Goal Achievement in Partially Known, Partially Observable Domains

Allen Chang and **Eyal Amir**
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{achang6,eyal}@cs.uiuc.edu

Abstract

We present a decision making algorithm for agents that act in partially observable domains which they do not know fully. Making intelligent choices in such domains is very difficult because actions' effects may not be known a priori (*partially known domain*), and features may not always be visible (*partially observable domain*). Nonetheless, we show that an efficient solution is achievable in STRIPS domains by using traditional planning methods. This solution interleaves planning and execution carefully. Computing each plan takes time that is linear in the planning time for the fully observable, fully known domain. The number of actions that it executes is bounded by a polynomial in the length of the optimal plan in the fully observable, fully known domain. Our theoretical results and preliminary experiments show that the algorithm scales to and perform wells on domains of $f > 50$ state features (thus, more than 2^{50} states).

1 Introduction

Agents that act in many real-world domains do not know the exact state of the world at any point in time. These domains are *partially observable* because agents cannot observe all the features of the world that might be relevant to them. For example, an agent crawling the World-Wide Web may press a button on a page, but may not see the immediate effect of its action (e.g., but it could see it, if it viewed another page).

Problems involving partial observability are especially difficult, and limited typically to very small domains (e.g., 100 states or 8 domain features) (Kaelbling, Littman, & Cassandra 1998; Kearns, Mansour, & Ng 2000; Bertoli *et al.* 2001; Bertoli & Pistore 2004). This is because any choice of action depends on the state of knowledge of the agent and the perceptions it receives, leading to a super-exponential computation in the number of steps and features in the domain.

In many such domains, the agent does not know how its actions affect the world a priori (*partially known domains*). For example, a World-Wide Web crawler may not know a priori how pressing any particular button affects the states of different web pages. Even after execution of an action, the crawler does not know that action's effects because they may not be reflected on the current page. More generally, the agent's task is to make decisions at the same time that it learns about its domain.

Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Acting in partially observable, partially known domains is particularly tricky. The main approaches involve at least exhaustive exploration of the domain (e.g., Reinforcement Learning in POMDPs (Jaakkola, Singh, & Jordan 1995; Even-Dar, Kakade, & Mansour 2005)) or advice about promising trajectories (Kearns, Mansour, & Ng 2000). Approaches that guarantee convergence to a solution do so only in the limit of an infinite number of steps. Most importantly, if the goal of the system changes, the policy must be recomputed, and little use is made of knowledge accumulated in previous runs.

This paper identifies an important tractable case of particular interest to the AI community, namely, domains in which actions are known to be deterministic and without conditional effects (*STRIPS* actions). We present an algorithm that interleaves planning and execution and that is guaranteed to reach the goal within a bounded number of steps. Specifically, the number of steps that we execute before reaching the goal is polynomial in the number of steps of an optimal deterministic plan in a fully observable, fully known domain. The computation time involved in our algorithm is linear in the time taken by a SAT-style planner in a fully observable, fully known domain.

Our algorithm iterates: it selects an action, executes it, and updates a logical representation of the possible effects of actions and the possible current states (a *transition belief state*). Our algorithm selects actions by calling a SAT-style planning subroutine that finds plans that could *possibly* work. After each action executes, the algorithm updates a transition belief state, limiting the set of possible states and transition models (action effect models) that could be true for our agent.

There are several key steps in making this algorithm and its proof succeed. First, we observed that we are guaranteed to learn something about the transition model, if the plan does not succeed (if it succeeds, then the algorithm succeeded by definition). Second, we find that encoding the transition belief state is not difficult, following results about learning in partially observable domains (Amir 2005; Chang & Amir 2005). Finally, as plans are executed and more knowledge is acquired by the agent, new plans that incorporate this knowledge are generated.

In Section 2 of this paper, we present the semantics for the problem. In Section 3 we present our decision-making algorithm. In Section 4 we give theoretical results for the decision making algorithm. Finally, in Section 5, we give experimental results for the algorithm.

2 Problem definition

In this section we formally describe the framework defining the problem we are attempting to solve. We give a description of a transition system, the world with which the agent interacts. Let a *planning domain* be defined as the tuple $D = \langle \mathcal{P}, \mathcal{S}, \mathcal{A}, \phi^{(0)}, G \rangle$ where:

- $\mathcal{P}$ is a finite set of propositional fluents.
- $\mathcal{S} \subseteq 2^{\mathcal{P}}$ is a set of states (where $2^{\mathcal{P}}$ is the power set of $\mathcal{P}$).
- $\mathcal{A}$ is a finite set of actions.
- $\mathcal{T} = 2^{\mathcal{S} \times \mathcal{A} \times \mathcal{S}}$ is the set of all possible transition relations.
- $\mathcal{W} = \mathcal{S} \times \mathcal{T}$ is a set of all possible worlds.
- $\phi^{(0)} \subseteq \mathcal{W}$ is the initial transition belief state.
- $G \subseteq \mathcal{S}$ is the set of goal states.

A *state* $s \in \mathcal{S}$ is a subset of $\mathcal{P}$ that contains all fluents that hold in the state. For any transition relation $R \in \mathcal{T}$ and action $a \in \mathcal{A}$, $R(s, a, s')$ means that state s' is one possible result of taking action a from state s. Each *possible world* $w \in \mathcal{W}$ is a state, transition-relation pair. A *transition belief state* $\phi \subseteq \mathcal{W}$ is a set of possible worlds the agent believes contains the current actual world.

Formally, the problem we are attempting to solve is, given the initial transition belief state $\phi^{(0)}$, the agent must select actions in order to reach one of the goal states. There is some actual world state $s^{(0)}$ and actual action model R such that $\langle s^{(0)}, R \rangle \in \phi^{(0)}$. After taking an action a, the current state s becomes a new state s' such that $R(s, a, s')$. In general, the agent cannot observe all the fluents of $\mathcal{P}$. Instead after taking each action, the agent may receive an observation o in the form of a logical formula over $\mathcal{P}$ that holds in the current state s.

In the case of partial observability, a decision-making agent cannot in general know beforehand whether each action it decides to take will fail or succeed. Instead, we assume that the agent knows whether each action it attempts fails or succeeds (after trying the action). More precisely, we assume that there is a special additional proposition OK that is observed by the agent after taking each action such that OK is true if and only if the action succeeded. An action succeeds if and only if the preconditions of the action were met.

