.

1.1. Reinforcement learning in sequential decision support systems

Aggarwal [1] proposed a taxonomy of sequential decision support systems that span up to four levels, being the distinction between a single decision process and multiple decision process the division at the first level. At the fourth level we encounter the sequential decision processes that can be expressed as MDPs and solved by classical methods of dynamic programming (DP) [5,6]. Very recently, reinforcement learning methods started to appear in many research works on sequential decision support systems [10]. One of the advantages of using RL methods in DSSs is that usually RL systems requires to tune-up a less number of parameters than other kinds of model based methods while at the same time it allows to construct an approximation method to find a good trade off between accuracy and responsiveness [29]. However, Fard and Pineau [10] argue that reinforcement learning cannot be applied directly to decision support systems, such as those in medical domains, as they often suggest highly prescriptive policies and leave little room for the user's input (e.g. to inject expert knowledge or clues).

In this sense, Fard and Pineau [10] propose a method based on a novel definition of non-deterministic policies that should allow more flexibility in the user's decision-making process. In the same line, the proposed Dyna-$\mathcal{H}$ confront this applicability issues by allowing the introduction of knowledge to the system, in the form of a heuristic function $\mathcal{H}$. The function $\mathcal{H}$ can express a mapping between decisions and a heuristic action-value expectation allowing thus to take effective advantage of any a priori knowledge of the task to be solved and also, more important, to effectively express the user's preferences for particular kinds of solutions, for instance, by using a fuzzy inference system to define $\mathcal{H}$. We refer the reader to the Fard and Pineau [10] paper to find a list of recent application cases of RL to DSSs.

1.2. Paper overview

The structure of the paper is as follows. In Section 2, the strategies that are going to be applied and compared are briefly described. The proposed Dyna-$\mathcal{H}$ algorithm, the main contribution of the paper, is defined in Section 3. Section 4 describes the experimental scenario. The experimental results obtained by the different algorithms are discussed in Section 5. Finally, the last Section 6 is dedicated to state the conclusions and further work.

2. Search, reinforcement learning and planning

The algorithms that are going to be compared are briefly described in this section. A new algorithm based on the Dyna architecture [27,26], that combines heuristic on-line search and *Q*-learning is presented. We focus on solving path planning problems for homogeneous agents in homogeneous environments.

2.1. Heuristic search, the A^* algorithm

The predominant state-space planning methods in artificial intelligence are collectively known as heuristic search. Unlike other planning methods, heuristic search is not concerned with changing the approximate, value function, but only with improving the actions selection given the current value function.

In heuristic search, for each state encountered, a large tree of possible alternatives is considered. The approximate value function is applied to the leaf nodes, and then backed up at the previous state towards the root. The backing up in the search tree is just the same as in the max-backups. This backing up stops at the state-action nodes of the current state. Once the backed-up values of these nodes are computed, the best of them is chosen as the current action, and the rest of the values are discarded. In conventional heuristic search no effort is made to save the backed-up values and the value function, once designed, never changes as a result of the search. However, it would be reasonable to allow the value function to be improved over time, using either the backed-up values computed during the heuristic search or by any other method.

Heuristic methods such as A^* based algorithms have been widely applied. Actually, in the game development community, the most popular path-planning is to divide the environment into a grid that can be explored using these A^* based algorithms. This approach works very well in computer games as it always retrieves

the shortest path, if exists. This heuristic search ranks each node by an estimate of the best route through that node. It combines the tracking of the previous path length of Dijkstra's algorithm [8], with the heuristic estimate of the remaining path from best-first search. Since some nodes may be processed more than once, in order to find better paths later, it is necessary to keep track of them in a list. Adding this heuristic score to the nodes stored in Dijkstras priority queue, the number of nodes visited during the search can be effectively pruned down.

A^* has a couple of interesting properties. It is guaranteed to find the shortest path, as long as the heuristic estimate is admissible. That is, it is never greater than the remaining distance to the goal. It makes the *most efficient* use of the heuristic function: *no search that uses the same heuristic function and finds optimal paths will expand fewer nodes than* A^*, not counting tie-breaking among nodes of equal cost. A^* turns out to be very flexible in practice.

2.2. Reinforcement learning

Reinforcement learning [16,26] goes back to the very first stages of Artificial Intelligence and Machine Learning, and it has several applications in the Intelligent Knowledge Engineering Systems domain. They have been also successfully applied to game playing [19].

Under a constrained environment, the learning agent can perceive a set S of distinct states, which are normally characterized by a number of dimensions, and it has a set A of possible actions at each state. Reinforcement learning tasks are generally discrete. At each time step t, the agent observes the current state s_t and chooses a possible action a_t, which leads to the succeeding state $s_{t+1} = d(s_t, a_t)$. Then, the environment generates a reward $r(s_t, a_t)$. These rewards can be positive, zero or negative and can have a delay. In other words, some actions and their state transitions may bring low rewards in short term, while they will lead to state-action pairs with a much higher reward later. On the contrary, an action in a given state may receive an immediate high reward, whereas it makes the agent enter into a path where the following actions have very low or even negative rewards. Therefore, the task of the agent is to learn a policy $\pi : S \rightarrow A$, to achieve the maximum accumulative reward over time.

