Planning with Partial Observability by SAT

Saurabh Fadnis[0000−0001−9307−281X] and Jussi Rintanen[0000−0001−5983−0074]

Aalto University, Department of Computer Science

Abstract. Geffner & Geffner (2018) have shown that finding plans by reduction to SAT is not limited to classical planning, but is competitive also for fully observable non-deterministic planning. This work extends these ideas to planning with partial observability. Specifically, we handle partial observability by requiring that during the execution of a plan, the same actions have to be taken in all indistinguishable circumstances. We demonstrate that encoding this condition directly leads to far better scalability than an explicit encoding of observations-to-actions mapping, for high numbers of observations.

1 Introduction

Geffner and Geffner [7] have shown how SAT yields an effective method for solving non-deterministic fully observable (conditional) planning problems. This is the first time SAT has been directly used for solving a broad class of problems outside deterministic planning, by only a polynomial number of SAT calls in the size of the plan being constructed. This approach is in strong contrast with earlier constraint-based approaches, which have required formalisms stronger than SAT, for example Σ_2^p-hard SSAT [11, 12] or QBF [16], or separate calls to SAT solvers for plan generation and verification [6] (again going up to Σ_2^p) and using SAT as a sub procedure of an otherwise exponential search algorithm, even when restricting to plans of a polynomial size.

We view as the core idea in Geffner & Geffner's work that contingent planning is in NP whenever *all executions of a plan* have a representation that has polynomial size. Our work demonstrates that *the same applies also to the far harder problem of planning with partial observability.*

1.1 Background

Finding plans in classical planning, with one initial state and deterministic actions, can be represented as propositional formulas of a size that is linear in the number of actions in a plan. The formula with length parameter n is satisfiable if and only if there is a sequence of n actions that reaches a goal state from the given initial state. For plans with a polynomial length, the NP-complete problem of finding them can be done with a SAT solver [8, 9].

When a plan can have multiple alternative executions, planning is harder. Conditional planning, with branching program-like plans, is in the complexity class Σ_2^p for poly-sized plans, and hence – in general – believed to be outside

the reach of the NP-complete SAT problem, and to require the more powerful framework of quantified Boolean formulas (QBF) [15, 21, 16].

The idea that Σ_2^p complexity was somehow inherent to practically significant conditional planning was broken by Geffner & Geffner [7] who showed that – under full observability – the two separate NP computations for plan search and for plan verification collapse to a single NP computation, if execution graphs and plans have the same form. Plans are (possibly cyclic) graphs, with non-terminal nodes associated with actions, and executions are viewed as paths in the graph. The states in a node are represented by the literals that are true in it.

Given a problem instance of size m and an $n \geqslant 0$, Geffner & Geffner generate a propositional formula so that any satisfying assignment represents a graph that has n nodes and represents a conditional plan and all of its executions. The size of the formula is polynomial in n and m. The formula leaves the structure of the plan open, and it is the SAT solver that chooses the (positive) literals in each node, the action in each node, and the outgoing arcs for each node.

1.2 Contributions

Our contributions are as follows. We present the first *SAT-based* encodings of *succinctly* represented (state variable based) planning problems under *partial observability*. Earlier works either use more powerful (and less scalable) formalisms than SAT such as QBF or effectively simulate such [6, 16], use a non-succinct enumerative representation [5], or cover full observability only [7, 14].

We use 3-valued (partial) execution graphs to represent all possible executions of a plan. Earlier 3-valued representations [1] have low complexity and are scalable, but lead to incompleteness, as most state sets do not have a 3-valued representation. We will show how *case analysis* on state variables marked *unknown* makes the 3-valued approach *complete*, that is, being able to represent a plan whenever one exists, by selectively eliminating partiality.

Finally, we show how an *implicit* representation of *small memory* plans can make the approach better scalable. Instead of accurately keeping track of the belief state, only an abstraction of the belief state is maintained, as a state as in a finite automaton, in order to distinguish between different execution histories. Plans are mappings from memory states and observations to next actions and next memory states. As the number of observables increases, the sizes of these mappings increase exponentially, making an explicit representation impractical: a smaller and smaller fraction of all observation combinations actually occur in any execution of a plan. A main result is that replacing an explicit encoding of small memory plans (as in [5]) by an implicit encoding can lead to substantial scalability improvements. We can still guarantee *the existence* of these mappings, and they can be easily extracted from satisfying assignments.

2 Planning with Partial Observability

A key idea in Geffner & Geffner's representation of branching plans as graphs is that each path can represent multiple possible executions as the nodes corre-

spond to *partial* states which determine the values of some state variables only. A partial state is essentially a representation of a *set* of states. For example, with state variables a, b, c and d, the partial state represented by the partial valuation $\{a = 1, b = 0\}$ corresponds to the set of those four states that match the partial valuation on a and b, and assign any value combination to c and d.

Example 1. In this navigation problem moving into a wall is not allowed, and hence it must be possible to detect which cells are next to a wall (observations $N = y_4$, $S = y_0$, $E = x_4$, $W = x_0$, indicating which wall(s) the robot is next to).

Considering the non-wall locations as possible starting locations, the plan that first moves to north until the north wall is encountered, and then moves west until the NW corner is encountered, is depicted on the right in Figure 1.

One graph that represents all executions of this plan is partly given in Figure 1. The rest of graph would be similar and repeats the "move west" action when north wall is observed until the NW corner location is reached.