For reasons of tractability, we assume all actions are STRIPS with unconditional effects in this paper. Additionally, we assume that the preconditions of the actions are known, so only the effects of actions need to be learned.

Example 2.1. Consider a domain where an agent is in a room with a locked door (see Figure 2.1). In its possession are three different keys, and suppose the agent cannot tell from observation only which key opens the door. The goal of the agent is to unlock the door. This domain can be represented as follows: let $\mathcal{P} = \{\mathsf{locked}\}$ where locked is true if and only if the door is locked. Let $\mathcal{S} = \{s_1, s_2\}$ where $s_1 = \{\mathsf{locked}\}$ (the state where the door is locked) and $s_2 = \{\}$ (the state where the door is unlocked). Let $G = \{s_2\}$ represent the goal. Let $\mathcal{A} = \{\mathsf{unlock}_1, \mathsf{unlock}_2, \mathsf{unlock}_3\}$ be the three actions where the agent tries unlocking the door using each of the three keys. Let $R_1 = \{\langle s_1, \mathsf{unlock}_1, s_2 \rangle, \langle s_1, \mathsf{unlock}_2, s_1 \rangle, \langle s_1, \mathsf{unlock}_3, s_1 \rangle\}$ represent the transition relation that key 1 unlocks the door (but the other keys do not). Likewise, define R_2 and R_3 in similar fashion. A transition belief state where the state of the world is fully known, but the action model is only partially known, is given by $\phi = \{\langle s_1, R_1 \rangle, \langle s_1, R_2 \rangle, \langle s_1, R_3 \rangle\}$.

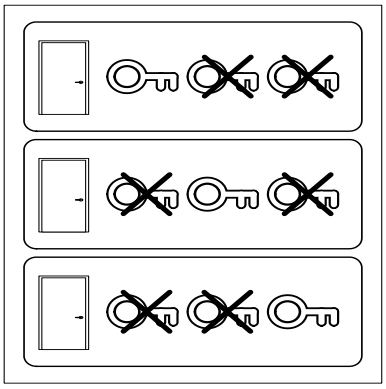

Figure 2.1: Locked door with unknown key domain

We would like the agent to be able to open the door despite not knowing which key to use. To do this, the agent will learn the actual action model (i.e., which key opens the door). In general, not only will learning action model be useful in achieving the immediate goal, but such knowledge will be useful as the agent attempts to perform other tasks in the same domain. □

2.1 Representing Transition Belief States

In general, it is impractical to represent transition belief states explicitly as sets of worlds. Instead, we represent transition belief states symbolically: each transition belief state is represented by a propositional logic formula, each model of which corresponds to a world in the transition belief state. First, define the set of propositional variables $L = \bigcup_{a \in \mathcal{A}, f \in \mathcal{P}} \{a^f, a^{\neg f}, a^{f\circ}\}$. Intuitively, each propositional variable a^f ($a^{\neg f}$) holds if and only if "action a causes f ($\neg f$) to hold after it is executed." Each propositional variable $a^{f\circ}$ holds if and only if "action a does not affect f." Transition belief states are represented as propositional formulas (which we will call transition belief formulas) containing propositions from the vocabulary $L \cup \mathcal{P}$. Each model of a transition belief formula corresponds to a world in the corresponding transition belief state. (Likewise, we assume the goal G is specified as a propositional formula over the vocabulary $\mathcal{P}$.) Therefore, from here forth, we will use ϕ to denote a transition belief state and the transition belief for-

Algorithm 1 PLAN-AND-EXECUTE($\phi^{(0)}, G$)

Inputs Initial transition belief formula $\phi^{(0)}$, goal formula G

Returns *Success* if the goal is achieved, *Failure* otherwise

1. Set $\phi \leftarrow \phi^{(0)}$
2. Set $\pi \leftarrow$ FIND-CANDIDATE-PLAN(ϕ, G)
3. For i from 1 to length of π
   (a) Set a_i to be the ith action in the plan.
   (b) Execute action a_i, and receive observation o_i.
   (c) Set $\phi \leftarrow$ PRE-STRIPS-SLAF$[\langle a_i, o_i \rangle](\phi)$.
   (d) If $o_i \models \neg OK$, go to step 2.
   (e) If $o_i \models G$, then return *Success*.
4. Go to step 2.

mula representing it, interchangeably.

Example 2.2. Consider the domain from Example 2.1. The transition belief state ϕ can be represented by the transition belief formula:

$$\begin{aligned}&\mathsf{locked} \wedge \\ &((\mathsf{unlock}_1^{\neg\mathsf{locked}} \wedge \mathsf{unlock}_2^{\mathsf{locked}\circ} \wedge \mathsf{unlock}_3^{\mathsf{locked}\circ}) \vee \\ &(\mathsf{unlock}_1^{\mathsf{locked}\circ} \wedge \mathsf{unlock}_2^{\neg\mathsf{locked}} \wedge \mathsf{unlock}_3^{\mathsf{locked}\circ}) \vee \\ &(\mathsf{unlock}_1^{\mathsf{locked}\circ} \wedge \mathsf{unlock}_2^{\mathsf{locked}\circ} \wedge \mathsf{unlock}_3^{\neg\mathsf{locked}}))\end{aligned}$$

3 Planning and Execution Algorithm

Now we present an algorithm (Algorithm 1) that interleaves planning and execution for achievement of goals in the presence of partial observability and uncertainty about the action model. The algorithm maintains a transition belief formula ϕ containing the agent's current knowledge about the state of the world and the effects of actions. In each iteration, the algorithm finds a candidate plan, using subroutine FIND-CANDIDATE-PLAN, and executes the actions of the plan. Next, it updates its knowledge, using subroutine PRE-STRIPS-SLAF, according to the executed actions and received observations. Subroutines FIND-CANDIDATE-PLAN and PRE-STRIPS-SLAF are described in detail in subsequent subsections.

We make the following assumptions about the problem domain. These assumptions guarantee that the algorithm achieves the goal, if achieving the goal is possible.

