Engineering Applications of Artificial Intelligence 55 (2016) 70–82

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

Online learning for optimistic planning

Lucian Buşoniu [a,*], Alexander Daniels [b], Robert Babuška [b]

[a] Department of Automation, Technical University of Cluj-Napoca, Romania
[b] Delft Center for Systems and Control, Delft University of Technology, the Netherlands

ARTICLE INFO

Article history:
Received 10 July 2015
Received in revised form
29 January 2016
Accepted 9 May 2016

Keywords:
Optimal control
Machine learning
Markov decision processes
Optimistic planning
Near-optimality analysis

ABSTRACT

Markov decision processes are a powerful framework for nonlinear, possibly stochastic optimal control. We consider two existing optimistic planning algorithms to solve them, which originate in artificial intelligence. These algorithms have provable near-optimal performance when the actions and possible stochastic next-states are discrete, but they wastefully discard the planning data after each step. We therefore introduce a method to learn online, from this data, the upper bounds that are used to guide the planning process. Five different approximators for the upper bounds are proposed, one of which is specifically adapted to planning, and the other four coming from the standard toolbox of function approximation. Our analysis characterizes the influence of the approximation error on the performance, and reveals that for small errors, learning-based planning performs better. In detailed experimental studies, learning leads to improved performance with all five representations, and a local variant of support vector machines provides a good compromise between performance and computation.

1. Introduction

Markov decision processes (MDPs) are often used to model sequential decision-making problems in artificial intelligence (Sutton and Barto, 1998; Sigaud and Buffet, 2010), but also work for optimal control problems in engineering, economics, operations research, medicine, etc. (Bertsekas, 2012). Online planning methods solve MDPs locally, by using a model to run an explorative search through the space of solutions (e.g. action sequences) from the current state, after which a first action is selected and applied. The system then reaches its new state and the process is repeated. A computationally effective strategy is to search the solution space optimistically (Munos, 2014), focusing on regions most likely to contain an optimal solution. The resulting *optimistic planning* (OP) algorithms (Munos, 2014; Buşoniu et al., 2012) integrate insights from several fields of artificial intelligence: reinforcement learning, bandit theory, and planning/graph search (Bertsekas, 2012; La Valle, 2006), as well as from optimization. A variety of OP methods are available, including Upper Confidence Trees (Kocsis and Szepesvári, 2006) which produced a competitive Go player (Gelly et al., 2006), OP for Deterministic systems (OPD) (Hren and Munos, 2008), OP for stochastic MDPs with discrete next-state distributions (OPMDP) (Buşoniu and Munos, 2012) or with general distributions (Bubeck and Munos, 2010), OP for continuous actions (Weinstein and Littman, 2012; Buşoniu et al., 2013a), etc.

We focus here on OPD and OPMDP, which explore a tree representation of the possible solutions from the current state. Every node on the planning tree is labeled by a state and an upper bound (called b value) on that state's optimal value. Exploiting the b values, at each iteration the algorithms refine further an optimistic partial solution, with the largest upper bound on the value. OPD and OPMDP guarantee near-optimality bounds as a function of the computation invested (Hren and Munos, 2008; Buşoniu and Munos, 2012). We refer to both algorithms collectively as 'OP'.

An important drawback of OP methods in their original form is that they discard the current tree right after applying the current action, and start over from scratch at the next step. However, the trees at consecutive steps will cover similar states, so the data can be reused to improve the quality of the search. Therefore, in this paper we propose to reuse data by learning online a b function from the state–b value pairs on the trees developed at previous steps. At the current step, the b function learned in this way is used to initialize newly created leaves with informed b values, rather than the uninformed values used in the original OP. The resulting approach is called OP with learning (L-OP). We provide a general analysis of the planning performance with learned b values, taking into account the tradeoff between two novel effects that did not appear in standard OP: the b values are closer to the optimal values than in OP (improving performance) – but due to approximation errors, they may no longer be true upper bounds and some nodes may be expanded non-optimistically (decreasing performance).

* Corresponding author.
E-mail addresses: lucian.busoniu@aut.utcluj.ro (L. Buşoniu), alexanderdaniels87@gmail.com (A. Daniels), r.babuska@tudelft.nl (R. Babuška).

http://dx.doi.org/10.1016/j.engappai.2016.05.003
0952-1976/

We continue by describing several b value approximators. The first is specific to OP, and given a Lipschitz value function it guarantees that b values always remain true upper bounds. The other approximators are standard: neural networks, local linear regression, and least-squares support vector machines; and for the latter, we also introduce a local, less computationally intensive variant. All techniques except neural networks are memory-based, so a procedure for memory management is additionally provided. We explain how these techniques can be applied to online L-OP, discuss their computational complexity, and evaluate them in practically relevant experiments on deterministic and stochastic problems. These experiments show that learning can achieve better performance with less computation, so it is useful for online control.