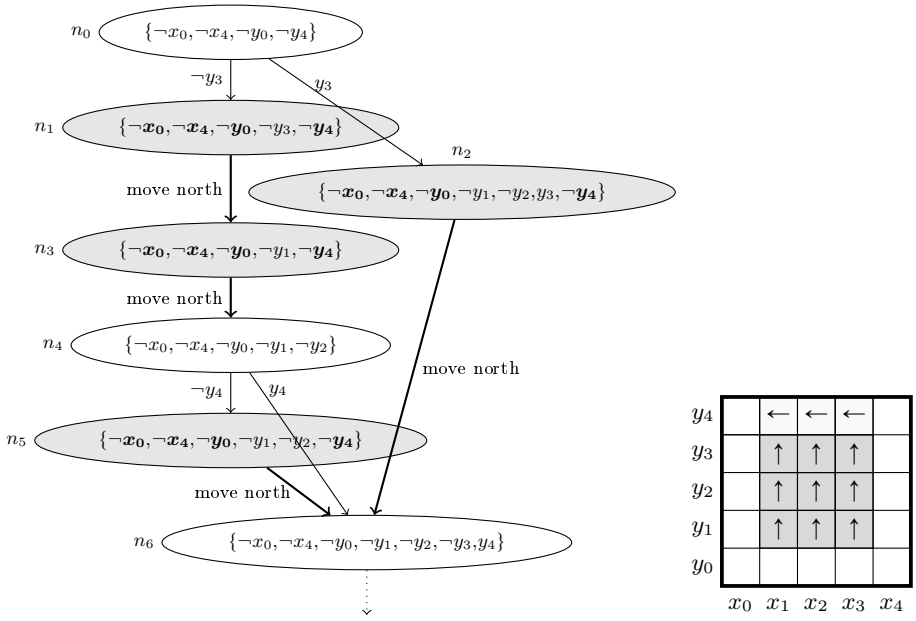

Fig. 1. Execution of a grid navigation problem (part)

In this example, the topmost branching (node n_0 in the graph) is on the variable y_3. We call such branching *Case Analysis* and it is discussed in the next subsection. The thing to note here is that y_3 is not observable, and in general the branching is not directly related to observations, but to the different execution paths of a given plan. Despite the branching being on an unobservable variable,

the graph is still a faithful representation of all executions of the plan. The key property here is that if an action is taken, then it is the same action that is taken in all mutually indistinguishable states with respect to observations: move north in all (partial) states that are not next to any wall (nodes n_1,n_2,n_3 and n_5 in figure 1), and move west whenever next to the north wall.

In this example there is no need to remember any of the previous belief states. In cases where this memory is needed, the same action is taken in all mutually indistinguishable states with respect to observations and *memory*. If the plan includes memory, more fine-grained choice of actions is possible. We consider small-memory plans [10, 13, 5], which means that at execution time there are only a small number of possible "memories" in the execution mechanism, and they – together with the observations – determine the next action and memory. We represent the different memories as integers from 0 to some small $M_{\max}$.

2.1 Case Analysis

For 3-valued partial representations, if some state variable values are *unknown*, the determination of which observations can be made, or what are the effects of actions with conditional *if-then* effects becomes problematic. This necessitates *case analysis* on the values of a state variable x with an unknown value: a node in the plan will have two successors, with x true in one and x false in the other.

Example 2. Consider a node n_0 with literals $\{p, \neg q\}$, and for which the plan assigns the action $a_1 = (\neg q, \{\text{IF } r \text{ THEN } \{p, q\} \text{ ELSE } \{\neg p, \neg q\}\})$. Since the value of r is not known in n_0, the values of p and q in the successor node cannot be determined based on the information in n_0.

If some of these values need to be known, a case-analysis on r is performed in n_0 instead, before executing the action. Now n_0 has two successor nodes, one with literals $\{p, \neg q, r\}$ and the other $\{p, \neg q, \neg r\}$. In both nodes the action a_1 is taken. The successor of the first node has literals $\{p, q, r\}$ and the successor of the second node has $\{\neg p, \neg q, \neg r\}$.

This example shows that we can always make enough of the state explicit so that values of sufficiently many observables can be determined so that actions can be chosen, and literals in successor nodes can be determined.

3 Formal Definition of Planning

We view *states* as valuations of Boolean state variables. If X is the set of state variables, then a state $s : X \to \{0, 1\}$ assigns a value to every state variable in X. By identifying states and Boolean valuations, we can directly use definitions from the Propositional Logic to talk about states. For example, we can say that a formula ϕ is *true in a state* s if $s \models \phi$, that is, this formula ϕ is true in the valuation s. We denote the set of all states (over some fixed set X) by $\mathbb{S}$.

Next, problem instances are formally defined. *Atomic effects* are of the form $x := 0$ or $x := 1$, where x is a state variable. A *conditional effect* has the form

IF ϕ THEN e, where ϕ is a formula over X, and e is a set of atomic effects. An action is associated with a set of atomic and conditional effects. For simplicity, we don't discuss non-deterministic actions in this work.

Definition 1 (Problem Instance). *A* problem instance in planning *is a tuple* $\Pi = \langle X, A, I, G, O \rangle$ *where*

- X *is a set of state variables,*
- A *is a set of actions* (p, e)*, where*
  - p *is a formula over* X*, and*
  - e *is a set of effects,*
- *the* initial states *are represented by a formula* I *over* X*,*
- *the* goal states *are represented by a formula* G *over* X*, and*
- *the* observations *are a set* O *of formulas over* X*.*

4 Execution Graphs

Our introduction of *case analysis* nodes in the execution graphs allows to complete, on demand, the approximated information sufficiently that every problem instance has a solution as a 3-valued execution graph.

