Automatic Derivation of Finite-State Machines for Behavior Control*

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

Héctor Palacios
Universidad Simón Bolívar
Caracas, Venezuela
hlp@ldc.usb.ve

Héctor Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, SPAIN
hector.geffner@upf.edu

Abstract

Finite-state controllers represent an effective action selection mechanisms widely used in domains such as video-games and mobile robotics. In contrast to the policies obtained from MDPs and POMDPs, finite-state controllers have two advantages: they are often extremely compact, and they are general, applying to many problems and not just one. A limitation of finite-state controllers, on the other hand, is that they are written by hand. In this paper, we address this limitation, presenting a method for deriving controllers automatically from models. The models represent a class of contingent problems where actions are deterministic and some fluents are observable. The problem of deriving a controller is converted into a conformant problem that is solved using classical planners, taking advantage of a complete translation into classical planning introduced recently. The controllers derived are 'general' in the sense that they do not solve the original problem only, but many variations as well, including changes in the size of the problem or in the uncertainty of the initial situation and action effects. Several experiments illustrating the automatic derivation of controllers are presented.

Introduction

Figure 1(a) illustrates a simple 1×5 grid where a robot, initially at one of the two leftmost positions, must visit the rightmost position, marked B, and get back to A. Assuming that the robot can observe the mark in the current cell if any, and that the actions *Left* and *Right* deterministically move the robot one unit left and right respectively, the problem can be solved by a contingent planner or a POMDP solver, resulting in the first case in a contingent tree, and a function mapping beliefs into actions in the second (Levesque 1996; Kaelbling, Littman, and Cassandra 1999). A solution to the problem, however, can also be expressed in a simpler manner as the finite-state controller shown in Fig. 1(b). Starting in the controller state q_0, this controller selects the action *Right*, whether A or no mark is observed ('$-$'), until observing B. Then the controller switches to state q_1 where it selects *Left* as long as no mark is observed.

The finite-state controller displayed in the figure has two features that make it more appealing than contingent plans and POMDP policies: it is very compact (it involves two state only), and it is very general. Indeed, the problem can be changed in a number of ways and the controller will still work. For example, the *size of the grid* can be changed from 1×5 to $1 \times n$, the agent can be placed *initially* anywhere in the grid (except at B), and the actions can be made *non-deterministic* by the addition of 'noise'. This generality is well beyond the power of contingent plans or exact POMDP policies that are tied to a particular state space. For these reasons, finite-state controllers are widely used in practice, from controlling non-playing characters in video-games (Buckland 2004) to mobile robots (Murphy 2000; Mataric 2007). Memoryless controllers or policies (Littman 1994) are widely used as well, and they are nothing but finite-state controllers with a single state. The additional states provide controllers with a memory that allows different actions to be taken given the same observation.

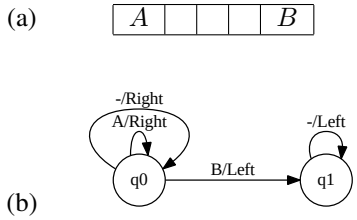

Figure 1: (a) A planning problem where an agent initially in one of the two leftmost positions has to get to B and then back to A. These two marks are observable. (b) A 2-state controller that solves this problem and many variations. The circles are the controller state, and an edge $q \rightarrow q'$ labeled o/a says to do a when observing o in the controller state q, switching then to q'.

The benefits of finite-state controllers come at a price: unlike contingent trees and POMDP policies, they are usually not derived automatically from a model but are written by hand; a task that is non-trivial even in the simplest cases. There have been attempts for deriving finite-state controllers for POMDPs with a given number of states (Meuleau et al. 1999; Poupart and Boutilier 2003; Bernstein et al. 2009), but the problem is then solved approximately with no correctness guarantees.

