Finite-State Controllers Based on Mealy Machines for Centralized and Decentralized POMDPs

Christopher Amato
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
camato@cs.umass.edu

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Shlomo Zilberstein
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
shlomo@cs.umass.edu

Abstract

Existing controller-based approaches for centralized and decentralized POMDPs are based on automata with output known as Moore machines. In this paper, we show that several advantages can be gained by utilizing another type of automata, the Mealy machine. Mealy machines are more powerful than Moore machines, provide a richer structure that can be exploited by solution methods, and can be easily incorporated into current controller-based approaches. To demonstrate this, we adapted some existing controller-based algorithms to use Mealy machines and obtained results on a set of benchmark domains. The Mealy-based approach always outperformed the Moore-based approach and often outperformed the state-of-the-art algorithms for both centralized and decentralized POMDPs. These findings provide fresh and general insights for the improvement of existing algorithms and the development of new ones.

Introduction

Sequential decision-making under uncertainty is an important and active field of study in Artificial Intelligence. The partially observable Markov decision process (POMDP) is used to model sequential decision-making problems in which there are stochastic actions and noisy sensors. Actions are chosen based on imperfect system information in order to maximize a long-term objective function. When multiple cooperative agents are present, the decentralized POMDP (DEC-POMDP) model can be used. This model allows a team of agents to be represented, each of which makes decisions based solely on possibly noisy local information. The goal of a DEC-POMDP is to maximize a joint objective function while executing actions in a decentralized manner, as opposed to the centralized policies in POMDPs.

Centralized and decentralized POMDPs are models that can represent a wide range of realistic problems, but are often difficult to solve. Discounted infinite-horizon problems (Sondik 1978), which proceed for an infinite number of steps and their rewards are discounted at a geometric rate, have been shown to be undecidable (Madani, Hanks, and Condon 2003). Even ϵ-optimal approaches (Hansen 1998; Bernstein et al. 2009) for these problems are intractable except for very small problem sizes. These approaches use finite-state controllers to represent polices, with actions chosen at each node in the controller and with controller transitions that depend on the observations seen.

Due to the difficulty in finding near optimal solutions, approximate methods are often desirable. Many such controller-based approaches have been developed for infinite-horizon POMDPs (Meuleau et al. 1999; Poupart and Boutilier 2003; Poupart 2005; Amato, Bernstein, and Zilberstein 2009) and DEC-POMDPs (Szer and Charpillet 2005; Bernstein, Hansen, and Zilberstein 2005; Bernstein et al. 2009; Amato, Bernstein, and Zilberstein 2009). Finite-horizon methods have also been extended to solve infinite-horizon problems. For centralized POMDPs, these approaches, such as point-based methods (Pineau, Gordon, and Thrun 2003; Smith and Simmons 2005; Spaan and Vlassis 2005; Bonet and Geffner 2009), build up a solution at each problem step until the discount factor causes further actions to not substantially change the quality of the solution. Point-based approaches often work well in practice, but rely on a manageable set of belief points and a solution that does not require a large number of steps to build. In contrast, controller-based approaches allow a concise, inherently infinite-horizon representation, that is more appropriate for certain problem types.

Approximation algorithms using finite-state controllers have been shown to provide high-quality solutions in a wide range of POMDPs and DEC-POMDPs. All current approaches are based on a type of automata with output called Moore machines. In this paper, we show that these methods can be improved by using controllers based on Mealy machines rather than Moore machines. A Mealy machine allows a more concise representation as well as additional structure that can be exploited by the solution methods. Our contribution is very general since all existing algorithms that are based on controllers can be adapted to use Mealy machines, often leading to higher quality solutions. We demonstrate this improved performance on a range of benchmark problems for centralized and decentralized POMDPs.

The rest of this paper is structured as follows. We begin with a general description of automata with output as well as POMDPs and DEC-POMDPs. We also show how agent policies can be represented using automata. Then, we briefly discuss the related work and how existing algorithms based on controllers can be adapted to use Mealy machines.

We also provide experiments comparing the use of Mealy and Moore machines as well as leading POMDP and DEC-POMDP algorithms. We conclude with a discussion of the key aspects of using Mealy machines.

Automata with output

An automaton with output produces an output string when processing an input string. There are two main types of machines, Moore and Mealy. Moore machines associate output symbols with nodes and Mealy machines associate output symbols with transitions.

