Policies that Generalize: Solving Many Planning Problems with the Same Policy

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

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

Abstract

We establish conditions under which memoryless policies and finite-state controllers that solve one partially observable non-deterministic problem (PONDP) generalize to other problems; namely, problems that have a similar structure and share the same action and observation space. This is relevant to *generalized planning* where plans that work for many problems are sought, and to *transfer learning* where knowledge gained in the solution of one problem is to be used on related problems. We use a logical setting where uncertainty is represented by sets of states and the goal is to be achieved with certainty. While this gives us crisp notions of solution policies and generalization, the account also applies to probabilistic PONDs, i.e., Goal POMDPs.

1 Introduction

Consider a vacuum cleaning robot in a linear 1×5 grid, with the robot initially in the leftmost cell, and each of the n cells being either clean or dirty [Russell and Norvig, 2002]. If the robot can sense whether the current cell is clean or not, and whether it is in the last (rightmost) cell, the goal of having all cells clean can be achieved with the policy "if dirt, suck it; if clean and not in the last cell, move right". This is an example of a memoryless policy that maps observations into actions. Compact memoryless policies and finite-state machines are heavily used in robotics and video games as they provide a simple language for writing controllers that are robust and general [Murphy, 2000; Buckland, 2004]. For example, the vacuum cleaning policy for the 1×5 grid will also work in a 1×10 grid, and actually in *any* $1 \times n$ grid. Moreover, it will work no matter how dirt is distributed, and even when the actions of moving right and sucking dirt may fail to achieve their effects sometimes. All these problem variations indeed appear to share a *common structure*, aside from the same set of actions and observations, and once the policy is derived from one problem it applies to all the variations. It is however not straightforward to pinpoint what is the common structure that supports such generalization. Indeed, what does the 1×5 instance with dirt all over has in common with the 1×10 instance where dirt is in the second cell only? What do these instances have in common with non-deterministic variants where actions may fail? And, what part of the structure is missing when the policy does *not* work?

Our aim in this paper is to provide a characterization of the common structure that allows for policy generalization. We use a logical setting where uncertainty is represented by sets of states and the goal is to be achieved with certainty. In this setting, the notions of solution policy and generalization become very crisp: a policy solves a problem or not, and it generalizes to another problem or not. We thus ignore considerations of costs, quality or rewards, and do not consider probabilities explicitly. Our results, however, do apply to the probabilistic setting where goals are to be achieved with probability 1. This is a result of the correspondence between *proper policies* in stochastic systems and *strong cyclic policies* in purely non-deterministic ones [Bertsekas, 1995; Cimatti *et al.*, 1998; Geffner and Bonet, 2013]. The theme of policies that are general has surfaced in generalized planning [Hu and De Giacomo, 2011; Srivastava *et al.*, 2011a] and reinforcement learning [Taylor and Stone, 2009], and the relation to these research threads will be discussed.

The paper is organized as follows. We introduce the model and memoryless policies, define reductions, consider refinements, and extend the results to finite-state controllers. We also illustrate the ideas on examples and discuss related work.

2 Model, Policies, Solutions

A *partially observable non-deterministic* problem (PONDP) is a tuple $P = \langle S, S_0, T, A, F, o \rangle$ where

1. S is a finite set of states s,
2. S_0 is the set of possible initial states $S_0 \subseteq S$,
3. T is the set of goal states $T \subseteq S$,
4. $A(s) \subseteq A$ is set of actions applicable in $s \in S$,
5. $F(a, s) \subseteq S$ denotes the successor states, $s \in S, a \in A(s)$
6. o is the observation function $o(s) \in \Omega_o$ where Ω_o is the set of possible observations.

PONDPs are similar to POMDPs with deterministic sensing (Partially Observable Markov Decision Processes; [Kaelbling *et al.*, 1998]) and in particular to *Goal* POMDPs where goals have to be achieved with certainty [Bonet and Geffner, 2009]. A key difference is that uncertainty is represented by

sets of states rather than probability distributions. For policies that must achieve the goal with certainty, however, the distinction is irrelevant; all that matters is which observations and state transition are possible, not their exact numerical probabilities [Bertsekas, 1995]. Similarly, noisy sensing can be compiled into deterministic sensing by extending the states with the observations [Chatterjee and Chmelík, 2015].

The basic notions are reviewed next. A sequence $\tau = \langle s_0, s_1, \ldots \rangle$ of states is a *state trajectory* in P iff $s_0 \in S_0$ and for every state s_{i+1} in τ there is an action a_i in $A(s_i)$ such that $s_{i+1} \in F(a_i, s_i)$. A trajectory τ *reaches* a state s if τ contains s, and τ is *goal reaching* if it reaches a goal state. A state s is *reachable* if there is a trajectory that reaches s.

A *memoryless policy* is a partial function $\mu : \Omega_o \to A$ where $\mu(\omega)$ is the action prescribed in all the states s with $o(s) = \omega$, with the only restriction that the action $\mu(\omega)$ must be applicable at s; $\mu(o(s)) \in A(s)$. The trajectories generated by μ, called the *μ-trajectories*, are the finite or infinite sequences $\langle s_0, s_1, \ldots \rangle$ where $s_0 \in S_0$, and $s_{i+1} \in F(\mu(o(s_i)), s_i)$ for all the states s_{i+1} in the sequence. A μ-trajectory is *maximal* if it is infinite or it if it ends in a state s for which $\mu(o(s))$ is undefined. Sometimes actions are included in μ-trajectories which are then denoted as $\langle s_0, a_0, s_1, a_1, s_2, \ldots \rangle$ where $a_i = \mu(o(s_i))$. A state is reachable by μ if there is a μ-trajectory that reaches the state.