In this work, we present the model-based method for deriving finite-state controllers automatically that we recently

*This AAAI-10 Nectar paper is based on (Bonet, Palacios, and Geffner 2009).

introduced (Bonet, Palacios, and Geffner 2009). The models represent a class of contingent problems where actions are deterministic and some fluents are observable. The task of deriving a controller for such models is converted into a conformant planning problem that is solved by state-of-the-art classical planners, taking advantage of a complete transformation (Palacios and Geffner 2009). A conformant problem is a contingent problem with no sensing whose solutions, like solutions of classical problems, are action sequences.

Model, Language, and Control

Finite-state controllers are derived from a simple but expressive model for contingent planning in which actions are *deterministic* and may have conditional effects but *no preconditions*, and *sensing is passive* meaning that the set of observable fluents is fixed and does not change with the actions taken.

More precisely, we consider a class of *control problems* of the form $P = \langle F, I, A, G, R, O, D\rangle$, where F, I, G and A denote the set of (primitive) fluents, the initial and goal situations, and the set of actions respectively, as in a classical planning problem, except that I is not a set of literals but a set of clauses that accommodate uncertainty. In addition, R stands for a set of *non-primitive fluents* defined in terms of the primitive fluents by means of a collection D of *axioms* or *ramification rules*, and O is a subset of R that represent the *observable fluents*. Likewise,

- A state s is a truth valuation over the primitive fluents F that defines values for all non-primitive fluents R through the axioms $r \Leftarrow C$ in D (Thiébaux, Hoffmann, and Nebel 2005).
- I is given by a set of F-clauses so that the possible initial situations are the truth valuations over F that satisfy I.
- Actions have empty preconditions and conditional effects of the form $C \rightarrow C'$, where C is a set of literals over $F \cup R$ and C' is a set of literals over F only.
- The observable fluents $O \subseteq R$ denote the information perceived by the agent: an observation o is the set of O-literals that are true in a given state. The observation that corresponds to state s is denoted as $o(s)$, and the set of all observations as O^*.

While the solution of control problems can be expressed in many forms, including policies mapping belief states into actions, contingent plans, and trees, we consider solutions that are represented by *finite-state controllers* (FSCs) of the form $\mathcal{C} = \langle Q, A, O^*, \delta, q_0\rangle$, where Q is a set of *states or nodes*,[1] A and O^* are sets of actions and observations, $\delta : O^* \times Q \rightarrow A \times Q$ is a (partial) transition function that maps *observation and node pairs* into *action and node pairs*, and $q_0 \in Q$ is the initial node. The nodes serve as the controller memory allowing the selection of different actions given the same observation. A FSC with one node is memoryless.

Controllers are represented in two ways: graphically, using circles to represent nodes and edges with labels to represent transitions (Fig. 1), and as sets of tuples $t = \langle o, q, a, q'\rangle$ that express that δ is defined on $\langle o, q\rangle$ and maps it to $\langle a, q'\rangle$.

[1] Henceforth, we refer to *controller states* as *nodes*.

A controller $\mathcal{C}$ provides an specification of the action a_{i+1} to do next after a given observation-action sequence $\langle o_0, a_0, \ldots, a_i, o_{i+1}\rangle$. The action to do at time $i = 0$ is a if $t = \langle o_0, q_0, a, q'\rangle$ is in $\mathcal{C}$ and $o(s_0) = o_0$, and q' is the controller node that results at time $i = 1$. Similarly, the action to do at time i in the state s is a if $t = \langle o, q, a, q'\rangle$ is in $\mathcal{C}$, $o(s) = o$, and the controller node at time i is q.

A controller *solves* a problem if all the state trajectories that it produces, starting from an initial state $s_0 \in I$, reach a goal state. This is a weak form of solution as it does not demand that all such trajectories *terminate* in a goal state. This difference does not matter when the goals are observable but is relevant otherwise.

