Minimizing Necessary Observations for Nondeterministic Planning

Robert Mattmüller, Manuela Ortlieb, and Erik Wacker

Research Group Foundations of AI, University of Freiburg, Germany
{mattmuel,ortlieb,wackere}@informatik.uni-freiburg.de

Abstract. Autonomous agents interact with their environments via sensors and actuators. Motivated by the observation that sensors can be expensive, in this paper we are concerned with the problem of minimizing the amount of sensors an agent needs in order to successfully plan and act in a partially observable nondeterministic environment. More specifically, we present a simple greedy top-down algorithm in the space of observation variables that returns an inclusion minimal set of state variables sufficient to observe in order to find a plan. We enhance the algorithm by reusing plans from earlier iterations and by the use of functional dependencies between variables that allows the values of some variables to be inferred from those of other variables. Our experimental evaluation on a number of benchmark problems shows promising results regarding runtime, numbers of sensors and plan quality.

Keywords: AI planning, nondeterministic planning, partial observability, observation actions

1 Introduction

When an autonomous agent interacts with its environment, it does so via *sensors* and *actuators* [8]. We consider an agent acting in a partially observable nondeterministic environment and equipped with an appropriate offline planning component. In particular, we consider the *sensors* this agent needs to be fitted with. Assuming that sensors are expensive, e.g., regarding power consumption, weight, or financially, it can be worthwhile trying to minimize the set of sensors necessary to solve a certain planning task. For example, it might turn out that in some specific robotic application, an RGB-D camera can handle all the observations a laser scanner would be used for, thus obviating the latter. In this paper, we study the problem of minimizing the set of necessary sensors, *not* the problem of minimizing the number of occurrences of observations in a plan. This makes a difference, since, e.g., excluding a laser scanner from the set of sensors makes the robot cheaper, but may lead to more complicated behavior and more compensatory observations at runtime. We simplify the problem by assuming that the amount of sensors needed is proportional to the number of variables that may have to be observed. Given a planning domain, this reduces our problem to finding a minimal set of (schematic) state variables such that every possible planning task from that domain is solvable if those and only those (schematic) variables are observable. We further simplify our problem by searching for sufficient sets of (grounded) state variables on the level of planning tasks instead of the level of planning domains. Strictly speaking, the resulting set of state variables is only sufficient for one specific planning task. But if that task is reasonably chosen (featuring all interesting aspects of the underlying domain), the set of observation variables found for that task can again be lifted to the schematic level of the underlying planning domain. Moreover, searching for sufficient observation variables on the instantiated level has the advantage of being more fine-grained than searching on the schematic level. This can potentially show that some ground instances of a predicate are necessary observations, whereas other ground instances of the same predicate are not necessary, possibly leading to a more fine-grained choice of sensors. Consider for example a nondeterministic version of the Blocksworld domain where the PutOnBlock(A, B) actions can have the undesired outcome that the moved block A is dropped to the table. All other actions are deterministic. If the initial state is completely known, it turns out that it is sufficient to observe the values of the variables Clear(B) for all blocks B for which an action PutOnBlock(A, B) occurs in the plan for some other block A. This is because the initial belief state is a singleton belief state and whenever a nondeterministic action of the form PutOnBlock(A, B) is applied to a singleton belief state, the resulting belief state will contain exactly two world states (one with A on B, the other with A on the table) that can be distinguished by observing the variable Clear(B). Notice that depending on the initial and goal state, not all instances of the Clear predicate may have to be observed, but for all nontrivial tasks, at least one of them has to be. Lifting this back to the domain level shows that observing the schematic Clear predicate is sufficient to solve all tasks from this domain. Regarding sensors this means that it is sufficient to install an overhead camera that is only able to observe if blocks are clear, but not in which configurations they are stacked. To our knowledge, there is little previous work on this topic. Huang et al. [5] study the related problem of finding an approximately minimal set of observation variables for strong planning, given a set of observation variables $\mathcal{V}$ (including possibly derived variables) and a *fixed* strong plan π that works under the assumption that all variables from $\mathcal{V}$ are observable. Their algorithm then reduces $\mathcal{V}$ to a sufficient subset for π still to work by first identifying all state pairs that need to be distinguishable for π to work and then identifying all variables from $\mathcal{V}$ necessary to actually distinguish those. Unlike in their approach, here we retain flexibility regarding the choice of the plan and allow any (strong cyclic) plan to be found as long as the observation variables found along with it are sufficient for the plan to be executable.

