Planning with partial observability using SAT solvers

Kinga Anna Bohus

MASTER THESIS

SUPERVISOR
Prof. Dr. Hector Geffner

Master in Intelligent and Interactive Systems
Department of Information and Communication Technologies
Universitat Pompeu Fabra

September 2016
Abstract

The problem of planning with partial observability is considered to be one of the most challenging problems in automated planning. The traditional approach to solve planning problems with uncertainty is to search for the goal in belief space. More recently, translation-based approaches have been proposed where partially observable planning (or contingent planning) problems are converted into non-deterministic, fully observable problems or even classical planning problems. However, even when the belief representation or the translation is compact, the planning problems produced by all of these approaches are difficult to solve.

SAT-based approaches for classical planning were found useful in solving problems that are inherently difficult for heuristic search algorithms. The SAT approach has the potential to be successful in contingent planning too, as it can exploit the parallelism that result from multiple branches. In this thesis, a SAT-based approach is introduced for partially observable, deterministic planning. The encoding is found to produce full offline plans with good quality, although future work is needed in order to scale up.
## Contents

List of Figures vii  
List of Tables ix  

1 INTRODUCTION 1  
1.1 Automated Planning 2  
1.2 Basic state model for classical planning 2  
1.2.1 Languages for Classical Planning 3  
1.3 Algorithms for Classical Planning 4  
1.4 Classical Planning as SAT 5  

2 PLANNING WITH PARTIAL OBSERVABILITY 9  
2.1 The Contingent Planning Model 9  
2.1.1 Language for Contingent Planning 10  
2.1.2 Solutions and Solution Forms 10  
2.2 Current Approaches and Formulations 12  

3 CONTINGENT PLANNING AS SAT 15  
3.1 Basic encoding of contingent problem 15  
3.2 Experimental Evaluation 19  

4 DISCUSSION 23
List of Figures

1.1 Example of a domain description in PDDL . . . . . . . . . . 5
1.2 Example of a formula in CNF . . . . . . . . . . . . . . . . . . 5
2.1 Contingent problems in PDDL . . . . . . . . . . . . . . . . . . 11
List of Tables

3.1 Results for contingent planning domains. .......................... 21
Chapter 1

INTRODUCTION

A central task in Artificial Intelligence (AI) research is programming autonomous agents capable of making decisions in order to solve a task. The problem of selecting actions (also called the control problem) for reaching a goal has been addressed following three main approaches. In the programming-based approach, the control is hardwired - specified fully by the programmer. In the learning-based approach, the best action is induced from experience by trial-and-error (reinforcement learning) or based on information provided by an instructor (classification). Finally, in the model-based approach, the solution is derived automatically from a suitable model of the problem. Automated Planning is the model-based approach to action selection. It is domain-independent, flexible and it focuses on the ability to make a decision based on reasoning. The main challenge of this approach is the autonomous exploitation of problem structure for scaling up to large and meaningful problems.

The classical model for planning is an action selection problem with known initial and goal states, deterministic actions and complete observability. The classical planning model can be represented by a directed graph where the nodes are the states and the directed edges are the actions mapping one state to the other. The solution to the problem is a plan that corresponds to a directed path in the graph between the initial node and the goal node. Thus, classical planning is equivalent to path-finding in a directed graph and can be solved by using graph search algorithms.

The model for partial observable planning is a variation of the classical model that features both uncertainty and feedback. The initial state is unknown and partial information about the current state of the system is available via observations. The solution to the problem is a closed loop controller where each action to be done at step $i$ depends on the actions and observations collected up to that point.
Most of the current state-of-the-art solvers are based on heuristic search. In these algorithms the goal plays an active role in the search by means of heuristic functions \( h(s) \) that provide a quick estimate of the cost to reach the goal from state \( s \). The search is driven by these estimates and prefers to visit nodes with better heuristic value. Most current state-of-the-art planners are based on heuristic search, extended with the use of automatically obtained structural information about the problem (helpful actions, landmarks) and multi-queue search architecture.

Another approach for solving classical planning problems is to express them as boolean satisfiability (SAT) problems. In the SAT approach, the planning problem is translated to a CNF formula and solved by (general) SAT solving algorithms.

1.1 Automated Planning

Automated planning is the model-based approach of autonomous behavior. The action selection problem is made up of three parts: the model that expresses the dynamics, feedback and goals of the agent; the languages that express these models in compact form; and the algorithms that use the representation of the models for generating the behavior.