A *transition* in P is a triplet (s, a, s') such that $s' \in F(a, s)$ and $a \in A(s)$. A transition (s, a, s') occurs in a trajectory $\langle s_0, a_0, s_1, a_1, \ldots \rangle$ if for some i, $s = s_i$, $a = a_i$, and $s' = s_{i+1}$. A trajectory τ is *fair* if it is finite and maximal, or if for any two transitions (s, a, s') and (s, a, s'') in P for which $s' \neq s''$, if one transition occurs infinitely often in τ, the other occurs infinitely often in τ as well.

A policy *μ solves P* iff every fair μ-trajectory is goal reaching. If μ solves P, μ is said to be a solution to P, and a *valid* policy for P. Solutions are thus *strong cyclic policies* [Cimatti *et al.*, 1998]. A solution μ is *strong* when the fair μ-trajectories are all *finite* and end in a goal state.

3 Characterizing the Common Structure

We define a structural relation among PONDPs and establish properties that follow from it. The reduction of a problem P into a "smaller" problem P' allows us to establish sufficient conditions under which a policy that solves P' also solves the "larger" problem P.

Definition 1 (Reduction). *Let $P = \langle S, S_0, T, A, F, o \rangle$ and $P' = \langle S', S'_0, T', A', F', o' \rangle$ be two problems with the same sets of observations; i.e., $\Omega_o = \Omega_{o'}$. A function $h : S \to S'$ reduces P to P' iff*

- *R1. $A'(h(s)) \subseteq A(s)$ for all $s \in S$,*
- *R2. $o(s) = o'(h(s))$ for all $s \in S$,*
- *R3. if $s' \in F(a, s)$ then $h(s') \in F'(a, h(s))$ for all $s, s' \in S$ and $a \in A'(h(s))$,*
- *R4. if $s_0 \in S_0$ then $h(s_0) \in S'_0$ for all $s_0 \in S_0$,*
- *R5. if $s \notin T$ then $h(s) \notin T'$ for all $s \in S$.*

The intuition for R1–R5 is that a reduction 1) doesn't introduce non-applicable actions, 2) preserves observations, and 3) maps transitions into transitions, initial states into initial states, and non-goal states into non-goal states.

Reductions are not symmetric in general, and thus, they are different than other types of relations among structurally similar problems like bisimulations [Milner, 1989]. In particular, deterministic systems can be reduced into non-deterministic systems but the opposite reduction may not be feasible.

Example: Countdown. Let us first consider the problem of driving a state counter from some value to zero using the actions Inc and Dec, and a sensor that tells if the goal has been achieved. The problem is the tuple $P_n = \langle S, S_0, T, A, F, o \rangle$ where $S = \{0, 1, \ldots, n\}$, $S_0 = S$, $T = \{0\}$, A contains the actions Inc and Dec, $o(s)$ is 0 or "+" respectively iff s is 0 or positive, and $s' \in F(a, s)$ iff $s' = s + 1$ when $a = Inc$ and $s < n$, $s' = s - 1$ when $a = Dec$ and $s > 0$, and else, $s = s'$. The simple policy μ where $\mu(\omega) = Dec$ if $\omega = +$ solves the problem. The function $h(s) = 0$ if $s = 0$ and $h(s) = 1$ if $s > 0$ maps the problem P_n into the *abstract* and smaller problem $P' = \langle S', S'_0, T', A', F', o' \rangle$ where all positive states in P_n are represented by a single state. More precisely, S' contains just the *two states* 0 and 1, $S'_0 = S'$, $T' = T$, $A' = A$, $o(s')$ is 0 or + according to whether s' is 0 or 1, $F'(a, 0) = F(a, 0)$ for $a \in \{Inc, Dec\}$, $F'(Dec, 1) = \{1, 0\}$, and $F'(Inc, 0) = F'(Inc, 1) = \{1\}$. The function h reduces the *deterministic* problem P_n, for any n, into a single *non-deterministic* problem P' where non-determinism is used in the transition $F'(Dec, 1) = \{1, 0\}$ to capture the possible transitions from different states s in P_n that are reduced to the same state $h(s) = 1$ in P'.

Clearly, the h-mapping satisfies R1, R2, R4, and R5 as it doesn't introduce non-applicable actions (P and P' have the same actions and they are always applicable), s and $h(s)$ give rise to the same observations, initial states of P_n map into initial states of P', and non-goal states are preserved (i.e. $s \neq 0$ implies $h(s) \neq 0$). It is also simple to check that h satisfies condition R3. Thus, h reduces P_n to P'. Since this reduction works for any $n > 1$, this means that all problems P_n can be reduced to the problem P' where a single state is being used to represent n states in P_n. The memoryless policy $\mu(\omega) = Dec$ if $\omega = +$ solves P' and each problem P_n. ∎

We turn now to the conditions under which a policy that solves a problem P' will also solve problems that can be reduced to P'. First of all, from the definition above, it follows that when h reduces P into P', any memoryless policy μ for P' is a memoryless policy for P; i.e., μ is *executable* in P:[1]

Theorem 2 (Executability). *If μ is a policy for P' and h reduces P into P', then μ is a policy for P.*

We consider next the trajectories τ generated by the policy μ in P, and the sequences $h(\tau)$ that such trajectories induce over P', where $h(\tau)$ is the trajectory $\tau = \langle s_0, a_0, s_1, a_1, \ldots \rangle$ with the states s_i replaced by $h(s_i)$; i.e., $h(\tau) = \langle h(s_0), a_0, h(s_1), a_1, \ldots \rangle$. The first result is that when h is a reduction of P into P', h maps trajectories τ generated by a policy μ in P into sequences $h(\tau)$ that are μ-trajectories in P':

[1] Formal proofs omitted for lack of space.