Below we show a basic formalization of (2-valued) execution graphs without partiality, and then provide a proof sketch of the *completeness* of our approach by showing that any execution graph can be represented in terms of a (sometimes much more compact) 3-valued execution graph. The key result maps every non-approximate 2-valued execution graph to a 3-valued execution graph that includes case analysis nodes so that all executions in the former are represented also in the latter. Additionally, the approach is *sound*, so the 3-valued representation only represents solutions that are representable in the 2-valued framework.

Definition 2 (Execution graphs). *Given an instance* $\Pi = \langle X, A, I, G, O \rangle$*, an* execution graph *is* $G = \langle N, E, P, S, M, B \rangle$ *where*

- N *is a finite set of* nodes*,*
- $E \subseteq N \times N$ *are the* arcs *of the graph,*
- *the (partial) function* $P : N \to A$ *maps non-terminal nodes to actions,*
- *the function* $S : N \to \mathbb{S}$ *assigns a state to every node.*
- $M = \{0, \ldots, M_{max}\}$ *is the set of* memory states *with* $M_{max} \geqslant 1$*, and*
- *the function* $B : N \to M$ *assigns a memory state to every node.*

The memory state with the highest index $M_{\max}$ is for indicating that a goal state has been reached.

Definition 3 (Solutions as execution graphs). *An execution graph* $G = \langle N, E, P, S, M, B \rangle$ *is a* solution *to* $\Pi = \langle X, A, I, G, O \rangle$ *if the following hold.*

1. *The graph* $\langle N, E \rangle$ *is acyclic (it has no directed cycles.)*
2. $\{s \in \mathbb{S} \mid s \models I\} = \{S(n) \mid n \in N, n \text{ has no parents}\}$
3. $S(n) \models G$ *for every* $n \in N$ *such that* $B(n) = M_{max}$

4. $B(n) = 0$ *for every node* n *with no parent*
5. $B(n) = M_{max}$ *for every node* n *that has no successors*
6. $S(n) \models p$ *if* $P(n) = (p, e)$
7. *For* $x \in X$ *and nodes* n' *with a parent,* $S(n') \models x$ *iff there is* $n \in N$ *such that* $(n, n') \in E$ *and* $P(n) = (p, e)$*, and either*
   - $(x := 1) \in e$,
   - *(IF* ϕ *THEN* $x := 1) \in e$ *and* $S(n) \models \phi$*, or*
   - $S(n) \models x$ *and* $(x := 0) \notin e$ *and* $S(n) \not\models \phi$ *for all (IF* ϕ *THEN* $x := 0) \in e$.
8. $S(n') \models \neg x$ *analogously*
9. *For all* $n_1, n_2 \in N$*, if* $P(n_1) \neq P(n_2)$*, then either* $B(n_1) \neq B(n_2)$ *or for some* $\omega \in O$*,* $S(n_1) \models \omega$ *iff* $S(n_2) \not\models \omega$.
10. *For all* $(n_1, n_1'), (n_2, n_2') \in E$*, if* $B(n_1') \neq B(n_2')$*, then either* $B(n_1) \neq B(n_2)$ *or for some* $\omega \in O$*,* $S(n_1) \models \omega$ *iff* $S(n_2) \not\models \omega$.

Condition 7 guarantees that the changes in the values of state variables between a node and its successor exactly correspond to the changes caused by the action in that node.

Conditions 9 and 10 express the *distinguishability* between two situations during the execution of a plan: if two situations cannot be distinguished through the memory or the observations, then the same actions have to be taken in both, and the next memory states have to be the same.

An execution graph represents every possible execution $s_0, \ldots, s_n$ explicitly, and different executions (state sequences) are represented by different paths in the graph. Hence, for any problem instance for which an exponential number of different states has to be considered, the size of the execution graph is exponential. Next we define *partial execution graphs* that can represent large numbers of states and executions more compactly..

5 Partial Execution Graphs

Partial execution graphs can be exponentially smaller than execution graphs if many executions share the same structure. The nodes in partial execution graphs are labelled with *partial states* which only determine the values of a subset of state variables. One partial state represents all states that assign the same values to the represented state variables. This same idea has been used by Geffner & Geffner in their work on planning with full observability [7].

Definition 4. *A* partial state *is a partial function* $z : X \to \{0, 1\}$.

Toggle the table of contents We denote the set of all partial states by $\mathbb{S}_p$. Clearly, $\mathbb{S} \subset \mathbb{S}_p$. A partial state z represents all states that do not disagree on the value of any state variable. That is, z represents $\{s \in \mathbb{S} \mid s(x) = z(x)$ or $z(x)$ is not defined, for all $x \in X\}$.

Definition 5. *A* partial execution graph $G = \langle N, E, P, C, S, M, B \rangle$ *for a problem instance* $\Pi = \langle X, A, I, G, O \rangle$ *consists of*

- *a finite set* N *of* nodes,
- *a set* $E \subseteq N \times N$ *of* arcs,
- *a (partial) function* $P : N \to A$ *that assigns an action to some of the nodes,*
- *a (partial) function* $C : N \to X$ *that assigns a state variable to some of the nodes (for case analysis),*
- *a function* $S : N \to \mathbb{S}_p$ *that assigns a partial state to every node.*
- $M = \{0, \ldots, M_{max}\}$ *is the set of* memory states *with* $M_{max} \geqslant 1$, *and*
- *the function* $B : N \to M$ *assigns a memory state to every node.*

The function C indicates in which nodes a case analysis is performed. If $C(n) = x$, then node n has two successor states n_1 and n_2 so that $S(n_1) \models x$ and $S(n_2) \models \neg x$. From now on, we assume all formulas to be in Negation Normal Form (NNF)[1]. We define truth of a formula in a partial state as follows.