2 Preliminaries

We formalize partially observable nondeterministic (POND) planning tasks using a *finite-domain representation* for the state variables similar to the formalization of Ortlieb and Mattmüller [6]. A *POND planning task skeleton* is a tuple $\Pi = \langle \mathcal{V}, B_0, B_\star, \mathcal{A}, \mathcal{W} \rangle$ consisting of the following components: $\mathcal{V}$ is a finite set of *state variables* v, each with a finite *domain* $\mathcal{D}_v$ and an *extended domain* $\mathcal{D}_v^+ = \mathcal{D}_v \uplus \{\bot\}$, where $\bot$ denotes the *undefined* or *don't-care* value. A *partial state* is a function s with $s(v) \in \mathcal{D}_v^+$ for all $v \in \mathcal{V}$. We say that s is *defined* for $v \in \mathcal{V}$ if $s(v) \neq \bot$. A *state* is a partial state s such that its *scope* $\mathrm{scope}(s) = \{v \in \mathcal{V} \mid s(v) \neq \bot\}$ is $\mathcal{V}$. The set of all states s over $\mathcal{V}$ is denoted as $\mathcal{S}$, and the set of all *belief states* B over $\mathcal{V}$ is denoted as $\mathcal{B} = 2^{\mathcal{S}}$. Depending on the context, a partial state s_p can be interpreted either as a *condition*, which is *satisfied* in a state s iff s agrees with s_p on all variables for which s_p is defined, or as an *update* on a state s, resulting in a new state s' that agrees with s_p on all variables for which s_p is defined, and with s on all other variables. The *initial belief state* B_0 and the *goal description* $B_\star$ of a task skeleton are both belief states. A belief state B is a *goal belief state* iff $B \subseteq B_\star$. $\mathcal{A}$ is a finite set of *actions* of the form $a = \langle \mathit{Pre}, \mathit{Eff} \rangle$, where the *precondition* Pre is a partial state, and the *effect* Eff is a finite set of partial states eff, the *nondeterministic outcomes* of a. The *application* of a nondeterministic outcome eff to a state s is the state $\mathit{app}(\mathit{eff}, s)$ that results from updating s with eff. The application of an effect Eff to s is the set of states $\mathit{app}(\mathit{Eff}, s) = \{\mathit{app}(\mathit{eff}, s) \mid \mathit{eff} \in \mathit{Eff}\}$ that might be reached by applying a nondeterministic outcome from Eff to s. An action is *applicable* in a state s iff its precondition is satisfied in s, and it is applicable in a belief state B if it is applicable in all $s \in B$. Actions are applied in belief states and result in belief states. The application of an action in a belief state B is undefined if the action is inapplicable in B. Otherwise, the application of an action $a = \langle \mathit{Pre}, \mathit{Eff} \rangle$ to B is the set $\mathit{app}(a, B) = \{\mathit{app}(\mathit{eff}, s) \mid \mathit{eff} \in \mathit{Eff}, s \in B\}$. Finally, $\mathcal{W} \subseteq \mathcal{V}$ is the set of variables that are possibly observable.

A POND planning task skeleton still lacks observations, which we define next. An *observation* is simply a variable $o \in \mathcal{W}$. The application of an observation o to B is the set of nonempty belief states that result from splitting B according to possible values of o, i.e., $\mathit{app}(o, B) = \{\{s \in B \mid s(o) = d\} \mid d \in \mathcal{D}_o\} \setminus \{\emptyset\}$. Now, a *POND planning task* is a tuple $\Pi[\mathcal{O}] = \langle \Pi, \mathcal{O} \rangle$ consisting of a POND planning task skeleton Π and a finite set of observations $\mathcal{O}$. All actions and observation applications have unit cost. We will sometimes abuse notation and refer to a planning task by Π as well. POND planning tasks as defined above induce nondeterministic transition systems where the nodes are the (reachable) belief states and where there is an arc from a belief state B to a belief state B' labeled with an action a (or observation o) iff a (or o) is applicable in B and $B' = \mathit{app}(a, B)$ (or $B' \in \mathit{app}(o, B)$). Given a POND planning task, we seek a strong cyclic plan [2] solving the task, i.e., a partial mapping π from belief states to applicable actions or observations such that for all belief states B reachable from the initial belief state B_0 following π, B is either a goal belief state, or π is defined for B *(π is closed)* and at least one goal belief state is reachable from B following π *(π is proper)*. For a plan π, by $\mathcal{B}_\pi$ we denote the set of belief states for which π is defined, i.e., the set of non-goal belief states reachable following π, including the initial belief state.

3 Minimizing Necessary Observations

We can formalize the problem of finding minimal sets of observations sufficient to solve a planning task either in terms of cardinality minimality or of inclusion minimality. For cardinality minimality, we get the following search problem.

*Problem 1 (*ObserveCardMin*).*
Input: A POND planning task skeleton $\Pi = \langle \mathcal{V}, B_0, B_\star, \mathcal{A}, \mathcal{W} \rangle$.
Output: A cardinality minimal set of observations $\mathcal{O} \subseteq \mathcal{W}$ for Π such that there exists a strong cyclic plan for $\Pi[\mathcal{O}]$, or None if no such set $\mathcal{O}$ exists.

To classify the problem ObserveCardMin complexity theoretically, we need a theorem by Rintanen.

Theorem 1 (Rintanen, 2004 [7]). *The strong cyclic plan existence problem for POND planning,* PlanExPOND*, is* 2-Exptime*-complete.* □

Rintanen's formalism differs slightly from ours in that his variables are propositional instead of finite-domain, that he encodes initial and goal states symbolically using formulas, and that he allows conditional effects. Neither of those differences affects the 2-Exptime-completeness result. Using Rintanen's result, we can immediately prove the following theorem.