Reinforcement learning agents are connected to the environment by perceptions and actions. On each step of the interaction with the environment, the agent receives as input the current state and the value of that state. This value is the reward. The agent records the reward signal and updates the policy based on the information received about the reward so far.

Q-learning [30] is a popular method of model-free reinforcement learning. It can also be viewed as a method of asynchronous dynamic programming (DP) [5,6]. Reinforcement learning provides agents with the capability of learning from interactions with the environment, to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to receive or build maps (models) of the domains [12].

Learning proceeds similarly to Sutton's method of temporal differences (TD) [26]: an agent tries an action at a particular state, and evaluates its consequences in terms of the immediate reward or penalty it receives and its estimate of the value of the state to which is taken. By trying all actions in all states repeatedly, it learns which ones are the best overall, judged by long-term discounted cumulative reward [28].

A probabilistic approach is commonly used in Q-learning. A straightforward strategy is the ϵ-greedy method, where the probability of making a random choice is handled by the parameter ϵ. In every step, with probability $1 - \epsilon$, the agent fully exploits the information stored in the Q-values, and with probability ϵ the agent chooses a random action in order to explore the state space. In the exploration mode, the ϵ-greedy method assumes equal selection probabilities for any possible action, whereas the chance of selecting a better action may be increased by taking the current value distribution between alternatives such as in the soft-max methods [26].

2.3. The Dyna architecture

Planning is usually referred to any computational process that takes a model as input and produces or improves a policy to interact with the modeled environment. Although there are different approaches, state-space planning is mainly a search through the state space for an optimal path. Actions cause transitions from one state to another, and value functions are computed over states.

In on-line planning, new information is gained from the interaction with the environment and may change the model. If decision-making and model-learning are both computation-intensive processes, it may be necessary to divide the available computational resources between them. Dyna [27], is a reinforcement learning architecture that easily integrates incremental reinforcement learning and on-line planning.

Algorithm 1. Dyna-Q algorithm, as proposed by Sutton [27].

1: Initialize $Q(s,a), Model(s,a) \forall s \in \mathcal{S},\ a \in \mathcal{A}$
2: **repeat** {for each episode}
3: $s \leftarrow$ current (non terminal) state
4: $a \leftarrow \epsilon$-*greedy*(s,Q)
5: execute a; observe s' and r
6: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
7: $Model(s,a) \leftarrow s', r$
8: **for** $i = 1$ **to** N **do**
9: $s \leftarrow$ random previously observed state
10: $a \leftarrow$ random action previously taken in s
11: $s', r \leftarrow Model(s,a)$
12: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
13: **end for**
14: **until** s' is terminal

The possible relationship between experience, model and values for Dyna-Q are described in Fig. 1. Each arrow shows a relationship of influence. Note how experience can improve the model and therefore the value function, either directly or indirectly. It is the latter, which is sometimes called indirect reinforcement learning, which is involved in planning. In Algorithm 1, where Dyna-Q is described, $Model(s,a)$ denotes the contents of the model (predicted next state and reward, respectively) for state-action pair (s,a). Direct reinforcement learning, model-learning, and planning are implemented by steps 6, 7 and 8, respectively. If steps 7 and 8 were omitted, the remaining algorithm would be one-step tabular Q-learning.

Dyna-Q includes all of these processes: planning, acting, model-learning, and direct RL, continually. The planning method is the random-sample one-step Q-planning. The direct RL method is the

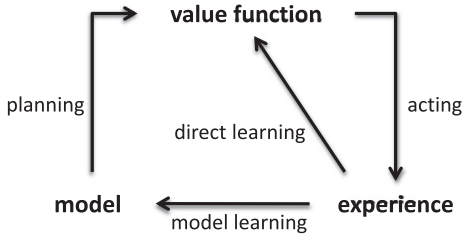

Fig. 1. Information flow in the Dyna architecture.

one-step Q-learning. The model-learning method is table-based and assumes the world is deterministic. After each transition, the model records the prediction that will deterministically follow. Thus, if the model is queried with a state-action pair that has been experienced before, it simply returns the last-observed next state and next reward as its prediction. During planning, the algorithm randomly samples only of state-action pairs that have been previously experienced. Conceptually, planning, acting, model-learning, and direct RL occur simultaneously and in parallel in Dyna agents [26].

3. A heuristic planning reinforcement learning algorithm based on the Dyna architecture

Here we propose a heuristic planning strategy to incorporate into a Dyna agent the advantages of a particular heuristic, in order to find the shortest paths in grid like environments, e.g. RPGs. A heuristic search method, as a search method after all, can be defined in terms of traversing a search tree. However, instead of generating all possible solution branches, a heuristic method selects branches more likely to produce successful outcomes than others. It is selective at each decision point. The proposed method incorporates the ability of heuristic search, e.g. A^*, to focus on specific search subtrees in order to make the searching more efficient. At the same time, the method learns online as any other common reinforcement learning algorithm and does not requires a complete model of the environment before staring to search.