Theorem 3. *For a reduction h of P into P', if μ is a policy for P' and $\tau = \langle s_0, a_0, s_1, a_1, \ldots\rangle$ is a μ-trajectory in P, then $h(\tau)$ is μ-trajectory in P'.*

From R5, it follows that if the μ-trajectory τ doesn't reach a goal in P, the μ-trajectory $h(\tau)$ doesn't reach a goal in P' either. If μ is a solution of P', however, and the μ-trajectory $h(\tau)$ is *fair*, then $h(\tau)$ must reach the goal in P'. Hence,

Theorem 4 (Structural Generalization). *Let μ be a policy that solves P' and let h be a reduction from P into P'. Then μ solves P if h maps the* fair *μ-trajectories τ in P into trajectories $h(\tau)$ that are* fair *in P'.*

We'll illustrate the use of this theorem and some of its corollaries below. A first corollary is that a *sufficient* condition for a policy μ to generalize from P' to a problem P that can be reduced to P' arises when μ is *terminating* in P:

Theorem 5. *Let μ be a policy that solves P'. The policy μ solves a problem P that can be reduced to P' if all the* fair *μ-trajectories in P are finite.*

One way in which we'll be able to show the termination of a policy μ on problems P is by showing that the policy μ is *monotonic*, meaning that when action $\mu(o(s))$ is applied to any (non-goal) state s, then s will not be reachable from the resulting states s' while following the policy. Often this can be shown with structural arguments in a simple manner. For example, in the Countdown problem, if $s > 0$ and $\mu(o(s)) = Dec$, then s won't be reachable from the resulting state if the policy μ doesn't include the action Inc in some (reachable) state. Clearly if a policy μ is *monotonic* in P, then μ must terminate in P. In other cases, however, showing termination may be as hard as showing that μ solves the problem itself.

Example: Countdown (continued). The problem P_n with $n+1$ states $i = 0, \ldots, n$ reduces to the non-deterministic problem P' with 2 states through the function $h(0) = 0$ and $h(i) = 1$ for $i > 0$. The policy $\mu(\omega) = Dec$ for $\omega = +$ solves P' because $(1, Dec, 0) \in F'$ and hence fair μ-trajectories that start in the state 1 in P' reach goal 0. Theorem 5 says that μ solves P_n for any $n > 1$ if μ-trajectories in P_n terminate, which holds since μ is *monotonic* (see above).

It's also important to see variations on which the generalization does *not* work. For this, let P'_n be a problem like P_n but with a "buggy" Dec action that sets the counter back to n from 1; i.e., with $F(Dec, 1) = \{n\}$. The function h still reduces P'_n into P' because the new transition $n \in F(Dec, 1)$ requires the transition $h(n) \in F'(Dec, h(1))$ which is true in F' as $h(n) = h(1) = 1$ and $1 \in F'(Dec, 1)$. Yet, the μ-trajectories τ that start in any state $i > 0$ in P'_n, consist of the sequence $i, i-1, \ldots, 1$ followed by the loop $n, n-1, \ldots, 1$. Such trajectories τ are *fair* in P'_n, that is deterministic, but result in trajectories $h(\tau) = \langle 1, Dec, 1, Dec, \ldots\rangle$ that are *not* fair in P', as the transition $(1, Dec, 0)$ in P' never occurs. Thus, h reduces P'_n into P' but does *not* reduce fair trajectories in P'_n into fair trajectories in P', and Theorem 4 can't be used to generalize μ from P' into P'_n. ∎

4 The Structure of Abstract Problems

In the Countdown example, the non-deterministic problem P' is used to show that the policy that solves the instance P_1 generalizes to any instance P_n, $n > 1$. The problem P', however, is very similar to P_1; indeed, it is the problem P_1 with an additional state transition (s, a, s') for $a = Dec$ and $s = s' = 1$. As we will see, this pattern is common, and we will often show that a solution to P generalizes to another problem Q in the same class by considering a problem P' that is like P but augmented with an *extra set of possible transitions* E. We make explicit the structure of such problems P' by writing them as $P + E$. Recall that a transition (s, a, s') is in P when a is applicable in s and s' is a possible successor, i.e., $a \in A(s)$ and $s' \in F(a, s)$.