1.2 Basic state model for classical planning

The classical planning model \( Q \) is defined as the tuple \( Q = (S, s_0, S_G, A, f, c) \) where:

- \( S \) is a finite set of states,
- \( s_0 \in S \) is the known initial state,
- \( S_G \subseteq S \) is the set of goal states,
- \( A \) is the set of operators (the actions),
- \( f \) is the state transition function representing the deterministic changes in the system, \( f: S \times A \rightarrow S: (s, a) \mapsto f(s, a) = s' \), and
- \( c \) is the cost function maps each operator to a non-negative cost; \( c: A \mapsto \mathbb{R}_0^+ \).
The set of actions applicable in state $s$ are represented by $A(s) \subseteq A$ and applying the action $a \subseteq A(s)$ in state $s$ results in state $s' = f(s, a)$.

A solution or plan in this model is a sequence of applicable actions achieving a goal state from the initial state of the problem. A plan $\pi$ for a classical problem $P = \langle S, s_0, S_G, A, f \rangle$ is an action sequence $\pi = [a_0, ..., a_n]$ that generates a state sequence $[s_0, s_1, ..., s_{n+1}]$, such that:

- all actions are applicable, and $s_{i+1} = f(s_i, a_i)$, for $0 \leq i \leq n$.
- the sequence terminates in a goal state: $s_{n+1} \in S_G$

The quality of the plan can be evaluated by the cost of the plan:

$$cost(\pi) = \sum_{i=0}^{n} c(a_i)$$

A plan is optimal if it has minimum cost. In most cases a common cost structure is assumed, where all actions cost $c(a, s) = 1$. In this case, the cost of the plan is equal to its length $|\pi|$ and the optimal plan is the one with the minimum number of actions.

### 1.2.1 Languages for Classical Planning

Languages for expressing classical planning models in compact form use factored representation instead of explicitly representing states. A factored representation uses boolean (or multivalued) state variables and represents states as a conjunction, where each state is a complete assignment of the state variables. The operators, applicability and transition functions can also be described in terms of these variables.

The simplest classical planning language, STRIPS is based on boolean state variables, called fluents or atoms, so each variable indicates whether a proposition about the world is true or false in a given state. A planning problem is a tuple $P = \langle F, I, O, G \rangle$ where

- $F$ represents a set of atoms,
- $O$ represents the set of actions,
- $I \subseteq F$ represents the initial situation, and
- $G \subseteq F$ represents the goal.
In STRIPS, the actions \( o \in O \) are represented by three sets of atoms called the Add, Delete and Precondition lists, denoted as \( \text{Add}(o) \), \( \text{Del}(o) \), \( \text{Pre}(o) \). The Add and Delete lists describe the atoms that an action makes true and false respectively. The Precondition list describes which atoms have to be true in order for the action to be applicable. A STRIPS problem \( P = \langle F, I, O, G \rangle \) encodes implicitly the classical state model \( S(P) = \langle S, s_0, S_G, A, f \rangle \) where

- the states \( s \in S \) are the possible collections of atoms over \( F \), where an atom \( p \in F \) is true in \( s \) iff \( p \in s \),
- the initial state \( s_0 \) is \( I \),
- the goal states \( s \in S_G \) are the states for which \( G \subseteq s \)
- the actions \( a \) in \( A(s) \) are the ones in \( O \) with \( \text{Prec}(a) \subseteq s \),
- the state transition function is \( f(a, s) = (s \setminus \text{Del}(a)) \cup \text{Add}(a) \), so that the resulting state \( s' \) is \( s \) but with the atoms in \( \text{Del}(a) \) deleted and the atoms in \( \text{Add}(a) \) added.

Planning problems are generally expressed in the Planning Domain Description Language (PDDL). PDDL accommodates the STRIPS language along with a number of additional syntactic constructs. Problems in PDDL are expressed in two parts: one about the general domain and the other about a particular problem instance. See Figure 1.1 for an example of a PDDL domain description.

### 1.3 Algorithms for Classical Planning

The classical planning problem corresponds to a path finding problem in a directed graph with nodes representing states and edges representing actions. An applicable action sequence corresponds to a directed path in the graph, and a plan for the classical planning problem is a directed path from the initial state to a goal state. Graph search algorithms can be used for finding plans for the classical planning model. The size of the graph is exponential in the number of variables, which makes blind search methods ineffective.

A key development in modern planning research was the realization that search could be guided by heuristics derived automatically from the problem [Bonet and Geffner, 2001]. The heuristics are based on relaxations, such as the delete-relaxation: a problem \( P^+ \) is produced where atoms are added exactly as in \( P \) but never deleted. Finding a plan for the relaxed problem
can be done efficiently, and the solution gives an approximation for the cost of a plan.