3.1. Sampling from the worst trajectories (the nightmares metaphor)

Contrary to intuition, the proposed sampling strategy consist in using a learned model of the environment and traveling across it using the worst trajectories with respect to some heuristic index (e.g. a priori knowledge of the domain), receiving thus the worst rewards. However, this lead the algorithm to find the solution faster that using any other a priori better approach.

Sampling from "bad" trajectories using simulated experience has a very interesting analogous in human behavior: nightmares. This analogy suggests that such strategy can be considered as an interesting candidate hypothesis about the role of nightmares in human behavior, assigning thus a specific function to this behavior: a tool used by the brain to reorganize some goal oriented behaviors using the resting time to learn based on imagination (simulated experience). Furthermore, Fig. 9 (in Section 4) show different trajectories using this sampling strategy. As can be seen, these trajectories present some discontinuities (abrupt jumps) and also pass through the walls, i.e. violates the physical laws; things that are very common in dreams.

The analogous heuristic, in this case, to the $\mathcal{H}$ function, could be associated with the so called value-systems, which shape human behavior [9,25]. Indeed, there is a growing body of research about value-systems in robotics and autonomous agents in order to design robots with adaptive, lifelong learning behavior, because this values-systems are a way for robots to behave autonomously through spontaneous, self-generated activity [22]. In connection with autonomous agents many kinds of different value-systems, based on some aspects of human behavior related to motivation, e.g. curiosity driven, intrinsic motivated, novelty detection, have been proposed. However, it seems that there is (up to our knowledge) no publication along this line of research relating the study of dreams and value-systems with the reinforcement learning and planning field.

3.2. The Dyna-$\mathcal{H}$ algorithm

In RPGs and grid world like environments in general, it is common to use the Euclidian or city-clock distance functions as an effective heuristic. In this case study, the euclidian distance is used for the heuristic ($\mathcal{H}$) planning module. However, in general, $\mathcal{H}(s,a)$ represents a general function that gives a guess about *how bad* is to take action a in state s, e.g. the euclidian distance between the state s' and the *goal* position (1).

$$\mathcal{H}(s,a) = \|s' - goal\|^2, \tag{1}$$

where the s' state is the result of the model query: $s' = Model(s,a)$.

Hence, given the heuristic $\mathcal{H}$, the heuristic action h_a is defined as:

$$h_a(s,\mathcal{H}) = \underset{a}{argmax}\,\mathcal{H}(s,a), \tag{2}$$

where $h_a(s,\mathcal{H})$ is the worst action following ($\mathcal{H}$), e.g. the action that yields the higher distance from the goal. Algorithm 2 describes the steps of this strategy.

Algorithm 2. The proposed Dyna-$\mathcal{H}$ heuristic planning algorithm

```
Initialize Q(s,a), Model(s,a) ∀s ∈ S, a ∈ A
repeat {for each episode}
  s ← current (non terminal) state
  a ← ε-greedy(s,Q)
  execute a; observe s' and r
  Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
  Model(s, a) ← s', r
  for i = 1 to N do
    a ← h_a(s,H)
    if s,a ∉ Model then
      s ← random previously observed state
      a ← random action previously taken in s
    end if
    s', r ← Model(s,a)
    Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') − Q(s,a)]
    s ← s'
  end for
until s' is terminal
```

4. Experimental scenario

The Dyna-$\mathcal{H}$ heuristic planning algorithm have been evaluated and compared in terms of learning rate to the one-step Q-learning and Dyna-Q algorithms for the same problem.

The experiment consists of searching for optimal paths, i.e. the shortest path with the lowest cost between two states. To study this problem in the context of reinforcement learning, we assume that it is a Markov decision process, where there is a set of possible states and a set of actions. A typical problem in path-finding is obstacle avoidance. The simplest approach to this problem is to ignore obstacles until jumping into them. This approach is simpler because it makes few demands: all that it needs is the relative position of the entity and its goal, and whether the immediate vicinity is blocked. For many game situations, this is good enough. But there are scenarios where the only intelligent approach would be to plan the entire route in advance.

In this paper, the playing space is represented with square tiles as a 39×36 grid (Fig. 2). The obstacles are walls that are set randomly (in gray). The state is the tile or position where the entity is located. Neighboring states would vary depending on the game and the local situation. The cost of going from one position to another can represent many things: in this case it is computed as the simple distance between the two positions, which in RL terminology is equivalent to set $r = -1$ for all non-terminal state transitions, minimizing thus the total distance, i.e. finding an

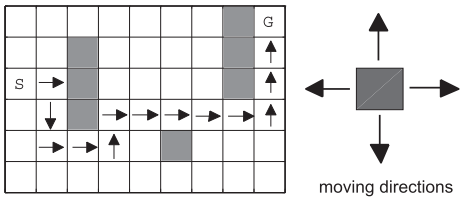

Fig. 2. The experimental scenario, starting point (S), goal (G), obstacles (gray), and a sample trajectory.

optimal path. The grid is represented as a two dimensional matrix of 39 rows and 36 columns. This matrix establishes the communication between nodes or states; each node can be related up to four neighbors, depending on the type of each node, i.e. up ($\uparrow$), down ($\downarrow$), left ($\leftarrow$) and right ($\rightarrow$).