Formulation

The key result in our work is that the problem of deriving a controller $\mathcal{C}_N$ with N nodes for a control problem P can be translated into the problem of solving a *conformant planning problem* P_N (Bonet, Palacios, and Geffner 2009). This translation performs basically the following tasks:

1. it translates the observations $o \in O^*$ and the controller nodes $q \in Q$ into fluents in P_N,
2. it sets the fluent (corresponding to) q_0 to true in the initial situation, and sets all fluents q, with $q \neq q_0$, and all fluents o to false,
3. it makes the effects of the actions a in P conditional on each observation o and fluent q by defining 'controller actions' $b(t)$, for each tuple $t = \langle o, q, a, q'\rangle$, that behave like a when o and q are true, and replace q by q',
4. it captures the effects of the actions on the non-primitive fluents by means of a single 'ramification action', and
5. it assures that all controller actions used in a solution are pairwise consistent; i.e., that no plan contains actions $b(\langle o, q, a, q'\rangle)$ and $b(\langle o, q, a', q''\rangle)$ with $\langle a, q'\rangle \neq \langle a', q''\rangle$.

The problem P_N is conformant because the uncertainty in the initial situation of P is transferred into the uncertainty about the initial situation of P_N, while the observations $o \in O^*$ are *compiled away* into the conditional effects of the actions of P_N.

For solving P_N, a sound and complete translation K_{S0} that transforms P_N into a classical planning problem $K_{S0}(P_N)$ is used (Palacios and Geffner 2009). The resulting classical planning problem, $K_{S0}(P_N)$, is solved with either a sequential suboptimal heuristic-search planner, or an optimal parallel SAT-based planner. For lack of space, we omit further details. We mention, however, that the translation is *sound and complete*, meaning that there is a controller $\mathcal{C}_N$ with N nodes for solving P iff there is a plan for the classical planning problem $K_{S0}(P_N)$. In that case, the controller can be read off the plan. There is no free lunch though, and both the translation of P into P_N, and the translation of P_N into $K_{S0}(P_N)$ are exponential in the worst case. The experiments below, however, show that this worst-case behavior is not necessarily a practical impediment.

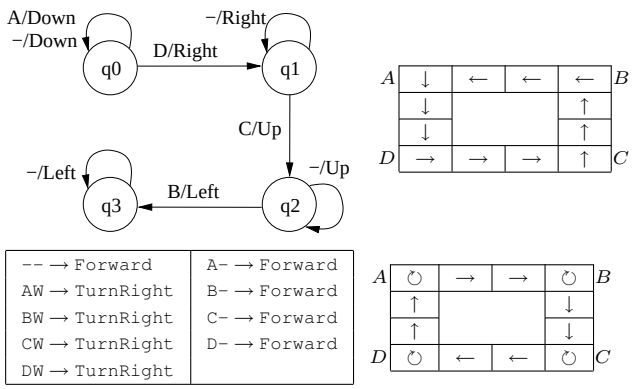

Figure 2: *Top:* 4-state controller obtained for the instance of Hall-A shown on right, with resulting execution. *Bottom:* memoryless controller obtained for the instance of Hall-R shown on right, with resulting execution. In Hall-A agent moves in each of the four directions, in Hall-R it only moves forward and rotates. Both controllers generalize to Halls of any size and work in the presence of noisy actions.

Experiments

We computed controllers for several problems as described next; further details can be found in Bonet, Palacios, and Geffner (2009).

Halls

The problem in Fig. 1 is the version 1×5 of the Halls domain. The $n \times n$ version, includes four $1 \times n$ halls arranged in a square, and observable marks A, B, C, D at the four corners. Starting in A, the robot has to visit all the marked positions and return to A. We consider two representations for the problem: in Hall-A, there are four actions that move the robot along each compass direction, in Hall-R, there are actions to move forward and to turn 90° left or right, and the presence of a wall in front of the robot can be detected (W).