### 1.4 Classical Planning as SAT

The Boolean Satisfiability Problem (SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. A Boolean formula, also called propositional logic formula is built from variables, operators AND (conjunction, denoted \( \land \)), OR (disjunction, \( \lor \)), NOT (negation, \( \neg \)) and parentheses. For general SAT solving, the Boolean expression is converted into an equivalent formula in conjunctive normal form (CNF). A clause is a disjunction of literals (a literal is a variable or the negation of a variable). A formula in conjunctive normal form is a single clause or a conjunction of clauses.

\[
(A \lor B) \land (\neg B \lor C \lor \neg D) \land (D \lor \neg E)
\]

Figure 1.2: Example of a formula in CNF
Even though the SAT problem is NP-complete, recently developed algorithms led to dramatic advances in solver performance.

The SAT approach for planning has been introduced in the mid 90s [Kautz et al., 1992]. The basic idea is to take a STRIPS problem $P = (F, I, O, G)$ and a planning horizon $n$, produce a CNF formula $C(P,n)$ and solve it by a general SAT solver. The CNF formula includes propositions $p_0, p_1, ..., p_n$ for each atom $p \in F$ and propositions $a_0, a_1, ..., a_{n-1}$ for each action $a \in O$. The subformulas that make up the CNF formula $C(P,n)$ are

**Init:** $p_0$ for each $p \in I$, $\neg q_0$ for each $q \in F$ such that $q \notin I$.

**Goal:** $p_n$ for each $p \in G$.

**Actions:** For $i = 0, 1, ..., n - 1$ and each action $a \in O$:

- $a_i \supset p_i$ for each $p \in Pre(a)$
- $a_i \supset p_{i+1}$ for each $p \in Add(a)$
- $a_i \supset \neg p_{i+1}$ for each $p \in Del(a)$

**Persistence:** For $i = 0, 1, ..., n - 1$ and each fluent $p \in F$ where $O(p^+)$ and $O(p^-)$ are the actions that add and delete $p$ respectively,

- $p_i \land \bigwedge_{a \in O(p^-)} \neg a_i \supset p_{i+1}$
- $\neg p_i \land \bigwedge_{a \in O(p^+)} \neg a_i \supset \neg p_{i+1}$

**Seriality:** For each $i = 0, 1, ..., n - 1$, if $a \neq a'$, $\neg (a_i \land a'_i)$

If the formula $C(P,n)$ is not satisfiable, then there is no plan of length $n$ solving the problem $P$, while if $C(P,n)$ is satisfiable, the plan given by the actions that are true at time $i = 0, 1, ...$ is a plan that solves $P$. In the SAT approach the planning horizon is increased one by one from $n = 0$ until a satisfying assignment for $C(P,n)$ is found. In this formulation the first plan found is the shortest possible plan, i.e. the optimal plan assuming action costs are uniform.

The above encoding does not scale up as well as heuristic search planners. A number of variations have been introduced to increase the power of the approach. One is to allow a set of actions to be done in parallel. This reduces the horizon where the first solution is found, which significantly increases the performance. Other improvements involve the introduction of NO-OP actions, and the use of lower bounds for initializing the planning horizon [Kautz and Selman, 1999]. These advances were made possible by building
on Graphplan [Blum and Furst, 1997], which became a powerful preprocessing step for SAT-based methods.

More recently, Jussi Rintanen introduced other refinements, including the use of heuristics for variable selection in the SAT solver and better memory management for dealing with the millions of clauses in the encodings of planning problems [Rintanen, 2012]. While SAT-based planners do not yet scale up as well as the best heuristic search planners, the gap has narrowed down considerably. In addition, SAT can do much better on certain inherently difficult problems than the heuristic search counterparts [Hoffmann et al., 2007].
Chapter 2

PLANNING WITH PARTIAL OBSERVABILITY

Classical planning describes the action selection problem under the assumption that the environment is fully observable and actions are deterministic. To solve problems with incomplete information or uncertainty, we need models that go beyond the scope of classical planning.

In presence of uncertainty, the true state of the environment is unknown, yet partial information might be available to the agent from sensors. The two main models of planning with incomplete information are conformant planning and contingent planning. The former considers problems where no sensing is available, the latter involves partial observability of the environment. Here, we introduce the model for contingent planning, and discuss the current formulations and approaches for solving partially observable planning problems.