Theorem 2. ObserveCardMin *is* 2-Exptime*-complete.*

Proof. To show that ObserveCardMin is 2-Exptime-hard, we polynomially reduce PlanExPOND to ObserveCardMin. POND planning tasks in the sense of PlanExPOND have exactly the same form as our POND planning tasks skeletons $\Pi = \langle \mathcal{V}, B_0, B_\star, \mathcal{A}, \mathcal{W} \rangle$. Viewing such a POND planning task Π as an input to ObserveCardMin, we see that the output of ObserveCardMin is different from None iff Π is a positive instance of PlanExPOND. To see that ObserveCardMin $\in$ 2-Exptime, we have to give a 2-Exptime algorithm solving the problem. The naïve algorithm that iterates over all (exponentially many) candidate subsets $\mathcal{O} \subseteq \mathcal{W}$, tests whether $\Pi[\mathcal{O}]$ is solvable, and returns a cardinality minimal set $\mathcal{O}$ for which this is the case, is such an algorithm, because each test if $\Pi[\mathcal{O}]$ is solvable is in 2-Exptime according to Theorem 1, and at most exponentially many such tests have to be performed. □

For inclusion instead of cardinality minimality we get a similar result.

*Problem 2 (*ObserveInclMin*).*
Input: A POND planning task skeleton $\Pi = \langle \mathcal{V}, B_0, B_\star, \mathcal{A}, \mathcal{W} \rangle$.
Output: An inclusion minimal set of observations $\mathcal{O} \subseteq \mathcal{W}$ for Π such that there exists a strong cyclic plan for $\Pi[\mathcal{O}]$, or None if no such set $\mathcal{O}$ exists.

Theorem 3. ObserveInclMin *is* 2-Exptime*-complete.*

Proof. Similar to proof that ObserveCardMin is 2-Exptime-complete. □

Clearly, cardinality minimal solutions are also inclusion minimal, but not every inclusion minimal solution is also cardinality minimal. Although both ObserveCardMin and ObserveInclMin are 2-Exptime-complete, we expect ObserveCardMin to be even more challenging in practice, since, in the worst case, the complete space of subsets of $\mathcal{W}$ has to be exhausted, whereas for ObserveInclMin a greedy top-down or bottom-up search in the space of those subsets is sufficient. Therefore, in the following we restrict our attention to practically solving ObserveInclMin.

3.1 Greedy Top-Down Search

When looking for minimal sets of observations, we can restrict our attention to variables that may ever need to be observed because they are either unknown initially or for which there is an action that makes them unknown.

Definition 1. *Let B be a belief state, $v \in \mathcal{V}$ a variable, and $a = \langle Pre, Eff \rangle \in \mathcal{A}$ an action. Then*

1. v is known in B *iff there exists a value $d \in \mathcal{D}_v$ such that $s(v) = d$ for all states $s \in B$.*
2. a makes v unknown *iff there are two nondeterministic effects $eff, eff' \in Eff$ such that $eff(v) = d$ for some value $d \in \mathcal{D}_v$ with $d \neq Pre(v)$ and $eff'(v) \neq d$.*
3. v may need to be observed *iff v is not known in the initial belief state B_0 or there exists an action $a \in \mathcal{A}$ that makes v unknown.*

Since whenever $\Pi[\mathcal{O}]$ is solvable, also $\Pi[\mathcal{O}^*]$ is solvable, where $\mathcal{O}^*$ is the set of all $v \in \mathcal{O}$ that may need to be observed, for the rest of this paper we assume that all $v \in \mathcal{W}$ may need to be observed.

Algorithm 1 shows a simple greedy algorithm that solves ObserveInclMin.

Algorithm 1 Simple Greedy Algorithm for ObserveInclMin

```
1:  function simpleGreedySearch($\Pi$):
2:      if $\Pi[\mathcal{W}]$ is unsolvable then
3:          return None
4:      Compute some plan $\pi$ for $\Pi[\mathcal{W}]$
5:      Let $\mathcal{O}$ be the set of variables actually observed in $\pi$
6:      Let $\mathcal{O}' = \mathcal{O}$
7:      for all $o \in \mathcal{O}'$ do
8:          if $\Pi[\mathcal{O} \setminus \{o\}]$ is solvable then
9:              Set $\mathcal{O}$ to $\mathcal{O} \setminus \{o\}$
10:     return $\mathcal{O}$
```

It is obvious that Algorithm 1 runs in doubly exponential time. We can also prove correctness of the algorithm.

Theorem 4. *Algorithm 1 correctly solves problem* ObserveInclMin.