1. $s \models_3 x$ if $s(x) = 1$, and $s \models_3 \neg x$ if $s(x) = 0$
2. $s \models_3 \alpha \wedge \beta$ iff $s \models_3 \alpha$ and $s \models_3 \beta$
3. $s \models_3 \alpha \vee \beta$ iff $s \models_3 \alpha$ or $s \models_3 \beta$

Here it is critical that we do not define a formula $\neg\phi$ to be true if ϕ is not true (as in 2-valued logic), because the value of ϕ might be undetermined due to the partiality of the state.

Define cubeof(s) that maps a partial state s to a corresponding conjunction of literals $\text{cubeof}(s) = \bigwedge(\{x \in X \mid s(x) = 1\} \cup \{\neg x \mid x \in X, s(x) = 0\})$.

Definition 6. *A partial execution graph* $G = \langle N, E, P, C, S, M, B\rangle$ *is a* solution *to* $\Pi = \langle X, A, I, G, O\rangle$ *if the following hold.*

1. *The graph* $\langle N, E\rangle$ *is acyclic (it has no directed cycles.)*
2. $I \models_3 \delta_1 \vee \cdots \vee \delta_m$, *where* $n_1, \ldots, n_m$ *are the nodes with no parents, and* $\delta_i = \text{cubeof}(S(n_i))$ *for every* $i \in \{1, \ldots, m\}$
3. $S(n) \models_3 G$ *for every* $n \in N$ *such that* $B(n) = M_{max}$
4. $B(n) = 0$ *for every* $n \in N$ *with no parent*
5. $B(n) = M_{max}$ *for every* $n \in N$ *that has no successors*
6. $S(n) \models_3 p$ *if* $P(n) = (p, e)$, *for every* $n \in N$
7. *For every* $n \in N$, *exactly one of the following holds.*
   - $C(n)$ *is defined*
   - $P(n)$ *is defined*
   - $B(n) = M_{max}$
8. *If* $C(n) = x$ *for some* $x \in X$, *then* n *has exactly two successors,* n_1 *and* n_2, *and* $S(n_1) \models_3 x$ *and* $S(n_2) \models_3 \neg x$.
9. *If* $C(n)$ *is defined and* $(n, n') \in E$, *then* $B(n) = B(n')$.
10. *If* $C(n) = x$ *and* $s = S(n)$, *then* $s(x)$ *is not defined.*
11. *If* $S(n') \models_3 x$ *for* $n' \in N$ *that has a parent* $n \in N$, *then either*
    (a) $(n, n') \in E$ *and* $P(n) = (p, e)$ *and*
    - $(x := 1) \in e$,

[1] In NNF, a formula contains only connectives $\vee$, $\wedge$ and $\neg$, and all negations $\neg$ are directly in front of an atomic proposition.

- *(IF ϕ THEN $x := 1) \in e$ and $S(n) \models_3 \phi$, or*
- *$S(n) \models_3 x$ and $(x := 0) \notin e$ and $S(n) \not\models_3 \phi$ for all (IF ϕ THEN $x := 0) \in e$.*

(b) or n is a case analysis node with $C(n) = x$,

(c) or n is a case analysis node with $C(n) \neq x$ and $S(n) \models_3 x$.

12. *$S(n') \models_3 \neg x$ analogously*
13. *For all $n_1, n_2 \in N$, if $P(n_1) \neq P(n_2)$, then either $B(n_1) \neq B(n_2)$ or for some $\omega \in O$, either $S(n_1) \models_3 \omega$ and $S(n_2) \models_3 \neg\omega$, or $S(n_1) \models_3 \neg\omega$ and $S(n_2) \models_3 \omega$.*
14. *For all $n_1, n_2 \in N$ and n_1', n_2' such that $(n_1, n_1'), (n_2, n_2') \in E$, if $B(n_1') \neq B(n_2')$, then either $B(n_1) \neq B(n_2)$ or for some $\omega \in O$ either $S(n_1) \models_3 \omega$ and $S(n_2) \models_3 \neg\omega$, or $S(n_1) \models_3 \neg\omega$ and $S(n_2) \models_3 \omega$.*

A key feature of partial execution graphs, similarly to the work by Geffner & Geffner [7], is that some state variable values may be "forgotten" when going from a node to its successor. Increasing the partiality this way allows parts of a plan to be more general in being applicable for more states.

An important question about partial execution graphs is whether they can represent any solution, as expressible by an execution graph.

Proposition 1. *For every execution graph G, there is a partial execution graph G' that represents exactly the same solution.*

Proof. Sketch: The most trivial way to eliminate all partiality is to enumerate all states that satisfy the initial state formula I, and create an initial node for each of those states s. So the partial states in all initial states are (total) states. The partial execution graph in this case does not contain any case analysis nodes, and its structure is exactly the same as that of the execution graph. Stated differently, execution graphs are a special case partial execution graphs, only without case analysis nodes, and with (total) states instead of partial states.

6 Encodings of Partial Execution Graphs

Next we describe the encoding of partial execution graphs as propositional formulas. Table 1 lists the atomic propositions used in the encoding.

State variable x is true in node n if P_x^n holds and false if N_x^n holds. Otherwise its value is unknown (could be either true or false.) To refer to the truth of arbitrary formulas in a node, we define L_n^ϕ as the formula obtained from ϕ by transforming it to the Negation Normal Form (NNF) and then (for all $x \in X$) replacing subformulas $\neg x$ by N_x^n, and finally replacing subformulas x by P_x^n. Hence L_n^ϕ is true in a partial state z iff ϕ is true in all states represented by z.

In the rest of the section, the schema variables n, a, x (possibly with subscripts or other embellishments) are instantiated with all possible nodes, action names or state variable names, unless stated otherwise.

