Replanning in Domains with Partial Information and Sensing Actions

Guy Shani
Department of Information Systems Engineering
Ben-Gurion University of the Negev
shanigu@bgu.ac.il

Ronen I. Brafman
Department of Computer Science
Ben-Gurion University of the Negev
brafman@cs.bgu.ac.il

Abstract

Replanning via determinization is a recent, popular approach for online planning in MDPs. In this paper we adapt this idea to classical, non-stochastic domains with partial information and sensing actions. At each step we generate a candidate plan which solves a classical planning problem induced by the original problem. We execute this plan as long as it is safe to do so. When this is no longer the case, we replan. The classical planning problem we generate is based on the T_0 translation, in which the classical state captures the knowledge state of the agent. We overcome the non-determinism in sensing actions, and the large domain size introduced by T_0 by using state sampling. Our planner also employs a novel, lazy, regression-based method for querying the belief state.

Introduction

Replanning is a popular, recent approach for planning under uncertainty. Replanning algorithms, such as FF-replan [Yoon *et al.*, 2007], invoke an offline classical planner with a deterministic domain that is generated from the original stochastic domain. The solution plan for the deterministic problem is executed until an outcome unexpected by the deterministic planner is observed. At that point, the planner is called again, etc. Although the basic idea may appear simplistic, recent extensions [Yoon *et al.*, 2008; Kolobov *et al.*, 2010] that tie it well with the powerful technique of state sampling, work well in non-deterministic, fully observable domains.

In this paper we extend the replanning method to contingent domains with partial observability and uncertainty about the initial state. To do this, we could, in principle, select a concrete initial state, and plan as if it was the true initial state of the world. If the actions are deterministic (which we assume for the present), this immediately yields a classical planning problem. We can execute its solution until an unexpected observation occurs, and then replan using a new initial state consistent with this information. However, as the classical planner plans as if it has complete information, it will never make explicit effort to obtain information. For example, such a planner will not use pure sensing actions or make detours from the greedy path in order to obtain information.

We overcome this problem by adapting the more sophisticated knowledge-based translation introduced in [Palacios and Geffner, 2009]. This method generates a classical planning problem that implicitly represents and reasons about the agent's state of knowledge, rather than the state of the world. The goal condition now is that the agent *know* that the goal is true. However, this transformation suffers from two problems: First, it may lead to classical domains containing a number of propositions that is exponential in the input size. Second, it transforms actions with sensing into non-deterministic actions [Albore *et al.*, 2009]. We use state sampling to overcome both problems. To reduce the size of the classical domain generated we sample a small subset S'_I of the possible initial states, and ignore all others. To remove non-determinism, we select a distinguished initial state $s'_i \in S'_I$. Whenever an observation is made, our model ensures that the observed value is consistent with s'_i being the true initial state. The resulting classical planning problem emulates the state of knowledge of an agent that initially believes that S'_I are the only possible states of the world, when the true initial state is s'_i.

If the true initial state of the world is s'_i, then the generated plan will lead us to the goal. Otherwise, at some point we will recognize a problem: either because we make an observation that is inconsistent with s'_i, or because we realize that we are about to execute an action whose preconditions do not necessarily hold, or because we do not reach the goal at the end. To recognize these problems, we must maintain some information about the set of states that are currently possible – known as the *belief state*. To recover from these problems, we replan with an updated state of information.

Like other planners that operate under uncertainty, we must somehow represent the current belief state in order to determine at each step which actions are applicable (i.e., whether their preconditions hold) and if the goal has been reached. Previous planners use one of two general methods: a compact, explicit representation of the set of possible states, e.g., using BDDs [Cimatti and Roveri, 2000] or logical formulas [To *et al.*, 2009], or an implicit representation via a propositional formula that captures the history of execution (e.g., CFF [Hoffmann and Brafman, 2006], Contingent-FF [Hoffmann and Brafman, 2005]) — an approach that dates back to situation calculus [McCarthy and Hayes, 1969; Reiter, 2001]. The explicit representation supports fast inference, but its size can become prohibitive, its update step can be more complicated, and no representation is known to be effective on all domains. The history-based approach is easier to maintain and update, but requires calling a SAT solver to determine the validity of conditions at each state. Our planner takes the history based method to an extreme, and only maintains the history of execution: the initial belief state, the actions executed, and the observations made. To determine whether c holds, we regress $\neg c$ through the executed plan. The resulting formula c_I is the weakest condition on the initial state under which the execution of the current plan would result in a state in which $\neg c$ holds. If c_I is consistent with the initial belief state b_I, then we know that c does not necessarily hold. Otherwise, c must be known. Like CFF, we cache this information, and use it to simplify formulas during the regression process. The advantage of this lazy approach is that it is even simpler to maintain and update. Moreover, the formulas produced during regression are focused on the condition in question, as opposed to the general formula maintained by CFF, and so with the help of simplification, are quite small in practice.