Formally, a Moore machine is a tuple $\langle Q, \Sigma, \Omega, \delta, \lambda, q_0 \rangle$ where Q is the set of nodes, Σ and Ω are the input and output alphabets, $\delta : Q \times \Sigma \rightarrow Q$ is the *deterministic* transition function, $\lambda : Q \rightarrow \Omega$ is the *output* function, and $q_0 \in Q$ is the initial node. On an input string $x_1, \ldots, x_n \in \Sigma^*$, the machine outputs the string $\lambda(q_0), \ldots, \lambda(q_n) \in \Omega^*$ where $q_i = \delta(q_{i-1}, x_i)$ for $i = 1, \ldots, n$.

A Mealy machine is a tuple $\langle Q, \Sigma, \Omega, \delta, \lambda, q_0 \rangle$ where Q, Σ, Ω and q_0 are as before, but $\lambda : Q \times \Sigma \rightarrow \Omega$ is the output function that associates output symbols with transitions of the automaton. Given an input string $x_1, \ldots, x_n \in \Sigma^*$, the machine outputs $\lambda(q_0, x_1), \ldots, \lambda(q_{n-1}, x_n) \in \Omega^*$ where $q_i = \delta(q_{i-1}, x_i)$ for $i = 1, \ldots, n$. Examples of two nodes Moore and Mealy machines are shown in Figure 1.

Both models are equivalent in the sense that for a given machine of one type, there is a machine of the other type that generates the same outputs. However, it is known that Mealy machines are more *succinct* than Moore machines; given a Moore machine M_1, one can find an equivalent Mealy machine M_2 with the same number of nodes by constraining the outputs produced at each transition from a common node to be the same. However, given a (general) Mealy machine, the equivalent Moore machine has $|Q| \times |\Omega|$ nodes (Hopcroft and Ullman 1979).

In this paper, we are interested in *stochastic automata*. These automata are similar to those presented above except that probability distributions are used for transitions and output. That is, for a Moore machine the transition function is $\delta : Q \times \Sigma \rightarrow \Delta Q$ (where ΔQ is the set of probability distributions over Q) and the output function is $\lambda : Q \rightarrow \Delta\Omega$. For Mealy machines, we use a *coupled* or *joint* function $\tau : Q \times \Sigma \rightarrow \Delta(Q \times \Omega)$ that specifies both the transition and output functions.

POMDPs and DEC-POMDPs

POMDPs and DEC-POMDPs are mathematical models for sequential decision-making under uncertainty and partial information. Formally, a POMDP is a tuple $\langle S, A, O, P, R, \gamma \rangle$ where S, A and O are finite sets of states, actions and observations respectively, P denotes both the transition and the observation models, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, and γ is the discount factor. The transition model is specified by probabilities $P(s'|s, a)$ for the system resulting in state s' after applying action a while at state s, and the observation model by $P(o|s', a)$ for the system generating observation o when it enters state s' after the application of action a.

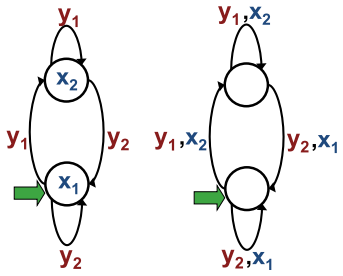

Figure 1: Two node deterministic Moore and Mealy machines with output $x_i \in \Omega$ and input $y_i \in \Sigma$ and an arrow designating the initial node.

The goal is to maximize the expected total cumulative reward for an initial state of the system that is distributed according to $b(\cdot)$. In the infinite-horizon problem, the decision making process unfolds over an infinite sequence of steps and rewards are discounted at a geometric rate using γ.

A DEC-POMDP is a generalization of the POMDP model for the case of multiple distributed agents that cooperate in order to maximize a joint reward. Formally, a DEC-POMDP can be defined as a tuple $\langle I, S, \{A_i\}, \{O_i\}, P, R, \gamma \rangle$, where I is a finite set of agents, A_i and O_i are finite sets of actions and observations for agent $i \in I$, and S, P, R remain the same as a POMDP except now, each depends on all agents. That is, the transition probabilities, $P(s'|s, \vec{a})$, define transitioning from state s to s' when the set of actions $\vec{a}$ is taken by the agents, $R(s, \vec{a})$ is the immediate reward for being in state s and taking the set of actions $\vec{a}$, and $P(\vec{o}|s', \vec{a})$ is the probability of seeing the set of observations $\vec{o}$ given the set of actions $\vec{a}$ was taken which results in state s'.