2.1 The Contingent Planning Model

The deterministic contingent planning model $Q$ is defined as the tuple $Q = \langle S, s_0, S_G, A, O, f \rangle$ where:

- $S$ is a finite set of states,
- $S_0 \subseteq S$ is a set of possible initial states,
- $S_G \subseteq S$ is the set of goal states,
- $A$ is the set of operators (the actions),
• $f$ is the state transition function that maps states to states, $f : S \times A \rightarrow S : (s, a) \mapsto f(s, a) = s'$, and
• an observation function $O(s, a)$ that associates to each state, action pair a set of possible observations (sensor model).

2.1.1 Language for Contingent Planning

To describe the partially observable problem in compact form, the state variable description has to include an encoding of the sensor model. The contingent planning problem can be defined as a tuple $P = \langle F, A, O, I, G \rangle$, where:

• $F$ is a set of Boolean fluents of the problem,
• $A$ is a set of actions, where every action is described by its preconditions and (conditional) effects.
• $I$ is a set of clauses over $F$ defining the initial situation
• $G$ is a set of fluents $G \subseteq F$, describing the goal states
• $O$ is the sensing model, a set of sensing actions where every action is described by a set of literals $C$ and a positive literal $L \in F$. If the precondition $C$ holds in a state, the hidden truth value of $L$ can be observed.

The semantics of a contingent planning problem in PDDL include two main additions compared to classical problems: in the domain description there are actions that have an observe tag and no effects, and in the problem there is at least one $Oneof()$ formula (see Figure 2.1.).

2.1.2 Solutions and Solution Forms

Planning under uncertainty requires a model for the incomplete knowledge of the agent, described as a belief state [Bonet and Geffner, 2000]. In a given situation, the belief state is the set of states deemed possible by the agent. These models are logical, rather than probabilistic, as the beliefs are represented by a set of states, not a probability distribution. In this setting, plans or policies are evaluated by the cost in the worst case rather than their expected cost.

A planning task with uncertainty can be thought of similarly as the classical task, but with belief states instead of states. A planning task can be
Figure 2.1: Contingent problems in PDDL.

seen as a path-finding problem in a directed graph where the nodes are belief states. The applicable actions in a belief state $b$ are the ones that have preconditions that are true in any of the possible states, and for an action $a$ the transition function maps belief state $b$ to another belief state $b_a$:

$$b_a = \{s'|\exists s \in b, s' = f(s,a)\}.$$

If the belief state for the agent is $b$ and an observation $o$ is obtained after applying the action $a$, the new belief state $b^o_a$ is given by the states in $b_a$ that are compatible with $o$:

$$b^o_a = \{s|s \in b_a, o \in O(s,a)\}.$$

The strong solution or simply solution of a planning problem with sensing is a choice of actions that ensures that all the resulting executions reach a goal belief (a belief state where all possible states are goal states) in a finite number of steps. When a solution plan does not exist for all the states of the initial belief, a weak solution is a plan that reaches a belief state that contains at least one goal state.

Searching in belief space is a problem exponentially larger than searching for solution plans in the classical setting, as the number of possible belief states is exponential in the number of states. Offline planning considers the problem of generating a complete policy (in this deterministic case, a plan tree), while online methods only select the actions to do next and replan after every observation. Offline methods are more complex, as they need to
generate the whole plan tree. Here, our main focus is on full offline contingent planning.

2.2 Current Approaches and Formulations

Contingent planning has been considered one of the most challenging problems in automated planning [Haslum and Jonsson, 1999]. One of the common approaches to contingent planning is to transform it into a non-deterministic (AND/OR) search problem in belief space. The main computational challenges is the belief representation employed in these methods, as using belief states themselves is impractical due to their exponential size. To address this, [Bertoli et al., 2001] proposed the use of binary decision diagrams (BDDs) to represent belief states in their planner MBP. Later, [Bryce, 2006] used BDDs to represent literals and actions in the planning graph for computing heuristics used in the contingent planner POND. The BDD representation is more compact than the belief state itself, but still very large and sensitive to the order of variables, which is why these approaches were not able to scale well.

A quite different approach is represent belief states implicitly through the action sequences that led to them from the initial belief state. This representation was employed within a heuristic forward search scheme in the planners Conformant-FF and Contingent-FF [Hoffmann and Brafman, 2006, Hoffmann and Brafman, 2005]. The representation has the advantage of requiring very little memory compared to previous methods. The trade-off is that these planners incur a lot of repeated computations.