Proof. Clearly, Algorithm 1 returns NONE iff no set $\mathcal{O} \subseteq \mathcal{W}$ exists such that $\Pi[\mathcal{O}]$ admits a strong cyclic plan. If there is a solution, then to see that Algorithm 1 returns an inclusion minimal set $\mathcal{O} \subseteq \mathcal{W}$ such that $\Pi[\mathcal{O}]$ admits a strong cyclic plan, we have to show that $\Pi[\mathcal{O}]$ is solvable and that $\Pi[\mathcal{O}']$ is unsolvable for all proper subsets $\mathcal{O}' \subsetneq \mathcal{O}$. Let $\mathcal{O}_0$ be the set of observation variables computed in line 5 and $o_1, \dots, o_n$ be the order in which $\mathcal{O}_0$ is traversed by the algorithm, and let $\mathcal{O}_i$ be the set $\mathcal{O}$ after the i-th iteration, $i = 1, \dots, n$. The fact that $\Pi[\mathcal{O}]$ is solvable follows inductively from the fact that $\Pi[\mathcal{O}_0]$ is solvable and that $\mathcal{O}_{i+1} \neq \mathcal{O}_i$ only if $\Pi[\mathcal{O}_{i+1}]$ is still solvable. To see that $\Pi[\mathcal{O}']$ is unsolvable for all proper subsets $\mathcal{O}' \subsetneq \mathcal{O}_n$, let $o_i \in \mathcal{O}_n$. Then in the i-th iteration, o_i was not removed, because $\Pi[\mathcal{O}_{i-1} \setminus \{o_i\}]$ would have been unsolvable. But since $\mathcal{O}_n \subseteq \mathcal{O}_{i-1}$ and hence $\mathcal{O}_n \setminus \{o_i\} \subseteq \mathcal{O}_{i-1} \setminus \{o_i\}$, also $\Pi[\mathcal{O}_n \setminus \{o_i\}]$ would be unsolvable (since removing observations only reduces solvability). $\square$

Algorithm 1 is simple, but quite inefficient. Specifically, when testing if the task remains solvable after deleting a variable (line 8), no plan information from the previous step is reused. In the next subsection, we investigate the reuse of portions of plans not affected by making a particular variable unobservable.

3.2 Plan Reuse

Suppose we know a plan π for $\Pi[\mathcal{O}]$ and want to test if there also exists a plan for $\Pi[\mathcal{O} \setminus \{o\}]$ for some $o \in \mathcal{O}$. Instead of replanning, we can try to reuse the portions of π *before the first splits of belief states with respect to o*.

Let $\pi : \mathcal{B}_\pi \to \mathcal{A} \cup \mathcal{O}$ be a plan for $\Pi[\mathcal{O}]$, and let $o \in \mathcal{O}$. Then by $\mathit{knownpos}(\pi, o)$ we refer to the set of all belief states $B \in \mathcal{B}_\pi \cup \{\mathit{app}(\pi(B'), B')) \mid B' \in \mathcal{B}_\pi\}$ such that v is known in B, and by $\mathit{safepos}(\pi, o)$ to the set of all belief states $B \in \mathit{knownpos}(\pi, o)$ that are reachable from B_0 following π along paths passing exclusively through belief states in $\mathit{knownpos}(\pi, o)$. By $\mathit{gaps}(\pi, o)$ we refer to those $B \in \mathit{safepos}(\pi, o)$ that (a) are not goal belief states, (b) for which the successor belief state $B' = \mathit{app}(\pi(B), B))$ following π is not in $\mathit{knownpos}(\pi, o)$, and (c) there exists a path from B to a goal belief state following π such that some action along that path is the observation of o. Intuitively, $\mathit{gaps}(\pi, o)$ is the set of belief states that form the fringe between the portion of the plan π that we can reuse and the portion we need to recompute. In Algorithm 2 we will iteratively fill those gaps, starting from shallow ones and working our way down to the deeper ones. To that end, we associate with each belief state $B \in \mathit{gaps}(\pi, o)$ a value $\mathit{depth}(B)$ that denotes the length of the shortest execution sequence of actions (and observations) leading from B_0 to B following π. By π_o we denote the part of π that lies before $\mathit{gaps}(\pi, o)$. More formally, we can view π as a set of pairs (B, a), where B is a belief state and $a = \pi(B)$ is an action or observation. Then such a pair from π is also contained in π_o if $B \in \mathit{safepos}(\pi, o) \setminus \mathit{gaps}(\pi, o)$. We call π_o the *plan π restricted to safe prefixes with respect to o*.

For all gap belief states $B \in \mathit{gaps}(\pi, o)$, in order to replan for B we need to solve the planning task that is like $\Pi[\mathcal{O} \setminus \{o\}]$, but with its initial belief state replaced by B. We refer to a planning task skeleton (or a planning task) Π with initial belief state replaced by B as $\mathit{\Pi}\langle B\rangle$. Therefore, the task we have to replan for is $\mathit{\Pi}[\mathcal{O} \setminus \{o\}]\langle B\rangle$. Algorithm 2 shows a greedy algorithm with plan reuse to (approximately) solve ObserveInclMin.