The model is decentralized because each agent i does not know the actions applied or the observations received by other agents, observing only the fragment o_i of the joint observation $\vec{o}$. Thus, agents seek to maximize a joint objective by choosing actions based solely on local information.

Finite-state controllers

Finite-state controllers can be used to represent policies for agents in an elegant way since an agent can be conceptualized as a device that receives observations and produces actions. For POMDPs a single controller provides the policy of the agent, while for DEC-POMDPs a set of controllers, one per agent, provides the joint policy of the agents.

In general, controllers of unbounded size may be needed to represent optimal policies, while controllers of bounded size represent policies of bounded memory. In the latter case, improved policies can be obtained by considering stochastic policies (Singh, Jaakkola, and Jordan 1994) that correspond to stochastic controllers. Due to space limitations, we only discuss the use of controllers for POMDPS, yet the case for DEC-POMDPs is similar.

Moore machines

In the standard POMDP, observations are generated only after the application of actions. This feature makes the

use of Moore machines natural for representing policies as shown in the following. Let $M = \langle Q, O, A, \delta, \lambda, q_0 \rangle$ be a stochastic Moore machine denoting the stochastic policy for the agent and assume that the environment is initially in state s_0. The first action applied by the agent, as dictated by M, is a_0 with probability $\lambda(q_0)(a_0)$. The system makes a transition to state s_1 with probability $P(s_1|s_0, a_0)$ and generates observation o_1 with probability $P(o_1|s_1, a_0)$. This observation triggers a transition in M to node q_1 with probability $\delta(q_0, o_1)(q_1)$, a new action a_1 with probability $\lambda(q_1)(a_1)$, and so on. The infinite-horizon discounted reward incurred by policy M when the initial state is s and the initial node of the automaton is q can be denoted by $V_M(q, s)$ and satisfies: $V_M(q, s) =$

$$\sum_a P(a|q) \left[R(s, a) + \gamma \sum_{s', o, q'} P(s', o|s, a) P(q'|q, o) V_M(q', s') \right]$$

where $P(a|q) \doteq \lambda(q)(a)$, $P(q'|q, o) \doteq \delta(q, o)(q')$ and $P(s', o|s, a) \doteq P(s'|s, a)P(o|s', a)$. The value of M at the initial distribution is $V_M(b) \doteq \sum_s b(s) V_M(q_0, s)$.

Note that these representations differ slightly from those used by some researchers (Bernstein et al. 2009; Bernstein, Hansen, and Zilberstein 2005) in that the transition no longer depends on the action in the Moore case. This is a minor difference that has limited significance.

Mealy machines

Mealy machines cannot be used directly to represent policies since these assume that observations are generated before actions. To see this, recall that a Mealy machine generates an output only when it makes a transition. Thus, to generate an action, it first needs an observation.

This issue is easily solved by adding a new state s^*, new action a^* and new observation o^* so that the system starts at s^*. The rewards satisfy $R(s^*, a^*) = 0$, and $R(s^*, a) = -M$ if $a \neq a^*$ and $R(s, a^*) = -M$ if $s \neq s^*$ where M is a very large integer; the observations are $P(o^*|s, a^*) = 1$; and the transitions are $P(s^*|s^*, a) = 1$ if $a \neq a^*$ and $P(s|s^*, a^*) = b(s)$ where $b(\cdot)$ is the initial distribution of the POMDP.

The new POMDP is such that action a^* should be applied only once in state s^*. This application generates o^* as the first observation. Therefore, one can shift the stream of action-observations by one position so that it starts with observations followed by actions as required. Furthermore, since $R(s^*, a^*) = 0$, the value function of the new POMDP satisfy $V_\pi^{new}(b^*) = \gamma V_\pi(b)$ for any policy π, where b^* is the new initial distribution defined by $b^*(s^*) = 1$. This means that the relative ranking of policies is carried over from the original POMDP into the new POMDP; in particular, the optimal policies of both POMDPs are the same.

Let us illustrate the use of a Mealy machine $M = \langle Q, O, A, \tau, q_0 \rangle$ to control a (modified) POMDP. As said before, the first action to apply in the initial state s^* is a^* that generated the observation o^*. After this action, the resulting state is s with probability $b(s)$. At this point, M is used to control the process by generating a new action a and node q with probability $\tau(q_0, o^*)(q, a)$, the system changes state to s' and generates observation o with probabilities $P(s'|s, a)$ and $P(o|s', a)$, and the execution continues.