One of the most successful approaches was introduced in the CLG planner by [Albore et al., 2009]. They used a translation-based method, mapping the contingent problem into a non-deterministic problem in state space. They introduced the concept of contingent width and showed that a polynomial translation $X(P)$ is possible if the contingent problem $P$ has bounded contingent width. The non-deterministic, fully observable problem $X(P)$ is solved by using a relaxation $X^+(P)$ that is a classical planning problem. The CLG planner combines the execution model $X(P)$ and a heuristic model $H(P)$ and selects the actions in closed-loop fashion, invoking the planner recursively on the resulting states.

Finally, the planners CNFct and DNFct use the AND/OR forward search algorithm PrAO [To et al., 2011], going back to the idea of heuristic search in belief state. The planners rely on DNF and CNF representation of belief states and a heuristic function based on the number of satisfied subgoals and the number of known literals in the belief state. Their representation of
belief states is quite efficient, and these planners rival CLG in solving offline contingent planning problems.
Chapter 3

CONTINGENT PLANNING AS SAT

We have seen that standard approaches to contingent planning search for the goal in belief space. More recently, translation-based approaches have been developed where contingent problems are converted into fully observable, non-deterministic problems, and even classical problems [Palacios et al., 2014]. These translations are not always compact, and in the worst case exponential in the number of initial states. One problem with all these approaches, even the compact translations, is that the resulting planning problems are not easy to solve, as the various heuristics developed do not reason very well about multiple branches.

The motivation for trying a SAT approach to contingent planning is to exploit the parallelism that result from multiple branches and reason about different branches in parallel too. The ideas in the formulation are not new, but there is currently no contingent planner using this approach. Even though the SAT approach is likely to be slower than the current best planners, we expect to find plans with better quality, i.e. fewer actions in the full contingent plan. In fact, the SAT formulation ensures that the solution found will minimize the length of the longest branch, minimizing the number of actions in the "worst case".

3.1 Basic encoding of contingent problem

As with the classical planning problem, the encoding of the contingent problem $P$ with planning horizon $n$ is a CNF formula $C(P, n)$. We denote the sub-problems of the contingent problem with $j = 1, ..., J$, i.e. the initial situation is one of $J$ possible states. The CNF formula $C(P, n)$ includes propositions
$p_j^0, p_j^1, \ldots, p_j^n$ for each atom $p \in F$ and propositions $a_j^0, a_j^1, \ldots, a_j^{n-1}$ for each action $a \in O$. In addition, there are a set of atoms $D(j, k)_0, D(j, k)_1, \ldots D(j, k)_n$ describing the interaction between subproblems $j$ and $k$, which we call accessibility relations. These atoms encode the knowledge of the true state of the world, specifically whether we can tell two subproblems $j, k$ apart at time $t$. In the initial situation all $D(j, k)_0$ is false, meaning that we cannot tell apart one possible initial state from another. These atoms become true if we observe a token that is true in one subproblem and false in the other one.

The basic CNF encoding $C(P, n)$ consists of the set of clauses:

For each subproblem $j = 1, \ldots, J$ :

**Init:** $p_j^0$ for each $p^j \in I^j$, $\neg q_j^0$ for each $q_j^i \in F$ such that $q_j^i \notin I^j$

**Goal:** $p_j^n$ for each $p \in G$,

**Actions:** For $i = 0, 1, \ldots, n - 1$ and each action $a \in A \cup O$:

- $a_i^j \supset p_i^j$ for each $p \in \text{Pre}(a)$
- $a_i^j \supset p_{i+1}^j$ for each $p \in \text{Add}(a)$
- $a_i^j \supset \neg p_{i+1}^j$ for each $p \in \text{Del}(a)$

**Persistence:** For $i = 0, 1, \ldots, n - 1$ and each fluent $p \in F$ where $O(p^+) \text{ and } O(p^-)$ are the actions that add and delete $p$ respectively,

- $p_i^j \land \bigwedge_{a \in O(p^-)} \neg a_i^j \supset p_{i+1}^j$
- $\neg p_i^j \land \bigwedge_{a \in O(p^+)} \neg a_i^j \supset \neg p_{i+1}^j$

**Seriality:** $\neg (a_i^j \land a_i'^j)$, for each $i = 0, \ldots, n - 1$, if $a \neq a'$

For every pair of subproblems $(j, k)$, where $j \neq k$ :