Definition 6 (Admissible Extensions). *Let μ be a policy for problem P. An* extension *E to P* given *μ is a set of triplets (s, a, s') for states s and s' in P and $a = \mu(s)$ such that (s, a, s') is not a transition in P. The extension is* admissible *if each state s' in such triplets is reachable by μ in P.*

Definition 7 (Structured Abstractions). *Let μ be a policy for P and let E be an admissible extension to P given μ. $P + E$ denotes the problem P augmented with the transitions in E.*

In Countdown, P' is $P_1 + E$ where $E = \{(s, Dec, s)\}$ for $s = 1$ is an admissible extension. Clearly,

Theorem 8. *If μ is a policy that solves P, and E is an admissible extension of P given μ, μ solves $P + E$.*

For using the structure of problems $P + E$ for generalization, however, a corresponding notion of fairness is required:

Definition 9 (P-fairness). *Let μ be a policy for P and let E be an admissible extension of P given μ. A μ-trajectory τ in $P + E$ is* P-fair *if for any transition (s, a, s') in P that occurs infinitely often in τ, other transitions (s, a, s'') in P occur infinitely often in τ too.*

In other words, a trajectory τ generated by μ in $P + E$ is P-fair if it doesn't skip P-transitions forever although it may well skip E transitions. Our main result follows:

Theorem 10 (Main). *Let μ be a policy that solves P and let h be a reduction from a problem Q into $P + E$ where E is an admissible extension of P given μ. Then μ solves Q if h maps the* fair *μ-trajectories τ in Q into trajectories $h(\tau)$ that are* P-fair *in $P + E$.*

Thus a policy μ for P generalizes to Q if 1) Q is structurally similar to P in the sense that Q can be reduced to $P + E$, and 2) the reduction h maps μ-trajectories τ in Q into $h(\tau)$ trajectories in $P + E$ that eventually evolve according to P, not $P + E$. Theorem 4 is a special case of this result when E is empty. When P is deterministic, P-fairness can be shown by bounding the number of E-transitions in the executions:

Theorem 11. *Let τ be a μ-trajectory for the problem $P + E$. If P is deterministic and transitions (s, a, s') from E occur a* finite *number of times in τ, then τ is P-fair.*

5 Projections and Restrictions

We consider next two ways of replacing the problems P and Q mentioned in the theorems by suitable simplifications.

5.1 Projections over the Observations

A standard way by which a problem P can be reduced into a smaller problem P' is by projecting P onto the space of observations. We formalize this in terms of a function h_{obs} and a projected problem P^o:

Definition 12. *For $P = \langle S, S_0, T, A, F, o\rangle$, the function h_{obs} is defined as $h_{obs}(s) = o(s)$, and the problem P^o as the tuple $P^o = \langle S', S'_0, T', A', F', o'\rangle$ with*

1. $S' = \{\omega \mid \omega \in \Omega_o\}$,
2. $A'(\omega) = \cap_s A(s)$ *for all s in S such that* $o(s) = \omega$,
3. $o'(\omega) = \omega$,
4. $\omega' \in F'(a, \omega)$ *iff there is* $s' \in F(a, s)$ *such that* $\omega = o(s)$, $\omega' = o(s')$, *and* $a \in A'(\omega)$,
5. $S'_0 = \{o(s) \mid s \in S_0\}$,
6. $T' = \{o(s) \mid s \in T\}$.

By construction, the function h_{obs} complies with conditions R1–R4 for being a reduction of problem P into P^o but it does not necessarily comply with R5. For this, an extra condition is needed on P; namely, that goals states s are observable; i.e., $o(s) \neq o(s')$ when $s \in T$ and $s' \notin T$:

Theorem 13. *If $P = \langle S, S_0, T, A, F, o\rangle$ is a problem with observable goals, $h_{obs}(s)$ reduces P into P^o.*

It follows that if a policy μ solves P^o, μ solves P too if the goals are observable and μ terminates in P.

5.2 Restricted Problems

For showing that a policy μ generalizes from a problem P to a family of problems Q, it is not necessary to use reductions that consider all actions and states. Suitable restrictions can be used instead:

Definition 14. *A restriction P_R of $P = \langle S, S_0, T, A, F, o\rangle$ is a problem $P_R = \langle S_R, S_0, T_R, A_R, F_R, o_R\rangle$ that is like P except for 1) the sets $A_R(s)$ of actions applicable at state s may exclude some actions from $A(s)$, and 2) the set S_R of states excludes from S those that become unreachable. T_R is $T \cap S_R$, while F_R and o_R are the functions F and o in P restricted to the states in S_R.*

A particular type of restriction P_R arises when the sets of applicable actions is restricted to contain a single action, and in particular, the action determined by μ. We denote with P_μ the restriction P_R of P where $A_R(s) = \{\mu(o(s))\}$ for each state s. For a policy μ to generalize from P to Q, it suffices to consider functions h that reduce Q_μ into $P_\mu + E$:

Theorem 15. *Let μ be a policy that solves P and let h be a reduction from Q_μ into $P_\mu + E$ where E is an admissible extension of P given μ. Then μ solves Q if h maps the* fair *μ-trajectories τ in Q_μ into trajectories $h(\tau)$ that are P-*fair *in $P_\mu + E$.*

Restrictions also enable the use of the reduction h_{obs} in problems where the goals are *not* observable. Indeed, P_μ reduces to P^o_μ if the states s that may be confused with goal states s' in P are not *reachable* by μ in P.