P_x^n, N_x^n	State variable x is true or false in node n, respectively.
${}^oP_\omega^n, {}^oN_\omega^n$	Observation ω or $\neg\omega$ is made in n, respectively, for $\omega \in O$.
(n, a)	Action a is applied in node n.
(n, a, n')	n' is the next node after action a is applied in node n.
(n, n')	n' is a successor node of n.
A_n	Node n is an action node
CA^n	n is a case analysis node.
CA_x^n	Case analysis is done in node n on variable x.
$\mathrm{CA}_{n,n'}^1$	n' is the successor node with $P_x^{n'}$ when case analysis is done on x in n.
$\mathrm{CA}_{n,n'}^0$	n' is the successor node with $N_x^{n'}$ when case analysis is done on x in n.
(n, m)	node n has the memory state m.
$Z_{i,j}^m$	Nodes i and j have the same memory state m.
$Z_{i,j}^\omega$	Nodes i and j are indistinguishable w.r.t. observation ω.
$Z_{i,j}$	Nodes i and j are indistinguishable w.r.t. memory state and all observations.

Table 1. Atomic propositions used in the encoding

6.1 Nodes and arcs

State variables are true, false or unknown (1). Preconditions are true (2).

$$\neg P_x^n \vee \neg N_x^n \qquad (1) \qquad (n, a) \rightarrow L_n^\phi \text{ where } \phi \text{ is the precondition of } a \qquad (2)$$

The acyclicity of the encoding is handled by instantiating all formulas referring to arcs from node n to a successor n' so that the index of n' is strictly higher than the index of n.

Node n has at least one action if and only if A_n is true (3), and A_n excludes case analysis in the same node (4). The variable (n, a) is defined by (5). Action nodes have exactly one successor node (6). [2]

$$\mathrm{A}_n \leftrightarrow \bigvee_{a \in A} (n, a) \qquad (3) \qquad \neg(\mathrm{A}_n \wedge \mathrm{CA}^n) \qquad (4)$$

$$(n, a) \leftrightarrow \bigvee_{n' \in N \setminus \{n\}} (n, a, n') \qquad (5)$$

$$\mathrm{A}_n \rightarrow \mathrm{exactly1}(\{(n, n') \mid n' \in N\}) \qquad (6)$$

Anything true in a successor node of an action node is an effect of the action or something that was true already and not made false by the preceding action.

$$(n, n') \wedge \mathrm{A}_n \wedge P_x^{n'} \rightarrow L_n^{\mathrm{Ppc}_x \vee (x \wedge \neg \mathrm{Npc}_x)} \qquad (7)$$

$$(n, n') \wedge \mathrm{A}_n \wedge N_x^{n'} \rightarrow L_n^{\mathrm{Npc}_x \vee (\neg x \wedge \neg \mathrm{Ppc}_x)} \qquad (8)$$

where $\mathrm{Ppc}_x = \chi_1 \vee \cdots \vee \chi_m$ and is the disjunction of the conditions χ_i under which action a_i makes x true, and Npc_x is the same for making x false. Each χ_i consists of the action variable (n, a_i) and additionally for conditional effects the condition under which x becomes *true* or *false*. This encodes Condition 11a.

[2] To encode the constraints *exactly-one* and *at-most-one* for $\phi_1, \ldots, \phi_k$, we use the quadratic encoding $\neg\phi_i \vee \neg\phi_j$ for all $1 \leqslant i < j \leqslant k$. Better encodings exist [18].

For the implicit encoding in Section 6.4 we have to enforce this explicitly by

$$\neg(n, a_1) \vee \neg(n, a_2) \text{ whenever } a_1 \neq a_2 \tag{9}$$

Note that (7) and (8) allow the effects of an action to be true in the successor node, but they do not have to be. This is because those values might not be needed, and the successor node is more *general* the fewer values are made explicit. This is as in the work by Geffner & Geffner [7].

6.2 Case Analysis

The following formulas encode how case analysis is done on some x so that a node has two successors, respectively with x true and false.

$$\mathrm{CA}^n \leftrightarrow \bigvee_{x \in X} \mathrm{CA}^n_x \tag{10}$$

For case analysis nodes, there is some variable to do the case analysis on, and there are two successor nodes (respectively for x and $\neg x$).

$$\bigvee_{n' \in N \setminus \{n\}} \mathrm{CA}^1_{n,n'} \rightarrow \bigvee_{x \in X} \mathrm{CA}^n_x \tag{11}$$

$$\bigvee_{n' \in N \setminus \{n\}} \mathrm{CA}^0_{n,n'} \rightarrow \bigvee_{x \in X} \mathrm{CA}^n_x \tag{12}$$

$$\mathrm{CA}^n_x \rightarrow \mathrm{exactly1}_{n' \in N \setminus \{n\}} \mathrm{CA}^1_{n,n'} \tag{13}$$

$$\mathrm{CA}^n_x \rightarrow \mathrm{exactly1}_{n' \in N \setminus \{n\}} \mathrm{CA}^0_{n,n'} \tag{14}$$

Case analysis is on at most one variable (15). Case analysis on x is only possible if its value is unknown (16). If node n does case analysis on x, then n has two successor nodes, and they respectively have x and $\neg x$ (17-18). The only new facts in the successor nodes of a case analysis node are x and $\neg x$ (19-22).

$$\mathrm{atmost1}(\{\mathrm{CA}^n_x \mid x \in X\}) \tag{15}$$

$$\mathrm{CA}^n_x \rightarrow \neg P^n_x \wedge \neg N^n_x \tag{16}$$

$$\mathrm{CA}^n_x \wedge \mathrm{CA}^1_{n,n'} \rightarrow P^{n'}_x \tag{17}$$

$$\mathrm{CA}^n_x \wedge \mathrm{CA}^0_{n,n'} \rightarrow N^{n'}_x \tag{18}$$

$$\mathrm{CA}^1_{n,n'} \wedge P^{n'}_x \rightarrow \mathrm{CA}^n_x \vee P^n_x \tag{19}$$

$$\mathrm{CA}^1_{n,n'} \wedge N^{n'}_x \rightarrow N^n_x \tag{20}$$

$$\mathrm{CA}^0_{n,n'} \wedge N^{n'}_x \rightarrow \mathrm{CA}^n_x \vee N^n_x \tag{21}$$

$$\mathrm{CA}^0_{n,n'} \wedge P^{n'}_x \rightarrow P^n_x \tag{22}$$

The last six are for nodes n and n' such that $n \neq n'$.