The infinite-horizon discounted value incurred by policy M when the initial state is s, the observation is o and the node of M is q is denoted by $V_M(q, o, s)$ and satisfies: $V_M(q, o, s) =$

$$\sum_{a, q'} P(q', a|q, o) \left[R(s, a) + \gamma \sum_{s', o'} P(s', o'|s, a) V_M(q', o', s') \right]$$

where $P(q', a|q, o) \doteq \tau(q, o)(q', a)$. In this case, the value of M at b is $V_M(b) \doteq \sum_s b(s) V_M(q_0, o^*, s)$.

For a given controller size, Mealy machines are more powerful than Moore machines in the following sense (a similar result also holds for DEC-POMDPs). The proof is straightforward and omitted due to lack of space. Also note that in many cases, the Mealy controller will have strictly greater value for a given controller size.

Theorem 1. *Let $P = \langle S, A, O, P, R, \gamma \rangle$ be a POMDP with initial state distribution b, and N a fixed positive integer. Then, for each stochastic Moore controller M_1 with N nodes, there is an stochastic Mealy controller M_2 with N nodes such that $V_{M_2}(b) \geq V_{M_1}(b)$.*

Proof. (sketch) This proof follows directly from the fact that a Mealy machine can represent a Moore machine exactly using the same number of nodes. As mentioned above, this is accomplished by constraining the Mealy machine to choose the same output for each input symbol. For POMDP control, this results in choosing the same action for each observation at a given node. Thus, there is always a Mealy machine that has the same value as a given Moore machine of any size N for any POMDP. □

We can also show that in some cases, a Mealy machine can represent higher-valued solutions with the same number of nodes. For instance, consider a POMDP whose optimal policy is reactive (a mapping from single observations to actions). This can be achieved with a one node Mealy machine (because actions depend on observations seen), but not with a one node Moore machine (because actions depend only on that single node). No matter what action probabilities are defined, the Moore machine must use the same action distribution at each step regardless of which observation is seen. In fact, a Moore machine of size $|O|$ is needed to produce the same reactive policy as the one produced from a one node Mealy machine. Thus, the greater representational power of the Mealy machine can provide higher quality solutions with the same controller size.

Previous work

Until now, only Moore machines have been used to describe policies for POMDPs and DEC-POMDPs. Optimal approaches build up larger controllers over a number of steps using "backups" (Hansen 1998; Bernstein et al. 2009), while approximate approaches typically seek to determine the action selection and node transition parameters that produce a high-valued fixed-size controller. Approximate POMDP methods include those utilizing gradient ascent (Meuleau et al. 1999), linear programming (LP) (Poupart and Boutilier 2003; Poupart 2005) and nonlinear programming (NLP) (Amato, Bernstein, and Zilberstein 2009). Approximate

DEC-POMDP algorithms have also been developed using heuristic search (Szer and Charpillet 2005), linear programming (Bernstein, Hansen, and Zilberstein 2005) and nonlinear programming (Amato, Bernstein, and Zilberstein 2009). Recently, Mealy machines have been used in automated planning for representing controllers that are synthesized with classical planners (Bonet, Palacios, and Geffner 2009).

Mealy controllers for (DEC-)POMDPs

We conjecture that all of the above methods can be improved by adapting them to use Mealy machines. This would allow them to produce higher value with smaller controllers. Optimal approaches would proceed exactly as before (since Mealy machines can represent Moore machines exactly), but after each "backup" an improvement step can be used to improve the value of the current controller. This improvement step is similar to the use of linear programming to increase the controller values by Bernstein et al. (2009), but utilizing a Mealy controller formulation instead of a Moore formulation. Since Mealy machines can produce higher value for a given controller size, this can allow higher quality solutions to be found more quickly.

Approximate approaches can be improved by utilizing Mealy machines as well. We describe how bounded policy iteration (BPI) (Poupart and Boutilier 2003) and the NLP formulation (Amato, Bernstein, and Zilberstein 2009) can be adapted for solving POMDPs. Decentralized BPI (Bernstein, Hansen, and Zilberstein 2005) and the NLP formulation (Amato, Bernstein, and Zilberstein 2009) for DEC-POMDPs can be adapted in a similar way by optimizing a controller for each agent while including constraints to ensure decentralized execution.

Bounded policy iteration

Bounded policy iteration utilizes linear programming to improve the action selection and node transition parameters of the controller. That is, it iterates through the nodes of the controller and seeks to find parameters that increase the value of that node for all states while assuming that the other parameters and values remain fixed. The algorithm terminates once no further improvements can be achieved. Improvements such as growing the controller or biasing the improvements based on information from the initial state can also be conducted. We describe the improvement of a fixed-size controller without bias. Mealy controllers can be grown in the same way as described by Poupart and Boutilier (2003) and bias can be added in a similar way to that described by Poupart (2005).