The resulting planner – SDR (*Sample, Determinize, Replan*) – compares favorably with CLG, which is the current state-of-the-art contingent planner. On most domains it reaches the goal state faster, sometimes by an order of magnitude, can solve problems that CLG cannot solve, and its plans have similar size.

Problem Definition

We focus on planning problems with partial observability and sensing actions (PPOS). We shall assume that actions are deterministic, following our current implementation. However, because the replanning approach was originally conceived to deal with stochastic effects of actions, it is well suited for non-deterministic actions as well, and we briefly discuss the needed extensions later.

Formally, PPOS problems can be described by a quadruple: $\langle P, A, \varphi_I, G\rangle$, where P is a set of propositions, A is a set of actions, φ_I is a propositional formula over P that describes the set of possible initial states, and $G \subset P$ is the goal propositions. In what follows we will often abuse notation and treat sets of literals as a conjunction of the literals in the set, as well as an assignment of value to propositions appearing in this set. For example, $\{p, \neg q\}$ is also treated as $p \wedge \neg q$ and as an assignment of *true* to p and *false* to q.

A state of the world, s, assigns a truth value to all elements of P. A *belief-state* is a set of possible states, and the initial belief state, $b_I = \{s : s \models \varphi_I\}$ defines the set of states that are possible initially. An action $a \in A$ is a three-tuple, $\{pre(a), effects(a), obs(a)\}$. $pre(a)$ is a set of literals denoting the action's preconditions. $effects(a)$ is a set of pairs (c, e) denoting conditional effects, where c is a set (conjunction) of literals and e is a single literal. Finally, $obs(a)$ is a set of propositions, denoting those propositions whose value is observed following the execution of a. We assume that a is well defined, that is, if $(c, e) \in effects(a)$ then $c \wedge pre(a)$ is consistent, and that if both $(c, e), (c', e') \in effects(a)$ and $s \models c \wedge c'$ for some state s then $e \wedge e'$ is consistent. In current benchmark problems, either the set *effects* or the set *obs* are empty. That is, actions either alter the state of the world but provide no information, or they are pure sensing actions that do not alter the state of the world, but there is no reason for this to be the case in general.

We use $a(s)$ to denote the state that is obtained when a is executed in state s. If s does not satisfy all literals in $pre(a)$ then $a(s)$ is undefined. Otherwise, $a(s)$ assigns to each proposition p the same value as s, unless there exists a pair $(c, e) \in effects(a)$ such that $s \models c$ and e assigns p a different value than s. Observations affect the agent's belief state. We assume all observations are deterministic and accurate, and reflect the state of world prior to the execution of the action.[1] Thus, if $p \in obs(a)$ then following the execution of a, the agent will observe p if p holds now, and otherwise it will observe $\neg p$. Thus, if s is the true state of the world, and b is the current belief state of the agent, then $b_{a,s}$, the belief state following the execution of a in state s is defined as:

$$b_{a,s} = \{a(s') | s' \in b, s' \text{ and } s \text{ agree on } obs(a)\}$$

That is, all states where the agent would receive the same observation if it executes action a as if it was at state s. The new belief state corresponds to the progression through a of all state in the old belief state b that assign the propositions in $obs(a)$ the same values as s does.

The SDR Planner

The replanning approach we pursue is an online method. The algorithm generates a plan, and executes it as long as the preconditions of the next action hold in all possible states. When this is no longer true, it generates a new plan online, and continues. Like other online planners, such as CLG [Albore *et al.*, 2009], the method does not attempt to generate a complete plan,[2] but needs only return a valid action to execute at each step. Replanning often returns a sequence of actions, rather than a single action.