6 Examples

Dust-Cleaning Robot. There is a $1 \times n$ grid whose cells may be dirty or not, and a cleaning robot that is positioned on the leftmost cell. The task for the robot is to clean all the cells. The actions allow the robot to move right and to suck dirt. Formally, $P = P_n$ is the problem $\langle S, S_0, T, A, F, o\rangle$ whose states are of the form $\langle i, d_1, \ldots, d_n\rangle$, where $i \in [1, n]$ denotes the robot location and $d_k \in \{0, 1\}$ denotes whether cell k is clean or dirty. The set S_0 contains the 2^n states $\langle 1, d_1, \ldots, d_n\rangle$ where the robot is at the leftmost cell. The set T of goals contains all states $\langle i, d_1, \ldots, d_n\rangle$ where each $d_k = 0$. The actions in A are $Suck$ and $Right$, both applicable in all the states, with $F(Suck, \langle i, d_1, \ldots, d_i, \ldots, d_n\rangle) = \{\langle i, d_1, \ldots, 0, \ldots, d_n\rangle\}$, $F(Right, \langle i, d_1, \ldots, d_n\rangle) = \{\langle i + 1, d_1, \ldots, d_n\rangle\}$ for $i < n$, and $F(Right, \langle i, d_1, \ldots, d_n\rangle) = \{\langle i, d_1, \ldots, d_n\rangle\}$ for $i = n$. The observation function o is such that $o(\langle i, d_1, \ldots, d_n\rangle)$ is the pair $\langle o_1, o_2\rangle$ where $o_1 \in \{e, m\}$ reveals the end of the corridor (e), and $o_2 \in \{d, c\}$ whether the current cell is dirty or clean.

We consider a reduction of the *deterministic* problem P_n with $n \times 2^n$ states into the problem $P + E$ with 8 states where P is P_2, and E comprises two transitions (s, a, s') where $s = \langle 1, d'_1, d'_2\rangle$, $a = Right$, and s' is either $\langle 1, 0, d'_2\rangle$ or $\langle 1, 1, d'_2\rangle$. That is, $P + E$ is like P_2 but when the robot is not in the rightmost cell, the action $Right$ can "fail" leaving the robot in the same cell, making the cell either clean or dirty.

The policy μ, $\mu(\langle o_1, o_2\rangle) = Suck$ if $o_2 = d$, and $\mu(\langle o_1, o_2\rangle) = Right$ if $o_1 = m$ and $o_2 = c$, solves the problem P_2 and can be shown to solve any problem P_n. For this, the restriction Q_μ of $Q = P_n$ can be reduced into the restriction $P_\mu + E$ for $P = P_2$ with the function $h(\langle i, d_1, \ldots, d_n\rangle) = \langle i', d'_1, d'_2\rangle$ such that $i' = 1$ and $d'_1 = d_i$ for $i < n$, and else $i' = 2$ and $d'_1 = 0$, with $d'_2 = d_n$ in both cases. In addition, since the policy μ terminates in any problem P_n as it's *monotonic* (actions $Suck$ and $Right$ in μ have effects that are not undone by μ), the trajectories $h(\tau)$ in Theorem 15 must be finite and fair, from which it follows that μ solves P_n for any n and any initial configuration of dirt. ■

Wall Following. We consider next the problem of finding an observable goal in a room with some columns. The agent can sense the goal and the presence of a wall on its right side and in front. The actions are move forward (F), rotate left (L) and right (R), and rotate left/right followed by a forward movement (LF/RF). An $n \times m$ instance is a tuple $P_{n,m} = \langle S, S_0, T, A, F, o\rangle$ whose states s are of the form (c, d) where $c \in [1, n] \times [1, m]$ is a cell in the grid and $d \in \{n, s, e, w\}$ is the heading of the agent. The initial state is $(1, 1, e)$ and the observable goal state is associated with a goal cell. The state transitions are the expected ones, except that actions that try to leave the grid and actions that are applied at states (c, d) where c is in a column have no effect. For simplicity, the columns are modeled as observable cells to be avoided. The observation function o maps each state s into an observation in $\Omega_o = \{$★, ▷⋮, ▷|, ▷⋮$\}$ where $o(s) =$ ★ if $s \in T$, and $o(s)$ is ▷⋮, ▷|, or ▷⋮ according to whether the agent senses wall on its right side and front.

The wall-following policy μ that we consider executes F when observing ▷⋮, L when observing ▷|, and RF when ob-

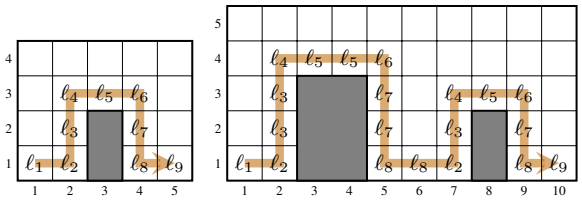

Figure 1: Two wall following problems P (Left) and Q (Right), along with μ-trajectories that end at the goal. Policy μ for P solves problems Q with any number of columns of any width and height.

serving ▷⋮. We want to illustrate how this policy generalizes to the class of $P_{n,m}$ problems consisting of any number of columns separated by two or more spaces, where the columns have any height shorter than $m-1$ and any (positive) width, and the goal is in first row to the right of an empty cell. The policy generalizes to a larger class of problems but these restrictions are sufficiently general and make the analysis simpler.

Figure 1 shows the problem P in this class that we use to show that μ solves the other problems such as the one shown on the right. The set of transitions E needed to reduce any Q_μ to $P_\mu + E$ includes the triplets (s, a, s') where $a = F$ and $\langle s, s' \rangle$ is either $\langle (\ell_8, e), (\ell_2, e) \rangle$, or $\langle (\ell_i, d), (\ell_i, d) \rangle$, for $i = 3, 5, 7, 8$, any heading d, and the cells ℓ_i shown on the left of Figure 1.