The Mealy version of BPI breaks the problem in two parts: improving the first node and improving the other nodes. The parameters for the initial node q_0 can be improved by updating the probabilities of choosing actions and transitioning to other nodes from q_0 with the following LP that has variables $\{\hat{P}(q,a|q_0,o^*) : q \in Q, a \in A\}$:

$$\max \quad \sum_{a,q} \hat{P}(q,a|q_0,o^*) \sum_s b(s) \Big[R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) V(q,o',s') \Big]$$

subject to

$$\sum_{a,q} \hat{P}(q,a|q_0,o^*) = 1 \,, \hat{P}(q,a|q_0,o^*) \geq 0 \text{ for } q \in Q, a \in A.$$

The parameters for other controller nodes q can be adjusted similarly with the following LP that has the variables $\{\epsilon\} \cup \{\hat{P}(q',a|q,o) : q' \in Q, a \in A\}$:

$$\max \ \epsilon$$

$$\text{subject to} \quad V(q,o,s) + \epsilon \leq \sum_{a,q'} \hat{P}(q',a|q,o) \Big[R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) V(q',o',s') \Big] \ \forall s \in S,$$

$$\sum_{a,q'} \hat{P}(q',a|q,o) = 1, \ \hat{P}(q',a|q,o) \geq 0 \text{ for } q' \in Q, a \in A$$

Algorithmic benefits In this version of BPI, we iterate through each pair $\langle q,o \rangle$ rather than just through each node q as is done with Moore machines. Similar to the Moore version, parameters are updated if better values can be found for all states of the system, but because each q and o is used, there are more opportunities for improvements and thus some local optima may be avoided. Also, the linear program used for the first node allows a known initial state to be directly incorporated in the solution. Lastly, because a Mealy machine is used, higher value can be represented with a fixed size, thus improving the potential of the algorithm for that size. Due to space limitations, we leave this promising area of research for future work.

Nonlinear programming

The nonlinear programming formulation for POMDPs seeks to optimize the action and node transition parameters for the whole controller in one step. This is achieved by allowing the values to change by defining them as variables in the optimization. The fixed-size controller is then optimized with respect to a known initial state, $b(\cdot)$, using any NLP solver. An optimal solution of the NLP formulation would produce an optimal Moore controller for the given size, but this is often intractable, causing approximate methods to be used.

The Mealy formulation of the NLP is shown in Table 1. This NLP defines an optimal Mealy controller of given size. Its variables are $\{\hat{V}(q,o,s) : q \in Q, o \in O, s \in S\} \cup \{\hat{P}(q',a|q,o) : q,q' \in Q, a \in A, o \in O\} \cup \{\hat{P}(q,a|q_0,o^*) : q \in Q, a \in A\}$. Like the Moore case, this problem is difficult to solve optimally, but approximate solvers can be used to produce locally optimal solutions for a given size. Observe that for the NLP one does not need to extend the sets A, O and S with a^*, o^* and s^* respectively.

Algorithmic benefits As with other approaches, Mealy machines provide a more efficient method to exploit a given controller size. For the NLP formulation, one drawback is that more nonlinear constraints are needed because the value function now includes three items (node, observation and state) instead of the two (node and state) required for Moore machines. This may make the NLP more difficult to solve. In the next section we discuss how to use the structure of the Mealy machine in order to simplify the NLP.

$$\max \quad \sum_s b(s) \sum_{a,q} \hat{P}(q,a|q_0,o^*) \left[R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) \hat{V}(q,o',s') \right]$$

$$\text{subject to} \quad \hat{V}(q,o,s) = \sum_{a,q'} \hat{P}(q',a|q,o) \left[R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) \hat{V}(q',o',s') \right] \quad \text{for } q \in Q, o \in O, s \in S,$$

$$\sum_{a,q'} \hat{P}(q',a|q,o) = 1 \text{ for } q \in Q, o \in O, \quad \sum_{a,q'} \hat{P}(q',a|q_0,o^*) = 1$$

Table 1: Nonlinear program representing an optimal Mealy machine POMDP policy of a given size with variables $\hat{P}(q,a|q_0,o^*)$, $\hat{P}(q',a|q,o)$ and $\hat{V}(q,o,s)$. Additional constraints also ensure that probabilities are at least zero.