The high level scheme of SDR is described in Algorithm 1. *size* is a parameter that governs the size of the set of world states S'_I. The algorithm generates a classical plan by determinising the contingent planning problem starting from the current belief state, and executes the resulting plan as long as it is valid. We now explain the generation (translation) of the classical planning problem (Step 6), and the validation of goal and actions preconditions (Steps 3 and 11). The validation process is closely related to our representation of histories and beliefs (Steps 2, 14).

Generating a Classical Planning Problem

Given the input PPOS $\pi = \langle P, A, \varphi_I, G\rangle$ and current belief state b, we generate a classical planning problem π_c – $\langle P_c, A_c, I_c, G_c\rangle$ as follows: First, we generate a set of states S satisfying b. The size of this set is a parameter (denoted *size*, above), which we typically set to a very small value (e.g.,

[1] One can choose to have the observations reflect the state of the world following the execution of the action at the price of a slightly more complicated notation below.

[2] This can be done via simulation. We do not pursue this here.

Algorithm 1 SDR

```
Input: PPOS Problem: $\pi = \langle P, A, \varphi_I, G\rangle$, Integer: size
 1: $b$ :=initial belief state
 2: $h := \phi$, the empty history
 3: while $G$ is not known at the current belief state do
 4:     Select a distinct set of states $S'_I$ consistent with $b$ and $h$ s.t. $|S'_I| \leq$size
 5:     Select a state $s'_i \in S'_I$
 6:     Generate a deterministic plan $C$ from $\pi, S'_I, s'_i, h$
 7:     Find a solution $\mathcal{P}$ for $C$
 8:     if no solution exists then
 9:         return failure
10:     end if
11:     while $\mathcal{P} \neq \emptyset$ and pre(first($\mathcal{P}$)) is known at the current belief state do
12:         $a$ :=first($\mathcal{P}$)
13:         execute $a$, observe $o$
14:         regress $o$ through $h$ obtaining $\phi_{o,h}$
15:         append $< a, o >$ to $h$
16:         $b :=$ update the initial belief state given $\phi_{o,h}$
17:         Remove $a$ from $\mathcal{P}$
18:     end while
19: end while
```

2,3). Next, we select one distinguished state $s'_i \in S'_I$. We now generate a classical planning problem as follows[3]:

Propositions $P_c = P \cup \{Kp, K\neg p | p \in P\} \cup \{Kp/s', K\neg p/s' | p \in P, s' \in S'_I\} \cup \{K\neg s' | s' \in S'_I\}$. That is: the original set of propositions P; propositions that capture the agent's state of knowledge: Kp denotes knowing that p is true, whereas Kp/s denotes knowing that p is true if s was the initial state. $K\neg s$ denotes knowing that s was not the initial state.

Actions For every action $a \in A$, A_c contains an action a_c defined as follows:

pre(a_c) = *pre*$(a) \cup \{Kp | p \in$ *pre*$(a)\}$. That is, the agent must *know* that the preconditions are true prior to applying the action.

for every $(c, e) \in$ *effects*(a), *effects*(a_c) contains the following conditional effects:

- (c, e) – the original effect.
- (Kc, Ke) – if we know that the condition c holds prior to executing a, we know its effect holds following a.
- $(\neg K\neg c, \neg K\neg e)$ – if c is not known to be false, prior to executing a, e will not be known to be false afterwards.
- $\{(Kc/s', Ke/s') | s' \in S'_I\} \cup \{(\neg K\neg c/s', \neg K\neg e/s') | s' \in S'_I\}$ – the above, conditioned on the initial states.
- $\{(p \wedge K\neg p/s', K\neg s'), (\neg p \wedge Kp/s', K\neg s') | p \in$ *obs*$(a), s' \in S'_I\}$ – rule out possible initial states in our set S'_I if the observation is inconsistent with them.
- $\{(p, Kp), (\neg p, K\neg p) | p \in$ *obs*$(a)\}$ – observing the value of p.