The function h that reduces Q_μ into $P_\mu + E$ maps states (c, d) in Q_μ into states $(g(c), d)$ in $P_\mu + E$, while preserving the heading d. The function g is obtained from the cell labeling shown in Fig. 1; namely, $g(c) = c'$ if cell c in Q_μ has the label c' which corresponds to a particular cell in P_μ on the left. While we are showing the function g for the particular problem Q shown, g can be defined to apply to any problem in the class.

In order to apply Theorem 15, we need to prove that h is a reduction, that E is an admissible extension of P given μ, and that the single trajectory τ generated by μ in Q maps into a trajectory $h(\tau)$ where E-transitions occur a finite number of times. The first two parts are left to the reader. The right part of Fig. 1 shows the trajectory τ that is finite so that the last condition is true as well. One could prove such termination by monotonicity arguments but more insight can be gained by looking at the trajectory $h(\tau)$ where states s in τ are replaced by $h(s)$. The sequence of cells in the states appearing in $h(\tau)$ corresponds to the sequence of labels ℓ_i shown in the figure on the right. Interestingly, it's possible to show that the set of label sequences that result for the different problems Q for which Theorem 15 applies given P, E, and h as above, corresponds to a *regular language*; namely, the language that is captured by the regular expression $\ell_1^+(\ell_2^2\ell_3^+\ell_4\ell_5^+\ell_6\ell_7^+\ell_8\ell_8^+)^*\ell_9$. ∎

7 Finite-State Controllers

A *finite-state controller* (FSC) is a tuple $\mathcal{C} = \langle Q, A, \Omega, \delta, q_0 \rangle$ where Q is the set of *controller states*, A and Ω are sets of actions and observations, $q_0 \in Q$ is the initial controller state, and δ is (partial) transition function that maps *controller state and observation pairs* $\langle q, o \rangle \in Q \times \Omega$ into *action and controller state pairs* $\langle a, q' \rangle \in A \times Q$. A controller $\mathcal{C}$ can also be represented by tuples $\langle q, o, a, q' \rangle$ that express that the transition function δ maps the pair $\langle q, o \rangle$ into the pair $\langle a, q' \rangle$.

A controller $\mathcal{C}$ generates a $\mathcal{C}$-trajectory made of interleaved sequences of pairs and actions where the pairs bundle a problem state s with a controller state q. That is, a $\mathcal{C}$-trajectory is a sequence $\langle (s_0, q_0), a_0, (s_1, q_1), a_1, \ldots \rangle$ where $s_0 \in S_0$ and q_0 is the initial controller state, $s_{i+1} \in F(a_i, s_i)$ for $i \geq 0$, and $\mathcal{C}$ contains the tuples $\langle q_i, o_i, a_i, q_{i+1} \rangle$ where $o_i = o(s_i)$ for $i \geq 0$. A controller $\mathcal{C}$ for a problem P must be executable and hence we assume that a tuple $\langle q, o, a, q' \rangle$ in $\mathcal{C}$ implies that $a \in A(s)$ for every state s with $o(s) = o$. A controller $\mathcal{C}$ solves P if all the fair $\mathcal{C}$-trajectories τ that it generates reach a goal state, where a trajectory is fair if it is finite and maximal, or transitions $\langle (s, q), a, (s', q') \rangle$ appear infinitely often in τ when transitions $\langle (s, q), a, (s'', q'') \rangle$ appear infinitely often for $s'' \neq s'$ and $s' \in F(a, s)$. The key difference with memoryless policies is that the action a_i selected at step i does not depend only on the observation o_i but also in the controller state q_i. The notions and results for memoryless controllers extend in a natural way to FSCs:

Theorem 16. *If $\mathcal{C}$ is a controller that solves P' and h reduces P into P', then $\mathcal{C}$ is controller for P, and for every $\mathcal{C}$-trajectory τ generated by $\mathcal{C}$ in P, there is a $\mathcal{C}$-trajectory $h(\tau)$ generated by $\mathcal{C}$ in P where $h(\tau)$ is τ with the (problem) states s replaced by $h(s)$.*

Theorem 17. *Let $\mathcal{C}$ be a finite-state controller that solves P and let h reduce Q into $P + E$ where E is admissible extension of P given $\mathcal{C}$. Then $\mathcal{C}$ solves Q if the* fair *$\mathcal{C}$-trajectories τ in Q map into trajectories $h(\tau)$ that are P-*fair *in $P + E$.*

A $\mathcal{C}$-trajectory τ is P-fair in $P + E$ when the sequence of states and actions in τ is P-fair in the sense defined above (no P-transitions skipped for ever). As before, the function h does not need to reduce Q into $P + E$ when the controller $\mathcal{C}$ is given, it suffices to reduce $Q_\mathcal{C}$ into $P_\mathcal{C} + E$ where $P_\mathcal{C}$ is the restriction of P under $\mathcal{C}$. That is, if P_T is defined as the problem P with the transitions (s, a, s') that are not in T excluded, $P_\mathcal{C}$ is P_T where T is the set of transitions (s, a, s') in P that appear in some $\mathcal{C}$-trajectory. With this notion,

