Derivation of compact policies for fully observable non-deterministic planning problems using SAT solvers

Author: Tomás Geffner

MASTER THESIS UPF / YEAR 2016

DIRECTOR
Dr. Héctor Geffner Departament of Information and Communication Technologies (DTIC)
Abstract

We propose a new SAT-based approach to fully observable non-deterministic (FOND) planning problems where compact policies are obtained from polynomial CNF encodings by running SAT solvers. Unlike recent, related SAT approaches to fully and partially observable planning, the encodings represent actions, atoms, and controller states, but not full states that are exponential in number. Likewise, the size of the policies can be exponentially smaller than the number of states reachable with those policies. The strengths and weaknesses of the new approach is then compared empirically with those of existing, state-of-the-art approaches, mostly based on greedy and revisable forms of replanning. The analysis sheds light also on the strengths and limitations of the current set of benchmarks used in the area, and in the notion of “interesting” (non-trivial) FOND and MDP problems.
## Contents

1 INTRODUCTION ................................................. 7  
   1.1 Autonomous behavior .................................. 7  
   1.2 Planning models ....................................... 7  
   1.3 Representation of planning models .................. 9  
      1.3.1 STRIPS ........................................... 9  
      1.3.2 PDDL ........................................... 10  
   1.4 Thesis outline ........................................ 11  

2 FULLY OBSERVABLE NON DETERMINISTIC PLANNING .............. 13  
   2.1 FOND model .......................................... 13  
   2.2 Classical planners for FOND problems .............. 14  
      2.2.1 FF-Replan: A greedy, on-line classical replanner 14  
      2.2.2 PRP: A careful, off-line classical replanner ...... 15  
   2.3 Probabilistic interesting problems .................. 15  

3 SAT APPROACH TO FOND ..................................... 17  
   3.1 SAT for classical planning problems .............. 17  
   3.2 SAT for non deterministic planning problems ..... 18  
   3.3 Our approach ........................................ 19  
   3.4 Basic encoding ....................................... 20  
   3.5 Properties ........................................... 21  
   3.6 Improved encoding: Removing Symmetries .......... 25  
   3.7 Results ................................................ 27  
   3.8 Variations ............................................ 31  
      3.8.1 Strong Planning and partial observability ...... 31  

4 CONCLUSIONS .................................................. 33  
   4.1 Summary .............................................. 33  
   4.2 Future work ........................................... 33
Chapter 1

INTRODUCTION

1.1 Autonomous behavior

A central problem regarding intelligent behavior is selecting what action to do next. In Artificial Intelligence (AI) three different approaches have been used to address this problem: programming-based, learning-based, and model-based approach [7]. The first one consists on specifying how should the agent behave in each possible situation it may encounter explicitly, i.e. the programmer anticipates every possible situation that the agent may face and programs the agent’s response. This approach might work for simple problems, but lacks flexibility; if the problem is slightly modified, the programmer has to modify every rule accordingly. Furthermore, if the agent reaches a situation not considered by the programmer, it would not know how to react. In the second approach, learning-based, the agent learns how to act while interacting with the environment and receiving penalties and rewards. The objective is simple: maximize the accumulated reward. Finally, in the model-based approach, the behavior is not pre-programmed nor learned but derived automatically from a given model that describes the actions, sensors and goals. The fact that this approach uses a model is both an advantage, as it is domain-independent and flexible, and a disadvantage, since models for complex domains are not easy to obtain. Also, solving the model, even the simplest ones, is computationally intractable. The model based approach to intelligent behavior is called planning.

1.2 Planning models

There is a wide variety of models used to represent different types of planning problems. All of them can be seen as variations of the state model $S = (S, s_o, S_G, A, f, c)$ underlying classical planning problems:
• S: finite and discrete state space
• $s_0 \in S$: known initial state
• $S_G \subseteq S$: non empty set of goal states
• A and $A(s)$: set of actions and set of actions applicable in state s, respectively
• $f(a, s)$: deterministic transition function
• $c(s, a)$: positive action costs

A solution to classical planning problems is an applicable sequence of actions (plan) that maps the initial state into a goal state. More precisely, a plan $\pi = a_1, a_2, ..., a_n$ must generate a sequence of states $s_0, s_1, ..., s_n$ such that $s_i = f(a_i, s_{i-1})$, $a_i \in A(s_{i-1})$ and $s_n \in S_G$.

Another model is the one used for partially observable planning problems, which is a variation of the classical planning model that features uncertainty about the initial and next states, and partial information about the current hidden state. It consists of

• S: finite and discrete state space
• $S_0 \subseteq S$: non empty set of possible initial states
• $S_G \subseteq S$: non empty set of goal states
• A and $A(s)$: set of actions and set of actions applicable in state s, respectively
• $f(a, s)$: non deterministic transition function (returns set of possible successor states)
• O: set of observations
• $O(s, a) \subseteq O$: sensor model such that $o \in O(s, a)$ means that $o$ may be observed in state $s$ if $a$ was the last action done
• $c(s, a)$: positive action costs

Solutions to this kind of problems must take observations into account; thus, solutions are functions that map the stream of past actions and observations into actions, or, more precisely, beliefs states into actions. The resulting belief state, after a sequence of actions and observations, represents the set of states that are
possible at that given moment, and it summarizes all the relevant information in order to select an action.

Finally, a fully observable model is just a partially observable one in which the wholes states are observable. In such a case, the set of observations and the the sensor model can be omitted.