In addition, for each literal l (w.r.t. P) we have a merge action that let's us conclude absolute knowledge from knowledge conditional on the initial state:

- *pre*(merge(l)) = $\{Kl/s' \vee K\neg s' | s' \in S'_I\}$.
- *effects*(merge(l)) = $\{($*true*$,Kl)\}$

Initial State $I_c = \bigwedge_{l:s'_i \models l} l \quad \bigwedge_{s' \in S'_I} \neg K\neg s' \quad \bigwedge_{s' \in S'_I, s' \models l} Kl/s'$, where the first conjunct sets the value of the regular propositions to their value in the distinguished state $s \in S$. The other two conjuncts state that all initial states in S are possible, and sets the conditional knowledge to correspond to these states.

Goal $G_c = KG$

Above we used Kc as a shorthand notation for $Kl_1 \wedge \cdots \wedge Kl_m$, where $c = l_1 \wedge \cdots \wedge l_m$, and $\neg K\neg c$ as a shorthand notation for $\neg K\neg l_1 \wedge \cdots \wedge K\neg l_m$. The latter is actually stronger than $\neg K\neg c$ in the appropriate logic of knowledge, and implies that we are more "quick" to forget knowledge. However, this is compensated by the fact that in the case of conditional knowledge (i.e., $\neg K\neg c/s'$) the two are equivalent. Consequently, we can always deduce this knowledge back using merge actions, when appropriate. Finally, recall that we assume the effect e is a literal.

The above is a variant of the method of [Palacios and Geffner, 2009; Albore *et al.*, 2009], with some changes:

1. Tags are restricted to the set of sampled initial states S'_I. This bounds the size of the classical problem generated, as opposed to those generated by CLG for its heuristic estimation, allowing us to scale up better. However, for problems with small conformant width, one can use their complete set of tags, with some theoretical benefits discussed later.
2. We keep a copy of the original set of propositions, initialized to their value in s'_i. The action definition ensures that the value of these propositions correspond to their value in the real world when s'_i is the initial state.
3. The result of sensing is deterministic. Following 2 above, the observed values correspond to the values of these propositions when s'_i is the initial state.

Non-deterministic Actions. The replanning approach was introduced in the context of stochastic planning, and the basic planning problem solved at each step was obtained by determinizing the effects of these actions. We can pursue an identical strategy. Suppose that action a has a conditional effects $(c, e_1 \vee e_2)$. In that case a_c will have one of two possible conditional effects (c, e_1) or (c, e_2). In fact, planners that use determinization often choose different determinizations for different instances of the same actions. In addition, we need to remove the agent's knowledge following the execution of a non-determnistic action. That is, in the above case, we must add $(c, \neg Ke_1)$ and $(c, \neg Ke_2)$.

Bias for Observation. It is possible to add to the SDR planner a simple bias for observation. Namely, at each step, if it is possible to sense the value of an unknown proposition without affecting the state, we perform this observation. As we will see, in the context of current planning benchmarks,

[3] Our description here focuses on deterministic actions. The extension to non-deterministic actions is discussed later on.

in which sensing actions do not affect the state of the world, this is a very effective bias.

Histories and Queries

As explained above, we pursue a history-based approach, where we maintain the sequence of executed actions and observations, and use it in conjunction with the initial belief state to reason about the properties of the current state. We also maintain limited information on each belief state to reduce the computational burden of such reasoning.

Querying for current state properties

Throughout the online process we maintain the initial state formula and the history of actions and observations made. We use this information to check, prior to applying any action a, whether its preconditions hold in the current state. We must also check whether the goal conditions hold in the current state to determine whether the goal has been achieved. To check whether a condition c holds, we regress $\neg c$ through the current history, obtaining $\bar{c}_I$. A world state currently satisfies $\neg c$ iff it is the result of executing the current history in an initial state satisfying $\bar{c}_I$. Thus, if $\bar{c}_I$ is *inconsistent* with φ_I, we know that c holds in all states currently possible.

The formulas generated during the regression process can grow rapidly with history. To avoid such growth, we maintain partially specified belief states (PSBS) throughout the history. This helps us simplify the formula generated at each step. For example, suppose our regressed formula at an intermediate belief state b has the form $\varphi \wedge l$, where l is a literal that is known to hold at b. Then, we need only regress φ back from b. Or, if we know that $\neg l$ holds in b, we can immediately conclude that the regressed formula will be inconsistent with the initial belief state.