**Unknown initial state:** $\neg D(j, k)_0$

**Sensing:** For $i = 0, 1, \ldots, n - 1$ and each sensing action $a \in O$ where $o$ is the observation token of $a$:

- $a_i^j \land a_i^j \land \neg a_i^k \supset D(j, k)_{i+1}$
- $a_i^j \land \neg a_i^j \land a_i^k \supset D(j, k)_{i+1}$
- $\neg D(j, k)_i \land a_i^j \land a_i^j \land a_i^k \supset \neg D(j, k)_{i+1}$
- $\neg D(j, k)_i \land a_i^j \land \neg a_i^j \land \neg a_i^k \supset \neg D(j, k)_{i+1}$
- $D(j, k)_i \supset D(j, k)_{i+1}$

**Restrictions:** $\neg D(j, k)_i \land a_i^j \supset a_i^k$ for each action $a \in O$
The clauses from classical planning are included for each subproblem \( j \). The clauses for seriality and action preconditions are added for both normal and sensing actions. The additional clauses describe the interactions between every pair of subproblems. Initially the accessibility relations \( D(j, k) \) are false. As long as the atom \( D(j, k) \) is false, actions have to be the same in both subproblems. Once there is an observation that differentiates two subproblems, \( D(j, k) \) becomes true.

The persistence of the accessibility relations cannot be encoded the same way as the persistence of fluents, because the actions that ”add” \( D(j, k) \) are not known \textit{a priori}. To encode persistence, clauses are added that explicitly say, if \( D(j, k) \) is false, and an action is done but the observation is the same for both subproblems (or there is no observation), \( D(j, k) \) remains false. Once \( D(j, k) \) becomes true, it remains true.

The above encoding works for small problems but does not scale up well. Some additions and modifications help with scaling up.

**Parallel actions.** The success of the SAT approach to solve planning problems was largely due to the development of parallel plan encodings. Here, the same cannot be done for regular actions due to the interactions between subproblems. However, sensing actions do not change the value of state variables, only the \( D(j, k) \) atoms. The encoding can be modified the following way:

- At each time step a sensing action \( a_t \) can be done which can make the atom \( D(j, k)_t \) true.
- At the same time step, a regular action \( a_t \) can be done which effects the value of state variables \( p_{t+1} \).

This way the horizon where the solution is found can be decreased.

**Action splitting.** In many cases, actions take two or more parameters. Consider the operator \( Move(x_1, x_2, x_3) \) in the Blocks domain, which describes moving block \( x_1 \) from on top of block \( x_2 \) onto block \( x_3 \). The planning actions are obtained by grounding operator descriptors over object symbols, i.e. assigning the variables \( x_1, x_2, x_3 \) to objects. If the number of blocks in the problem is \( B \), the number of actions added for each time step is \( \binom{B}{3} \), i.e. cubic in the number of blocks. We can split the operator to predicates \( Move[k](x) \), which represent the k-th argument of the operator. The conjunct \( Move[1](x_1) \land Move[2](x_2) \land Move[3](x_3) \) represents \( Move(x_1, x_2, x_3) \). This way the number of variables is reduced to only \( 3B \). In addition, the clauses describing per-
sistence, action preconditions and effects become shorter and more expressive. For example the action \textit{Move}(A, B, C) adds the predicate \textit{Clear}(B). In the new formulation the same predicate is added by \textit{Move}[2](B), meaning that any action that removes one block from \textit{B} adds \textit{Clear}(B). Now the Add list is reduced to one variable instead of the variables for all the actions where \textit{B} is the second argument.

This split representation of actions has been subject of a significant body of work in SAT-based planning since the early days of the approach (e.g. [Ernst et al., 1997], [Kautz et al., 1992]). The formulation has been abandoned because of the parallel planning approach. The problem is interference: if at time \textit{t} both actions \textit{Move}(A, B, C) and \textit{Move}(D, E, F) are done, the representation returns true for \textit{Move}[1](A), \textit{Move}[2](B), \textit{Move}[3](C), \textit{Move}[1](D), \textit{Move}[2](E), \textit{Move}[3](F). This can imply any of the 8 actions with these parameters, e.g. \textit{Move}(A, E, C), \textit{Move}(A, B, F) and so on.

**Observation tokens.** As it was already mentioned, the sensing actions that change accessibility relation atoms \textit{D}(j, k) cannot be known in advance. Consider an example from the Localize domain. In this domain, the agent is initially in one of several cells, and the goal is to get to a specific cell. The agent can sense whether an adjacent cell is free, i.e. the action sense-left reveals if there is a cell to the left or a wall. Because the agent moves around, we cannot initially know which sensing action will reveal information about which initial location we started from.