Arcs are induced by case analysis or by actions only.

$$(n, n') \leftrightarrow \left(\mathrm{CA}^1_{n,n'} \vee \mathrm{CA}^0_{n,n'} \vee \bigvee_{a \in A} (n, a, n') \right) \tag{23}$$

6.3 Initial and Goal Nodes

We assume the initial state formula to be in DNF as $I = \phi_1 \vee \cdots \vee \phi_k$. Let $\Phi_I = \{\phi_1, \ldots, \phi_k\}$. We require that for each $\phi \in \Phi_I$, at least one of the nodes $n_0, \ldots, n_{k-1}$ does not falsify any of the literals in ϕ (and hence all initial states for ϕ are "included" in some initial node), and that node has memory 0. We have for every $\phi \in \Phi_I$ the following.

$$\bigvee_{i=0}^{k-1} ((n_i, m_0) \wedge \neg L^{\neg\phi}_{n_i} \wedge \bigwedge_{\omega \in O} (\neg {}^o P^{n_i}_\omega \wedge \neg {}^o N^{n_i}_\omega)) \tag{24}$$

The unique goal node n_g has memory $m_{M_{\max}}$ and the formula G holds in n_g, and other node has memory $m_{M_{\max}}$.

$$(n_g, m_{M_{\max}}) \wedge L^G_{n_g} \qquad (25) \qquad \neg(n, m_{M_{\max}}) \text{ for all } n \neq n_g \qquad (26)$$

6.4 Encoding of Small-Memory Plans Implicitly

The implicit encoding, which does not make the plans explicit and only guarantees that one exists that matches the execution graph, is the one better scalable than the explicit encoding when there is a high number of observation combinations. It consists of the following.

$$\text{exactly1}((n, m_0), \ldots, (n, m_{M_{\max}})) \qquad (27)$$

For case analysis nodes, the successor nodes have the same memory.

$$\text{CA}^1_{n,n'} \rightarrow ((n, m) \leftrightarrow (n', m)) \text{ for } m \in M \qquad (28)$$

$$\text{CA}^0_{n,n'} \rightarrow ((n, m) \leftrightarrow (n', m)) \text{ for } m \in M \qquad (29)$$

Formulas (30-32) define the indistinguishability of two nodes w.r.t. memory, observations, and both respectively as $Z^m_{i,j}$, $Z^\omega_{i,j}$ and $Z_{i,j}$.

$$(n_i, m) \wedge (n_j, m) \rightarrow Z^m_{i,j} \text{ for } i < j, m \in M \qquad (30)$$

$$\neg({}^oP^{n_i}_\omega \wedge {}^oN^{n_j}_\omega) \wedge \neg({}^oN^{n_i}_\omega \wedge {}^oP^{n_j}_\omega) \rightarrow Z^\omega_{i,j} \qquad (31)$$

$$Z^m_{i,j} \wedge \bigwedge_{\omega \in O} Z^\omega_{i,j} \rightarrow Z_{i,j} \qquad (32)$$

If a node has observation ω, then ω must hold in the preceding node.

$$(n, n') \wedge {}^oP^{n'}_\omega \rightarrow L^\omega_n \qquad (33) \qquad (n, n') \wedge {}^oN^{n'}_\omega \rightarrow L^{\neg\omega}_n \qquad (34)$$

If two action nodes are indistinguishable, same action must be taken in both, and successor nodes must have the same memory state.

$$Z_{i,j} \wedge \text{A}_{n_i} \wedge \text{A}_{n_j} \rightarrow ((n_i, a) \leftrightarrow (n_j, a)) \qquad (35)$$

$$Z_{i,j} \wedge (n_i, n') \wedge (n_j, n'') \wedge \text{A}_{n_i} \wedge \text{A}_{n_j} \rightarrow ((n', m) \leftrightarrow (n'', m)) \qquad (36)$$

6.5 Encodings of Small Memory Plans Explicitly

Here we briefly describe the encoding of explicitly represented small-memory plans. The standard way of representing small-memory plans is by explicitly encoding the whole mapping from all observation combinations and the memory state to the action to be taken and the new memory state [5]. A finite memory plan is an automaton with

- $k \geqslant 1$ memory states $M = \{m_1, \ldots, m_k\}$,
- mapping $A_{M,O} : M \times O \rightarrow A$ from the current memory state and observation to an action,

– mapping $M_{M,O} : M \times O \to M$ from the current memory state and observation to the next memory state.

New atoms in the encoding:

– (n, ω): observation ω is observed in node n.
– $(n, \mathcal{O})$: $\mathcal{O}$ is the combination of observations observed in node n. To define the atoms, $\mathcal{O}$ is enumerated by valuation of all possible observations ω. For k observations and node n, there are 2^k different valuations and $(n, \mathcal{O})$ atoms.
– $(\mathcal{O}, m, a)$: action a is mapped to observation combination $\mathcal{O}$ and the current memory state is m.
– $(\mathcal{O}, m_i, m_j)$: the memory state m_j is mapped to observation combination $\mathcal{O}$ and the memory state is m_i.