1.3 Representation of planning models

1.3.1 STRIPS

The oldest language used to represent planning models is STRIPS [6]. A planning problem in STRIPS is a tuple \( P = (F, I, G, O) \) where

- \( F \): set of atoms (boolean variables)/propositions of interest
- \( I \subseteq F \): initial situation
- \( G \subseteq F \): goal
- \( O \): set of actions

In STRIPS each action is characterized by an Add list, which includes the atoms that become true after applying the action, a Delete list, which includes the atoms that are set to false after applying the action, and a Precondition list, which includes the atoms that must be true in a state for the action to be applicable. A problem \( P = (F, I, G, O) \) encondes a state model \( S = (S, s_0, S_G, A, f, c) \) where

- The states \( s \) are collections of atoms from \( F \)
- The initial state \( s_0 \) is \( I \)
- The set of goal states \( S_G \) is composed of every state \( s \) such that \( G \subseteq s \)
- The set of actions \( A \) is given by \( O \), and \( A(s) \) is composed by actions \( o \) such that \( Precondition(o) \subseteq s \)
- The transition function \( f(s, a) \) is given by \( f(s, a) = s \cup Add(a) - Delete(a) \)
- The action cost is 1

The state representation obtained using STRIPS is domain independent; that is, any problem expressed in this language can be used as an input to a solver programmed to process inputs expressed in STRIPS, regardless of the “contents” or “situation” described by the problem. For instance, the Blocks World and the
Logistics domain can be both expressed in STRIPS, even though they refer to completely different problems: building specific towers of blocks using a gripper and moving packages to target destinations using vehicles.

1.3.2 PDDL

PDDL [11] (Planning Domain Definition Language) extends STRIPS while providing a standard, machine-readable syntax. Problems in PDDL are expressed in two parts: one describing the general domain and other describing a particular instance or problem. For example, the Blocks World domain can be defined as follows in PDDL:

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x block ?y block)
               (on table ?x block)
               (clear ?x block)
               (handempty)
               (holding ?x block))

  (:action pick-up
   :parameters (?x block)
   :precondition (and (clear ?x) (on table ?x) (handempty))
   :effect
   (and (not (on table ?x))
        (not (clear ?x))
        (not (handempty))
        (holding ?x)))

  (:action put-down
   :parameters (?x block)
   :precondition (holding ?x)
   :effect
   (and (not (holding ?x))
        (clear ?x)
        (handempty)
        (on table ?x)))