A 4-state controller obtained for the 4×4 instance of Hall-A, and a memoryless controller obtained for a 4×4 instance of Hall-R are shown in Fig. 2. The arrows in the cells show the execution that results when the controller is applied to the initial state where the agent is at A. Both controllers generalize to Halls of any dimension, and work also in the presence of noise in both the initial situation and in the action effects. This *generalization* is achieved in spite of having inferred the controller from a fixed initial state, and results from the *change of representation*: sequential plans do not generalize as they do not represent finite-state controllers, unless we associate controller nodes with time indices. By removing the dependence on time indices, the generalization is achieved. Another way to look at the controllers, is as contingent plans in a language where looping constructs are allowed (Levesque 2005).

Blocks

Blocks is the problem of picking up a green block from a tower with n blocks of different colors. We encode the domain with three actions, $Unstack$, $Drop$, and $Collect$, that take no arguments and have conditional effects. The first is that unstack x clears y and puts x in the gripper, if x is clear and is on y; the second that a block can be discarded when held in the gripper; the third, that the goal is achieved if the collect action is done while a green block is being held (a collect action with a block of a different color results in a dead-end). The observable fluents are whether the top block in the tower is green ('G'), and whether a block is being held ('H'). Thus the condition that the block being held is green is not observable, and a memoryless controller would not solve the problem. A 2-state controller that generalizes to any number of blocks is shown in Fig. 3 (Left).

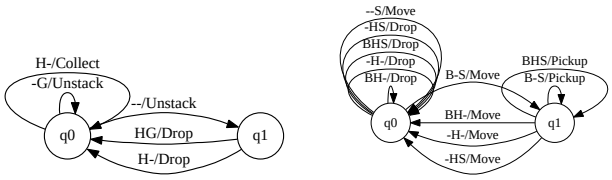

Figure 3: *Left.* Blocks: 2-state controller for collecting a green block in a tower, by observing whether the top block is green and whether an object is being held. *Right.* Gripper: 2-state controller for the instance $(3, 5)$ that consists of a robot with 3 grippers and an uncertain number of balls, from 1 to 5. The controller generalizes for problems with an arbitrary number of balls and grippers.

FNU → WanderForTrash	AAU → Grab
FAU → WanderForTrash	AFU → WanderForTrashcan
FFU → WanderForTrash	AFU → WanderForTrashcan
NAU → MoveToTrash	FNH → MoveToTrashcan
NNU → MoveToTrash	FAH → Drop
NFU → MoveToTrash	FAH → Drop
ANU → Grab	

Figure 4: Memoryless Controller for Trash Collecting: first position in observation vector refers to how far is the trash (Far, Near, At), the second to how far is the trash can (Far, Near, At), and the third, to whether trash is being held.

Gripper

In this problem, a robot must carry balls from room B to room A. The robot can move between rooms, and it can pick up and drop balls using its grippers. The observations consist of whether there are balls left in B ('B'), whether there is space left in the grippers ('S'), and whether the robot is holding some ball ('H'). The robot cannot directly observe its location yet it is initially at room A with certainty. The instance (n, m) refers to a problem with n grippers and an uncertain number of balls in room B, that could range from 1 to m. Fig. 3 (Right) shows the controller obtained for the instance $(3, 5)$ which also works for problems (n, m) for arbitrary n and m. The robot goes first to B, and picks up balls until no space is left in the grippers, then it moves to A, where it drops all the balls, one by one, repeating the cycle, until no balls are left in B and the robot is not holding any ball. In this case, the goal is observable and is true when neither B nor H are true.