Partially-specified states

For each step of execution of the current plan we maintain a list of literals known to hold at that state. All propositions that do not appear among the literals in this list are unknown. We call this construct a *partially-specified belief state*, and it serves as a cache. When we execute an action a, we propagate this set forward and update it as follows:

- If (c, l) is an effect of a, and c must be true before a's execution, we add l and remove $\neg l$.
- If l and $\neg l$ may both be true (that is, l is known in the PSBS), a deletes l if l holds (possibly conditional on other conditions that necessarily hold), and a does not add l when $\neg l$ (and some other conditions possible) holds, we can conclude that $\neg l$ must be true after the execution of a.
- If, when checking for the validity of a condition c via regression, we learn that a literal l is valid in the current belief, we update the PSBS and its successors accordingly.

When an observation is obtained, we update the corresponding PSBS by adding its value. Then, we regress this observation backwards to the initial state. We update the initial state by conjoining to it the result of this regression. We also update all intermediate partial belief states, if possible.

Sampling states

Our previous description of the sampling method used to generate the classical plan was not completely accurate. Recall that we said that we select a set S'_I from the current belief state b. However, as we do not explicitly maintain the current belief state, we do the following: we sample a set of states S'_I from the initial belief state b_I (possibly modified by observations), and then progress these states forward with the current history to obtain a set of states that are currently possible.

Relations to CFF

Our method is similar to that of CFF [Hoffmann and Brafman, 2006], but lazier. CFF maintains a single complete formula that describes both the history and the initial state jointly. This requires using a different copy of the state variables for each time point. An action applied at step t is conjoined with this formula as clauses that describe the value of variables in time $t + 1$ as a function of their value in time t. To determine whether condition c holds at the current belief state, the current formula is conjoined with $\neg c$. If the resulting formula is consistent, we know that there are possible states in which $\neg c$ holds. Otherwise, we know that c is valid. For example, if c is the goal condition, we know that a plan was found. If c is the preconditions to some action a, we know that a can be applied safely. CFF also caches such information discovered by simplifying the formula whenever such a conclusion is obtained, via unit propagation.

The regression method that we use can be though of as constructing for each query only the part of the CFF formula that is needed for answering the current query. The downside of this approach is that we could, in principle, reconstruct the same formula, or parts of a formula repeatedly. The advantage is that the formulas that we construct are much smaller and easier to satisfy than the complete CFF formula.

Theoretical Properties

Our focus in this paper is the algorithm's description and its practical efficacy. Given our space constraints, we only briefly touch upon theoretical properties. Soundness can take various meanings in the context of online planning. A reasonable requirement is that the method used for checking the validity of preconditions is sound, which is true for SDR's regression-based method. The ideas underlying PAC proofs for RL algorithms [Kearns and Singh, 2002] are useful in proving the completeness of replanning algorithms with deterministic actions. The typical assumption made is that the state space is connected (and *ergodic* in the case of MDPs), i.e., there are no "dead-ends". This is a very strong assumption, and domains violating it are the Achilles heel of most greedy algorithms, replanning included. Thus, completeness results under this assumption can only provide a "sanity check" for replanning algorithms. Assuming connectedness, to prove completeness one needs to establish a property called *efficient explore or exploit (E^3)* in PAC-RL. E^3 holds when, within a bounded number of steps, the online algorithm will either reach the goal or reduce the uncertainty – i.e., rule out an initial state of the world. Such an algorithm eventually reaches the goal or learns the true state of the world. SDR (with a sound and complete classical planner) satisfies E^3

Table 1: Comparing CLG (execution mode) to SDR with and without the observation bias. For domains with conditional actions (*localize* and *medpks*) results for CLG cannot be computed. We denote *TF* when the CLG translation failed, *CSU* when CLG cannot run a simulation with a uniform distribution, and *PF* where the CLG planner failed, either due to too many predicates or due to timeout.