5. Experimental results

Figs. 3–10 show the results of the simulations. As explained before, we have compared the performance of three algorithms: one-step Q-learning (Figs. 3 and 6), Dyna-Q (Figs. 4 and 7) and the proposed heuristic planning Dyna-$\mathcal{H}$ algorithm (Figs. 5 and 8).

As in Dyna maze [26], all the tests were based on the one-step Q-learning algorithm with a set of fixed parameters. The initial action values are zero, i.e. $Q(s,a) = 0$, the step-size parameter is $\alpha = 0.1$, and the exploration parameter was fixed to $\epsilon = 0.1$. When selecting greedily among actions, ties were broken randomly. For each algorithm, the learning curve shows the number of steps taken by the agent in each episode, averaged over 30 runs, each run consisting on a randomly generated labyrinth except from the staring (1,4) and goal (28,34) positions that remained constant during all experiments. Each random labyrinth was obtained using the same probability distribution (normal with $\mu = 0$, $\sigma = 0.3$) for every square tile of the grid, as shown in (4).

$$\phi(x) = \mathcal{N}(\mu = 0, \sigma^2 = 0.3^2), \tag{3}$$

$$tiletype = sgn(abs(Round(\phi(x)))); \tag{4}$$

where $tiletype = 1$ means that there is an obstacle and $tiletype = 1$ indicates a free tile.

For each different algorithm, the initial seed for the random number generator was held constant, hence, all are evaluated on the same set of 30 different grid configurations. For Dyna-Q and Dyna-$\mathcal{H}$, the number of planning steps was fixed to 10. All experiments ran for up to 100 episodes.

Fig. 3 shows the learning curve of the one-step Q-learning algorithm. As it can bee seen, this is the slowest method and thus it serves as a standard for comparisons. The Q-learning agent presents a very slow convergence curve and in fact it never found the optimal policy. It started with 2000 steps and showed a constant policy improvement during the 100 episodes, ending with approximately 1400 time steps. In Fig. 6 the best path found by the one-step Q-learning algorithm is shown.

Fig. 4 shows the learning rate of the Dyna-Q algorithm. As expected, the Dyna agent improved the learning curve regarding the on-step Q-learning algorithm. The Dyna-Q agent presents a "reasonable" convergence curve. However, it never found the optimal policy. It started with 2000 steps and showed a high policy improvement up to episode 40, were the agent continued improving but with a slower rate, almost constant (linear like) factor during the remaining 60 episodes, ending the learning with around 400 time steps. Although it presents a good behavior, it could not found the optimal trajectory during the simulation time. Next we present some examples of the kind of solution trajectories generated by each algorithm. These solutions corresponds to the first experiment of each algorithm evaluation. In Fig. 7 the best path found by Dyna-Q algorithm is shown.

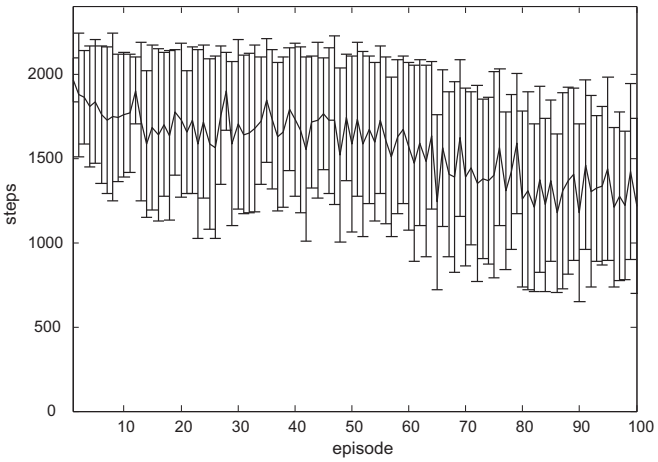

Fig. 3. Average learning curve over 30 runs for the one-step tabular Q-learning algorithm.

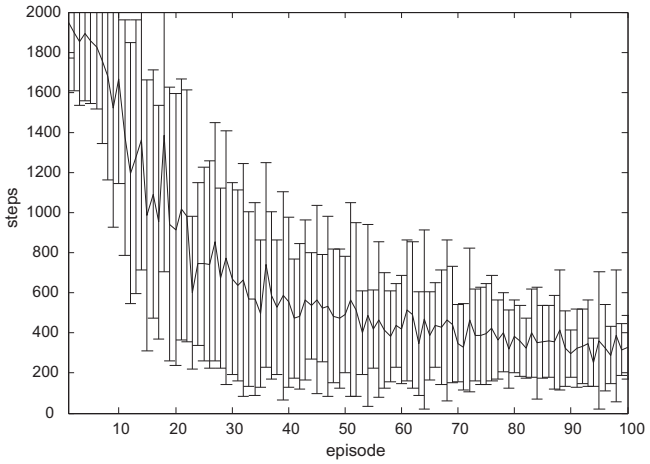

Fig. 4. Average learning curve over 30 runs for the Dyna-Q model with random sample with 10 planning steps.