Theorem 18. *Let $\mathcal{C}$ be a finite-state controller that solves P and let h reduce $Q_\mathcal{C}$ into $P_\mathcal{C} + E$ where E is admissible extension of P given $\mathcal{C}$. Then $\mathcal{C}$ solves Q if the* fair *$\mathcal{C}$-trajectories τ in $Q_\mathcal{C}$ map into trajectories $h(\tau)$ that are P-*fair *in $P_\mathcal{C} + E$.*

Example: Visual Marker. This problem is from Bonet *et al.* [2009], inspired by the use of deictic representations [Chapman, 1989; Ballard *et al.*, 1997], and where a visual marker needs to be placed over a hidden green block in a scene that contains a number of block towers. The marker can be moved one cell at a time in each of the four directions. Observations are 'G' for the green block, and pairs where the first component tells whether the marker is next to the table ('T') or not ('−'), and the second whether the cell beneath the marker is empty ('C') or a non-green block ('B'). An (n, m) instance represents a scene in a $n \times m$ grid where the towers do not reach the 'roof', the marker is initially located at the empty cell $(1, 1)$, and exactly one block is green. A simple instance P is shown in Fig. 2 along with the controller $\mathcal{C}$ derived by Bonet *et al.*, while a larger instance Q is shown in Fig. 3(b).

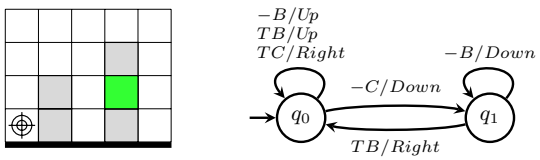

Figure 2: *a)* In problem P on the left, marker shown as 'eye' must be placed on hidden green block. *b)* FSC $\mathcal{C}$ with 2 states that solves P and any problem in the class.

The states in P are the pairs (c, g) where $c, g \in [1, 5] \times [1, 4]$ are the cells for the marker and the hidden green block respectively. The actions affect the cell for the marker leaving the other cell intact. The initial states are the pairs (c, g) where $c = (1, 1)$ and g is one of the 3 cells in the second tower which we denote as g_i for $i = 1, 2, 3$ from the bottom up. The goal states are the pairs (g_i, g_i) for $i = 1, 2, 3$. The observation function in P encodes the map as in Wall Following, so that the observation for the state (c, d) for $c = (2, 2)$ and $d = (4, 3)$ is $-B$, and for $c = d = (4, 3)$ is G.

For proving that the controller $\mathcal{C}$ for P generalizes to any problem Q, we consider a general reduction function h and extension E. The states $s = (c, g)$ in $Q_\mathcal{C}$ are mapped into states $h(s) = (f_1(c, g), f_2(g))$ in $P_\mathcal{C}$ that decompose h into two functions: one f_2, mapping goal cells into goal cells, the other f_1, mapping marker cells into marker cells with a dependence on the goal cell. The function $f_2(g)$ is $(4, 1)$ if $g = (x, 1)$, $(4, 2)$ if $g = (x, 2)$, and $(4, 3)$ if $g = (x, y)$ for $y > 2$. For describing the function $f_1(c, g)$ from states $s = (c, g)$ in $Q_\mathcal{C}$ into marker cells in $P_\mathcal{C}$, we label the latter as ℓ_1 to ℓ_8 as shown in Fig. 3 and consider two cases. If $c = (x, y)$, $g = (x', y')$ and $y = y'$, then the marker is in the goal tower in Q. Such states $s = (c, d)$ are mapped into cells in the goal tower in P; i.e. $f_1(c, d) = (4, y'')$ where $y'' = y$ if $y < 3$ and $y'' = 3$ otherwise. On the other hand, if $y \neq y'$, $f_1(c, d)$ is mapped into the unique cell in $\{\ell_1, \ldots, \ell_4\}$ in $P_\mathcal{C}$ that give rise to the same observation as (c, g) in $Q_\mathcal{C}$. Fig. 3(b) shows the values of the function $f_1(c, g)$ for $g = (5, 2)$ and the various cells c in $Q_\mathcal{C}$ for the particular problem Q shown.

The set of transitions E required for h to reduce $Q_\mathcal{C}$ into $P_\mathcal{C} + E$ expresses the idea in which the $\mathcal{C}$-trajectories τ in $Q_\mathcal{C}$ get mapped into trajectories $h(\tau)$ in $P_\mathcal{C} + E$ that go back and forth between the non-goal towers and empty cells in P until a goal tower is reached in Q. For this, 27 transitions (s, a, s') need to be added to P in E. If $s = (c, g)$ and $s' = (c', g')$, g and g' must be equal, and the 9 transitions for each of the three $g = g_i$ goal cells in P are as follows. For $a = Right$, if $c = (1, 1)$, c' is $(1, 1)$ and $(4, 1)$, and if $c = (2, 1)$, c' is $(1, 1)$, $(2, 1)$, and $(4, 1)$. For Up and $Down$, if $c = (2, 2)$, $c' = c$, and for Up only and $c = (4, 2)$, $c' = c$. Finally, for $a = Up$ and $c = (2, 1)$, $c' = (2, 3)$. We lack the space for explaining why all these transitions are needed or to prove formally that E is an admissible extension of $P_\mathcal{C}$ given $\mathcal{C}$ and that h does indeed reduce $Q_\mathcal{C}$ into $P_\mathcal{C} + E$. Theorem 18 implies that $\mathcal{C}$ solves Q if $\mathcal{C}$-trajectories τ in Q map into trajectories $h(\tau)$ that contain a finite number of E-transitions. One such trajectory τ is displayed in Fig. 3 for the problem Q shown that follows from the initial state where the hidden green block is at $(5, 2)$. The labels show the $P_\mathcal{C}$-cells visited by the trajectory $h(\tau)$ induced over P. ∎