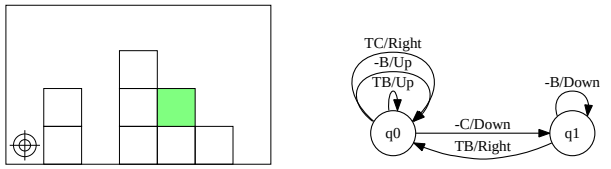

Figure 5: *Left:* The visual marker shown as an 'eye' must be placed on a green block in the blocks-world scene shown, where the location of the green block is unknown. *Right:* The controller derived for the instance that works also for any number and configuration of blocks

Trash Collecting

Figure 4 shows a controller for a trash-can collecting robot (Connell 1990; Murphy 2000; Mataric 2007), that can wander for a target object until the object is near, can move to an object if the object is near, can grab an object if located right where the object is, and can drop an object if the object is being held. The task is to move around, wandering for trash, and when a piece of trash is being held, wander for a trash can to drop it. In the encoding, the observable fluents are trash-held, far-from-X, near-X and at-X where X is trash or trashcan. The observations correspond to vectors ABC where A refers to how far is the trash (Far, Near, At), B refers to how far is the trash can (Far, Near, At), and C refers to whether the trash is being held (Held, Unheld).

Moving a Visual Marker

The second blocksworld problem is about placing a visual marker over a green block whose location is unknown (Fig. 5), and is inspired by the use of deictic representations (Chapman 1989; Ballard et al. 1997). The visual marker, initially at the lower left corner, can be moved along the four compass directions between the cells of the scene, one cell at a time. The observations are whether the cell beneath the marker is empty ('C'), is a non-green block ('B'), or is a green block ('G'), and whether it is on the table ('T') or not ('$-$'). We obtained controllers for different instances yet none generalized over arbitrary configurations. An instance (n, m) contains m horizontal cells and n blocks in some disposition. By restricting the left/right movements of the visual marker to the level of the table (i.e., when 'T' is observed to be true), we obtained a controller that works for *any* number of blocks and cells. The controller is shown on the right of Fig. 5; it basically searches for a tower with a green block from left to right, going all the way up to the top in each tower, then going all the way down to the table, and moving to the right, and iterating in this manner until the visual marker reaches a green block.

Summary

We have presented the method for deriving finite-state controllers automatically recently introduced by Bonet, Palacios, and Geffner (2009). The problem of deriving a controller is converted into a conformant planning problem that is then transformed into a classical planning problem and solved by state-of-the-art planners. The controllers derived in this way are often general in the sense that they do not solve the original problem only, but many variations too, including changes in the size of the problem or in the uncertainty of the initial situation and action effects.

In the future, we would like to investigate the guarantees on the generalization achieved by the resulting controllers, the synthesis of controllers when actions have nondeterministic effects, and the problem of termination when the goal is not observable.

Acknowledgements The work of H. Geffner is partially supported by grant TIN2009-10232, MICINN, Spain.

References

Ballard, D.; Hayhoe, M.; Pook, P.; and Rao, R. 1997. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences* 20(04):723–742.

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

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

Buckland, M. 2004. *Programming Game AI by Example*. Wordware Publishing, Inc.

Chapman, D. 1989. Penguins can make cake. *AI magazine* 10(4):45–50.

Connell, J. H. 1990. *Minimalist Mobile Robotics*. Morgan Kaufmann.

Kaelbling, L. P.; Littman, M.; and Cassandra, A. R. 1999. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Levesque, H. 1996. What is planning in the presence of sensing? In *Proc. AAAI*, 1139–1146.

Levesque, H. 2005. Planning with loops. In *Proc. IJCAI*, 509–515.

Littman, M. L. 1994. Memoryless policies: Theoretical limitations and practical results. In Cliff, D., ed., *From Animals to Animats 3*. MIT Press.

Mataric, M. J. 2007. *The Robotics Primer*. MIT Press.

Meuleau, N.; Peshkin, L.; Kim, K.; and Kaelbling, L. P. 1999. Learning finite-state controllers for partially observable environments. In *Proc. UAI*, 427–436.

Murphy, R. R. 2000. *An Introduction to AI Robotics*. MIT Press.

Palacios, H., and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research* 35:623–675.

Poupart, P., and Boutilier, C. 2003. Bounded finite state controllers. In *Proc. NIPS*, 823–830.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1–2):38–69.