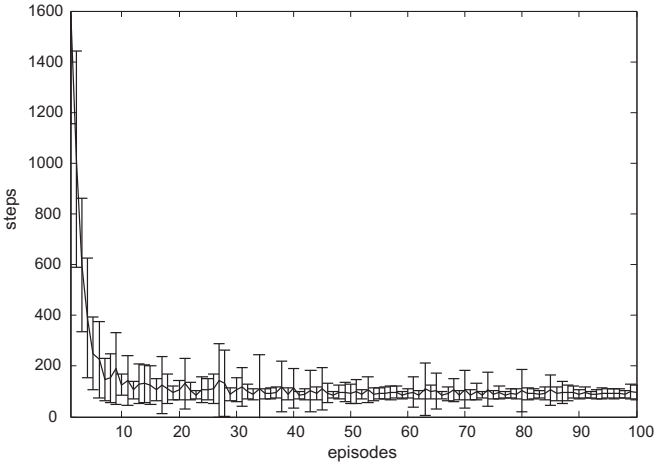

Fig. 5. Average learning curve over 30 runs for the proposed Dyna-$\mathcal{H}$ heuristic planning algorithm with 10 planning steps.

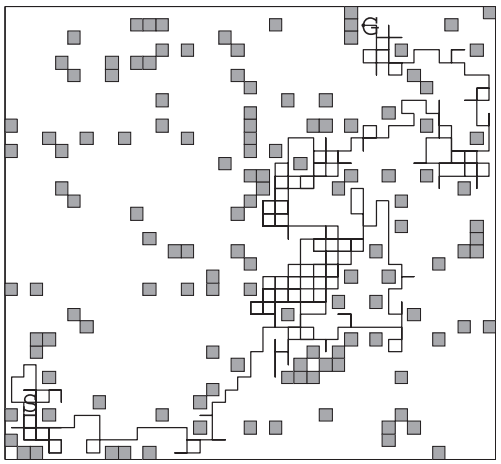

Fig. 6. Trajectory describing the best path found by the one-step Q-learning algorithm after 100 episodes for the first experiment.

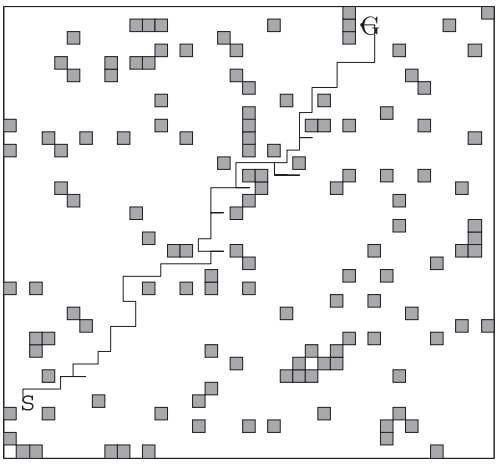

Fig. 8. Trajectory describing the best path found by the proposed Dyna-$\mathcal{H}$ heuristic planning algorithm (10 planning steps) after 100 episodes for the first experiment.

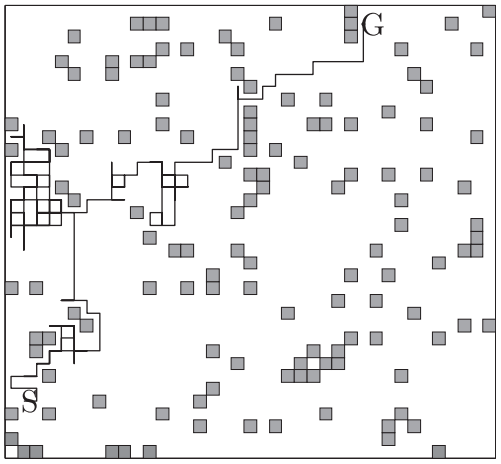

Fig. 7. Trajectory describing the best path found by the Dyna-Q planning algorithm (10 planning steps) after 100 episodes for the first experiment.

Fig. 5 shows the behavior of the proposed heuristic planning algorithm. As it is possible to see, the heuristic-planning agent improved a lot regarding the learning curve in comparison to the other algorithms. It presents an exponential convergence until the optimal policy is found. It started with 1600 steps and reduced them drastically up to episode 10, where it reaches the optimum (80 steps per episode). This means a high improvement both in the learning speed and the quality of the policy found. In Fig. 8 the best path found by Dyna-$\mathcal{H}$ algorithm is shown. It can be seen that the generated path is very close to the optimal path.

Fig. 9 shows several examples of the trajectories generated by the heuristic sampling planning procedure. The trajectories shown are all taken from the first episode of the first experiment of the Dyna-$\mathcal{H}$ algorithm and represent successive time steps of the episode. In these images, it is possible to appreciate clearly that the trajectories generated by the heuristic sampling strategy are almost the worst or very bad with respect to the solution, i.e. sampling from the worst trajectories, as defined by the Dyna-$\mathcal{H}$ algorithm and that using these trajectories the algorithm learns extremely well.

In Fig. 10, the average learning curves of the three algorithms are shown. The difference in terms of learning rate exhibited by the Dyna-$\mathcal{H}$ algorithm is evident.