Condition 3.1. The domain meets the following criteria:

1. The initial belief $\phi^{(0)}$ is consistent with the actual state of the world.
2. The agent observes whether the goal is met at the end of each plan execution. That is, it observes the fluents appearing in G at the end of the plan execution.
3. The domain is strongly connected; it is always possible to achieve the goal through a sequence of actions from any reachable state.
4. All actions are deterministic (i.e., STRIPS), and have no conditional effects.

The first assumption states that the agent's initial knowledge is correct (that the actual world is actually contained within the initial belief $\phi^{(0)}$). The second assumption is required to ensure that the agent makes progress after each plan execution. Note that no other assumptions need to be made about the observation model of the domain. The third assumption ensures that the domain can be safely explored by the agent.

3.1 Efficient Learning and Filtering

As an agent takes actions and receives observations, its knowledge about the current state of the world and the action model of the world changes. Define the simultaneous learning and filtering (SLAF) operation over a transition belief state and a sequence of actions and observations to be the transition belief state containing the set of worlds consistent with the actions taken and the observations received. More precisely:

Definition 3.2. (SLAF Semantics) Let $\phi \subseteq \mathcal{W}$ be a transition belief state. The SLAF of ϕ with actions/action-failures and observations $\langle a_j, o_j \rangle_{1 \le j < t}$ is defined by

1. $SLAF[a](\phi) = \{\langle s', R\rangle | \langle s, a, s'\rangle \in R, \langle s, R\rangle \in \phi\}$ if a was successful
2. $SLAF[a](\phi) = \{\langle s, R\rangle | \langle s, a, \cdot\rangle \notin R, \langle s, R\rangle \in \phi\}$ if a failed
3. $SLAF[o](\phi) = \{\langle s, R\rangle \in \phi | o \text{ is true in } s\}$
4. $SLAF[\langle a_j, o_j\rangle_{i \le j \le t}](\phi) = SLAF[\langle a_j, o_j\rangle_{i < j \le t}](SLAF[o_i](SLAF[a_i](\phi)))$

Example 3.3. Consider the domain described in example 2.1. The progression of ϕ on the action unlock_1 is given by $SLAF[\mathsf{unlock}_1](\phi) = \{\langle s_2, R_1\rangle, \langle s_1, R_2\rangle, \langle s_1, R_3\rangle\}$. Likewise, the filtering of ϕ on the observation $\neg\mathsf{locked}$ (the door became unlocked) is given by $SLAF[\neg\mathsf{locked}](\phi) = \{\langle s_2, R_1\rangle\}$.

We now present an algorithm, PRE-STRIPS-SLAF (Algorithm 2), that performs the SLAF operation on a transition belief formula efficiently in such a way as to maintain formula compactness. As previously stated, we assume that actions are unconditional STRIPS, and that the action preconditions are known by the agent. That is, the agent must learn the effects of its actions, but does not need to learn the preconditions of its actions.

PRE-STRIPS-SLAF uses AE-STRIPS-SLAF as a subroutine. AE-STRIPS-SLAF is described in detail in (Amir 2005). It learns the effects of always-executable STRIPS actions (i.e., actions which never fail). That is, it performs learning in the absence of action failures. AE-STRIPS-SLAF requires that formulas are maintained in *fluent-factored* form:

Definition 3.4. A transition belief formula ϕ is *fluent-factored* if it can be written as $\phi = \bigwedge_{f \in \mathcal{P}} \phi_f$ where $\phi_f = (\neg f \vee expl_f) \wedge (f \vee expl_{\neg f}) \wedge A_f$ where $expl_f$, $expl_{\neg f}$, and A_f contain only propositions from $\bigcup_{a \in \mathcal{A}} \{a^f, a^{\neg f}, a^{f\circ}\}$.

Note that the fluent-factored formula containing no prior knowledge about the state of the world or the action model can be represented as $\bigwedge_{f \in \mathcal{P}} (\neg f \vee \top) \wedge (f \vee \top) \wedge \top$.

Algorithm 2 PRE-STRIPS-SLAF[$\langle a, o\rangle$](ϕ)

Inputs Action a and observation term o. ϕ a transition belief formula with the following factored form: $\phi = \bigwedge_i \bigvee_j \phi_{i,j}$, where each $\phi_{i,j}$ is a fluent-factored formula.

Returns Filtered transition belief formula ϕ.

1. If $o \models \neg OK$:
   (a) Set $\phi \leftarrow \phi \wedge \bigvee_i F(\neg l_i)$ where l_i are the literals appearing in a's precondition, and $F(l)$ is the fluent-factored formula equivalent to l (i.e., $F(l) = ((l \Rightarrow \top) \wedge (\neg l \Rightarrow \bot) \wedge \top) \wedge \bigwedge_{f \in \mathcal{P}}((f \Rightarrow \top) \wedge (\neg f \Rightarrow \top) \wedge \top)$)
   (b) Set $\phi_{i,j} \leftarrow$ AE-STRIPS-SLAF[o]($\phi_{i,j}$) for all $\phi_{i,j}$
2. Else ($o \models OK$):
   (a) For each $\phi_{i,j}$:
      i. Set $\phi_{i,j} \leftarrow$ AE-STRIPS-SLAF[P]($\phi_{i,j}$) where P is the precondition of a
      ii. Set $\phi_{i,j} \leftarrow$ AE-STRIPS-SLAF[$\langle a, o\rangle$]($\phi_{i,j}$)
3. Each $\phi_{i,j}$ is factored into $A_{i,j} \wedge B_{i,j}$ where $B_{i,j}$ contains all (and only) clauses containing a fluent from $\mathcal{P}$. For any i such that there exists B such that for all j, $B_{i,j} \equiv B$, replace $\bigvee_j \phi_{i,j}$ with $B \wedge \bigvee_j A_{i,j}$
4. Eliminate subsumed clauses in each $\phi_{i,j}$
5. Return ϕ

Condition 3.5. For *some* of our theorems, we assume that one of the following holds:

1. For every transition relation in ϕ, a maps states 1:1.
2. ϕ contains all its prime implicates.
3. There is at most one state s such that $\langle s, R\rangle \in \phi$ for any R.