However, in some domains the observation tokens coincide with the unknown initial fluents, and do not change over time. For example, consider the domain Doors where the agent has to move through a grid from a known initial cell to a goal cell. To get to the goal, the agent has to go through a door which can be in one of \textit{n} cells. The sensing action reveals if there is a door in an adjacent cell. In this case, the sensing action reveals the truth value of the unknown initial state and the sensing actions that add \textit{D}(j, k) can be identified in advance.

For domains with unchanging observation tokens, the persistence clauses for the relation atoms \textit{D}(j, k) can be written similarly as they are for the other atoms. Additionally, there is no need in this case to restrict sensing actions from being done in parallel. If the preconditions hold for several sensing actions, they can all be done at the same time step. This can make a difference in certain domains where there is a large number of subproblems but sensing actions can be done in parallel. An example is the domain Ebtcs, where the task is to find a bomb that can
be in one of \( n \) packages and dispose of it. The sensing action has no
precondition and the observation tokens are the same as the unknown
atoms in the initial state. Instead of horizon \( n \) the problem is solved
with horizon 1, because all the branches can be identified in the first
time step.

The translations were solved by MiniSat 2.2.0 core version starting from
horizon \( n = 1 \) and increased one by one until a solution was found.

### 3.2 Experimental Evaluation

**Planners.** We compare the translation \( C(P) \) with planners DNFct, CLG
and contingent-FF on contingent benchmarks. The time reported does
not include the parsing and encoding but includes all runs of MiniSat.
The experiments were restricted to use maximum 4 GB of memory and
two hours of runtime.

**Benchmarks.** We tested the planners on contingent benchmarks collected
from the distributions of CLG, Contingent-FF and DNFct.

**Summary of experimental results.** Table 3.1 reports the experimental
results in the form \( s/d \ (t) \), where \( s \), \( d \) and \( t \) denote the number of
actions in the solution, the depth of the solution tree and the overall
execution time respectively. We are mostly interested in the quality of
the solutions, the depth and size of the plans.

As expected, our approach finds the solutions with smallest depth. The
depth of the solution found by the SAT approach is by definition the
minimal depth possible. The solutions minimize the number of actions
to the goal in the "worst case".

The SAT approach does exceptionally well in the Block and Wumpus
domains in terms of the quality of the solutions. In the Block domain,
CLG finds plans not much worse than ours, but DNFct fails to solve the
larger problems. In Wumpus, the solutions found by our approach are
far better than DNFct or CLG, except for the largest problem, where
only CLG finds a solution.

In the Btcs domain, the quality of the plan and the run time of our
approach is about the same as DNFct. The reason why our translation
is so fast in this domain is mentioned earlier: the structure of this
domain allows our encoding to find the plan on the first try due to
parallel sensing actions.
In the domains Localize and Grid, the size of the solution is large compared to the solution found by at least one other planner. The reason for this is that the different branches are not optimized. Finally, in the Doors and Unix domains, the SAT approach does not scale up as well as other planners.

All in all, the encoding of the full contingent problem to SAT does better than Contingent FF, but does not reach the performance of the current best planners DNFct and CLG. The weakness of the full encoding is finding solutions for problems with many possible initial states. Doors-7 has 343 possible initial states, which means that for every time step 58 thousand new accessibility relation variables are added, and even more clauses.

The strength of the approach is best demonstrated by the Wumpus and the Block domains. In these problems, the different subproblems require more or less the same number of actions. This allows the full encoding to reason about subproblems in parallel, and does not ”waste resources” searching for solutions on branches that could be terminated much faster.
Table 3.1: Results for contingent planning domains.