	SDR-obs			SDR			CLG	
Domain	#Actions	#Replan	Time	#Actions	#Replan	Time	#Actions	Time
Wumpus05	37.9 (1.39)	3.8 (0.18)	2.8 (0.09)	**20.1** (0.76)	3.2 (0.2)	2.7 (0.13)	24 (0.58)	**1.5** (0.01)
Wumpus10	81.4 (4.86)	6.4 (0.51)	**11.9** (0.76)	**50.8** (4.1)	8 (0.56)	**10.9** (0.72)	57.4 (1.29)	28.3 (0.05)
Wumpus15	135 (8.34)	9.3 (0.77)	53.1 (3.26)	**92.5** (8.05)	11.8 (0.81)	**42** (2.71)	103 (3.87)	240.7 (0.65)
Wumpus20	163.1 (11.92)	9.6 (0.8)	**169 (12.78)**	**115.9** (9.88)	17.7 (1.57)	**156.1** (13.12)	160 (3.49)	1224.8 (1.65)
doors9	**53.5** (2.53)	8.6 (0.61)	**6.6** (0.38)	74.4 (2.54)	22.3 (1.02)	8.8 (0.42)	**53.8** (2.37)	29.3 (0.05)
doors11	84.4 (2.69)	13.6 (0.8)	22.9 (0.88)	100.4 (4.81)	28.2 (1.9)	**13.6** (0.88)	**73.1** (3.42)	95.7 (0.18)
doors13	**103.8** (4.95)	12.9 (0.93)	60.1 (3.38)	177.1 (7.39)	33.5 (1.73)	**25.1** (1.28)	**111.8** (4.93)	264.5 (0.54)
doors15	**147.4** (4.8)	16 (0.85)	183.7 (6.79)	234.7 (7.13)	40.5 (1.65)	**46.6** (1.81)	*PF*	
doors17	**210.5** (7.1)	21.7 (1.13)	541.1 (19.99)	306.8 (10.61)	60 (2.57)	**96.9** (4.15)	*PF*	
localize9	40.8 (2.83)	2.5 (0.21)	2.6 (0.12)	32 (2.8)	4.8 (0.28)	4.9 (0.18)	*CSU*	
localize11	57.7 (4.29)	3 (0.22)	5.9 (0.54)	50.1 (2.81)	6.7 (0.34)	11.7 (0.58)	*CSU*	
localize13	69.9 (5.01)	4 (0.31)	15.1 (1.43)	54.2 (2.98)	5.9 (0.31)	18.9 (1.21)	*PF*	
localize15	89.4 (5.39)	3.9 (0.28)	36.2 (2.79)	56.5 (5.22)	6.4 (0.39)	35.6 (3.54)	*PF*	
localize17	82.4 (7.6)	3.4 (0.27)	51 (7.41)	71.7 (4.39)	6.6 (0.28)	75 (7.09)	*PF*	
unix3	83.7 (9.63)	12.1 (1.2)	**4.1** (0.4)	123.1 (14.81)	16.8 (1.8)	5.2 (0.55)	**39.8** (5.41)	14.3 (0.13)
unix4	202.5 (29.24)	25.6 (3.29)	**11.4** (1.49)	236.2 (27.68)	29.4 (3.11)	12.4 (1.34)	**94.8** (11.41)	160.3 (3.12)
colorballs-9-1	101.4 (8.91)	14.3 (1.69)	**15.1** (1.74)	217.2 (22.98)	62.8 (6.97)	65.1 (7.38)	**80** (8.96)	98.2 (0.71)
colorballs-9-3	**240.9** (12.23)	22.8 (1.28)	3**3.2** (1.76)	660.2 (30.69)	156.6 (6.31)	209.1 (9.25)	**227.8** (12.15)	707.1 (3.44)
colorballs-9-5	**387.4** (14.42)	31.4 (1.29)	**66.1** (2.89)	874.6 (36.77)	212.1 (8.27)	346.8 (15.59)	*TF*	
colorballs-9-7	**490.1** (13.22)	34.5 (1.02)	**103.6** (2.93)	1343.9 (50.02)	310.9 (12.8)	693.3 (35.01)	*TF*	
cloghuge	57 (1.24)	5.6 (0.27)	**2.5** (0.09)	76.1 (2.1)	18 (0.86)	8 (0.36)	**52.1** (1.16)	4.7 (0.02)
ebtcs-70	**36.7** (3.92)	2 (0.04)	**0.7** (0.01)	35.6 (3.93)	33.6 (3.93)	15.8 (1.81)	**40.3** (3.79)	53.7 (0.27)
elog7	20.2 (0.22)	1.9 (0.06)	**0.5** (0.01)	20.2 (0.3)	3.3 (0.17)	1.3 (0.05)	**19.8** (0.24)	0.6 (0)
medpks150	**86.7** (8.97)	2 (0.04)	**34.9** (3.65)	**88.8** (8.5)	85.8 (8.51)	268 (26.9)	*CSU*	
medpks199	**89.9** (11.42)	2 (0)	**82.8** (10.95)	**89.9** (11.42)	86.9 (11.42)	502.9 (67.58)	*PF*	