As Sutton and Barto [26] comment, in the short term, sampling according to, for instance, the on-policy distribution helps to focus on states that are close descendants of the initial state. On the other hand, in the long run, focusing on the on-policy distribution may make the convergence worse because the most visited states have already their correct values. Sampling them is useless, whereas sampling other states may actually help. This can be the reason why the exhaustive, unfocused approach, works better in the long run, at least for small problems. Although it may seem the same case, the proposed planning process does exactly the contrary to what would be an optimal policy (the policy to which the on-policy distribution should converge), focusing on apparently not very promising branches. However, by sampling from the worst trajectories, the learned policy converge quickly to the optimal one.

In Fig. 11 an analysis of the convergence of the proposed Dyna-$\mathcal{H}$ algorithm, for different numbers of planning steps N is shown. The proposed heuristic planning algorithm have been tested for $N = 1, 5, 10$ and 25 planning steps. For $N = 1$, the algorithm converges in a few steps, around the 7th episode. However, it converges to a local suboptimal solution around 370 steps per episode. For $N = 5$, the algorithm also converges in around 7 episodes but it converges to a suboptimal solution that is significantly better than for the previous case, reaching an average of 250 steps per episodes. The cases of $N = 10$ and $N = 25$ show an identical convergence pattern as the $N = 5$ case but they reach better optimal policies.

It is quite significant that the case $N = 1$ presents the same convergence rate than much higher planning rates, but it finds much worse policies. However, dealing with problems where the system should save computational resources, it can achieve a good compromise between optimality and computational time. The learning curves for $N = 5$ up to $N = 25$ are identical, being the only difference the optimality of the policy reached, that is, the length of the path from the initial node to the goal. Again, this behavior is quite interesting since it indicates that the trade off between optimality and computational resources can be directly controller by tuning the number of planning steps.

6. Conclusions and further work

In this paper we have presented a novel reinforcement learning-planning algorithm, Dyna-$\mathcal{H}$, that integrates planning and learning into an online algorithm based on the well known Dyna architecture. The proposed method involves heuristic in the planning module. It incorporates the ability of A^* to focus on specific search subtrees in order to make the search more efficient by taking