The following theorem states the correctness of PRE-STRIPS-SLAF:

Theorem 3.6 ((Chang & Amir 2005)). *The following are true:*

1. *SLAF*$[a, o](\phi) \models$ *PRE-STRIPS-SLAF*$[a, o](\phi)$
2. *PRE-STRIPS-SLAF*$[a, o](\phi) \equiv$ *SLAF*$[a, o](\phi)$ *if Condition 3.5 holds.*

The first property states that PRE-STRIPS-SLAF always produces a safe approximation of the exact transition belief formula. That is, the exact transition belief state corresponding to SLAF[$\langle a, o\rangle$](ϕ) is always a subset of the transition belief state corresponding to PRE-STRIPS-SLAF[$\langle a, o\rangle$](ϕ). The second property gives a sufficient condition when filtering is exact.

The following theorem gives the time and space complexity for PRE-STRIPS-SLAF:

Theorem 3.7 ((Chang & Amir 2005)). *The following are true of PRE-STRIPS-SLAF:*

1. *The procedure takes time linear in the size of the formula.*
2. *If every fluent is observed every at most k steps and the input formula is in $m \cdot k$-CNF, then the filtered formula is in $m \cdot k$-CNF, where m is the maximum number of literals in any action precondition.*

Algorithm 3 FIND-CANDIDATE-PLAN(ϕ,G)

Inputs Transition belief formula ϕ, goal formula G

Returns Candidate plan $\pi = \langle a_1, \ldots, a_T\rangle$

1. Set $T \leftarrow 1$
2. Search for a satisfying assignment to equation (3.1).
3. If satisfying assignment found:
   (a) For t from 1 to T: Set a_t to be the action a such that $a^{(t)}$ is true in the satisfying assignment.
   (b) Return $\langle a_1, \ldots, a_T\rangle$
4. Set $T \leftarrow T + 1$ and goto step 1.

The last property states that, under the condition that every fluent is observed every at most k steps, the resulting filtered formula stays indefinitely compact (i.e., is in $m \cdot k$-CNF).

3.2 Finding Candidate Plans using SAT

Algorithm 1, PLAN-AND-EXECUTE, leverages the power of propositional satisfiability solvers in its decision making routine. This approach is inspired by previous approaches to encoding traditional (full knowledge) planning problems as satisfiability problems (Kautz & Selman 1992). Choice for this approach is motivated by the recent development of highly efficient satisfiability algorithms (Zhang *et al.* 2001), as well as empirical evidence for the efficacy of satisfiability-based planners (Kautz & Selman 1999).

As in previous satisfiability-based planning approaches, we encode our problem as a propositional formula, each model of which corresponds to a valid candidate plan. More specifically, for each $a \in \mathcal{A}$ and time step t, we create a propositional variable $a^{(t)}$. In any specific model corresponding to a plan, $a^{(t)}$ is true if and only if action a should be taken at time t. It is assumed that belief states are represented as logical formulas, as described previously. Let $\phi^{(t)}$ and $G^{(t)}$ represent the same formulas as ϕ and G, except that all fluents from $\mathcal{P}$ appearing in the formulas are annotated with time-stamp t.

The candidate plan is chosen to be the shortest plan that achieves the goal from some possible world in ϕ. Intuitively, such a plan is the shortest plan that can be found such that it is known that the agent will gain knowledge after the plan executes. Finding a shortest candidate plan that achieves G from some world in ϕ can be reduced to searching for a model for the following formula using a satisfiability checker:

$$\phi^{(0)} \wedge N \wedge \bigwedge_{1 \le t \le T} \left[C^{(t)} \wedge \bigwedge_{a \in \mathcal{A}} \left(a^{(t)} \Rightarrow R_a^{(t)} \right) \right] \wedge G^{(T)} \tag{3.1}$$

T, the length of the plan, is initially be set to 1 and is iteratively deepened by increments of 1 until the formula is

satisfiable and a model is found. Definitions for each of the component formulas of formula (3.1) are given below:

$$N \equiv \bigwedge_{a\in\mathcal{A}, f\in\mathcal{P}} (a^f \vee a^{\neg f} \vee a^{f\circ}) \wedge \neg(a^f \wedge a^{\neg f}) \wedge \neg(a^{\neg f} \wedge a^{f\circ}) \wedge \neg(a^f \wedge a^{f\circ})$$

Formula N encodes the constraint that for each action $a \in \mathcal{A}$ and fluent $f \in \mathcal{P}$, either action a causes f to hold, a causes $\neg f$ to hold, or a does not affect f. That is, N ensures that only consistent action models (where exactly one of a^f, $a^{\neg f}$, $a^{f\circ}$ hold) are considered.

$$R_a^{(t)} \equiv P_a^{(t)} \wedge \bigwedge_{f\in\mathcal{P}} ((a^f \Rightarrow f^{(t+1)}) \wedge (a^{\neg f} \Rightarrow \neg f^{(t+1)}) \wedge (a^{f\circ} \Rightarrow (f^{(t)} \equiv f^{(t+1)}))$$

Formula $R_a^{(t)}$ encodes the transition constraints at time t given that action a is taken. The sub-formula $P_a^{(t)}$ is the precondition of action a at time t. Formula $R_a^{(t)}$ ensures that that the precondition for action a holds at time t and the state of the fluents describing the world between times t and $t+1$ are consistent according to the particular valuations chosen for the action propositions a^f, $a^{\neg f}$, and $a^{f\circ}$.

$$C^{(t)} \equiv \left[\bigwedge_{\substack{a_1, a_2 \in \mathcal{A} \\ a_1 \neq a_2}} \neg(a_1^{(t)} \wedge a_2^{(t)}) \right] \wedge \bigvee_{a\in\mathcal{A}} a^{(t)}$$

Finally, formula $C^{(t)}$ encodes the constraints that exactly one action is taken at time t.

Consider any model of formula (3.1). The valuation given to the propositions of the form a^f, $a^{\neg f}$, $a^{f\circ}$ correspond to some action model consistent with the agent's knowledge about the world contained in ϕ. Likewise, the valuation given to the fluents at time 0 ($f^{(0)}$ for $f \in \mathcal{P}$) corresponds to some initial state consistent with the agent's knowledge. The candidate plan can be extracted from the valuation of the propositions $a^{(t)}$ for $a \in \mathcal{A}$. Because iterative deepening is used (and assuming the satisfiability solver is sound and complete), the candidate plan must be of the shortest possible length.