  (:action stack
   :parameters (?x block ?y block)
   :precondition (and (holding ?x) (clear ?y))
   :effect
```

10
(and (not (holding ?x))
  (not (clear ?y))
  (clear ?x)
  (handempty)
  (on ?x ?y)))

(:action unstack
  :parameters (?x - block ?y - block)
  :precondition (and (on ?x ?y) (clear ?x) (handempty))
  :effect
  (and (holding ?x)
       (clear ?y)
       (not (clear ?x))
       (not (handempty))
       (not (on ?x ?y)))))

This encoding describes the general domain. A specific domain instance can be defined as follows

(define (problem BLOCKS–4–0)
  (:domain BLOCKS)
  (:objects D B A C – block)
  (:INIT (CLEAR C) (CLEAR A) (CLEAR B) (CLEAR D) (ONTABLE C) (ONTABLE A) 
        (ONTABLE B) (ONTABLE D) (HANDEMPY))
  (:goal (AND (ON D C) (ON C B) (ON B A)))
)

The encoding defines a task with 4 blocks, in which initially each one is on the table, and specifies the specific tower that should be built. One of the possible solutions to this problem is the sequence

pick(B), stack(B, A), pick(C), stack(C, B), pick(D), stack(D, C)

1.4 Thesis outline

The thesis is organized as follows. First we define FOND problems, the three main kinds of solutions and describe two existing computational approaches. We then describe our SAT-based approach for FOND planning problems and report experiments that show the strengths and limitations of the new approach.
Chapter 2

FULLY OBSERVABLE NON DETERMINISTIC PLANNING

2.1 FOND model

A Fully Observable Non Deterministic (FOND) model is like a classical planning model, with the addition that actions may have non-deterministic effects. Thus, the only difference lays on the fact that the transition function \( f(a, s) \) is non deterministic, i.e. several different states \( s' \) may be the result of applying action \( a \) (non deterministic) in state \( s \), and which one will occur cannot be predicted a priori by the planner. In PDDL this non determinism is expressed using the keyword `oneof`. For example, a pick action in the Blocks World with two possible outcomes, one in which the agent was able to pick up the block and is currently holding it, and one in which the block slipped from the agent’s hand and is now on the table, is encoded in (the extended) PDDL as:

\[
(:\text{action} \ \text{pick-up} \\
:\text{parameters} \ (?b - \text{block}) \\
:\text{precondition} \ (\text{and} \ (\text{emptyhand}) \ (\text{clear} \ ?b) \ (\text{on-table} \ ?b)) \\
:\text{effect} \ (\text{oneof} \\
\hspace{1cm} (\text{and}) \\
\hspace{1cm} (\text{and} \ (\text{holding-one}) \ (\text{holding} \ ?b) \ (\text{not} \ (\text{emptyhand})) \\
\hspace{1cm} (\text{not} \ (\text{on-table} \ ?b)) \ (\text{not} \ (\text{clear} \ ?b)))))
\]

The first possible effect is empty, meaning that nothing changed (the block slipped and is back on the table), while the second effect represents the outcome for which the agent holds the block.

A solution to FOND problems represents a function that maps states into actions, also called a policy. Clearly, applying a policy can result in many differ-
ent executions depending on the alternative outcomes that may arise from non-deterministic actions. Three types of plans, policies, or solutions have been distinguished [5]:

**Weak policies** reach the goal in some of the possible executions

**Strong policies** reach the goal in all the possible execution in a finite number of steps

**Strong cyclic policies** reach the goal in all the possible execution under the assumption that non-determinism is probabilistic or fair

Our focus is on the computation of strong cyclic policies, although our approach can also be used with minor modifications for computing strong policies as well. Weak policies, on the other hand, can be trivially computed by a classical planner on the so-called all-outcome-relation of the FOND problem, where action non-determinism is assumed to be under the control of the planning agent. This idea is exploited for computing strong cyclic policies in greedy and careful replanners.

### 2.2 Classical planners for FOND problems

Currently, many solvers for FOND problems are based on classical planning solvers. As examples one could mention FF-Replan [13], winner of the 2004 International Probabilistic Planning competition (IPPC-04) [14] and top performer in IPPC-06 [3] domains, and PRP [12] (Planning for Relevant Policies), that represents the current state of the art for FOND planning. Basically, both use a classical planner to solve the problem assuming they can control which outcome occurs for every action, and replan as soon as an outcome that was not considered takes place.

#### 2.2.1 FF-Replan: A greedy, on-line classical replanner

FF-replan [13] doesn’t compute policies; just the action to do next. It’s an on-line planner. For this, it uses the all-outcome-relaxation of the FOND problem for computing a classical plan. It then applies this plan until reaching a goal state, or until a state is observed that is different than the state predicted. In the latter case, the classical planner is invoked again from the observed state, and the process repeats until a goal state is found.

This form of greedy on-line planning scales up as well as the underlying classical planner that is used. On the other hand, the approach may fail to reach a goal state in the presence of dead-ends, even when there exist strong or strongly cyclic plans that can always reach the goal by avoiding such dead-ends.
2.2.2 PRP: A careful, off-line classical replanner

The PRP planner also uses an underlying classical planner, but unlike FF-replan, it is complete; namely, it computes a strong cyclic policies for FOND problems when such policies exist. This is achieved by calling the classical planner from every state that is reachable with the policy computed so far, until the policy encodes one such classical plan for each such state. This process is accelerated by recording the states and the classical plans that map them into goal states, and by using regression for generalizing these plans to more states. When this process encounters a state from which the goal cannot be reached via a classical plan consistent with the policy so far, the state is marked as a dead-end, the actions leading to such states are marked as non-applicable in the states where they lead to marked states, and the whole process is started from scratch again, while keeping track of the actions that cannot be applied in certain (partial) states. The result of these backtracks is that the algorithm is complete. On the other hand, in some problems, the number of replanning episodes required may be exponential. We will actually show how such inefficient behavior can arise from the addition of irrelevant actions in otherwise simple and meaningful problems.

2.3 Probabilistic interesting problems

Due to the success of greedy, on-line replanning approaches in the first two MDP competitions, Little and Thiebaux [10] provided a class of “interesting” MDP or FOND problems where such approaches would be bound to fail. Such problems involve

- **Presence of (avoidable) deadends.** Should there be no deadends, a replanner will reach the goal with 100% probability, and usually it will find a solution faster than a probabilistic planner.