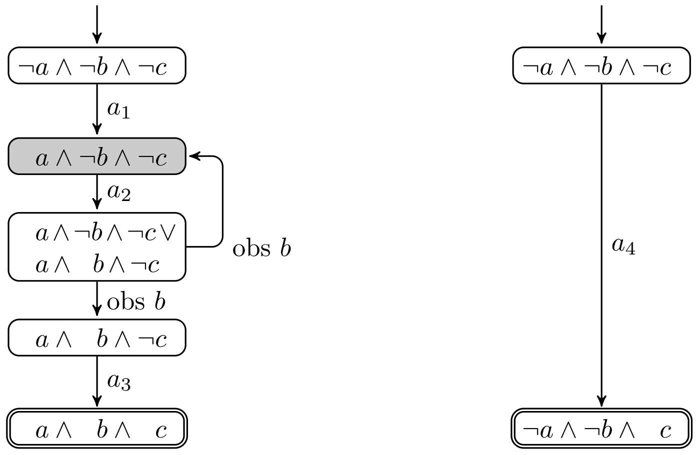
Algorithm 2 Greedy Algorithm with Plan Reuse for ObserveInclMin

```
1: function greedySearchWithPlanReuse($\mathit{\Pi}$):
2:     if $\mathit{\Pi}[\mathcal{W}]$ is unsolvable then
3:         return None
4:     Compute some plan $\pi$ for $\mathit{\Pi}[\mathcal{W}]$
5:     Let $\mathcal{O}$ be the set of variables actually observed in $\pi$
6:     Let $\mathcal{O}' = \mathcal{O}$
7:     for all $o \in \mathcal{O}'$ do
8:         Let $\mathcal{G}$ be $gaps(\pi, o)$
9:         Let $\pi_o$ be the plan $\pi$ restricted to safe prefixes with respect to $o$
10:        Set allGapsFillable to true
11:        while $\mathcal{G} \neq \emptyset$ and allGapsFillable do
12:            Pick $B \in \mathcal{G}$ with minimal $depth(B)$ and remove it from $\mathcal{G}$
13:            if $\mathit{\Pi}[\mathcal{O} \setminus \{o\}]\langle B\rangle$ is solvable with plan $\pi^B_{-o}$ then
14:                Merge $\pi^B_{-o}$ into $\pi_o$ (resulting in updated $\pi_o$)
15:                Trace $\pi_o$ and retain in $\mathcal{G}$ only those belief states that are non-goal
16:                        belief states reachable following $\pi_o$ for which $\pi_o$ is undefined.
17:                if $\mathcal{G} = \emptyset$ then
18:                    Set $\mathcal{O}$ to $\mathcal{O} \setminus \{o\}$
19:                    Set $\pi$ to $\pi_o$
20:            else
21:                Set allGapsFillable to false
22:    return $\mathcal{O}$
```

Like Algorithm 1, Algorithm 2 first tests for solvability, computes an initial plan, and then iteratively tries to remove variables from the observation set $\mathcal{O}$. Unlike Algorithm 1, when testing if o can be removed, it determines the gaps in π that arise if o is no longer observable and that need to be filled using new subplans. It also identifies the portion π_o of π that can be reused as it does not depend on o. If at least one gap cannot be filled, o is retained and the next observation variable is considered (line 21). In order to fill one gap B, a plan π^B_{-o} for $\mathit{\Pi}[\mathcal{O} \setminus \{o\}]\langle B\rangle$ is computed. This plan π^B_{-o} is then merged into π_o (line 14), which means that all entries from π^B_{-o} are added to π_o, and if both π_o and π^B_{-o} contain an entry for the same belief state B', then the entry from π^B_{-o} is used and that from π_o is overridden. More formally, in line 14 we set π_o to $\pi_o \oplus \pi^B_{-o}$, where $(\pi_o \oplus \pi^B_{-o})(B') = \pi^B_{-o}(B')$ if $\pi^B_{-o}(B')$ is defined, and $(\pi_o \oplus \pi^B_{-o})(B') = \pi_o(B')$, otherwise. In line 15/16, the algorithm removes from $\mathcal{G}$ all gaps that were "accidentally" *closed* by π^B_{-o} and need not be considered any longer, i.e., all gaps B' for which π^B_{-o} is defined, as well as all gaps "accidentally" *circumvented* by π^B_{-o} by a different choice further up in the plan. Finally, if no gaps are left, o is removed from $\mathcal{O}$, π is set to π_o, and the next observation variable is considered. Notice that traversing the gaps from shallow to deep is only used as a heuristic and not strictly necessary for the algorithm.

Regarding runtime, it is obvious that the POND planning steps that take doubly exponential time in the worst case still dominate the overall runtime, and that at most exponentially many such planning steps are necessary. All other computations are cheaper than 2-EXPTIME. Therefore, Algorithm 2 runs in doubly exponential time. We still need to show that the final plan π is really a strong cyclic plan for the task $\Pi[\mathcal{O}]$ with the final set $\mathcal{O}$, and we have to reason about inclusion minimality of $\mathcal{O}$. We first give an example showing that the returned set $\mathcal{O}$ is in general *not* inclusion minimal, and that the fact that we consider the removal of each observation variable only once is not the culprit. Instead, the reason is that some gap B might not be fillable without observing variable o, but completely replanning without observing o is possible and B simply does not occur in a completely replanned solution.