Solving the Mealy machine more efficiently

The structure of the Mealy machine can be used to identify unreachable state-observation pairs or dominated actions that can be removed without impact on optimality. The Mealy formulation can then be solved more efficiently while automatically determining important problem structure.

For given transition and observation dynamics, certain state-observation pairs cannot occur; if $P(s',o|s,a) = 0$ for every s and a, then the pair $\langle s',o \rangle$ does not need to be considered and thus neither does $V(q,o,s')$. This can drastically reduce the number of constraints in some problems.

Actions can also be removed based on their values given the last observation seen. For example, for a robot in a grid, if the last observation seen is that there is a wall directly in front of it, then trying to move forward is unlikely to be helpful. To determine which actions are useful after an observation, upper and lower bounds for the value of actions at each state after an observation can be computed. Actions whose upper bound values are lower than the lower bound values of another action can be removed without affecting optimality. In the POMDP case, the upper bound can be found using heuristics such as the MDP value or crossproduct MDP (Meuleau et al. 1999) and random or previously found solutions can be used for lower bounds.

Once upper and lower bounds of the value function are computed, an upper bound on choosing action a after being in state s and seeing o, $Q^U(o,s,a)$, is obtained with $Q^U(o,s,a) = R(s,a) + \gamma \sum_{s',o'} P(s',o'|s,a) V^U(o',s')$, while a lower bound $Q^L(o,s,a)$ is obtained similarly. An agent does not know the exact state it is in, and thus in general, an action must have an upper bound that is at most the lower bound for all states of the problem. That is, if there is a' such that $Q^U(o,s,a) \leq Q^L(o,s,a') \; \forall s \in S$ then the action a is dominated by a' with respect to o because a higher-valued policy can always be formed by choosing a' rather than a. This can be refined by noticing that typically, a subset of states are reachable after observing o, requiring that value is examined only $\forall s \in S^o$ where S^o is the set of states possible after seeing o. In these instances, action a does not need to be considered, allowing the corresponding parameters to be removed from $\{\hat{P}(q',a|q,o) : q' \in Q\}$.

Experimental results

We performed experiments on selected POMDP and DEC-POMDP benchmarks comparing approximate solutions of the Moore and Mealy NLP formulations with other leading approximate algorithms. These approaches represent the state-of-the-art in terms of both controller-based and general approximate approaches for infinite-horizon problems.

All Moore and Mealy experiments were conducted on the NEOS server (http://neos.mcs.anl.gov) using the snopt solver. They were initialized with random deterministic controllers and averaged over 10 trails. As described above, unreachable state-observation pairs and dominated actions were removed from the Mealy formulation. MDP and POMDP policies were used as upper bounds for POMDPs and DEC-POMDPs respectively, while reactive or previously found policies were used as lower bounds. Controller size was increased until the formulation could no longer be solved given the NEOS resource limitations (approximately 400MB and 8 hours).

Unless otherwise noted, other experiments were performed on a 2.8 GHz machine with 4GB of RAM. The code for HSVI2 and PERSEUS was used from the web sites of T. Smith and M. Spaan respectively and were run with a time limit of 90 minutes. For PERSEUS, 10,000 points were used and the average of 10 runs is provided. As experiments were conducted on different machines, results may vary slightly, but we expect the trends to remain the same.

Table 2 shows the results for three POMDPs benchmarks. The aloha problem is a networking domain using the slotted aloha scheme (Cassandra 1998) and the tag problem involves a robot that must catch and tag an opponent (Pineau, Gordon, and Thrun 2003). Because the original tag problem stops after the opponent is successfully tagged and thus is not fully infinite-horizon, we also provide results for a version in which the problem repeats rather than stopping. A discount factor of 0.999 was used for the Aloha domain and 0.95 was used for the tag problems.

In the first and third problems, Mealy machines provide the highest-valued solutions and generally use much less time than the other methods. In the second problem, the Mealy formulation is competitive with the state-of-the-art in terms of quality and time. In all cases Mealy machines outperform Moore machines.

Table 3 shows the results for three two agent DEC-POMDP benchmarks: the meeting in a grid problem (Bernstein, Hansen, and Zilberstein 2005), the box pushing domain (Seuken and Zilberstein 2007) and the stochastic Mars rover problem (Amato and Zilberstein 2009). On all of the DEC-POMDP domains a discount factor of 0.9 was used.

