# Induction and Learning of Finite-State controllers from Simulation

# (Extended Abstract)

M. Leonetti
Sapienza University of Rome
via Ariosto 25
00185 Rome, Italy
leonetti@dis.uniroma1.it

L. Iocchi
Sapienza University of Rome
via Ariosto 25
00185 Rome, Italy
luca.iocchi@dis.uniroma1.it

S. Ramamoorthy
The University of Edinburgh
10 Crichton Street
Edinburgh EH8 9AB, UK
s.ramamoorthy@ed.ac.uk

## ABSTRACT

We propose a method to generate agent controllers, represented as state machines, to act in partially observable environments. Such controllers are used to constrain the search space, applying techniques from Hierarchical Reinforcement Learning. We define a multi-step process, in which a simulator is employed to generate possible traces of execution. Those traces are then utilized to induce a non-deterministic state machine, that represents all reasonable behaviors, given the approximate models and planners used in simulation. The state machine will have multiple possible choices in some of its states. Those states are choice points, and we defer the learning of those choices to the deployment of the agent in the actual environment. The controller obtained can therefore adapt to the actual environment, limiting the search space in a sensible way.

## Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: ARTIFICIAL INTELLIGENCE: Problem Solving, Control Methods, and Search

## General Terms

Algorithms

## Keywords

Single Agent Learning, Robot planning, Agent development techniques, tools and environments

## 1. INTRODUCTION

Decision making in unknown environments is characterized by uncertainty at many and different levels. Part of the uncertainty can be captured by models for planning under partial observability, to which a great deal of attention has been payed in recent years. A paramount source of uncertainty lies in the assumptions behind such models themselves, especially if the problem has not been synthesized, but arises from an existing application. This is true regardless of how accurately the model has been designed or learned. Such uncertainty cannot be dealt with at planning time, and requires to monitor the execution in order to identify any discrepancies between what is expected and what is perceived.

Hierarchical Reinforcement Learning (HRL) [1] allows the designer to provide structure to the policies searched, constraining the exploration in fully observable domains. This is a fundamental aspect for real-world applications, as time is a strictly limited resource, and robotic agents are subject to wearing and tearing. The automatic definition of the aforementioned structures is still an open problem, and is usually carried out by hand.

In the following, we propose a method to generate agent controllers automatically, combining several ideas developed in the literature of planning under partial observability and reinforcement learning. We define a multi-step process, in which increasingly accurate models - generally too complex to be used for planning - are employed to generate possible traces of execution by simulation. Those traces are then utilized to induce a machine with non-deterministic states. Those states are *choice points*, and we defer the learning of those choices to the deployment of the agent in the actual environment.

## 2. GENERATING CONTROLLERS FROM SIMULATION

In this section we define the process to generate a finite state controller for a given problem under partial information.

### 2.1 Environments, problems, and controllers

We begin with the definition of a *dynamic environment* $\mathcal{E} = \langle \mathcal{A}, \mathcal{O}, \mathcal{S}, \mathcal{I}, \Delta, \Omega \rangle$ in which $\mathcal{A}$ is a finite set of actions, $\mathcal{O}$ is a set of observations, $\mathcal{S}$ is a set of states, $\mathcal{I} \subseteq \mathcal{S}$ is a set of initial states, $\Delta : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition relation, $\Omega : \mathcal{S} \rightarrow \mathcal{O}$ is the observation function. We assume to have available only $\mathcal{A}$ and $\mathcal{O}$, while all the other components of the environment are unknown. We also further assume, for this paper, that the environment is deterministic.

A *generalized planning problem* over an environment is a tuple (defined by Bonet et al [2]) $P = \langle F, I, A, G, R, O, D \rangle$ where: $F$ is a set of primitive fluents, $I$ is a set of F-clauses representing the initial situation, $A$ is a set of actions, $G$ is a set of literals representing the goal situation, $R$ is a set of non-primitive fluents, $O \subseteq R$ is the set of axioms defining the fluents in $R$. In the literature [2, 3], this problem specification is used to derive, by logic, a controller. Due to the uncertainty on the definition of the environment, we only assume to have $A$, and $G$. That is, we assume to know the

**Appears in:** *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

available actions and to be able to recognize the goal states. Such a problem cannot be solved directly by any planner, nor learned as a POMDP, as the model is largely unknown - including the description of the state space. Such a situation is described as *partial information*, which includes partial observability. Being a problem on a actual environment, however, experience can be gathered by acting in it.

A *controller* is a tuple $\mathcal{C} = \langle Q, A^*, O^*, \delta, q_0 \rangle$ where $Q$ is a set of states, $q_0 \in Q$ is the initial state, and $A^*$, $O^*$, and $\delta$ are the finite set of actions, set of observations, and transition relation respectively. The (partial) transition relation $\delta$ maps pairs $\langle q_i, o_i \rangle$ of controller states and observations into actions, and next states $q_{i+1}$. The controller is *deterministic* if given a pair $\langle q_i, o_i \rangle$, the action and consequently the next states are uniquely determined. A deterministic controller $C$ over an environment $\mathcal{E}$ produces, from each initial state $s_0$, a single trajectory $t_C(s_0) = \langle o_0, q_0, o_1, q_1, \ldots, o_f, q_f \rangle$. A non-deterministic controller, on the other hand, can produce a set of trajectories that we denote with $T_C(s_0)$.