when $S'_I = b_I$, because the plans it generates are executable in all possible states, and reach the goal or lead to an observation that rules out the distinguished initial state, s'_i. From E^3, completeness follows for SDR if the underlying classical planner is sound and complete. When $|b_I|$ is very large, two practical options maintain completeness: for problems with small conformant width, we can use the tag-generation method of [Palacios and Geffner, 2009]. Alternatively, we can start with a small value for *size* (step 4), increasing it if we reach an unsafe action without learning something new. It is an open question whether there is a suitable sampling scheme that satisfies E^3 when *size* remains constant.

Experimental Results

To demonstrate the power of our replanning approach we compared SDR to the state of the art contingent planner CLG [Albore *et al.*, 2009]. We use CLG in its so-called execution mode, where it becomes an online planner. We compare SDR to CLG on all domains from the CLG paper. We implemented our SDR algorithm using *C#*. The experiments were conducted on a Windows 7 machine with 4 2.83GHz cores (although we only use a single core) and 8GB RAM. We used FF [Hoffmann and Nebel, 2001] compiled under a Cygwin environment for solving the deterministic problems, and we used the Minisat SAT solver [Eén and Sörensson, 2003] to search for satisfying assignments (or lack there of). As Minisat does not provide random assignments, we also implemented a naive solver that generates random assignments for the true environment states.

We tested SDR with and without the observation bias described above. When observation bias is active, SDR checks at every state whether there are unknown (hidden) properties of the state for which there exists an immediate applicable sensing-only action. If so, the agent applies all these actions before continuing with plan execution. As we can see, in most benchmark domains, this observation bias resulted in faster execution, because it is typically beneficial to learn as much as you can concerning the hidden state.

Table 1 list the results for SDR with and without observation bias and CLG. For each method we report the average number of actions and the time (seconds) until the goal is reached over 25 iterations (standard error reported in brackets). Execution time for CLG includes translation time for the domain, CLG execution, and plan verification time in our environment, which adds only a few seconds to each execution. The translation timeout was 20 minutes, and CLG execution was also stopped after 20 minutes. For SDR we also report the average number of replanning episodes. We gave FF a timeout of 2 minutes for each replanning episode, but that timeout was never reached. In most cases FF solves the deterministic plan in a few seconds. For each domain we bolded the shortest plan and the fastest execution. SDR also computes much smaller translated domain descriptions, ranging from 10KB to 200KB. However, a direct comparison with CLG is impossible because SDR generates parameterized domains while CLG generates grounded translations, and we hence do not provide detailed results for model sizes. In some domains CLG's simulator does not support uniform sampling of initial states (denoted CSU in Table 1). As in these domains SDR scales up to larger instances than CLG, this problem is not crucial to the comparison.

As we can see in Table 1, SDR and SDR-obs are typically about one order of magnitude faster than CLG, and also scale up to larger problem instances. In domains all planners can solve, the efficiency (in terms of avg. steps to reach the goal) is mixed. CLG is clearly better on Unix, but in many other instances SDR-obs is better, and on Wumpus, SDR is better.

An important parameter of our algorithm is the size of S'_I — the number of initial states that the deterministic planner recognizes. As this number grows, the plan must distinguish between the various states and hence becomes more accurate. However, the more states we use the larger is the translation to the deterministic problem, and the more difficult it is for FF to handle. To examine the impact of this parameter we tested the performance of SDR as a function of the size of S'_I. As we show in Figure 1, the plan quality (the number of actions to reach the goal) of SDR does not change considerably with the number of states. The only domain where there is a significant benefit from adding more states is Wumpus, and there one sees no farther improvement beyond 6 states. As expected, the running time grows, with the growth in the number of states. We can conclude, hence, that, at least in current domains, there is no need to use more than a handful of states.