<table>
<thead>
<tr>
<th>Domain</th>
<th>C(P)</th>
<th>DNFct</th>
<th>CLG</th>
<th>ContFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block-3</td>
<td>5/4 (0.012)</td>
<td>5/4 (0.08)</td>
<td>6/4 (0.02)</td>
<td>5/4 (0.01)</td>
</tr>
<tr>
<td>Block-7</td>
<td>41/9 (1.28)</td>
<td>59/28 (12.6)</td>
<td>55/9 (4.67)</td>
<td>49/12 (0.47)</td>
</tr>
<tr>
<td>Block-11</td>
<td>101/16 (26)</td>
<td>OM</td>
<td>115/18 (36)</td>
<td>TO</td>
</tr>
<tr>
<td>Block-15</td>
<td>144/22 (1935)</td>
<td>OM</td>
<td>157/22 (97)</td>
<td>TO</td>
</tr>
<tr>
<td>Wumpus-5</td>
<td>45/12 (1.04)</td>
<td>1083/34 (2.47)</td>
<td>745/41 (1.13)</td>
<td>E</td>
</tr>
<tr>
<td>Wumpus-7</td>
<td>189/18 (114)</td>
<td>29k/74 (55)</td>
<td>6.5k/57 (10)</td>
<td>E</td>
</tr>
<tr>
<td>Wumpus-10</td>
<td>OM</td>
<td>280k/100 (2954)</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>Localize-5</td>
<td>193/22 (1.44)</td>
<td>49/23 (0.57)</td>
<td>112/24 (1.86)</td>
<td>53/53 (42)</td>
</tr>
<tr>
<td>Localize-7</td>
<td>470/35 (17)</td>
<td>80/36 (0.72)</td>
<td>231/37 (6.89)</td>
<td>MC</td>
</tr>
<tr>
<td>Localize-9</td>
<td>846/41 (74)</td>
<td>113/53 (0.84)</td>
<td>386/50 (21.2)</td>
<td>MC</td>
</tr>
<tr>
<td>Localize-11</td>
<td>1626/50 (257)</td>
<td>144/62 (1.2)</td>
<td>577/63 (64)</td>
<td>MC</td>
</tr>
<tr>
<td>Btcs-50</td>
<td>99/50 (0.4)</td>
<td>101/50 (0.13)</td>
<td>121/52 (0.02)</td>
<td>99/50 (11.96)</td>
</tr>
<tr>
<td>Btcs-70</td>
<td>139/70 (1.8)</td>
<td>139/70 (2.77)</td>
<td>140/140 (13.7)</td>
<td>139/70 (123)</td>
</tr>
<tr>
<td>Btcs-90</td>
<td>179/90 (5.2)</td>
<td>179/90 (5.4)</td>
<td>180/180 (41)</td>
<td>TO</td>
</tr>
<tr>
<td>Unix-2</td>
<td>48/37 (80)</td>
<td>48/37 (0.78)</td>
<td>50/39 (0.64)</td>
<td>48/37 (0.13)</td>
</tr>
<tr>
<td>Unix-3</td>
<td>MO</td>
<td>111/84 (2.02)</td>
<td>113/86 (5.88)</td>
<td>111/84 (3.84)</td>
</tr>
<tr>
<td>Grid-3</td>
<td>114/21 (21)</td>
<td>313/51 (1.97)</td>
<td>114/30 (0.94)</td>
<td>23/23 (0.06)</td>
</tr>
<tr>
<td>Grid-4</td>
<td>429/24 (385)</td>
<td>982/71 (2.9)</td>
<td>212/40 (1.87)</td>
<td>49/49 (0.14)</td>
</tr>
<tr>
<td>Grid-5</td>
<td>537/30 (974)</td>
<td>1337/81 (2.9)</td>
<td>872/51 (4.64)</td>
<td>46/46 (0.15)</td>
</tr>
<tr>
<td>Doors-5</td>
<td>159/16 (21)</td>
<td>146/18 (0.05)</td>
<td>144/16 (0.04)</td>
<td>E</td>
</tr>
<tr>
<td>Doors-7</td>
<td>MO</td>
<td>2193/53 (4.28)</td>
<td>2153/51 (7.6)</td>
<td>E</td>
</tr>
</tbody>
</table>


Chapter 4

DISCUSSION

In this thesis, we developed a SAT-based approach for partial observable planning. The encoding of classical planning as SAT was extended to the planning problem with unknown initial state and sensing. The encoding has proved to be useful for certain contingent planning problems, but scaling up is still a challenge.

There are a number of ways the encoding could still be improved. The current basic version does not employ almost any of the advances that made SAT so successful in the classical planning domain. The use of lower bounds for the horizon would decrease the time spent on searching for solutions by the SAT solver. The persistence axioms for the fluents could be reformulated with NO-OP actions, and there is probably more efficient ways to encode the persistence of the accessibility relations. The automatic extraction of invariants and mutexes in preprocessing could also be used for better performance.

Scaling up to problems with very many possible initial states could be done by only considering a certain set of subproblems and finding a plan only for them. If a solution is found, it is carried out until the first observation. When there is an observation, it is integrated and the planning is done again. Continuing this loop, the planner could find solutions with many subproblems. This modification of the approach could also be used for online planning.

In conclusion, the SAT-based approach of planning under partial observability has potential to improve on the plans found by the current best contingent planners. The current formulation is only scratching the surface of what can be achieved with SAT-based approaches in partially observable planning.
Bibliography


26