Explicit encoding of the small-memory plans is as follows.

$$\text{valuation}((n, \omega_1), (n, \omega_2) \dots) \leftrightarrow (n, \mathcal{O}) \tag{37}$$

Map each observation valuation to an observation combination $\mathcal{O}$ for all nodes.

$$\text{exactly1}((\mathcal{O}, m, a_1), (\mathcal{O}, m, a_2), \dots) \tag{38}$$
$$\text{exactly1}((\mathcal{O}, m_i, m_1), (\mathcal{O}, m_i, m_2), \dots) \tag{39}$$

Each pair of observation combination and memory state is mapped to exactly one action and to exactly one memory state.

$$(n_i, m_j) \wedge (n_i, \mathcal{O}) \wedge (\mathcal{O}, m_j, a) \wedge \mathrm{A}_{n_i} \to (n_i, a) \tag{40}$$
$$(n_i, m_j) \wedge (n_i, \mathcal{O}) \wedge (n_i, a) \ \to (\mathcal{O}, m_j, a) \tag{41}$$

Only the action mapped to the memory and the observation combination can be applied in an action node with that memory and observation combination. Also if an action is taken in a node, then the memory and the observation combination must be mapped to that action in the plan.

$$(n_i, n_j) \wedge (n_i, \mathcal{O}) \wedge (n_i, m_k) \wedge (\mathcal{O}, m_k, m_l) \wedge \mathrm{A}_{n_i} \to (n_j, m_l) \tag{42}$$
$$(n_i, n_j) \wedge (n_i, \mathcal{O}) \wedge (n_i, m_k) \wedge (n_j, m_l) \to (\mathcal{O}, m_k, m_l) \tag{43}$$

If memory state m_l is mapped to observation combination $\mathcal{O}$ and memory state m_k, then the successor of an action node with those observations and memory should have memory state m_l.

$$(n_i, n_j) \wedge (n_i, m_k) \wedge \mathrm{CA}^{n_i} \to (n_j, m_k) \tag{44}$$

The memory stays the same in the successor nodes after case analysis.

$$\text{exactly1}((n, m_1), (n, m_2), \dots) \tag{45}$$

Each node can have only one memory state.

$$(n, n') \wedge \neg \mathrm{CA}^n \wedge (n', \omega) \to L^\omega_n \tag{46}$$
$$(n, n') \wedge \neg (n', \omega) \to \neg L^\omega_n \tag{47}$$

If a node has an observation for ω, the formula ω must hold in the predecessor. In case analysis nodes, the observations are copied forward.

$$(n, n') \wedge \mathrm{CA}^n \to ((n', \omega) \leftrightarrow (n, \omega)) \tag{48}$$

7 Sizes of the Encodings

In the implicit encoding, as given in Section 6.4, the largest component is formula (36) with size $|N|^4 \times |M|$. Other dominant formulas are only quadratic with sizes $|N|^2 \times |A|$ (35), $|N|^2 \times |M|$ (28-30) and $|N|^2 \times |O|$ (31, 33, 34).

On the other hand, the dominant formulas in the explicit encoding of small memory plans have components of size $2^{|O|}$ simply because the mapping has an exponential size, making the encoding quickly impractical for higher numbers of observations.

There are a number of components quadratic in $|N|$ used by both the implicit and explicit encoding, addressing arcs between nodes, for example in Section 6.2.

8 Invariants

Invariants are formulas (over state variables) that hold in all reachable states. They reduce the search performed by a SAT solver. With our 3-valued partial states, invariants don't have an obvious representation as redundant constraints, unlike in classical planning, as their interaction with frame axioms becomes more complicated due to partiality. Our solution is to *compile* invariants to actions: if $l_1 \vee l_2$ is an invariant, and $\overline{l_1}$ is an effect, then include l_2 as an effect. This makes it unnecessary to handle invariants explicitly in (7) and (8). Additionally we add redundant constraints that help SAT solvers prune the search space: For an invariant $x \vee y$, we include $\neg N_x^n \vee \neg N_y^n$ in the encoding, to allow inferring $\neg N_y^n$ whenever N_x^n has been inferred. Note that inferring P_y^n would be too strong, as abstraction/generalization may call for not making the value of y explicit.

We use 2-literal invariants $l_1 \vee l_2$ that are found with invariant algorithms that apply to non-deterministic actions [17].

9 Experiments

We have a proof-of-concept implementation of our framework, and we have run computational experiments to demonstrate its potential. Table 2 shows statistics on solving a number of planning problems. All runs were with a 2400 second time limit and performed on Xeon E5 2680 2.50 GHz CPUs with a memory limit of 8 GB. We tested by increasing number of nodes by 5 and memory states by 2 until the instances became solvable. All experiments used the KisSAT solver [2].

The largest instances reported here have tens of thousands of states, which is outside the scalability of methods that represent all states explicitly [5]. While this shows good potential for this approach, it is currently not competitive with the state-of-the-art, specifically the DNF/CNF family of planners [19, 20] or reductions of partial observability to fully observable problems [3, 4].