A trajectory $t_C(s_0)$ is a solution of $P$ from an initial state $s_0$ iff the terminating observation $o_f$ is such that $o_f \models G$. We are assuming that $o_f \models G \Rightarrow s_f \models G$, that is, if an observation fulfills the goal specification, the underlying, unobservable, state is a goal state. A deterministic controller *solves* a problem $P$ over an environment $\mathcal{E}$ iff each trajectory $t_C(s_0)$, from each initial state $s_0$, is a solution from $s_0$. We say that a non-deterministic controller $C$ *can solve* a problem $P$ if $\exists t \in T_C(s_0)$ such that $t$ is a solution from $s_0$.

Finally, we define a *restriction* of a problem $P$ to $\hat{I} \subseteq I$ as the sub-problem $P(\hat{I}) = \langle F, \hat{I}, A, G, R, O, D \rangle$.

### 2.2 Simulators

We assume the existence of another environment $\mathcal{E}'$ on which we can define a problem $P'$. Informally, $\mathcal{E}'$ is a simulator for $\mathcal{E}$, and comprises the designer knowledge of the environment. The characteristic of a simulator is to be a model that provides an approximation of the environment $\mathcal{E}$, that is usually too complex to be used for planning. In such a model, however, experience can be gathered much more cheaply than in $\mathcal{E}$. We acknowledge the inescapable differences between $\mathcal{E}$ and $\mathcal{E}'$, and account for a learning phase to optimize the controller generated in the latter to act in the former. We do not define any direct relationship between $\mathcal{E}$ and $\mathcal{E}'$, we shall rather establish one through controllers.

Although we have a complete specification of both $\mathcal{E}'$ and $P'$, we only use them through simulation, that is, to generate trajectories. $P'$ can be partially observable, but does not necessarily have to. Furthermore, we assume the existence of a *decision maker* that can solve $P'$ in $\mathcal{E}'$.

### 2.3 Controller induction

The decision maker is deployed in $\mathcal{E}'$ to generate trajectories $t'(s'_0)$ that are solutions to $P'$.

Considering each action in $A'$ as a symbol, the set of trajectories that are solutions to $P'$, from which observations are removed, form a *language*. We induce a finite deterministic automaton $\mathcal{C}' = \langle Q', A', \emptyset, \delta', q'_0 \rangle$ that accepts such a language. Note how this is equivalent to a controller with an empty observation set.

Finally, we expand the edges of $\mathcal{C}'$ to accommodate the observations of $P$. We define a controller $\hat{\mathcal{C}} = \langle Q', A', O, \hat{\delta}, q'_0 \rangle$ obtained from $\mathcal{C}'$ such that for each observation $o \in O$, $\hat{\delta}$ connects a state $q'_i$ to a state $q'_j$ when observing $o$ and by executing $a' \in A'$, if and only if $\delta'$ connects $q'_i$ to $q'_j$ by executing $a'$.

If the controller $\hat{\mathcal{C}}$ obtained through the process just described can solve a reduction of $P$ in $\mathcal{E}$ we say that the composition of the decision maker, $\mathcal{E}'$, $P'$, and the method used to induce the automaton is *admissible*. This can be verified by executing $\hat{\mathcal{C}}$ in $\mathcal{E}$.

### 2.4 Reinforcement learning on controllers

The controller $\hat{\mathcal{C}}$, obtained through simulation and automaton induction, provides at the same time a constraint to the possible behaviors, and a partial specification of a solution for a problem that could not otherwise be solved. If such an automaton is deterministic no further improvement is possible, and it constitutes a completely specified solution to $P$. More interestingly, if such a controller is non-deterministic, learning in the actual environment can determine (and be limited to) the behavior at choice states. In order to generate a non-deterministic controller $\hat{\mathcal{C}}$ the decision maker must be able to produce more than a solution trajectory, from at least some initial states. The two options are not mutually exclusive.

A stochastic process is derived from $\hat{\mathcal{C}}$ according to the procedure presented by Leonetti and Iocchi [4]. The resulting process is Non-Markovian if the memory embedded in the controller and the observations is not a sufficient statistics for the reward. If that is the case, a policy can still be learned with a specific algorithm [6, 5].

## 3. CONCLUSION

We proposed a method to generate agent programs, in the form of state machines, by combining different components: an initial decision maker, available to the designer to include any previous knowledge about the task; a simulator, that is, a model too complex to be used for planning, but from which possible trajectories can be extracted; the induction of an automaton that accepts the extracted trajectories; and finally reinforcement learning, on the derived state machine, directly in the actual domain. Simulators are commonly employed, but are rarely integral parts in the development of agent programs. Constraining the final learning, through the interaction of all those components, significantly limit the search space in the actual domain.

## 4. REFERENCES

[1] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[2] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. of ICAPS*, pages 34–41, 2009.

[3] G. De Giacomo, F. Patrizi, and S. Sardina. Agent programming via planning programs. In *Proc. of AAMAS*, pages 491–498, 2010.

[4] M. Leonetti and L. Iocchi. Improving the performance of complex agent plans through reinforcement learning. In *Proceedings of AAMAS*, volume 1, pages 723–730, 2010.

[5] M. Leonetti, L. Iocchi, and S. Ramamoorthy. Reinforcement Learning Through Global Stochastic Search in N-MDPs. In *Proceedings of ECML-PKDD*, volume 2, pages 326–340, 2011.

[6] T. J. Perkins. Reinforcement learning for POMDPs based on action values and stochastic optimization. In *Proceedings of the National Conference on Artificial Intelligence*, pages 199–204, 2002.