Algorithm	Value	Size	Time
Aloha: $\|S\| = 90, \|A\| = 29, \|O\| = 3$			
Mealy	1,221.72	7	312
HSVI2	1217.95	5,434	5,430
Moore	1,211.67	6	1,134
PERSEUS	853.42	68	5,401
Tag: $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
PBPI[1]	-5.87	818	1,133
RTDP-BEL[1]	-6.16	2.5m	493
PERSEUS[1]	-6.17	280	1,670
HSVI2[1]	-6.36	415	24
Mealy	-6.65	2	323
biased BPI[1]	-6.65	17	250
BPI[1]	-9.18	940	59,772
Moore	-13.94	2	5,596
Tag Repeat: $\|S\| = 870, \|A\| = 5, \|O\| = 30$			
Mealy	-11.44	2	319
PERSEUS	-12.35	163	5,656
HSVI2	-14.33	8,433	5,413
Moore	-20.00	1	37

Table 2: Results for POMDP problems comparing Mealy and Moore machines and other algorithms. The size for the machines is the number of nodes in the controller. The size for other algorithms is the number of planes or belief points. The time is in seconds.

To put the results from the Moore and Mealy machines into perspective, we also include results from heuristic policy iteration with nonlinear programming (HPI w/ NLP) (Bernstein et al. 2009) and the goal-directed sampling algorithm (Amato and Zilberstein 2009). This goal-directed approach assumes special problem structure and thus is not a general algorithm. As such we would expect it to outperform the other algorithms.

In all three problems, the Mealy machine obtains higher quality solutions than the Moore machine or HPI with NLP. The Mealy formulation also outperforms the Goal-directed approach on the first problem and is competitive with it in the other domains, showing that much of the problem structure can be automatically discovered with our approach. This is accomplished with concise controllers and a very reasonable running time. Note that both the goal directed and HPI w/ NLP approaches use Moore machines as their policy representation. We believe that using Mealy machines would improve their value, but leave this for future work.

In a final experiment, we compare the quality of controllers obtained by utilizing fixed-size Mealy and Moore machines on the DEC-POMDP benchmarks. Table 4 shows the results of the comparison. As can be seen, Mealy machines always achieve better quality for a fixed size. Similar results were also obtained in the POMDP benchmarks. By

[1]These results are taken from PBPI: (Ji et al. 2007), RTDP-BEL: (Bonet and Geffner 2009), PERSEUS: (Spaan and Vlassis 2005), HSVI2: (Smith and Simmons 2005), biased BPI: (Poupart 2005) and BPI: (Poupart and Boutilier 2003)

Algorithm	Value	Size	Time
Meeting in a Grid: $\|S\| = 16, \|A_i\| = 5, \|O_i\| = 2$			
Mealy	6.13	5	116
HPI w/ NLP	6.04	7	16,763
Moore	5.66	5	117
Goal-directed[2]	5.64	4	4
Box Pushing: $\|S\| = 100, \|A_i\| = 4, \|O_i\| = 5$			
Mealy	143.14	4	774
HPI w/ NLP	95.63	10	6,545
Moore	50.64	4	5,176
Goal-directed[2]	149.85	5	199
Mars Rover: $\|S\| = 256, \|A_i\| = 6, \|O_i\| = 8$			
Mealy	19.67	3	396
HPI w/ NLP	9.29	4	111
Moore	8.16	2	43
Goal-directed[2]	21.48	6	956

Table 3: Results for DEC-POMDP problems comparing Mealy and Moore machines and other algorithms. The size refers to the number of nodes in the controller and the time is in seconds.

using approximate solvers, there may be problems for which this is not the case, but we are encouraged by the results.

We can also see that in these problems much larger Moore controllers are needed to represent the same value as a given Mealy controller. In fact, the solver often exhausted resources before this was possible. In the box pushing and mars rover problems, the largest solvable Moore machines were not able to achieve the same quality as a one node Mealy machine. In the meeting in a grid problem, a four node Moore machine is required to approximate the one node Mealy machine.

It is also worth noting that similar value was often found using Mealy machines without state-observation reachability or action elimination. For instance, in the meeting in a grid problem, three node controllers could be solved producing an average value of 5.87 (the same as the three node controller with the more efficient representation) in the somewhat longer time of 31s. Thus, even without using the controller structure to increase solution efficiency, the Mealy formulation can be beneficial.

Discussion

We presented a novel type of controller for centralized and decentralized POMDPs that is based on the Mealy machine. Existing controller-based algorithms can be adapted to use this type of machine instead of the currently used Moore machine. We adapted one such algorithm and our experiments show that Mealy machines can lead to higher-valued controllers when compared to the state-of-the-art approaches for both POMDPs and DEC-POMDPs.