- **Multiple goal trajectories.** If there is only one trajectory to the goal, it would be straightforward for a replanner to find it. Furthermore, should there be multiple trajectories to the goal, but none share common actions, the situation would be similar. Thus the authors state that one or more pairs of distinct goal trajectories such that their first \( n - 1 \) outcomes are the same and the \( n^{th} \) outcome is different is also a necessary condition for a problem to be considered *probabilistically interesting*.

- **Mutual exclusion.** There have to be choices that exclude, potentially or necessarily, other possible choices. If this is not the case, deadends would not be avoidable, and the probability of reaching one would not depend on
the actions chosen, but on the actions’ outcomes, that are not controlled by
the solver.

Nonetheless, the conditions that define the “probabilistic interesting prob-
lems” are good for defeating and showing the limitations of greedy on-line re-
planners, but not for defeating and showing the limitations of careful off-line re-
planners such as PRP that do rather well in such problems. On the other hand,
it is rather simple to envision scenarios where such careful replanners will suffer
too. In particular, FOND planners relying on classical replanners will not scale
up well in problems where there is an exponential number of classical plans in the
all-outcome relaxation such that few of such classical plans can be extended into
fully fledged strong cyclic policies. Moreover, if such few plans are considerably
longer than the other classical plans, the number of replanning episodes required
for finding a strong cyclic solution will be exponential, as practically all classical
planners have a bias towards solutions of lower cost or length. We will actually
see below how such situations arise from the addition of “irrelevant” actions and
atoms. Indeed, one such a problem will results from the tireworld when tires can
be picked and dropped, resulting in an exponential number of tire distribution and
in an exponential number of “short” plans that cannot be extended into strong
cyclic solutions.
Chapter 3

SAT APPROACH TO FOND

A limitation of approaches to FOND planning that rely on classical planners like PRP is that they reason with one complete execution at a time. As a result, they may discover that the partial policy constructed up to a point cannot be extended into a fully strong cyclic policy, leading in certain cases to exponentially many backtracks. In order to address this limitation, we formulate a novel, SAT-based approach to FOND planning that is capable of reasoning about all possible executions in “parallel”. For this, compact FOND policies are obtained from the satisfying assignments of a given, polynomial SAT (CNF) encoding of the FOND problem. Before introducing this encoding, we review compact SAT encodings proposed for classical planning, and non-compact (i.e., state-based) SAT encodings for FOND planning.

3.1 SAT for classical planning problems

The first SAT-based approach to classical planning problems was introduced in [9]. They introduced a way to encode a SAT formula such that a plan (of length N - fixed) that solves the problem exists if and only if the SAT formula is satisfiable. Furthermore, the plan can be easily obtained given the assignment that satisfies the formula. In order to generate the formula they use clauses that express the following: 1) propositions that are true in the initial state; 2) propositions that must be true in the goal state; 3) precondition of an action must be true in timestep \( i \) for the action to be applicable; 4) if an action applied in timestep \( i \) adds (deletes) an atom \( p \), \( p \) must be true (false) in timestep \( i + 1 \); 5) one and only one action occurs at each timestep; and 6) if an atom is not affected by an action, then its truth value remains the same.

The system starts searching for plans of length 1, if a solution is found it is returned, else it looks for solutions of length 2, and so on until it finds a valid
plan. An advantage of encoding the problem as a SAT formula mentioned by the authors is that it is relatively simple to express conditions and facts in any state of the world. For instance, using only one extra clause one could force that a particular action is applied at a particular timestep, or that a particular atom is true in at least one timestep, among other possible facts.

This SAT-based approach to classical planning can be extended to handle non-deterministic actions in a direct fashion; namely, by having the alternative outcomes as a one-of formula. However, the satisfying assignments of the resulting formula won’t encode strong plans or strong cyclic plans for a FOND problem, but just weak plans, where the action non-determinism is assumed to be (wrongly) under the control of the planning agent. For deriving strong and strong cyclic policies for FOND problems, a different type of SAT encoding is needed.

3.2 SAT for non-deterministic planning problems

Several authors have described SAT encodings for the problem of finding strong cyclic policies for FOND problems [1, 4]. An undesirable aspect of these encodings is that they are not compact; indeed, they introduce atoms and actions for each one of the possible states in the problem. We will address this limitation in the next section, but review first the encoding use by [4] for computing memoryless policies for partially observable non-deterministic problems and POMDPs (when the goal is to be achieved with probability 1, the exact values of the state transition probabilities do not matter, but just whether they are zero or different than zero).

In [4] the authors use a parameter $k$ to limit the length of the shortest path from a state $s$ reachable by the policy to a goal state. Their encoding uses the following variables:

- $A_{ij}$, $1 \leq i \leq |S|$, $1 \leq j \leq |A|$. $S$ and $A$ are the sets of states and actions, respectively. The boolean variable $A_{ij}$ indicates whether action $j$ has a probability greater than 0 of being applied in state $i$.

- $C_i$, $1 \leq i \leq |S|$ express whether state $i$ is reachable following the policy.

- $P_{ij}$, $1 \leq i \leq |S|$, $0 \leq j \leq k$ express whether there is a path to the goal of length at most $j$ from state $i$.

The clauses they use are the following:

- $\bigvee_{j \in A} A_{ij}$ for each state $i \in S$, ensures that at least one action is applied in every state
• $A_{ij} \iff A_{rj}$ if the same observation is observed in states $i$ and $r$. As two states in which the same observation is observed are not distinguishable (memoryless policy) the action applied in each should be the same

• $C_l$ expresses that the initial state is reachable by the policy

• $C_i \land A_{ij} \rightarrow C_l$ if the probability of going from state $i$ to state $l$ is greater than 0 when the action $j$ is applied. This clauses will assign true to the $C_l$ variables, if the state $l$ is reachable by the policy

• $P_{Gj}$ for all $0 \leq j \leq k$ expresses that the goal state is reachable from the goal state using a path of length at most $0$