Example 3.8. Reconsider the domain from example 2.1. Suppose the agent initially has no knowledge about the effects of the actions; that is, $\phi^{(0)} \equiv \mathsf{locked}$. One candidate plan which achieves the goal from a world in $\phi^{(0)}$ is to take the action unlock_1 (from some world where $\mathsf{unlock}_1^{\neg\mathsf{locked}}$ holds). Suppose the agent takes the action and observes that the door is still locked. The transition belief state becomes (after learning and filtering) $\phi \equiv \mathsf{locked} \wedge (\mathsf{unlock}_1^{\mathsf{locked}} \vee \mathsf{unlock}_1^{\mathsf{locked}\circ})$. Now, a candidate plan is to take the action unlock_2 (from some world where $\mathsf{unlock}_2^{\neg\mathsf{locked}}$ holds). Suppose after taking the action, the agent observes that the door is unlocked. Then the new transition belief state becomes $\phi \equiv \neg\mathsf{locked} \wedge (\mathsf{unlock}_1^{\mathsf{locked}} \vee \mathsf{unlock}_1^{\mathsf{locked}\circ}) \wedge \mathsf{unlock}_2^{\neg\mathsf{locked}}$ and the goal is achieved. Not only was the goal achieved, but the agent now knows that key 2 unlocks the door, while key 1 does not.

4 Correctness and Complexity of Planning Algorithm

4.1 Actions with Known Preconditions

It is not difficult to show that after executing each candidate plan, the agent must gain new knowledge. Once the agent has obtained full knowledge, the problem becomes equivalent to a classical planning problem. As a result, we can give the following theorem, which shows the correctness of the algorithm:

Theorem 4.1. *The Interleaved Planning and Execution algorithm achieves the goal in a finite number of steps if the domain meets Conditions 3.1 and 3.5.*

Proof. See subsection 8.1 in Appendix. □

More interestingly, given a stronger observation model, we can also give an upper bound on the number of actions required by the interleaved Planning and Execution algorithm to achieve the goal (it does not seem possible to give a useful bound without a stronger observation model or some other additional assumptions). In the stronger observation model, we assume that every fluent in $\mathcal{P}$ is observed once every at most k steps. Note that under this observation model, the transition belief formula maintained by the algorithm stays indefinitely compact.

For each state $s \in \mathcal{S}$, let $d(s)$, the distance of s, be the length of the shortest plan that achieves the goal from s (assuming full knowledge). For each action $a \in \mathcal{A}$, let $\mathsf{eff}(a)$ represent the set of fluents that are affected by action a. Let $\mathsf{pre}(a)$ represent the set of fluents appearing in the precondition of a. Finally, for each action a, let $v(a) = \sum_{a'\in\mathcal{A}} |\mathsf{pre}(a) \cap \mathsf{eff}(a')|$. The following theorem states the bound:

Theorem 4.2. *Suppose every fluent is observed once every at most k steps and the domain meets Condition 3.1. Then the Interleaved Planning and Execution algorithm achieves the goal after executing at most $O(r|\mathcal{A}|2^m 3^q + \max(r, 2k)|\mathcal{P}|(3|\mathcal{A}|)^k)$ actions, where $r = \max_{s\in\mathcal{S}} d(s)$, $m = \max_{a\in\mathcal{A}} |\mathsf{pre}(a)|$, and $q = \max_{a\in\mathcal{A}} v(a)$.*

Proof. See subsection 8.2 in Appendix. □

The idea behind the proof is to show that a certain amount of knowledge must be gained whenever a plan executes successfully (does not end in an action failure). Likewise, one can show that actions can fail only a finite number of times, so there must be a bound on the number of plans that end in action failures. Together, these two bounds can be combined to give the above bound.

4.2 Actions with Unknown Preconditions

Unfortunately, there are not yet efficient algorithms that maintain a transition belief formula efficiently and indefinitely compactly in the case that preconditions are unknown and must be learned. Nevertheless, the other portions of the Interleaved Planning and Execution algorithm (i.e., those portions unrelated to the SLAF operation) can be easily generalized to the case where action preconditions are unknown. If we call this the "generalized Planning and Exploring algorithm," we can give a similar bound to the one above:

Theorem 4.3. *Suppose every fluent is observed once every at most k steps and the domain meets Condition 3.1. The generalized Interleaved Planning and Execution algorithm achieves the goal after executing at most $O(r|\mathcal{A}|4^m 3^q + \max(r, 2k)|\mathcal{P}|(5|\mathcal{A}|)^k)$ actions, where $r = \max_{s \in \mathcal{S}} d(s)$, $m = \max_{a \in \mathcal{A}} |\mathit{pre}(a)|$, and $q = \max_{a \in \mathcal{A}} v(a)$.*

5 Experimental Results

The algorithm was tested on a virtual-world domain from (Hlubocky & Amir 2004). The domain consists of a small house containing a number of rooms and objects, generated in random configurations, with which the agent can interact. The agent's goal is to exit the house, but it is initially given no knowledge about the state of the world or the effects of the actions the agent can take.

Figures 5.1 and 5.2 show the number of actions taken and total solution time for 15 randomly generated problems in this domain. After solving each problem, the agent retains the knowledge about action effects it learned for use in solving the subsequent problems (however, at the start of every problem, the agent knows nothing about the state of the world). Thus, it is expected that the agent's performance will improve as it learns more domain knowledge by solving more problems. As the results indicate, in solving the first two problems, the agent performs a relatively large number of actions, because it has so little prior knowledge about the actions' effects. In these early stages, the agent performs a great deal of exploration, because for actions that have never been executed before, the agent "hypothesizes" that these actions will help it achieve the goal. After solving the first two problems however, the agent has acquired enough domain knowledge that it can solve the remaining problems much more quickly.

The domain contains approximately 50 features and 100 actions (note that the vocabulary for the belief state of such a domain contains over 15000 propositional variables). The algorithm was implemented with relatively unoptimized LISP code and uses zChaff (Zhang *et al.* 2001) as an external SAT solver. It is suspected that the algorithm will perform significantly better if written in optimized C, since a great deal of time is spent by the slower LISP code in constructing and manipulating propositional logic formulas.