Example 1. Consider the planning task skeleton $\Pi = \langle \mathcal{V}, B_0, B_\star, \mathcal{A}, \mathcal{W} \rangle$ with $\mathcal{V} = \{a, b, c\}$ and $\mathcal{D}_v = \{0, 1\}$ for all $v \in \mathcal{V}$. For ease of notation, we write (belief) states and preconditions as Boolean formulas over $\mathcal{V}$, and nondeterministic effects as sets of such formulas. Furthermore, $B_0 = \neg a \wedge \neg b \wedge \neg c$, $B_\star = c$, $\mathcal{W} = \{b\}$, and $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$, where $a_1 = \langle \neg a, \{a\} \rangle$, $a_2 = \langle \neg b, \{b, \top\} \rangle$, $a_3 = \langle b, \{c\} \rangle$, and $a_4 = \langle \neg a \wedge \neg b \wedge \neg c, \{c\} \rangle$. A solution π for $\Pi[\mathcal{W}]$ is depicted on the left below.



When b is made unobservable, we seek a plan for $\Pi[\emptyset]$. We can reuse the first two belief states from π up to the (only) gap state marked in gray. However, replanning with initial belief state $a \wedge \neg b \wedge \neg c$ without observing b will fail, since each action that makes c true needs b to be known or a to be false, neither of which can be accomplished from $a \wedge \neg b \wedge \neg c$ without observing b. Therefore, Algorithm 2 will return $\mathcal{O} = \{b\}$, whereas $\mathcal{O}^* = \emptyset$ would be an inclusion minimal solution, as witnessed by the plan in the right-hand part of the figure above. □

Therefore, in order to find inclusion minimal observation sets, one would still have to run Algorithm 1 initialized with the output of Algorithm 2. Next, we show that Algorithm 2 returns a valid solution.

Theorem 5. *Assume $\Pi[\mathcal{W}]$ is solvable and let π and $\mathcal{O}$ be the plan and the set of observation variables at termination of Algorithm 2. Then π is a strong cyclic plan for $\Pi[\mathcal{O}]$.*

Proof. Let $\mathcal{O}_0$ be the set of observation variables used in the initial plan π^0, let $o_1, \ldots, o_n$ be the order in which $\mathcal{O}_0$ is traversed by the algorithm, let $\mathcal{O}_i$ be the set $\mathcal{O}$ after the i-th iteration, $i = 1, \ldots, n$, and π^i the corresponding plan.

We show the claim by induction over i. The induction base is obvious, since π^0 is a strong cyclic plan for $\Pi[\mathcal{O}_0]$ by construction. For the inductive step from i to $i+1$, there are two cases. If $\mathcal{O}_{i+1} = \mathcal{O}_i$, then also $\pi^{i+1} = \pi^i$ and vice versa. Since π^i is a plan for $\Pi[\mathcal{O}_i]$, also π^{i+1} is a plan for $\Pi[\mathcal{O}_{i+1}]$. The more interesting case is the one where $\mathcal{O}_{i+1} = \mathcal{O}_i \setminus \{o_{i+1}\}$. We have to show that after all gaps are filled, the resulting composite plan solves $\Pi[\mathcal{O}_{i+1}]$. Let $B_i^1, \ldots, B_i^k \in \mathcal{G}$ be the set of all gaps that actually get filled, in that order (some gaps might be skipped if they are accidentally filled or avoided before the algorithm would explicitly take care of them). Let $\pi_j^i := \pi_{-o_{i+1}}^{B_i^j}$, $j = 1, \ldots, k$, be the corresponding plans to fill the gaps, and let $\pi_o^i := \pi_{o_{i+1}}^i$. Then the resulting plan π^{i+1} is $(\ldots(\pi_o^i \oplus \pi_1^i) \oplus \ldots) \oplus \pi_k^i$. We have to show that π^{i+1} is a strong cyclic plan for $\Pi[\mathcal{O}_{i+1}]$. It is clear that π^{i+1} does not observe any variable outside $\mathcal{O}_i$, and also it does not observe o_{i+1} by construction. What is left is showing that π^{i+1} is closed and proper. To see this, notice that by construction each execution of π^{i+1} starts out as an execution of π_o^i (for zero or more steps), then possibly executes π_1^i (for zero or more steps), then π_2^i and so on, with lower indices j of the executed subplan π_j^i only increasing. Eventually, it only follows π_o^i or π_j^i for some maximal $j \leq k$. Since π^i and all π_j^i, $j = 1, \ldots, k$, are closed and proper, the same holds for π^{i+1}. This concludes the proof. □