• $P_{ij} \iff \bigvee_{a \in A} [A_{ia} \land (\bigvee_{i' \in S: P(i,a,i') > 0} P_{i'i-1})]$ expresses that there is a path of at most $j$ steps from $i$ to the goal if and only if: 1) the probability of reaching a state $i'$ by applying an action $a$ in $i$ is greater than 0 (and $A_{ij}$ is true); and 2) there is a path of at most $j - 1$ steps from $i'$ to the goal

All this clauses are expressed as disjunctions and form a CNF. If the formula is satisfied, a winning strategy can be easily extracted using the truth values of the $A_{ij}$ variables. Should the formula be unsatisfiable for $k = |S|$, no memoryless winning strategy exists.

### 3.3 Our approach

The proposed new SAT encoding for computing policies for FOND problems using SAT solvers is designed to satisfy two requirements. First, the encoding must be polynomial and hence cannot introduce atoms and actions for each of the problem states. Second, they must represent policies in compact form, meaning that in some cases, the size of the policies as determined from the satisfying assignments will be exponentially smaller than the number of states that will be reachable by those policies. For this, the policies will map partial states into actions, where partial states are (possibly non-maximal) sets (conjunctions) of atoms. The SAT encodings accepts a $k$ parameter that represents the number of controller states; i.e., the number of partial states for which an action is prescribed. If the controller states are represented by $n$ and the states by $s$, executions define trajectories in the space of $(n, s)$ pairs starting with the initial controller state $n = n_0$ and the initial state $s = s_0$. The non-deterministic action $a$ is prescribed in the controller state $n$, if the atom $(n, a)$ is true. Similarly, the atom $p$ is part of the partial state associated with the controller state $n$, if $p(n)$ is true.
3.4 Basic encoding

For describing the encoding, we assume that each non-deterministic action $a$ is associated with a set $b_1, b_2, ..., b_m$ of deterministic actions; i.e. $a = b_1, ..., b_m$, $m > 1$, where each deterministic action $b_i$ is unique to each action $a$. We use $k$ as the parameter of the SAT encoding; it encodes the number of possible controller states (partial states in the policy). Variables used:

- $p(n)$: atom $p$ true in controller state $n$
- $(n, b)$: deterministic action $b$ is applied in controller state $n$
- $(n, b, n')$: $n'$ is the controller state obtained after applying deterministic action $b$ in controller state $n$
- $\text{Reachable}I(n)$: controller state $n$ reachable from initial controller state $n_0$ following the policy
- $\text{Reachable}G(n, j)$: goal controller state $n_G$ reachable from controller state $n$ in at most $j$ steps following the policy

Clauses used:

1. $\neg p(n_0)$ if $p \notin S_0$
2. $p(n_G)$ if $p \in G$
3. $(n, b) \rightarrow p(n)$ if $p \in \text{prec}(b)$. The precondition of action $b$ must be true in controller state $n$ for the action to be applicable
4. $(n, b) \rightarrow (n, b')$ if $b, b'$ are two possible deterministic outcomes of the same non-deterministic action. If a non-deterministic action is applied in a controller state, every possible outcome should be taken into account
5. $(n, b) \rightarrow \neg(n, b')$ if $b$ and $b'$ are deterministic outcomes of different non-deterministic actions. Only one non-deterministic action is applied in one controller state
6. $(n, b) \iff \bigvee_{n'}(n, b, n')$. Every action applied in a controller state leads to another controller state
7. $(n, b, n') \land \neg p(n) \rightarrow \neg p(n')$ if $p \notin \text{add}(b)$. Negative propagation of atoms; an atom cannot be true unless it’s added by an action
8. \((n, b, n') \rightarrow \neg p(n')\) if \(p \in \text{del}(b)\). Negative propagation of atoms; an atom cannot be true if it’s deleted by an action

9. \(\text{ReachableI}(n_0)\). The initial state \(n_0\) is always reachable from \(n_0\)

10. \((n, b, n') \land \text{ReachableI}(n) \rightarrow \text{ReachableI}(n')\). If controller state \(n\) is reachable from the initial state and action \(b\) leads from \(n\) to \(n'\), \(n'\) is reachable from the initial state

11. \(\text{ReachableG}(n_G, 0), \ldots, \text{ReachableG}(n_G, k)\) with \(k\) being the number of controller states. The goal state is reachable from the goal state in at most 0 steps

12. \(\neg \text{ReachableG}(n, 0)\) for \(\forall n \neq n_G\)

13. \(\text{ReachableG}(n, j + 1) \iff \bigvee_{b, n'}[(n, b, n') \land \text{ReachableG}(n', j)]\). \(n_G\) is reachable from \(n\) in at most \(j + 1\) steps if and only if \(n\) is connected through an action to \(n'\) and \(n_G\) is reachable from \(n'\) in at most \(j\) steps

14. \(\text{ReachableG}(n, j) \rightarrow \text{ReachableG}(n, j + 1); j < k\)

15. \(\text{ReachableI}(n) \rightarrow \text{ReachableG}(n, k)\). If controller state \(n\) is reachable from the initial state, the goal state should be reachable from \(n\)

For computing policies for a FOND problem \(P\), a SAT-solver is called over the CNF formula \(C(P, k)\) for \(k = 1\), increasing the \(k\) parameter by 1 until a solution is found. If the formula is unsatisfiable for \(k = |S|\), \(P\) has no solution. The approach however is meaningful computationally only when \(P\) is solvable through the use of compact policies (small values of \(k\) in relation to \(|S|\)).

### 3.5 Properties

The final system consists on a set of controller states, each one characterized by a set of positive atoms, connected by deterministic actions. The initial state is the pair \((n_0, s_0)\). Once an action is applied, the system moves to the next pair \((n_i, s_j)\).

**Property 1:** Let \(\Sigma\) be a satisfying assignment for the formula \(C(P, k)\) and let \(\pi_\Sigma\) be the policy defined by \(\Sigma\). If the pair \((n, s)\) is reachable by \(\pi_\Sigma\), and the atom \(p(n)\) is true in \(\Sigma\), then \(p\) is true in \(s\).

*Proof.* This is proved by induction in the length of the possible executions.

**Base case**
If atom \( p(n_0) \) is true in \( \Sigma \), \( p \) is true in \( s_0 \). This is forced by clause (1.) described in the previous section (\(-p(n_0) \) if \( p \not\in s_0 \)).

**Inductive case**

Suppose that the property is true for executions of length \( r \) and that the last transition in an execution of length \( r + 1 \) is \((n, s), b, (n', s')\) where \( b \) is one of the outcomes of the action \( a \) prescribed for the controller state \( n \). We want to show that if \( p(n') \) is true in \( \Sigma \), \( p \) must be true in \( s' \). Let’s assume otherwise; namely that \( p(n') \) is true in \( \Sigma \) but \( p \) is false in \( s' \). There are two cases to consider:

1. If \( p \not\in Add(b) \), since \( p \) is false in \( s' \), it must be false in \( s \). From the induction, this means that \( p(n) \) must be false in \( \Sigma \). Yet this is not possible, as formula (7.) of the encoding ensures that \( p(n') \) should then be false as well, in contradiction with the assumption.

2. If \( p \in Add(b) \), \( p \) must be true in \( s' \), which also contradicts the assumption.

As a result, \( p(n') \) implies that \( p \) must be true in \( s' \), establishing the theorem.