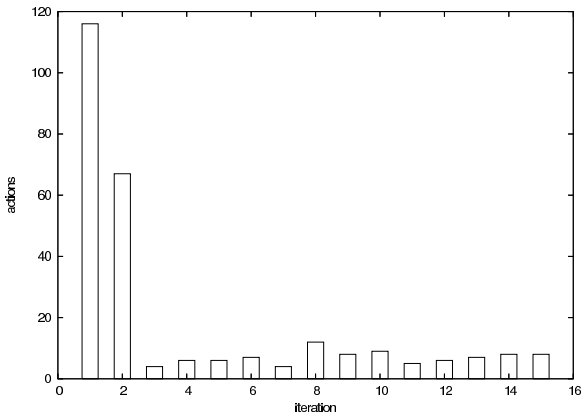

Figure 5.1: Number of actions that are required to achieve the goal as the algorithm solves more problems in the test domain.

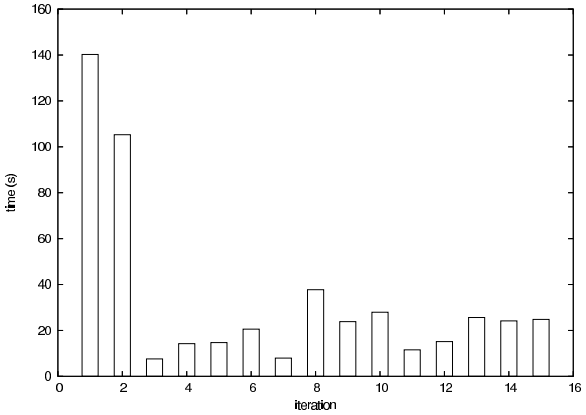

Figure 5.2: Time required to solve each problem as the algorithm solves more problems in the test domain.

6 Related Work

Recent approaches to decision making under partial observability include conformant and contingent planning (Hoffmann & Brafman 2005; Bertoli *et al.* 2001). In general, such approaches do not address the problem of ac-

quiring domain knowledge and reasoning with partial domain knowledge. Other approaches to acquiring planning domain knowledge include (Gil 1994), CaMeL (Ilghami *et al.* 2002), and (McCluskey, Richardson, & Simpson 2002). (Pasula, Zettlemoyer, & Kaelbling 2004) give an algorithm for learning probabilistic relational planning rules. All of these approaches are for fully observable domains. Other approaches for acquiring domain knowledge in the case of partial observability include (Yang, Wu, & Jiang 2005) and (Chang & Amir 2005), but such approaches treat the problem as a supervised learning problem, where it is assumed that the algorithm has access to plan traces produced by an expert agent.

A number of other approaches to the problem of integrated planning and learning of domain knowledge have been proposed. OBSERVER (Wang 1994) is a system that incrementally learns planning operators by first observing expert agents performing tasks in the domain, and then attempting to form plans of its own from its learned knowledge. OBSERVER differs in that it assumes access to some plan traces produced by an expert. Additionally, OBSERVER does not address the problem of partial observability, unlike our work. Finally, the agent's belief is not tracked exactly; instead, only a hypothesis about the domain operators is kept. Thus, for example, the agent cannot tell when it has learned the action model exactly, for example.

LOPE (Garcia-Martinez & Borrajo 2000) is a fully autonomous system that integrates learning, planning, and execution. LOPE, unlike our work, maps world states to observations perceived by the agent and deals with these observations only. That is, the "states" dealt with by LOPE are the inputs to the sensory system of the agent. Because only sensory information is considered, partial observability is dealt with implicitly, rather than explicitly. Unlike our system, the learning and planning algorithms are stochastic in nature, and resemble reinforcement learning in some respects.

Min-Max Learning Real-Time A* Search (Min-Max LRTA*) (Koenig 2001) is another algorithm that integrates learning, planning, and execution. However, LRTA* does not learn action models but rather learns estimates for distances from states to goals. Additionally, Min-Max LRTA* requires storage of heuristic information on a per-state basis, which may be infeasible for large domains with many states. Finally, performance bounds for LRTA* are specified over the number of states, rather than features, of the domains, and depend on the quality of the initial heuristic estimates provided to the algorithm.

There are numerous approaches that deal with Partially Observable Markov Decision Processes (POMDP) including (Jaakkola, Singh, & Jordan 1995), (Kearns, Mansour, & Ng 1999), and (Eyal Even-Dar & Mansour 2005). Current approaches are only practical for domains with small horizon times. Likewise, POMDPs solvers do not learn the action model of the domain, but rather directly generate a policy for maximizing utility. In planning domains, the goal may differ drastically for two problems from the same domain. Thus, it is not clear how to use a POMDP policy generated for achieving one goal to achieve a different goal in the same domain. Finally, it is not clear how to assign rewards for many planning domains in order to represent them as POMDPs.

7 Conclusion

We have presented a new algorithm for decision making in the presence of *both* partial observability and incomplete domain knowledge. Our algorithm interleaves planning and execution in order to achieve the goal in such domains. We have given a theoretical analysis for the algorithm showing correctness and a bound on the number of actions the algorithm outputs. We have also given empirical results for our algorithm. Additionally, we have discussed the assumptions and limitations of our approach.

Future work will include relaxing the assumptions we make as well as extending the capabilities of the algorithm. Our larger goal is to use the presented algorithm to create intelligent agents for interesting domains featuring partial observability and incomplete domain knowledge. Such domains include autonomous World-Wide Web agents and agents acting in rich virtual worlds, such as adventure games.

Acknowledgements This research was supported by the Defense Advanced Research Projects Agency (DARPA) grant HR0011-05-1-0040.

8 Appendix

8.1 Proof of Theorem 4.1

Proof. Let $m(\phi)$ represent the number of worlds in ϕ. It is easy to show that for any sequence of deterministic actions and observations $\langle a_i, o_i \rangle_{1 \leq i \leq n}$:

1. $m(SLAF[\langle a_i, o_i \rangle_{1 \leq i \leq n}](\phi)) \leq m(\phi)$ for any sequence of deterministic actions a_i and observations o_i
2. If $\sigma \subseteq \phi$ and $\sigma' = SLAF[\langle a_i, o_i \rangle_{1 \leq i \leq n}](\sigma)$ and $\phi' = SLAF[\langle a_i, o_i \rangle_{1 \leq i \leq n}](\phi)$, then $\sigma' \subseteq \phi'$
3. $SLAF[\langle a_i, o_i \rangle_{1 \leq i \leq n}](\phi) \subseteq SLAF[\langle a_i \rangle_{1 \leq i \leq n}](\phi)$.

