Artificial Intelligence 241 (2016) 103–130

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

A synthesis of automated planning and reinforcement learning for efficient, robust decision-making

Matteo Leonetti [a,c,*], Luca Iocchi [b], Peter Stone [a]

[a] Department of Computer Science, The University of Texas at Austin, 2317 Speedway, Stop D9500, Austin, TX 78712, USA
[b] Department of Computer, Control, and Management Engineering, Sapienza University of Rome, Via Ariosto 25, 00185 Rome, Italy
[c] School of Computing, University of Leeds, LS2 9JT Leeds, UK

ARTICLE INFO

Article history:
Received 17 August 2015
Received in revised form 15 April 2016
Accepted 20 July 2016
Available online 12 September 2016

Keywords:
Automated planning
Reinforcement learning
Autonomous robot
Robot learning
Answer set programming

ABSTRACT

Automated planning and reinforcement learning are characterized by complementary views on decision making: the former relies on previous knowledge and computation, while the latter on interaction with the world, and experience. Planning allows robots to carry out different tasks in the same domain, without the need to acquire knowledge about each one of them, but relies strongly on the accuracy of the model. Reinforcement learning, on the other hand, does not require previous knowledge, and allows robots to robustly adapt to the environment, but often necessitates an infeasible amount of experience. We present Domain Approximation for Reinforcement LearnING (DARLING), a method that takes advantage of planning to constrain the behavior of the agent to *reasonable* choices, and of reinforcement learning to adapt to the environment, and increase the reliability of the decision making process. We demonstrate the effectiveness of the proposed method on a service robot, carrying out a variety of tasks in an office building. We find that when the robot makes decisions by planning alone on a given model it often fails, and when it makes decisions by reinforcement learning alone it often cannot complete its tasks in a reasonable amount of time. When employing DARLING, even when seeded with the same model that was used for planning alone, however, the robot can quickly learn a behavior to carry out all the tasks, improves over time, and adapts to the environment as it changes.

1. Introduction

A great deal of work has been carried out in automated planning, and deliberation in general, while the application of such work to autonomous agents is much less developed [13]. The deployment of automated planning, especially in robotic tasks, faces many challenges, mostly due to high levels of uncertainty, which make completing a plan a significantly more difficult task than computing one. While low-level planning, such as navigation or motion planning, is gaining momentum in robotics, high-level planning is often avoided altogether. Behaviors are, instead, engineered by programming them directly.

The reason why plans are brittle can ultimately be attributed to imperfections in the models: relevant details overlooked, dynamics incorrectly represented, or assumptions violated. Nonetheless, making decisions in domains of any interest unavoidably involves abstraction and approximation, causing imperfect models to be widespread.

* Corresponding author at: School of Computing, University of Leeds, LS2 9JT Leeds, UK.
E-mail addresses: m.leonetti@leeds.ac.uk (M. Leonetti), iocchi@dis.uniroma1.it (L. Iocchi), pstone@cs.utexas.edu (P. Stone).

http://dx.doi.org/10.1016/j.artint.2016.07.004
0004-3702/

Execution Monitoring [24] and continual on-line planning [5] deal with unreliable models, but being able to recognize and react to failures might not be enough: an intelligent agent should learn from its mistakes and avoid them as much as possible in the future. The Reinforcement Learning (RL) [31] paradigm suits perfectly to this scenario, since it is based on trial and error, and the agent can have little knowledge about the domain before execution. Reinforcement Learning on its own, however, without prior knowledge of the environment, requires an enormous amount of experience. Applying RL methods directly is often infeasible in many practical cases, especially involving physical systems such as robots.