**Property 2:** Policies are applicable, i.e., if the execution reaches the joint state \((n, s)\) and \((n, a)\) is true in the satisfying assignment \( \Sigma \), then for each precondition \( p \) of \( a \), \( p(n) \) must be true in \( \Sigma \) as well. This follows immediately from clause 3 of the encoding.

**Property 3: Soundness.** If \( \Sigma \) is a satisfying assignment for \( C(P, k) \), the compact policy \( \pi_\Sigma \) encoded by \( \Sigma \) represents a strongly cyclic solution to \( P \).

**Proof.** If the policy reaches a joint state \((n, s)\), i.e. there is a path \((n_0, s_0), (n_1, s_1), ..., (n, s)\), \( \text{ReachableI}(n) \) will be set to true by clause (10.), thus forcing \( \text{ReachableG}(n, k) \) to be true (clause (15.)). In order to satisfy clause (13.) there must be a path \((n_1, s_1), (n_2, s_2), ..., (n_j, s_j)\) of at most \( k \) steps, with \( n_1 = n \) and \( s_1 = s \), such that \( n_j \) is \( n_G \) and \( s_j \) is a goal state (see property 3.1).

**Property 3.1:** If an execution reaches the pair \((n_G, s)\), \( s \) must be a goal state. This follows from clause (2.) of the encoding.
Property 4: Completeness. Let $\pi$ be a strong cyclic policy that solves $P$, and let $\pi'$ be a compact representation of $\pi$ obtained as follows: first, states $s$ not reachable are removed from the policy, second, each reachable state $s$ is replaced by the partial state $s' \subseteq s$ comprised of the atoms $p$ true in $s$, except for the atoms $p$ that are not relevant in the $\pi$-executions starting from $s$ (i.e. $p$ does not appear in the goal nor in the precondition of an action appearing later in a $\pi$-execution from $s$).

If $k$ is the number of partial states in $\pi'$, there is a satisfying assignment $\Sigma$ of $C(p, k)$ that encodes the compact policy $\pi'$; i.e. where there is a correspondence between the controller states $n$ and the partial states $s$ in $\pi'$, such that: 1) $\pi'(s') = a$ iff $(n, a)$ is true in $\Sigma$, and 2) $p$ is true in $s'$ iff $p(n)$ is true in $\Sigma$.

It may occur that two different states, $s_1$ and $s_2$ such that both result in the same partial state $s'$ given $\pi$, are assigned different actions, $a_1$ and $a_2$, by the policy $\pi$. In these cases $\pi'(s')$ may be assigned to any of the options without affecting the validity of the policy (i.e. $\pi'(s') = a_1$ and $\pi'(s') = a_2$ are both valid options).

Property 5: The size of the policy encoded by a truth assignment satisfying $C(P, k)$ can be exponentially smaller than the number of states reachable by that policy. That is, if we collect all the pairs $(n, s)$ reachable in some execution of the policy, the number of states $s$ reachable can be exponentially larger than the number of controller states $n$ reachable.

Proof. For this it suffices to provide an example, where there is a reduced policy $\pi'$ for which the number of reachable states $s$ is exponential in the number of controller states. Then from the proof of Property 4, this one follows.

Consider the Tireword domain, which consists on several locations that form a graph, and a car, able to drive from location $a$ to $b$ only if they are connected, that has to drive from the initial location to a goal location. The drive action is non deterministic, as the car can drive from $a$ to $b$ without inconveniences, or it can get a flat tire (but it will get to $b$). In the latter case the car cannot drive to another location unless it has an extra tire available to replace the damaged one. Finally, some of the locations have extra tires that the car can use, but cannot move to another location. A strong cyclic policy for this problem would be to go through a path such that every location in it has an extra tire.

In Fig. 3.1 and Fig. 3.2 we show a particular task and its solution, respectively.
Figure 3.1: Example of the tireworld domain. Square location have spare tires, round ones do not

Figure 3.2: Solution to the example shown in Fig. 3.1

It can be easily generalized that the number of controller states is linear in the number of locations in the correct path \((l_0, l_1, \ldots, l_m)\) - i.e. the path with extra tires in every location. However, the number of states reachable by the policy is exponential. When the car reaches the goal, it might have used the spare tire in location \(l_1\) or not, depending on whether it got a flat when going from \(l_0\) to \(l_1\). The same is valid for \(l_2, l_3, \ldots, l_{m-1}\). So, basically, there
are \(2^{m-2}\) possible states in which the car is in the goal and the policy was followed (spare tire or not in each location \(l_1, \ldots, l_{m-1}\)).

\[\square\]

### 3.6 Improved encoding: Removing Symmetries

The basic encoding showed before works, but can be improved. We introduced several changes to the formulation. Of these, removing symmetries provided the biggest improvement regarding the results. First we describe what do we mean by symmetries and how we dealt with them and then we explain other modifications.

Suppose that the finite state controller \(P\), shown next, solves a problem

\[
\begin{array}{c}
N_0 \longrightarrow N_1 \longrightarrow N_2 \longrightarrow N_3 \longrightarrow N_G
\end{array}
\]

Controller \(P'\) shown next would also solve the problem

\[
\begin{array}{c}
N_0 \longrightarrow N_1 \longrightarrow N_2 \longrightarrow N_3 \longrightarrow N_G
\end{array}
\]

If a problem is solved with \(N_C\) controller states, there are \((N_C - 2)!\) possible satisfying assignments for \(C(P, k)\) such that equivalent policies are derived. The only difference among these policies is the name assigned to each controller state. When the system tries to find a controller of size \(N_C\) this is not a problem, since the first permutation or name assignment tested would provide a valid policy. Instead, when the system searches for size \(m\) controllers, with \(m < N_C\), it tests every possible name assignation to the controller states, when checking only one would be enough. The number of possible permutations or name assignations can be reduced using the following symmetry breaking clauses:

1. \(\bigwedge_{j \leq i} \neg(n_j, n_g) \rightarrow (n_i, n_{i+1})\)

2. \(FirstG(n_i) \iff (n_i, n_g) \land \bigwedge_{j < i} \neg(n_j, n_g)\)

3. \(AfterG(n_i) \iff \bigvee_{j < i} FirstG(n_j)\)

4. \((n_a, n_b) \land AfterG(n_{b-1}) \rightarrow \bigvee_{i \leq a} (n_i, n_{b-1})\)
Clause (1.) forces the existence of an ordered path from the initial state to the goal (i.e. $n_0, n_1, n_2, ..., n_G$). Clauses (2.), (3.) and (4.) establish an order for the remaining controller states (that are not in this ordered path). Basically, they force that if there is an action from $n_i$ to $n_j$, there should be an action from $n_k$ to $n_{j-1}$ such that $k \leq i$.

For a valid controller there is always at least one permutation that satisfies these conditions, obtained as follows:

The ordered path from $n_0$ to $n_G$ will contain the first $m-1$ nodes and $n_G$, with $m$ being the path’s length. Let this path be called $Q$. The remaining nodes are labeled with their minimum distance to any node in $Q$ and their names are given following the resulting order (lower distance lower number). Let the minimum distance from $Q$ to node $n_i$ be $DQ(n_i)$. There are three possible situations:

- $DQ(n_b) = DQ(n_{b-1})$ and $n_b$ and $n_{b-1}$ have different lowest predecessors. In this case $n_{b-1}$ should be assigned as the successor of the lowest numbered node between the lowest predecessor of both nodes.

- $DQ(n_b) = DQ(n_{b-1})$ and $n_b$ and $n_{b-1}$ have equal lowest predecessors. In this case clause (4.) is satisfied $n_b$ and $n_{b-1}$ as are connected to the same node (which is for both the lowest predecessor).

- $DQ(n_b) > DQ(n_{b-1})$. In this case clause (4.) is also satisfied. We have $DQ(n_q) < DQ(n_k)$ and $q < k$ (with $n_q$ being the lowest predecessor of $n_{b-1}$ and $n_k$ the lowest predecessor of $n_b$), since the node’s values are assigned in increasing order of $DQ$.

Other (less important) modifications to the encoding include:

- Variables $(n_i, n_j)$, meaning that there is an action from $n_i$ to $n_j$ were added together with the clauses $(n_i, n_j) \iff \bigvee_b(n, b, n')$. The clause

$$ReachableG(n, j + 1) \iff \bigvee_{b,n'}[(n, b, n') \land ReachableG(n', j)]$$

was replaced by the simpler clause

$$ReachableG(n, j + 1) \iff \bigvee_{n'}[(n, n') \land ReachableG(n', j)]$$

1A node may have several predecessors. Suppose that $n_q$ is reachable with one action from nodes $n_i$, $n_j$ and $n_k$. The lowest predecessor of $n_q$ would be $n_l$ with $l = \min(i, j, k)$.
• Clauses \( \neg p_i(n) \lor \neg p_j(n) \) if \( p_i \) and \( p_j \) belong to the same mutex group\(^2\) were added.

• Usage of binary variable \( \text{ReachableI}(n, j) \), meaning that controller state \( n \) is reachable from \( n_0 \) in \( j \) steps. The clauses

\[
(n, b, n') \land \text{ReachableI}(n) \rightarrow \text{ReachableI}(n')
\]

were replaced by

\[
\text{ReachableI}(n', j + 1) \iff \bigvee_{n'} [(n, n') \land \text{ReachableI}(n, j)]
\]

And information about the \( h_1 \) heuristic was added using the clauses

\[
p(n) \rightarrow \neg\text{ReachableI}(n, j) \text{ if } j < h_1(p)
\]

• Clause (7.) is replaced by

\[
(n, n') \land \neg p(n) \rightarrow \neg p(n') \lor \bigvee_{b, p \in \text{add}(b)} (n, b)
\]

and

\[
(n, b, n') \land \neg p(n) \rightarrow \neg p(n') \text{ if } p \notin \text{add}(b) \text{ and another deterministic outcome } b' \text{ of the corresponding non deterministic action adds } p.
\]

This modification reduces drastically the number of clauses generated.

### 3.7 Results

In order to understand the strengths and weaknesses of the proposed computational approach to FOND problem solving, we compare the new SAT-based approach with PRP. We consider two sets of domains: one class of existing domains that are large but not “interesting” for “cautious replanners” and a new pair of domains that illustrate domains that will be challenging to FOND planners relying on classical planners (unless they appeal to more sophisticated mechanism for identifying compact dead-end conditions and for using such conditions in the classical replanning episodes).

\(^2\)mutex group: set of atoms such that no two atoms in the group can be true simultaneously
For the experiments a time out of 4 hours and a memory bound of 4 Gbs was used. In Table 3.1 the results obtained in existing FOND domains are shown. The number of instances per domain are shown as well as the number of instances solved by each planner. These are domains that are large but not not challenging for replanners.

<table>
<thead>
<tr>
<th>Domain</th>
<th># Instances</th>
<th>Our SAT</th>
<th>PRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobatics</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Beam walk</td>
<td>11</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Blocks world</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Earth oservation</td>
<td>20</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>Elevators</td>
<td>15</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>Faults</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>First responders</td>
<td>20</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Forest new</td>
<td>20</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Rectangle tireworld</td>
<td>15</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Tireworld</td>
<td>15</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Triangle tireworld</td>
<td>15</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Zenotravel</td>
<td>15</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>194</strong></td>
<td><strong>120</strong></td>
<td><strong>173</strong></td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of our SAT-based approach in relation to PRP in existing FOND domains

PRP performs better, solving 173/194 tasks while our SAT solver solves 120/194. Both solvers solve the smaller instances, but ours does not scale up as well. This was expected, since PRP is based on a classical planner able to solve problems with huge state spaces. Despite the fact that many of these problems are probabilistically interesting according to the definition given in [10], PRP solves them easily. We propose an explanation for this. Consider the Triangle Tireworld domain, which is just like the Tireworld domain with tasks generated following the pattern shown in Fig. 3.3.
The only policy that is strong cyclic is the one that guides the car through the longest path as there are spare tires in every location. A solver might be “tempted” to go through shorter paths, but this would not be a strong cyclic policy. As said above, despite that this problem satisfies the condition given in [10] and can be considered probabilistically interesting according to that criteria, PRP solves them easily. We believe this is because it is simple for the solver to “recover” from dead-ends. When PRP finds a deadend, it computes which pairs \((\text{partial state}, \text{action})\) lead to the deadend and labels them as forbidden pairs, restarts the search and avoids these forbidden pairs. In the Triangle Tireworld these deadends would be being in a location that do not have a spare tire while having a damaged tire. PRP thus concludes that driving to a location that does not have a spare tire leads to deadends. While the classical planner solves the problem the heuristic will still push the search towards these deadends, but the solver will avoid those actions. The idea we identify is related to how long does it takes to the solver to “recover” from those deadends and choose the correct path. For the Triangle Tireworld this is done quite fast.

Next we propose two domains that somehow deal with this issue, one is a variation of Tireworld which we call Tireworld-Drop, and the other is called Islands.

In the Tireworld-Drop domain the car has to go from an initial location to a goal, and can pick up a tire in a location, carry it around and drop it in another location. There are two types of roads 1) spiky, the car may get a flat tire when driving; and 2) normal, the car never gets a flat tire. There are two paths to the
goal, a long and a short one. The long one is composed entirely of normal roads, so the car would get to the goal with no problems, while the short one has two spiky roads between the last two pairs of locations and no extra tires there, so that there is no strong cyclic policy that goes that way. Also, there are several extra tires laying around close to the initial location in the short path.