The idea of combining learning with model-based algorithms is well-known, see e.g. Dyna (Sutton, 1990). A variety of related work can be identified in classical planning by noticing that OPD and OPMDP extend the A* (Hart et al., 1968) and AO* (Nilsson, 1980) algorithms to infinite-horizon control. The b value in OP plays a similar role to the heuristic in A* and AO*, and deriving good heuristics is recognized as essential for performance. Heuristics are often found offline, before planning starts, based e.g. on relaxed versions of the problem (Helmert et al., 2007; Yoon, 2006). Other methods learn heuristics incrementally from consecutive planning tasks (Thayer et al., 2011; Arfaee et al., 2010; Fink, 2007), and these have some common elements with our work, such as generalizing function approximation (Jaillet et al., 2010; Thayer et al., 2011). Nevertheless, most of these methods compute complete, optimal plans offline, whereas our focus is near-optimally solving infinite-horizon problems online. Closer to our online setting are the classical algorithms 'learning real-time A*' (Korf, 1985) and 'real-time dynamic programming' (Barto et al., 1995), which learn heuristics while planning online. Kolobov et al. (2009) combined an approximator with RTDP – but this technique works for goal-based MDPs with logical states, and is not applicable in our general-MDP setting. Compared to Korf (1985), Barto et al. (1995), Kolobov et al. (2009), we introduce function approximation to deal with large, continuous state spaces (as suggested for RTDP by Barto et al. (1995)), provide a near-optimality analysis in the context of OP, and a thorough empirical study with several types of approximators.

Our technique is to our knowledge the first to learn b values online in optimistic planning. A value function computed *offline* was used to improve the performance of the Upper Confidence Trees algorithm by Gelly and Silver (2007). Our previous work on OP, such as Buşoniu and Munos (2012), Buşoniu et al. (2013a, 2013b), never uses learning – with a single exception: Fonteneau et al. (2013), where the model (transition probabilities) is learned rather than the b values.

Next, Section 2 introduces the optimal control problem and the two OP methods. Section 3 describes learning for OP, including the methods for the deterministic and stochastic cases, their analysis, the five function approximators, and memory management. Section 4 gives the experimental results, and Section 5 concludes the paper.

2. Background

2.1. Problem setting

We consider discrete-time optimal control problems with states $x \in X$ and actions $u \in U$. When applying u_k, the state changes from x_k to x_{k+1} with probability $f(x_k, u_k, x_{k+1})$, where $f: X \times U \times X \to [0, 1]$ is the state transition function. A reward function $\rho: X \times U \times X \to \mathbb{R}$ measures the quality of transitions, $r_{k+1} = \rho(x_k, u_k, x_{k+1})$. We assume the following: (i) the state space X is compact and included in $\mathbb{R}^p$; (ii) the action space U consists of a finite number K of actions; (iii) rewards are bounded, and without loss of generality they are in $[0, 1]$; finally, (iv) applying any action in any state can only lead to a finite number M of possible next states. These assumptions are typical in artificial intelligence, where X, U, f and ρ are said to form a Markov decision process (MDP). In control engineering, (i) is not restrictive in practice, while action discretization (ii) and reward saturation (iii) reduce performance, but the loss is often manageable. Deterministic transitions are common, in which case (iv) holds implicitly; otherwise, it restricts the random disturbance to discrete phenomena such as switches.

A policy $\pi: X \to U$ describes the control behavior: which action $u = \pi(x)$ to apply in each state x. The policy's value $V^\pi: X \to \mathbb{R}$ is defined for each state x as the expected return obtained by following the policy from x:

$$V^\pi(x) = E\left\{\sum_{k=0}^{\infty} \gamma^k r_{k+1}\right\} = E\left\{\sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k), x_{k+1})\right\} \quad (1)$$

where $x_0 = x$, $x_{k+1} \sim f(x_k, \pi(x_k), \cdot)$, the expectation is taken over trajectories, and $\gamma \in (0, 1)$ is the discount factor. Since rewards are at most 1, an upper bound on any state value under any policy is $V_{\max} = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$. Next to the state value function V, the state-action value function Q is defined as:

$$Q^\pi(x, u) = E_{x' \sim f(x,u,\cdot)}\{\rho(x, u, x') + \gamma V^\pi(x')\} \quad (2)$$

The objective is to *optimally* control the system, so that the value function is maximized for all $x \in X$. This maximal, optimal value function is denoted $V^*(x)$, an optimal policy that achieves these values is denoted $\pi^*(x)$, and the corresponding optimal state-action value function is $Q^*(x, u)$. The relationship $V^*(x) = \max_u Q^*(x, u)$ holds.

2.2. Optimistic planning for deterministic systems

We focus first on the deterministic case where the transition function reduces to $x_{k+1} = f(x_k, u_k)$, since a single state x_{k+1} is reached with probability 1; and the reward function to $r_{k+1} = \rho(x_k, u_k)$, since the next state x_{k+1} – and hence the reward – are fully determined by x_k and u_k.[1]

We consider an online model-based planning algorithm called Optimistic Planning for Deterministic systems (OPD) (Hren and Munos, 2008), which at each step k explores the set of possible action sequences from the current state x_k. Such sequences are able to represent an optimal solution local to x_k. They are more general than the state-feedback policies $\pi(x)$, which can also represent optimal solutions, but sequences are convenient in planning so we will use them. At a high level, OPD iteratively refines promising action sequences until a computational budget n, related to the number of evaluations of the model f, is exhausted. Based on the return information accumulated about these sequences, OPD then chooses an action u_k that is as good as possible. This action is applied to the system and the procedure is repeated from the new state.

To formalize the algorithm, we relabel by convention the current time k to 0, so that x_k becomes x_0. Of course, the procedure works at any time step. The planning process can be visualized

[1] Here as well as in the sequel (e.g. for the b values), we slightly abuse the notation by using the same symbols to denote analogous but mathematically different objects in the stochastic and deterministic case. For example, the deterministic $\rho(x_k, u_k)$ is obtained by plugging x_{k+1} in the stochastic reward function, $\rho(x_k, u_k, x_{k+1})$. It will usually be clear from the context to which variant the text refers; when it is not, we make it explicit.