We propose to overcome the brittleness of the plans computed on a model through reinforcement learning in the environment, and to restrict the exploration of the environment through automated reasoning on the model. The resulting approach, Domain Approximation for Reinforcement LearnING (DARLING), exploits the synergy of the two methods allowing the agent, on the one hand, to relax the requirements on the planner, which can work on a simplified, abstract, representation of the domain. On the other hand, it allows the agent to take advantage of previous knowledge, for reducing the experience required by reinforcement learning. The automated reasoner provides a *rational* way to constrain the exploration and reduce the search space, while reinforcement learning eases the requirements on the accuracy of the model, which does not need to incorporate transition probabilities and action costs, even if the agent is in fact acting in a stochastic environment.

The main idea behind this work was introduced by Leonetti et al. [16] in the context of Hierarchical Reinforcement Learning, where a finite-state controller was induced by reasoning in Linear Temporal Logics (LTL), and used to constrain the exploration during a subsequent reinforcement learning phase. This article contributes a new formulation in terms of partial policies, an implementation of the planning phase through Answer Set Programming (ASP) instead of model-checking on LTL, and a thorough experimental validation conducted in several new domains. The use of ASP allowed us to scale the applicability of DARLING to real-world domains. In particular we deployed it to our autonomous mobile robots, carrying out and learning several tasks in a real-world office environment (the Gates-Dell Complex at the University of Texas at Austin).

2. Background and notation

This work leverages both reinforcement learning in Markov Decision Processes and automated reasoning in Answer Set Programming. In this section we introduce these two formalisms, along with the notation we will use in the rest of the article.

2.1. Markov Decision Processes

A Markov Decision Process is a tuple $D = \langle \mathcal{S}, \mathcal{A}, f, r \rangle$ where:

- $\mathcal{S}$ is a set of *states*.
- $\mathcal{A}$ is a set of *actions*. If not differently specified, every action is available in every state. When that is not the case, we denote with $\mathcal{A}(s)$ the set of actions available in state $s \in \mathcal{S}$.
- $f : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function. The function $f(s, a, s') = Pr(S_{t+1} = s' | S_t = s, A_t = a)$ is the probability that the current state changes from s to s' by executing action a. If $f(s, a, s') = \{0, 1\}$ the system is said to be *deterministic*, otherwise it is *stochastic*.
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R} \to [0, 1]$ is the reward function. The function $r(s, a, s', g) = Pr(R_{t+1} = g | S_t = s, A_t = a, S_{t+1} = s')$ is the probability of getting a reward g for being in state s, executing action a, and reaching state s'. The reward is said to be *deterministic* if $r(s, a, s', g) = \{0, 1\}\ \forall s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $g \in \mathcal{R}$. If the reward is negative, we will also refer to it as a *cost*.

We consider the system at discrete time steps. Let $t \in \mathbb{N}$ be the current time, and S_t be the state at time t. The agent interacts with the environment by choosing an action A_t and perceiving the next state S_{t+1}, such that:

$$S_{t+1} \sim f(S_t, A_t, \cdot) = Pr(S_{t+1} = s' | S_t = s, A_t = a),\ s, s' \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

It also receives a reward R_{t+1}:

$$R_{t+1} \sim r(S_t, A_t, S_{t+1}, \cdot)$$

If a state is never left, after it is entered for the first time, it is said to be a *terminal* or an *absorbing* state. If s is a terminal state then $S_{t+1} = s$ holds almost surely given that $S_t = s$. If an MDP has a terminal state it is said to be *episodic*.

The behavior of the agent is represented as a function $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ called a *(stationary) policy*, where $\pi(s, a) = Pr(A_t = a | S_t = s)$ is the probability of selecting action a in state s at time t. If $\pi(s, a) = \{0, 1\}\ \forall s \in \mathcal{S}$ and $a \in \mathcal{A}$ the policy is *deterministic*, in which case we will denote the action chosen by the policy as $a = \pi(s)$. A policy π and an initial state s_0 determine a probability distribution over the possible sequences ($\langle S_t, A_t, R_{t+1} \rangle$, $t \geq 0$). Given such a sequence, we define the *cumulative discounted reward* as:

$$G = \sum_{t \geq 0} \gamma^t R_{t+1}$$