Proceed by induction on $m(\phi)$.

Basis step: $m(\phi) = 1$. Then the problem becomes equivalent to planning without uncertainty. If the problem is solvable, a solution will be found in step 1.

Inductive step: $m(\phi) > 1$. By the assumption that the domain is strongly connected, some plan that achieves the goal for some world in ϕ will be found in step 1.

Suppose that executing the plan fails to achieve the goal and returns to step 1 after actions $a_1, \ldots, a_n$ are executed and observations $o_1, \ldots, o_n$ are received.

Let $\phi^{(i)}$ be the belief state of the algorithm after executing actions $a_1, \ldots, a_i$. We claim $m(\phi^{(n)}) < m(\phi)$. Suppose for contradiction $m(\phi^{(n)}) = m(\phi)$ (because we assume all actions are deterministic, it cannot be the case that $m(\phi^{(n)}) > m(\phi)$). We have $\phi^{(n)} \subseteq SLAF[a_1, \ldots, a_n](\phi)$ and thus $\phi^{(n)} = SLAF[a_1, \ldots, a_n](\phi)$.

Now consider the case that the plan aborted because the last action failed ($o_n \models \neg OK$). But the plan found in step 1 guarantees that $SLAF[a_1, \ldots, a_{n-1}](\phi)$ contains some

world where the precondition of a_n is satisfied. But $\phi^{(n)}$ was filtered on the action failure, contradiction.

Now consider the case that the plan executed successfully, but failed to achieve the goal ($o_n \not\models G$). But again, the plan found in step 1 guarantees that $SLAF[a_1, \ldots, a_n](\phi)$ contains some world that satisfies G. But $\phi^{(n)}$ was filtered on o_n, so no world in $\phi^{(n)}$ satisfies G. Contradiction.

By the semantics of SLAF, the actual world is in $\phi^{(n)}$. By the assumption that the domain is strongly connected, there exist a sequence of actions that achieve the goal from $\phi^{(n)}$. Therefore, by the induction hypothesis, the claim holds for the new belief state $\phi^{(n)}$. □

8.2 Proof of Theorem 4.2

Proof. Consider the total number of actions required to achieve the goal. We will separately bound the number of these actions contributed by "successful" plans (plans which did not end in an action failure) and the number of these actions contributed by "failed" plans (plans that ended in a failure).

Let $L = \bigcup_{f \in \mathcal{P}} \bigcup_{a \in \mathcal{A}} \{a^f, a^{\neg f}, a^{f\circ}, f\}$ denote the vocabulary of SLAF formulas we are dealing with. A model of a SLAF formula ϕ is a truth assignment $t : L \to \{0, 1\}$ that satisfies the formula. Let $\mathsf{models}(\phi)$ denote the set of all models of ϕ. Let $m(\phi) = |\mathsf{models}(\phi)|$ denote the number of models of ϕ.

Lemma 8.1. *If actions always succeed, then the algorithm requires $O(\max(r, 2k)(3|\mathcal{A}|)^k)$ steps to achieve the goal.*

Proof. In the case that actions always succeed, it can be shown that the transition belief formula ϕ can always be maintained in fluent-factored form (see (Amir 2005)). Suppose ϕ is fluent-factored, so $\phi = \bigwedge_{f \in \mathcal{P}} \phi_f$. Let $L_f = \bigcup_{a \in \mathcal{A}} \{a^f, a^{\neg f}, a^{f\circ}, f\}$ denote the restricted vocabulary of ϕ_f. Let $m_f(\phi_f) = |\{t|_{L_f} : t \in \mathsf{models}(\phi_f)\}|$ denote the number of unique models of ϕ_f after restriction to L_f.

It is easy to see that $m(\phi) = \prod_{f \in \mathcal{P}} m_f(\phi_f)$.

Let $\langle a_i, o_i \rangle_{1 \le i \le N}$ represent the sequence of actions and observations taken by the algorithm before the goal is achieved. Let $\phi^{(0)}$ represent the initial belief. Let $\phi^{(i)} = SLAF[\langle a_j, o_j \rangle_{1 \le j \le i}](\phi^{(0)})$ denote the ith belief after executing the first i actions and receiving the first i observations.

Consider the first plan executed by the algorithm after the first k steps. Let $\phi^{(a)}$ represent the belief before execution of the plan, and let $\phi^{(b)}$ represent the belief after execution of the plan (so we have $k < a < b$).

Note that $m_f(\phi_f^{(b)}) \le m_f(\phi_f^{(a)})$ for all $f \in \mathcal{P}$, (see proof of Theorem 4.1; note that we assume actions are deterministic). We also have $m(\phi^{(b)}) < m(\phi^{(a)})$ as argued in the same proof. Because $m(\phi) = \prod_{f \in \mathcal{P}} m_f(\phi_f)$, we must have for some f, $m_f(\phi_f^{(b)}) < m_f(\phi_f^{(a)})$.

Let x represent the last point less than or equal to a at which f was observed. Let y represent the first point greater than or equal to b at which f was observed. Because $m_f(\phi_f^{(a)}) \le m_f(\phi_f^{(x)})$ and $m_f(\phi_f^{(b)}) \le m_f(\phi_f^{(y)})$, we have $m_f(\phi_f^{(y)}) < m_f(\phi_f^{(x)})$. Since f was observed at both x and y, it follows that $\phi_f^{(a)} \equiv f^{(a)} \wedge A_f^{(a)}$ and $\phi_f^{(b)} \equiv f^{(b)} \wedge A_f^{(b)}$, where $f^{(i)}$ denotes the literal observed for f at step i. Since the A_f portion of each fluent-factored formula grows monotonically as filtering proceeds, $A_f^{(a)}$ must contain a subset of the clauses in $A_f^{(b)}$, and all clauses in these formulas contain action propositions only (i.e., they contain no fluents from $\mathcal{P}$). Therefore, it follows that $A_f^{(b)}$ must contain at least one new clause.