[2]Goal-directed results assume special problem structure and thus cannot be directly compared with general approaches such as the Mealy, Moore and HPI methods.

Type	Number of nodes 1	2	3	4	5
Meeting in a grid: $\|S\| = 16, \|A_i\| = 5, \|O_i\| = 2$					
Mealy	5.50	6.00	5.87	6.05	6.13
Moore	3.58	4.83	5.23	5.62	5.66
Box pushing: $\|S\| = 100, \|A_i\| = 4, \|O_i\| = 5$					
Mealy	123.46	124.20	133.67	143.14	
Moore	-1.58	31.97	46.28	50.64	
Mars rover: $\|S\| = 256$, $\|A_i\| = 6$, $\|O_i\| = 8$					
Mealy	18.92	19.17	19.67		
Moore	0.80	8.16			

Table 4: Results for Mealy and Moore machines of different sizes for DEC-POMDP benchmarks. A blank entry means that the controller of that size could not been computed given the resource restrictions of the NEOS server.

Mealy machines are beneficial for several reasons. First, they are more powerful than Moore machines, resulting in higher-valued solutions with the same representation size. Second, they possess special structure that can be straightforward to exploit. This includes start state information and knowledge from informative observations. The increased representational power can cause solving Mealy machines to be more difficult for a given controller size, but more concise representations may be sufficient. Also, as shown in our experiments, the ability to exploit domain structure may make solution methods that utilize Mealy machines more scalable than those that utilize Moore machines.

In the future, we plan to adapt other controller-based algorithms to utilize Mealy machines and measure the improvement in performance. Also, we would like to devise additional ways to exploit the structure of the Mealy machine and to explore the relationship between Mealy and Moore machines in order to obtain a better understanding of both types of machines.

Acknowledgements

This work was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0181 and by the National Science Foundation under Grant No. IIS-0812149.

References

Amato, C., and Zilberstein, S. 2009. Achieving goals in decentralized POMDPs. In *Proceedings of the Eighth International Joint Conference on Autonomous Agents and Multiagent Systems*, 593–600.

Amato, C.; Bernstein, D. S.; and Zilberstein, S. 2009. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Journal of Autonomous Agents and Multi-Agent Systems* DOI: 10.1007/s10458-009-9103-z.

Bernstein, D. S.; Amato, C.; Hansen, E. A.; and Zilberstein, S. 2009. Policy iteration for decentralized control of Markov decision processes. *Journal of AI Research* 34:89–132.

Bernstein, D. S.; Hansen, E. A.; and Zilberstein, S. 2005. Bounded policy iteration for decentralized POMDPs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, 1287–1292.

Bonet, B., and Geffner, H. 2009. Solving POMDPs: RTDP-Bel bs. point-based algorithms. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 1641–1646.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 34–41.

Cassandra, A. R. 1998. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. Ph.D. Dissertation, Brown University, Providence, RI.

Hansen, E. A. 1998. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, 211–219.

Hopcroft, J. E., and Ullman, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Ji, S.; Parr, R.; Li, H.; Liao, X.; and Carin, L. 2007. Point-based policy iteration. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, 1243–1249.

Madani, O.; Hanks, S.; and Condon, A. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence* 147:5–34.

Meuleau, N.; Kim, K.-E.; Kaelbling, L. P.; and Cassandra, A. R. 1999. Solving POMDPs by searching the space of finite policies. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 417–426.

Pineau, J.; Gordon, G.; and Thrun, S. 2003. Point-based value iteration: an anytime algorithm for POMDPs. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 1025–1032.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *Advances in Neural Information Processing Systems*, 16. 823–830.

Poupart, P. 2005. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes*. Ph.D. Dissertation, University of Toronto.

Seuken, S., and Zilberstein, S. 2007. Improved memory-bounded dynamic programming for decentralized POMDPs. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 344–351.

Singh, S.; Jaakkola, T.; and Jordan, M. 1994. Learning without state-estimation in partially observable Markovian decision processes. In *Proceedings of the Eleventh International Conference on Machine Learning*, 284–292.

Smith, T., and Simmons, R. 2005. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, 542–549.

Sondik, E. J. 1978. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research* 26:282–304.

Spaan, M. T. J., and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *Journal of AI Research* 24:195–220.

Szer, D., and Charpillet, F. 2005. An optimal best-first search algorithm for solving infinite horizon DEC-POMDPs. In *Proceedings of the Sixteenth European Conference on Machine Learning*, 389–399.