				Implicit			Explicit		
INSTANCE	V	A	O	n	m	time	n	m	time
doors2	6	12	4	10	6	7.78	15	6	17.59
medical002	6	10	2	10	4	2.85	10	4	1.48
medical003	8	11	3	10	4	3.28	10	4	3.75
medical004	9	11	3	15	4	332.86	15	4	94.64
medical005	11	12	4	20	6	1364.99	35	10	TO
bombRB1	3	3	2	10	4	1.82	10	4	0.77
bombRB2	5	6	4	10	4	1.9	20	8	1501.97
bombRB3	7	9	6	15	4	74.64	35	8	TO
bombRB4	9	12	8	15	4	346.75	15	4	OM
bts010	11	20	10	15	2	93.27	5	4	OM
gridXY13	3	3	2	10	4	1.37	10	4	0.97
gridXY15	5	3	2	15	4	52.09	50	8	TO
gridXY33	6	5	4	15	4	60.31	25	8	OM
rovers2	10	17	2	15	10	1505.43	15	10	180.66
elogistics1	13	24	6	15	6	66.21	15	8	770.26
elogistics3	19	39	9	20	6	795.27	10	4	OM
egrid2	42	100	20	15	10	1815.77	5	2	OM
logistics1	13	24	6	10	6	7.94	15	8	OM
logistics3	19	39	9	15	8	137.99	5	4	OM
medpks2	6	6	2	10	4	2.64	10	4	1.68
grid2	42	100	20	15	10	1816.62	5	2	OM
blocks3	15	51	15	10	4	3.41	5	2	OM
erovers2	10	17	2	15	10	1485.55	15	8	100.47

Table 2. Comparison of the implicit and explicit encodings. V: *number of variables*; A: *number of actions*; O: *number of observations*; TO: *time-out*; OM: *out-of-memory*

10 Conclusion

We have shown how planning with a complex actions and partial observability, can be effectively reduced to propositional logic and solved with SAT solvers. This is the first time that planning with partial observability has been solved with single SAT solver calls that both find a plan and determine its correctness, showing significant potential in the ideas presented by Geffner & Geffner [7].

Our framework is effective when belief states can be represented as conjunctions of literals. Complex dependencies between state variables, representable e.g. as disjunctions $a \vee b$, require making plan executions more explicit, increasing the size of the graphs and the search cost. More expressive belief state representations should be investigated.

An obvious inefficiency in our encoding is that the use of case analysis is not restricted in any way, for example allowing forgetting immediately followed by case analysis on the same variable. Another inefficiency worth further research is symmetry reduction for the graphs.

References

1. Baral, C., Kreinovich, V., Trejo, R.: Computational complexity of planning and approximate planning in the presence of incompleteness. Artificial Intelligence **122**(1), 241–267 (2000)
2. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. In: Proceedings of SAT Competition 2020 – Solver and Benchmark Descriptions, pp. 51–53. Department of Computer Science Report Series B, vol. B-2020-1, University of Helsinki (2020)
3. Bonet, B., Geffner, H.: Planning under partial observability by classical replanning: Theory and experiments. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 1936–1941 (2011)
4. Bonet, B., Geffner, H.: Flexible and scalable partially observable planning with linear translations. In: Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI-14). pp. 2235–2241. Citeseer (2014)
5. Chatterjee, K., Chmelik, M., Davies, J.: A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In: Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16). pp. 3225–3232. AAAI Press (2016)
6. Ferraris, P., Giunchiglia, E.: Planning as satisfiability in nondeterministic domains. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000) and the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-2000). pp. 748–753. AAAI Press (2000)
7. Geffner, T., Geffner, H.: Compact policies for non-deterministic fully observable planning as SAT. In: ICAPS 2018. Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling. pp. 88–96. AAAI Press (2018)
8. Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence. pp. 359–363. John Wiley & Sons (1992)
9. Kautz, H., Selman, B.: Pushing the envelope: planning, propositional logic, and stochastic search. In: Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference. pp. 1194–1201. AAAI Press (1996)
10. Lusena, C., Li, T., Sittinger, S., Wells, C., Goldsmith, J.: My brain is full: When more memory helps. In: Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference (UAI-99). pp. 374–381. Morgan Kaufmann Publishers (1999)
11. Majercik, S.M., Littman, M.L.: MAXPLAN: A new approach to probabilistic planning. In: Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems. pp. 86–93. Pittsburgh, Pennsylvania (1998)
12. Majercik, S.M., Littman, M.L.: Contingent planning under uncertainty via stochastic satisfiability. Artificial Intelligence **147**(1-2), 119–162 (2003)
13. Meuleau, N., Kim, K.E., Kaelbling, L.P., Cassandra, A.R.: Solving POMDPs by searching the space of finite policies. In: Uncertainty in Artificial Intelligence, Proceedings of the Fifteenth Conference (UAI-99). pp. 417–426. Morgan Kaufmann Publishers (1999)
14. Pandey, B., Rintanen, J.: Planning for partial observability by SAT and graph constraints. In: ICAPS 2018. Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling. pp. 190–198. AAAI Press (2018)
15. Rintanen, J.: Constructing conditional plans by a theorem-prover. Journal of Artificial Intelligence Research **10**, 323–352 (1999)

16. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07). pp. 1045–1050. AAAI Press (2007)
17. Rintanen, J.: Regression for classical and nondeterministic planning. In: ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence. pp. 568–571. IOS Press (2008)
18. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, Sitges, Spain, October 2005., pp. 827–831. Springer-Verlag (2005)
19. To, S.T., Pontelli, E., Son, T.C.: On the effectiveness of CNF and DNF representations in contingent planning. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 2033–2038. AAAI Press (2011)
20. To, S.T., Son, T.C., Pontelli, E.: A generic approach to planning in the presence of incomplete information: Theory and implementation. Artificial Intelligence **227**, 1–51 (2015)
21. Turner, H.: Polynomial-length planning spans the polynomial hierarchy. In: Logics in Artificial Intelligence, European Conference, JELIA 2002. pp. 111–124. No. 2424 in Lecture Notes in Computer Science, Springer-Verlag (2002)