Therefore, we see that after execution of the plan, at least one new clause must have been learned. After an additional $2k$ actions are executed, this argument can be repeated for subsequent plans. Note that there are fewer than $|\mathcal{P}|(3|\mathcal{A}|)^k$ possible clauses containing only action propositions (e.g., $a^f, a^{\neg f}, a^{f\circ}$) that can appear in the belief formula. Additionally, each plan is of length at most r. Together, these facts imply that the number of actions that can occur before the goal is achieved is at most $O(\max(r, 2k)|\mathcal{P}|(3|\mathcal{A}|)^k)$. □

Lemma 8.2. *Each action can fail at most $2^m 3^q$ times.*

Proof. Let $O_a = \bigcup_{a' \in \mathcal{A}} (\mathsf{eff}(a') \cap \mathsf{pre}(a))$. Let $L_a = \mathsf{pre}(a) \cup (\bigcup_{f \in O_a} \{a^f, a^{\neg f}, a^{f\circ}\})$.

Let $\sim$ denote the relation that two models are equal when restricted to L_a. That is, if $x, y \in \mathsf{models}(\phi)$, then $x \sim y$ iff $x|_{L_a} = y|_{L_a}$. The relation $\sim$ induces a partition of the models of ϕ into equivalence classes $\mathsf{models}(\phi)/\sim$. Let $m_a(\phi) = |\mathsf{models}(\phi)/\sim|$ denote the number of such equivalence classes.

Now, we claim $m_a(SLAF[\langle a_i, o_i \rangle_i](\phi)) \le m_a(\phi)$ for any sequence of deterministic actions and observations $\langle a_i, o_i \rangle_i$. It suffices to show that all models in each equivalence class are mapped to the same equivalence class after filtering. Clearly this is the case when filtering on an observation. Likewise, this is the case when we filter on an action, by the way in which L_a was constructed.

Furthermore, every time action a fails, $m_a(\phi)$ is decreased by at least 1. This is because in at least one of the equivalence classes, the fluents appearing in the precondition must not agree with the precondition. However, $1 \le m_a(\phi) \le 2^m 3^q$, and thus the claim follows. □

Finally, we can show that the bound holds. Consider $\langle a_i, o_i \rangle_{1 \le i \le N}$, the sequence of actions taken and observations received by the algorithm. Now consider the same sequence with all steps where actions failed removed. This sequence still achieves the goal; therefore, by the argument presented in claim 1, the number of actions from successful plans (plans that did not end in an action failure) in this sequence is bounded by $O(\max(r, 2k)|\mathcal{P}|(3|\mathcal{A}|)^k)$. Likewise, by claim 2, the number of actions from failed plans in this sequence is bounded by $O(r|\mathcal{A}|2^m 3^q)$. Thus the claim holds. □

References

Amir, E. 2005. Learning partially observable deterministic action models. In *Proc. Nineteenth International Joint*

Conference on Artificial Intelligence (IJCAI '05). Morgan Kaufmann.

Bertoli, P., and Pistore, M. 2004. Planning with extended goals and partial observability. In *Proceedings of the 14th Int'l Conf. on Automated Planning and Scheduling (ICAPS'04)*.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01)*, 473–478. Morgan Kaufmann.

Chang, A., and Amir, E. 2005. Learning partially observable deterministic action models (ii). Technical Report UIUCDCS-R-2005-2661, University of Illinois Urbana-Champaign, Department of Computer Science, UIUC, Urbana, IL, USA.

Even-Dar, E.; Kakade, S. M.; and Mansour, Y. 2005. Reinforcement learning in POMDPs. In *Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '05)*, 660–665. AAAI Press.

Eyal Even-Dar, S. K., and Mansour, Y. 2005. Reinforcement learning in pomdps without resets. In *IJCAI*.

Garcia-Martinez, R., and Borrajo, D. 2000. An integrated approach of learning, planning, and execution. *J. Intell. Robotics Syst.* 29(1):47–78.

Ghallab, M.; Hertzberg, J.; and Traverso, P., eds. 2002. *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*. AAAI.

Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the 11th International Conference on Machine Learning (ICML-94)*, 10–13.

Hlubocky, B., and Amir, E. 2004. Knowledge-gathering agents in adventure games. In *AAAI-04 Workshop on Challenges in Game AI*. AAAI Press.

Hoffmann, J., and Brafman, R. 2005. Contingent planning via heuristic forward search with implicit belief states. In Biundo, S.; Myers, K.; and Rajan, K., eds., *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS-05)*. to appear.

Ilghami, O.; Nau, D. S.; Muñoz-Avila, H.; and Aha, D. W. 2002. Camel: Learning method preconditions for htn planning. In Ghallab et al. (2002), 131–142.

Jaakkola, T.; Singh, S. P.; and Jordan, M. I. 1995. Reinforcement learning algorithm for partially observable Markov decision problems. In Tesauro, G.; Touretzky, D.; and Leen, T., eds., *Advances in Neural Information Processing Systems*, volume 7, 345–352. The MIT Press.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101:99–134.

Kautz, H. A., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, 359–363.

Kautz, H. A., and Selman, B. 1999. Unifying sat-based and graph-based planning. In Dean, T., ed., *IJCAI*, 318–325. Morgan Kaufmann.

Kearns, M. J.; Mansour, Y.; and Ng, A. Y. 1999. Approximate planning in large pomdps via reusable trajectories. In Solla, S. A.; Leen, T. K.; and Müller, K.-R., eds., *NIPS*, 1001–1007. The MIT Press.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2000. Approximate planning in large POMDPs via reusable trajectories. In *Proceedings of the 12th Conference on Neural Information Processing Systems (NIPS'99)*, 1001–1007. MIT Press.

Koenig, S. 2001. Minimax real-time heuristic search. *Artif. Intell.* 129(1-2):165–197.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An interactive method for inducing operator descriptions. In Ghallab et al. (2002), 121–130.

Pasula, H. M.; Zettlemoyer, L. S.; and Kaelbling, L. P. 2004. Learning probabilistic relational planning rules. In *Proceedings of the 14th Int'l Conf. on Automated Planning and Scheduling (ICAPS'04)*. AAAI Press.

Wang, X. 1994. Learning planning operators by observation and practice. In *Artificial Intelligence Planning Systems*, 335–340.

Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning planning operators by observation and practice. In *Proc. National Conference on Artificial Intelligence (AAAI '05)*.

Zhang, L.; Madigan, C. F.; Moskewicz, M. W.; and Malik, S. 2001. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, 279–285.