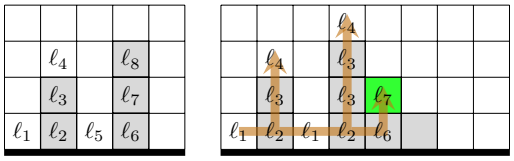

Figure 3: a) Labeling of the cells in P reached by controller $\mathcal{C}$. b) $\mathcal{C}$-trajectory τ in Q with P-cell labels in trajectory $h(\tau)$.

8 Related Work

Bonet *et al.* [2009] and Hu and De Giacomo [2013] show how to derive certain types of controllers automatically. The generality of these controllers has been addressed formally in [Hu and De Giacomo, 2011], and in a different form in [Srivastava *et al.*, 2011a] and [Hu and Levesque, 2011]. A related generalized planning framework appears in [Srivastava *et al.*, 2011b] that captures nested loops involving a set of non-negative variables that may be increased or decreased non-deterministically, and where the notion of *terminating strong cyclic policies* plays a central role. A key difference to these works is that our approach is not "top down" but "bottom up"; namely, rather than solving a (harder) generalized problem to obtain a general solution, we look at solutions to easy problems and analyze how they generalize to larger problems. Our reductions are also related to abstraction and transfer methods in MDP planning and reinforcement learning [Li *et al.*, 2006; Konidaris and Barto, 2007; Taylor and Stone, 2009]. One difference is that these formulations use rewards rather than goals, and hence the abstractions are bound to either preserve optimality (which is too restrictive) or to approximate reward. Our focus on goal achievement makes things simpler and crisper, yet the results do apply to the computation of proper controllers for Goal MDPs and POMDPs.

9 Discussion

We have studied the conditions under which a controller that solves a POND problem P will solve a different problem Q sharing the same actions and observations. A practical consequence of this is that there may be no need for complex models and scalable algorithms for deriving controllers for large problems, as often, controllers derived for much smaller problems sharing the same structure will work equally well. The shared structure involves a common set of actions and observations, which in itself, is a crisp requirement on *problem representation.* For example, policies for a blocksworld instance won't generalize to another instance when the actions involve the names of the blocks. The same holds for MDP policies expressed as mappings between states and actions. The appeal of so-called deictic or agent-centered representations [Chapman, 1989; Ballard *et al.*, 1997; Konidaris and Barto, 2007] is that they yield sets of actions and observations that are not tied to specific instances and spaces, and hence are general and can be used for representing general policies.

References

[Ballard *et al.*, 1997] D. Ballard, M. Hayhoe, P. Pook, and R. Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20(4):723–742, 1997.

[Bertsekas, 1995] D. Bertsekas. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific, 1995.

[Bonet and Geffner, 2009] B. Bonet and H. Geffner. Solving POMDPs: RTDP-Bel vs. Point-based Algorithms. In *Proc. IJCAI-09*, pages 1641–1646, 2009.

[Bonet *et al.*, 2009] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS-09*, pages 34–41, 2009.

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

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

[Chatterjee and Chmelík, 2015] K. Chatterjee and M. Chmelík. POMDPs under probabilistic semantics. *Artificial Intelligence*, 221:46–72, 2015.

[Cimatti *et al.*, 1998] A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. AAAI-98*, pages 875–881, 1998.

[Geffner and Bonet, 2013] H. Geffner and B. Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers, 2013.

[Hu and De Giacomo, 2011] Y. Hu and G De Giacomo. Generalized planning: Synthesizing plans that work for multiple environments. In *Proc. IJCAI*, pages 918–923, 2011.

[Hu and De Giacomo, 2013] Y. Hu and G De Giacomo. A generic technique for synthesizing bounded finite-state controllers. In *Proc. ICAPS*, pages 109–116, 2013.

[Hu and Levesque, 2011] Y. Hu and H. Levesque. A correctness result for reasoning about one-dimensional planning problems. In *Proc. IJCAI*, pages 2638–2643, 2011.

[Kaelbling *et al.*, 1998] L. Kaelbling, M. Littman, and T. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[Konidaris and Barto, 2007] G. Konidaris and A. Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, pages 895–900, 2007.

[Li *et al.*, 2006] L. Li, T. J. Walsh, and M. Littman. Towards a unified theory of state abstraction for MDPs. In *Proc. ISAIM*, 2006.

[Milner, 1989] Robin Milner. *Communication and concurrency*, volume 84. Prentice Hall, 1989.

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

[Russell and Norvig, 2002] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002. 2nd Edition.

[Srivastava *et al.*, 2011a] S. Srivastava, N. Immerman, and S. Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647, 2011.

[Srivastava *et al.*, 2011b] S. Srivastava, S. Zilberstein, N. Immerman, and H. Geffner. Qualitative numeric planning. In *Proc. AAAI*, pages 1010–1016, 2011.

[Taylor and Stone, 2009] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.