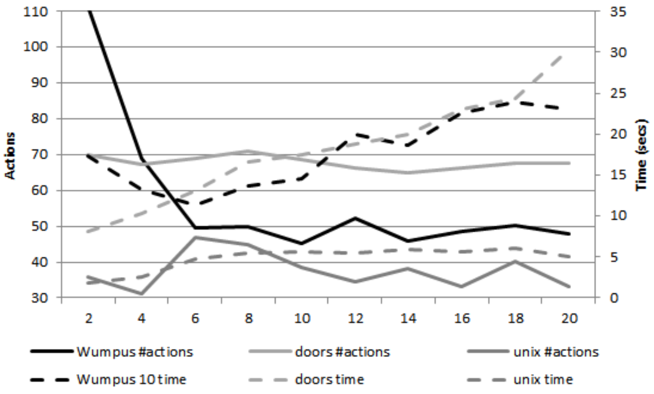

Figure 1: Effect of $|S'_I|$ — the number of initial states (tags) on the number of actions (solid) and the execution time (dashed) for Wumpus10, doors 9, unix 2.

Our results, combined with the simple method underlying our planner help highlight some deficiencies in current contingent planning benchmarks. First, there are no dead-ends, which make problems particularly amenable to replanning-based methods [Little and Thiebaux, 2007]. Second, the fact that we perform well when a sampled initial belief state of size 2 seems to imply that the solution is not too sensitive to the identity of the initial state. This is also related to the fact that the type and amount of conditional effects we see in current domains is quite limited. Finally, the success of the the sensing bias suggests that we should investigate domains in which actions carry a cost, and where sensing is not necessarily separate from action. Domains where sensing actions require substantial effort to attain their preconditions may also provide interesting insights.

Conclusion

We described SDR, a new contingent planner that extends the replanning approach to domains with partial observability and uncertainty about the initial state. SDR also introduces a novel, lazy method for maintaining information and querying the current belief state, and has nice theoretical properties. Our empirical evaluation shows that SDR improves the state of the art on current benchmark domains, scaling up much better than CLG. However, the success of its current simple sampling techniques also highlights the weakness of current benchmark problems.

Acknowledgement: The authors are grateful to Alexander Albore, Hector Geffner, and Son To for their help in understanding and using their systems. Ronen Brafman is partially supported by ISF grant 8254320, the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

References

[Albore *et al.*, 2009] A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *IJCAI*, pages 1623–1628, 2009.

[Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *JAIR*, 13:305–338, 2000.

[Eén and Sörensson, 2003] N. Eén and N. Sörensson. An extensible sat-solver. In *SAT*, pages 502–518, 2003.

[Hoffmann and Brafman, 2005] J. Hoffmann and R. I. Brafman. Contingent planning via heuristic forward search witn implicit belief states. In *ICAPS*, pages 71–80, 2005.

[Hoffmann and Brafman, 2006] J. Hoffmann and R. I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7):507–541, 2006.

[Hoffmann and Nebel, 2001] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Kearns and Singh, 2002] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[Kolobov *et al.*, 2010] A. Kolobov, Mausam, and D. Weld. Classical planning in mdp heuristics: with a little help from generalization. In *ICAPS*, pages 97–104, 2010.

[Little and Thiebaux, 2007] I. Little and S. Thiebaux. Probabilistic planning vs. replanning. In *In ICAPS Workshop on IPC: Past, Present and Future*, 2007.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Palacios and Geffner, 2009] H. Palacios and H. Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *JAIR*, 35:623–675, 2009.

[Reiter, 2001] R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[To *et al.*, 2009] S. T. To, E. Pontelli, and T. C. Son. A conformant planner with explicit disjunctive representation of belief states. In *ICAPS*, 2009.

[Yoon *et al.*, 2007] S. W. Yoon, A. Fern, and R. Givan. Ff-replan: A baseline for probabilistic planning. In *ICAPS*, 2007.

[Yoon *et al.*, 2008] S. W. Yoon, A. Fern, R. Givan, and S. Kambhampati. Probabilistic planning via determinization in hindsight. In *AAAI*, pages 1010–1016, 2008.