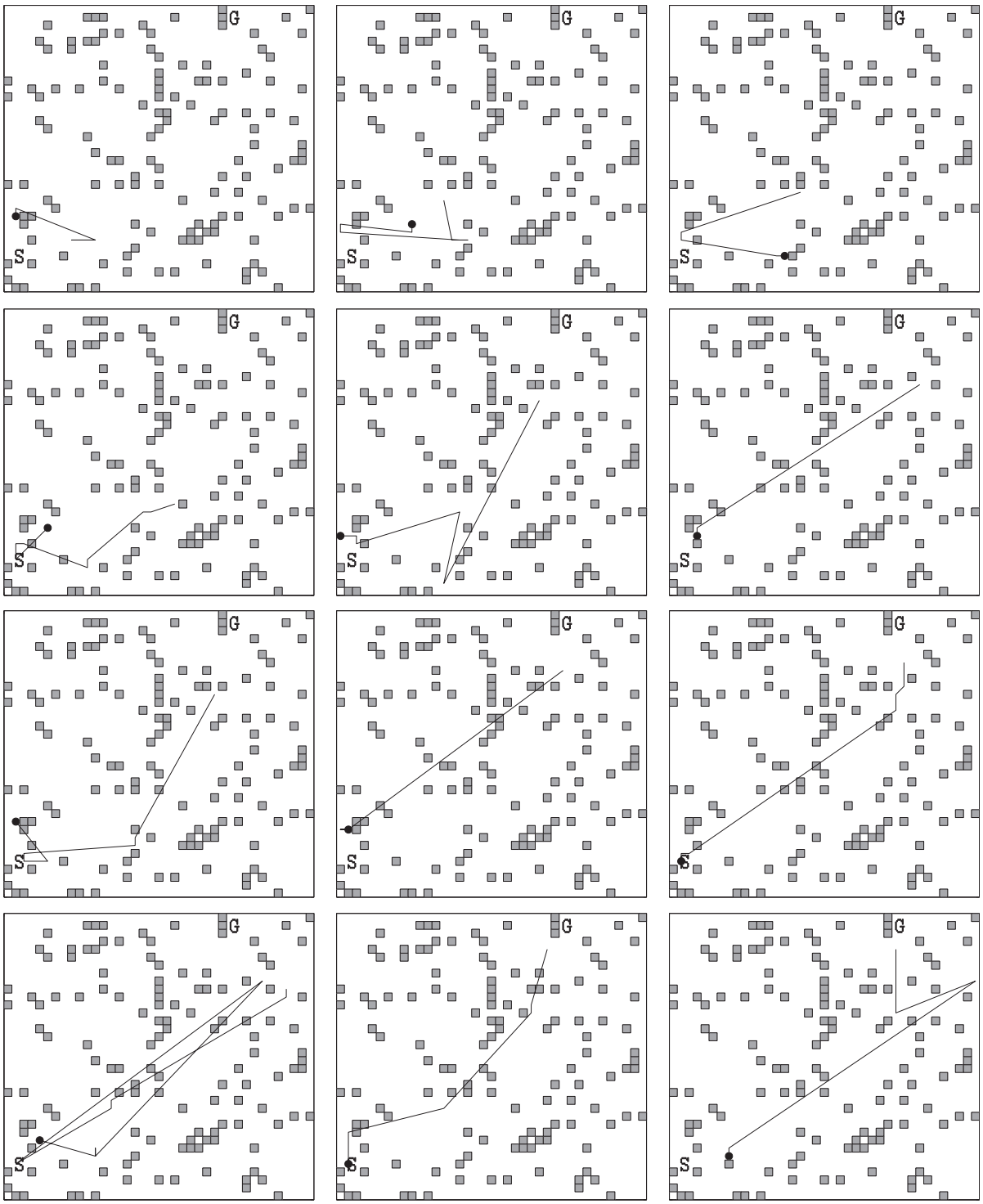

Fig. 9. Several sampling trajectories produced by the heuristic sampling for consecutive time steps during one episode.

advantage of the heuristic. Besides, it is a model free strategy that can be applied to sequential decision making problems under uncertainty.

A scenario to compare three learning algorithms: *Q*-learning, Dyna-*Q* and the proposed Dyna-$\mathcal{H}$, has been designed. The results (learning rate and convergence and policy found) obtained by all these methods have been shown and discussed. The new algorithm gives the best trajectories and the number of steps is reduced in more than the 90% with the Dyna-$\mathcal{H}$ strategy. From this results, we can conclude that the proposed Dyna-$\mathcal{H}$ heuristic planning algorithm is an effective strategy in path-finding problems and therefore for role-playing games.

Since the main difference between Dyna-*Q* and the proposed Dyna-$\mathcal{H}$ method is the use of a heuristic that guides the planning process when exploring the model, it makes sense to conclude that, under some well defined scenarios such as informed search methods, random sampling can be improved significatively.

In addition, we have shown a functional analogy between the proposed sampling from worst trajectories heuristic and the role of dreams (nightmares) in human behavior, suggesting a strong

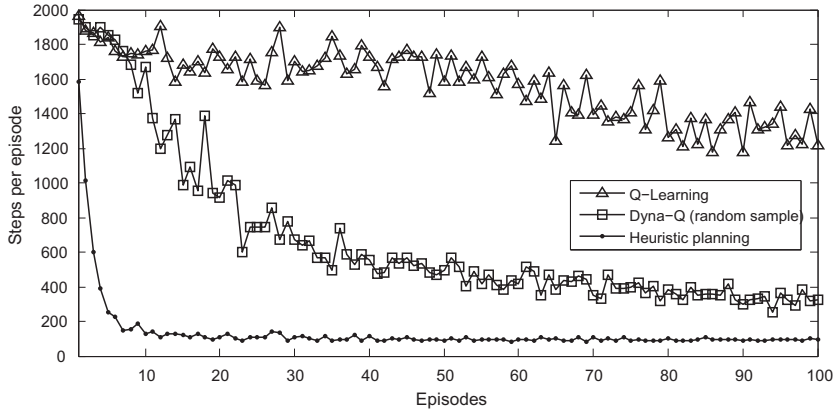

Fig. 10. Comparison of the average learning curve over 30 runs for Q-learning, Dyna-Q (random sample with 10 planning steps) and the proposed Dyna-$\mathcal{H}$ heuristic planning algorithm (with 10 planning steps).

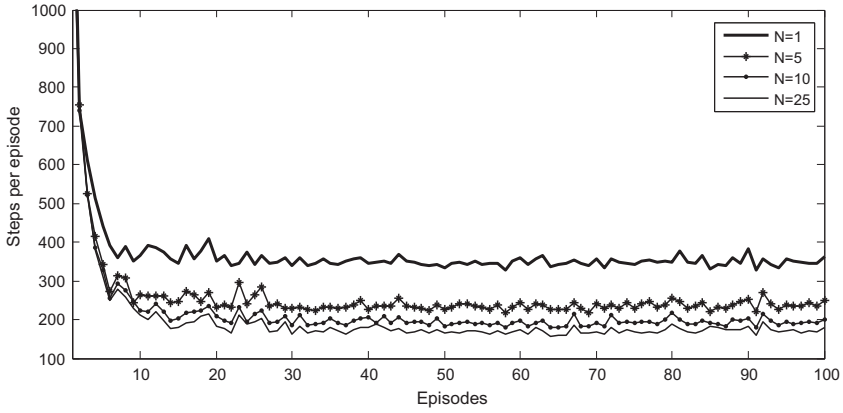

Fig. 11. Average learning rates (over 30 runs) of the proposed Dyna-$\mathcal{H}$ heuristic planning algorithm for different numbers of planning steps, $N = 1, 5, 10$ and 25.

relation of nightmares and pessimistic imagination with the reward system in the brain.

We expect the successful application of the proposed algorithm to many related problems. Further work should include the application of the proposed heuristic planning algorithm to different domains, for example, stochastic environments such as capture games for chaotic moving targets.