The Islands domain consists on two islands, the player starts in an initial location in island 1 and has to go to a final location located in island 2. There are two ways of going from one island to the other: swimming (short path) or through a bridge (long path). Walking on the bridge is deterministic, while swimming may get player to the other island or may lead it to its death. Also, there are some monkeys in each island; if one of these is on the bridge the player cannot walk through it. The actions available to the player are: move from a cell to an adjacent one, move a monkey, walk through the bridge, and swim.

The results in these two domains are shown in Table 3.2.

<table>
<thead>
<tr>
<th>Domain</th>
<th># Instances</th>
<th>Our SAT</th>
<th>PRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Islands</td>
<td>60</td>
<td>59</td>
<td>31</td>
</tr>
<tr>
<td>Tireworld-Drop</td>
<td>11</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>71</td>
<td>70</td>
<td>32</td>
</tr>
</tbody>
</table>

Table 3.2: Comparison of our SAT-based approach in relation to PRP in the two new FOND domains

It is seen that PRP managed to solve only one of the eleven tasks for the Tireworld-Drop (our solver solved all); the one with the smallest number of spare tires. For the other tasks it ran out of time.

When it comes to the Islands domain PRP did not solve many tasks (29 out of 60) because it ran out of memory (our solver solved 59). The explanation for this is really simple: from the initial state the player can swim to the goal (but might die while doing so), so in the relaxed domain obtained with the all outcome determinization swimming is clearly the best choice. However, when the solver detects it might die avoids that action, leaving the options of moving a monkey or moving the player to another location. Moving a monkey results in another state of heuristic value 1 in the all outcome relaxation, while moving the player away from the water may result in a state with an heuristic value greater then 1 (which is the correct action if the agent wants to send the player towards the bridge). Of course, the chosen action is the one that moves a monkey. This action is chosen every turn such that a huge number of nodes is generated before the player is moved. It is worth mentioning that the tasks in which the solvers were tested have two parameters: size of the islands ($n \times n$) and the number of monkeys. The islands’ size did not represent the main problem for PRP, as it solved problems for $n = 2, 3, 4, 5$ and 6, but the number of monkeys did, solving the tasks with smaller number of monkeys, and failing in those with more.
Another interesting result is the size of the policies found by our SAT-based approach. This is shown, for some of the tasks, in Table 3.3.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Task ID</th>
<th>Size policy (# controller states)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobatics</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>Beam walk</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>Blocks world</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>Earth observation</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>Elevators</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Faults</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>First responders</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>Tireworld</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>Triangle tireworld</td>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>Triangle tireworld</td>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>Zeno travel</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>Islands</td>
<td>59</td>
<td>12</td>
</tr>
<tr>
<td>Tireworld-Drop</td>
<td>5</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 3.3: Sizes of the policies found by our SAT-based approach in some of the tasks

The size of the policies found by the system goes up to 50 - 60 controller states. Whenever a problem requires a greater number of controller states the system runs into difficulties given that the number of variables and clauses becomes large. However, it is seen that for many problems, even large ones, the policies found are quite small. For instance, the 11th task of the tireworld domain has 36 locations and approximately 140 connections. Despite the huge number of possible states, the solution found is of size 5. Also, the 20th task of the blocks world has 10 blocks (the number of possible states is greater than 10!) and the solution found has 24 controller states. This is also seen for the two proposed domains; the 59th task of the islands domain has two $6 \times 6$ islands and 10 monkeys, that results in more than $72^{11}$ states, has a size 12 solution.

3.8 Variations

3.8.1 Strong Planning and partial observability

A simple modification can be done in order to obtain a system that looks for strong solutions, which consist on finite state controllers without cycles. This can be done, for instance, adding clauses of the form $\neg(n_i, n_j)$ for $j \leq i$. A controller has to be acyclic in order to satisfy these restrictions. Also, using this addition and the transformation described in [2] the resulting encoding can be used for problems with deterministic actions and partial observability. This transformation
is complete given: 1) non-unary clauses describing the uncertainty about the initial state represent invariants; 2) fluents that are hidden in the initial situation do not appear in the condition of conditional effects of (deterministic) actions; and 3) the state space is connected.
Chapter 4

CONCLUSIONS

4.1 Summary

We developed a SAT-based approach to FOND planning. Unlike replanning approaches relying on classical planners, our approach can reason about multiple alternative executions in parallel. Also, unlike previous SAT approaches to FOND planning, it makes use of compact, polynomial CNF encodings that do not grow with the number of problem states, and can capture policies that can reach a number of states that is exponential in the policy’s size.

For this, FOND problems \( P \) are mapped into CNF formulas \( C(P, k) \) where \( k \) is a parameter that defines the number of controller states. For a sufficiently large value of \( k \), the approach is complete, yet the approach is computationally meaningful when a small value of controller states suffice.

We have shown empirically that the proposed SAT-approach does not compete with replanners in problems that are simple but large, but performs much better in problems that are not that large but “probabilistically interesting”. Actually, we provided a further analysis of the classes of FOND problems that pose a severe challenge not only to greedy on-line replanners such as FF-replan, but also to cautious off-line replanners such as PRP.

4.2 Future work

- More effective encoding
- Implement necessary modifications to compute strong policies
- Implement the necessary modifications to solve partially observable problems
• Define more domains that satisfy the condition we added to the ones mentioned in [10]


**Bibliography**