3.3 Use of Functional Dependencies

We discuss one more way to speed up Algorithm 1 or 2, a preprocessing step that discards potential observation variables o whose value can be derived from the values of other observation variables. For our purpose, in order to discard o, it is sufficient that there are observation variables $o^1, \ldots, o^n$ that are not discarded and a function $f : \prod_{i=1}^n \mathcal{D}_{o^i} \to \mathcal{D}_o$ such that for all world states s in all belief states B encountered following π, $s(o) = f(s(o^1), \ldots, s(o^n))$. For states not encountered following the plan, this equality does not necessarily have to hold. We call a tuple $F = (o, \{o^1, \ldots, o^n\}, f)$ with this property a *functional dependency* in π, $\mathit{head}(F) = o$ the *head* of F, $\mathit{body}(F) = \{o^1, \ldots, o^n\}$ the *body* of F, and $\mathit{fun}(F) = f$ the *function* of F. We call a set of functional dependencies $\mathcal{F}$ *acyclic* if there are no $F_1, \ldots, F_k \in \mathcal{F}$ such that $\mathit{head}(F_i) \in \mathit{body}(F_{i+1})$ for all $i = 1, \ldots, k-1$ and $\mathit{head}(F_k) \in \mathit{body}(F_1)$. Given a plan π that observes only variables in $\mathcal{O}$ and an acyclic set of functional dependencies $\mathcal{F} = \{F_1, \ldots, F_m\}$ in π such that (a) for all $F, F' \in \mathcal{F}$, $\mathit{head}(F) \notin \mathit{body}(F')$ (if $\mathcal{F}$ is acyclic, this can be assumed without loss of generality), (b) for all $F \neq F' \in \mathcal{F}$, $\mathit{head}(F) \neq \mathit{head}(F')$, and (c) for all $F \in \mathcal{F}$, $\mathit{body}(F) \subseteq \mathcal{O}$, every occurrence of an observation o in π

such that $o = \mathit{head}(F)$ for some $F \in \mathcal{F}$ can be replaced by successive observations of (a subset of) the observation variables in $\mathit{body}(F)$. This produces a new plan $\pi_{\mathcal{F}}$ that observes only variables in $\mathcal{O} \setminus \{\mathit{head}(F) \mid F \in \mathcal{F}\}$.

We can extend Algorithms 1 and 2 accordingly as follows: After computing some plan π for $\Pi[\mathcal{W}]$, find an acyclic set of functional dependencies $\mathcal{F}$ in π with properties (a), (b) and (c) as above and replace π with $\pi_{\mathcal{F}}$. In our implementation, we use a simplified version of this idea: We only identify sets $\mathcal{X}$ of Boolean observation variables such that in each state reachable with π, exactly one $X \in \mathcal{X}$ is true. Each such set $\mathcal{X} = \{X_1, \dots, X_k\}$ induces k functional dependencies $F_1, \dots, F_k$, where $F_i = (X_i, \mathcal{X} \setminus X_i, f)$, $i = 1, \dots, k$, where $f(x_1, \dots, x_{k-1}) = 1$ if $x_1 = \dots = x_{k-1} = 0$, and $f(x_1, \dots, x_{k-1}) = 0$, otherwise. Regarding computation of such sets $\mathcal{X}_1, \dots, \mathcal{X}_n$, for all subsets of Boolean variables we test whether they satisfy the desired mutex property. Candidate sets that obviously cannot satisfy it (after inspection of the reachable states) are not generated. From the functional dependencies induced by the remaining sets $\mathcal{X}$, we keep an acyclic (but not necessarily maximal) subcollection $\mathcal{F}$ as above.

4 Experiments

We implemented Algorithms 1 and 2 on top of the myND planner [6] that uses LAO* search [3] guided the FF heuristic [4] applied to sampled world states of the belief state to be evaluated. Belief states and transitions between them are represented symbolically using BDDs [1]. Our benchmark domains are POND versions of the IPC domains Blocksworld and FirstResponders as well as the TidyUp domain concerned with a robot tidying up a number of tables in a number of rooms.

Figure 1 shows the runtimes until the final set of observation variables has been found. We used a memory limit of 8 GB and a time limit of 30 minutes per task. The first six instances on the x-axis are our Blocksworld instances, the next 15 are from the FirstResponders domain, and the remaining ones are from the TidyUp domain, sorted by difficulty per domain. We can see that the plan reuse from Algorithm 2 clearly pays off compared to the simple greedy algorithm. Use of functional dependencies leads to little extra time savings, except for some of the harder TidyUp instances. Figure 2 shows the cardinalities of the sets of observation variables $|\mathcal{W}|$ before and $|\mathcal{O}|$ after minimization, with the x-axis as before. In principle, different inclusion minimal sets are incomparable cardinality-wise, but in our experiments the three configurations shown in the figure agreed on the sizes of the sets found. In particular, the problem from Example 1 does not occur here. The initial observation set cardinalities the algorithm starts with are 36 for all Blocksworld instances, between 7 and 39 for FirstResponders, and between 7 and 25 for TidyUp.