Software

An open-source Matlab^{TM} implementation of the Dyna-$\mathcal{H}$ algorithm can be obtained from the following direction: http://www.dacya.ucm.es/jam/downloads/Dyna-H.rar.

Acknowledgments

This work is partially supported by Spanish Project DPI2009-14552-C02-01.

References

[1] A. Aggarwal, A taxonomy of sequential decision support systems, Informing Science 4 (4) (2001).
[2] E. Alpaydin, Introduction to machine learning, Adaptive Computation and Machine Learning, MI, 2004.
[3] C. Alvarez, M. Santos, V. López, Reinforcement learning vs. A^* in a role playing game benchmark scenario, in: D. Ruan, T. Li, Y. Xu, G. Chen, E. Kerre (Eds.), Computational Intelligence, Foundations and Applications, vol. 3, 2010, pp. 644–650.
[4] S. Bayili, F. Polat, Limited-damage A^*: a path search algorithm that considers damage as a feasibility criterion, Knowledge-Based Systems 24 (2011) 501–512.
[5] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.
[6] R.E. Bellman, S.E. Dreyfus, Applied Dynamic Programming, Princeton University Press, 1962.
[7] L. Busoniu, R. Babuska, B. De Schutter, A comprehensive survey of multiagent reinforcement learning, IEEE Transactions on Systems, Man, and Cybernetics, Part C 38 (2) (2008) 156–172.
[8] E.W. Dijkstra, A note on two problems in connection with graphs, Numerical Mathematics 1 (5) (1959) 269–271.
[9] G.M. Edelman, Neural Darwinism – The Theory of Neuronal Group Selection, Basic Books, 1987.
[10] M. Fard, J. Pineau, Non-deterministic policies in markovian decision processes, Journal of Artificial Intelligence Research 40 (2011) 1–24.
[11] A.D. Gilmore, An exploratory study of the airline ticket purchasing problem. Ph.D. Thesis, University of Cincinnati, Computer Science, 2008.
[12] M. Grzes, D. Kudenko, Online learning of shaping rewards in reinforcement learning, in: the 18th International Conference on Artificial Neural Networks, ICANN 2008, Neural Networks 23 (4) 2010 541–550.
[13] A. Iglesias, M. del Castillo, M. Santos, J. Serrano, J. Oliva, A Comparison Between Possibility and Probability in Multiple Criteria Decision Making, in: D. Ruan, J. Montero, J. Lu, L. Martínez, P. Dhondt, E. Kerre (Eds.), Computational Intelligence in Decision and Control, vol. 1, World Scientific, 2008, pp. 307–312.
[14] A. Iglesias, P. Martínez, R. Aler, F. Fernández, Reinforcement learning of pedagogical policies in adaptive and intelligent educational systems, Knowledge-Based Systems 22 (4) (2009) 266–270 (artificial Intelligence (AI) in Blended Learning - (AI) in Blended Learning).
[15] D. Janssens, Y. Lan, G. Wets, G. Chen, Allocating time and location information to activity-travel patterns through reinforcement learning, Knowledge-Based Systems 20 (2007) 466–477.
[16] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.
[17] I. Karamouzas, M.H. Overmars, Adding variation to path planning, Computer Animation and Virtual Worlds 19 (3–4) (2008) 283–293.

[18] S. Krishnamurthy, G. Thamilarasu, C. Bauckhage, Malady: a machine learning-based autonomous decision-making system for sensor networks. in: International Conference on Computational Science and Engineering, 2009. CSE'09, vol. 2, 2009, pp. 93 –100.
[19] M.L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: ICML, 1994, pp. 157–163.
[20] V. Lopez, M. Santos, J. Montero, Fuzzy specification in real estate market decision making, International Journal of Computational Intelligence Systems 3 (1) (2010) 8–20.
[21] J. Lu, D. Ruan, Guest editorial preface: intelligent knowledge engineering systems, Knowledge-Based Systems 20 (2007) 437–438.
[22] K. Merrick, A comparative study of value systems for self-motivated exploration and learning by robots, Autonomous Mental Development, IEEE Transactions on 2 (2) (2010) 119–131.
[23] T. Mitchell, Machine Learning, McGraw Hill, New York, 1997.
[24] D. Muse, C. Weber, S. Wermter, Robot docking based on omnidirectional vision and reinforcement learning, Knowledge-Based Systems 19 (5) (2006) 324–332.
[25] O. Sporns, N. Almássy, G.M. Edelman, Plasticity in value systems and its role in adaptive behavior, Adaptive Behavior 8 (2000) 129–148.
[26] R. Sutton, A. Barto, Reinforcement Learning, An Introduction, The MIT press, 1998.
[27] R.S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, SIGART Bulletin 2 (4) (1991) 160–163.
[28] G. Tesauro, Practical issues in temporal difference learning, Machine Learning 8 (1992) 257–277.
[29] D. Thapa, I. Jung, G. Wang, Agent based decision support system using reinforcement learning under emergency circumstances, Advances in Natural Computation (2005) 422.
[30] C.J. Watkins, P. Dayan, Technical note Q-learning, Machine Learning 8 (1992) 279.