We also conducted an experiment measuring the expected numbers of actions and observations necessary to reach a goal using the different intermediate plans π^i, expecting that plans with fewer allowed observations would tend to lead to longer execution sequences due to the need to replace simple, but forbidden

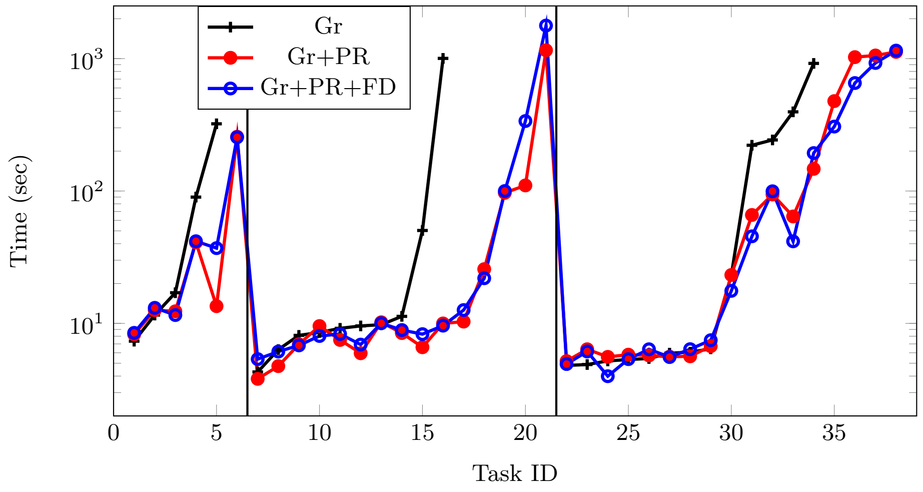


Fig. 1. Overall runtime needed for finding final observation set. Legend: Gr = greedy, PR = plan reuse, FD = functional dependencies.

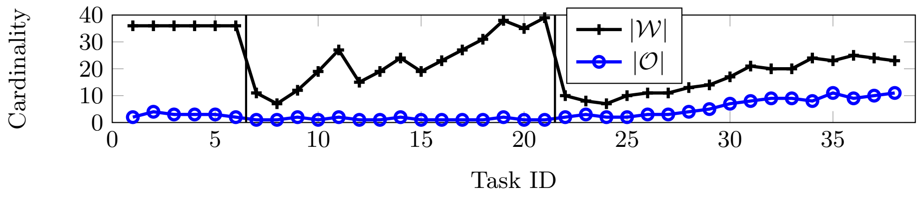


Fig. 2. Cardinalities of the observation sets before and after minimization.

observations by longer observation sequences. In a few instances we saw evidence supporting this (e.g., in one of the Blocksworld instance, the expected plan steps increases from 34 to 65.5), but this is an exception, and across all instances expected execution steps mostly stay the same or increase only minimally.

Finally, it is interesting to see which sets of variables are typically left as observation variables having domain knowledge in mind: In Blocksworld, in most of the runs most of the remaining predicates are instances of the OnTable predicate, followed by occasional uses of the Clear predicate as predicted in the example in the Introduction. Both of these predicates on their own are sufficient. In FirstResponders, we get an even clearer picture: Fires always need to be observed in order to make sure they are eventually extinguished. In all but one FirstResponders instance, nothing else but Fire predicates are observed, the only exception being the only task in which there is no road from a victim's location to a hospital, which necessitates treating him on scene. For that, observing his VictimStatus is identified as being necessary. In the TidyUp domain, we always get one sensing action for the status of the grippers, one for whether each relevant table is clean, one for each relevant door state (open or closed), one for the robot location, and one for each relevant cup on any of the tables.

5 Conclusion and Future Work

We presented an asymptotically optimal greedy top-down baseline algorithm for finding inclusion minimal observation sets and extended that algorithm by reusing plans from earlier iterations and using functional dependencies between observation variables. Our experiments showed superiority of the extended algorithm over the baseline algorithm in terms of runtime, whereas observation variable sets and strong cyclic plans of similar quality are generated.

For future work, we plan to complement the top-down procedure with a bottom-up procedure and to investigate variable ordering heuristics for the iteration over candidate variables for removal (based on an analysis of nondeterministic outcomes of operators, giving preference to more volatile variables). Moreover, we want to study the same problem on the domain instead of planning task level.

Acknowledgments. This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, see `http://www.avacs.org`).

References

1. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers 35(8), 677–691 (1986)
2. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence 147(1–2), 35–84 (2003)
3. Hansen, E.A., Zilberstein, S.: LAO*: A heuristic search algorithm that finds solutions with loops. Artificial Intelligence 129(1–2), 35–62 (2001)
4. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
5. Huang, W., Wen, Z., Jiang, Y., Wu, L.: Observation reduction for strong plans. In: Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI 2007). pp. 1930–1935 (2007)
6. Ortlieb, M., Mattmüller, R.: Pattern-database heuristics for partially observable nondeterministic planning. In: Proceedings of the 36th German Conference on Artificial Intelligence (KI 2013). pp. 140–151 (2013)
7. Rintanen, J.: Complexity of planning with partial observability. In: Proc. 14th International Conference on Automated Planning and Scheduling (ICAPS 2004). pp. 345–354 (2004)
8. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education, third edn. (2010)