Factorised Contingency Planning

Bram Ridder[1], **Michael Cashmore**[2], **Maria Fox**[2], **Derek Long**[2], **Daniele Magazzeni**[2]
Kings College London, London, WC2R 2LS
bernardus.cornelis@kcl.ac.uk[1], *firstname.lastname@kcl.ac.uk*[2]

Abstract

In this paper we consider one of the hardest problems in planning, contingency planning. Recent work has proposed translations for a specific class of contingency planning problems, characterised as *Deterministic POMDPs*, to classical planning problems. This class of contingency planning problems have deterministic actions and observations which makes it feasible to translate them into classical planning problems. This makes it possible to use mature classical planners like FF and Fast Downward to solve contingency planning problems. However, the translations proposed so far do not scale well and the results are not competitive with native contingency planners like POND and CLG. In this paper we improve upon previous translations by factorising the domain based on exploiting mutually independent observation actions. We show that our approach scales better compared to previous offline approaches in domains that are factorisable. For domains that do not factorise well we show that our approach is on-par with previous offline approaches.

1 Introduction

Contingency planning deals with planning problems where the initial state is not fully known and the outcome of actions and observations are non-deterministic. This class of problems is relevant to a wide range of real-life problems. For example, robotic systems have noisy sensors and the outcome of executing an action is uncertain to some degree. The recent interest in incorporating planning systems in robotic systems highlights the need for planning systems that can deal with uncertainty. Unfortunately, this class of problems is known as one of the hardest problems in planning (Rintanen 2004).

Early approaches encoded a contingent planning problem as conjunctive, and disjunctive, normal formulae. For example, Contingent-FF (Hoffmann 2005), POND (Bryce, Kambhampati, and Smith 2006), or the related planners CNF and DNF (To, Pontelli, and Son 2011). One of the largest problems contingency planners face is that they do not *scale* very well due to the representation of the belief state and the generation of the successor states.

In this paper we deal with a *restricted* set of contingent planning problems that can be characterised as *Deterministic POMDPs* (Bonet 2009). Unlike general contingency planning problems these problems have *deterministic* actions and observations. This restricted class of contingency planning problems can effectively be translated into equivalent *classical planning* problems (Albore, Palacios, and Geffner 2009). The encoding of belief states is much easier as all the uncertainty is in the initial state and all observation actions are deterministic. Belief states can be encoded by tracking which possible initial states conform to the outcome of observation actions. Successor states are generated by applying actions to all these states.

Online planners like SDR (Brafman and Shani 2012b) and MPSR (Brafman and Shani 2012a) use translations to reduce the complexity of the original problem by *sampling* a subset of all possible initial states and encoding the relaxed problem as a classical planning problem. The found solution is used as a heuristic for action selection. CLG (Albore, Palacios, and Geffner 2009) maps contingent problems to non-deterministic problems, it translates the latter to a relaxed problem that is solvable by a classical planner. The solutions of the relaxed problems are used an heuristic estimate. CLG can also be used as an *off line* planner, where it finds a complete solution for all possible initial states.

K-Planner (Bonet and Geffner 2011) further restricts the set of contingency planning problems by imposing that variables that are unknown in the initial state do not appear in the body of conditional effects. Contingent problems of this form are simple as they have a bounded contingent width of 1 (Bonet and Geffner 2014), so they can be translated into equivalent fully-observable non-deterministic planning problems that require no beliefs. Its translation is linear in the problem size for restricted contingent planning problems. This translation only works if the state-space is *fully connected*, because the translation assumes that the planner can choose the outcome of observations. It creates a partial solution and if the actual observation does not match the chosen observation during execution a replan is triggered. Subsequent translations like (Palacios, Albore, and Geffner 2014) can find *complete* solutions to the restricted contingent planning problems and do not require the state-space to be fully connected.

PO-PRP (Muise, Belle, and McIlraith 2014) uses an existing translation (Bonet and Geffner 2011) from restricted contingent planning problems to FOND and solves it using a

modified FOND planner (PRP (Muise, McIlraith, and Beck 2012)) to solve it. Their planner and translation scales much better than the previous state-of-the-art planner CLG, albeit on a smaller class of problems. One additional benefit of PO-PRP is that it can detect strong cycles and produce very compact plans.

In this paper we explore the idea of *factorising* the contingency planning problem in order to reduce the space state and increase the scalability of contingency planners. The key idea is that the outcomes of all observations are not all dependent. E.g. in a logistics domain where a driver is unsure about the location of his key for the truck and the location of a package he needs to pick up, it is clear that the location of the key and the whereabouts of the package are independent. In this paper we will explore how we can exploit these independence relationships and how we can extract them from the planning problem.

Factorisation has been used in the context of belief tracking for planning problems with sensing. For example, Causal Belief Tracking (Bonet and Geffner 2014) decomposes a problem into projected subproblems based on sets of variables that are *causally relevant* to each other. Variables are causally relevant to each other if the uncertainty of one affects the uncertainty of the other. Causal Belief Tracking is an orthogonal approach, and trivial in the set of problems we consider. We use the same kind of decomposition in order to to speed up the planning process by factorising the original planning problem.

The rest of the paper is organised as follows. In Section 2 we formally define the contingency planning problem, in Section 3 we explore how we can detect independence relationships, in Section 4 we present our translation of a contingency planning problem by factorising the domain, in Section 5 we present the results of our novel translation and we conclude in Section 6 with a discussion and future work.

2 Contingency Planning

In this section we define the contingency planning problem. We use the PDDL (Ghallab et al. 1998) problem representation used in Palacios et al. (Palacios, Albore, and Geffner 2014) (second translation).

Definition 1 — Problem Representation
A contingency planning problem $P = \langle F, S_0, A_F, A_O, G \rangle$ where F is a set of atoms, S_0 is the set of possible initial states, G is a conjunction over F that represents the goal that needs to be achieved, A_F is a set of actions that affect the world, and A_O is the set of *sensing* actions.

All the uncertainty is encoded in the initial belief state S_0. A belief state is maintained by tracking which facts $f \in F$ are true for each state $s \in S_0$. We denote that a fact f is true in state s as f/s.

Included in F is the proposition $(active)$. This proposition describes whether the state concurs with the sensing actions made so far. Initially all initial states s are *active*, i.e. $(active)/s$, $\forall s \in S_0$. Xf denotes that a fact $f \in F$ is true in all active states $S \subseteq S_0$.

Every action $a \in A_F \cup A_O$ has a precondition $pre(a)$. An action is only applicable if its precondition is satisfied by *all* active states. Every action $a \in A_F$ has a set of effects that are characterised by $a : C \rightarrow E$, where C is a formula that – when satisfied – adds the effect E.

An action $a_f \in A_O$ reveals the true value of the atom $f \in F$. Whenever an observation action a_f is performed, the planning problem is split into two branches. One branch contains all the states $S_\top \subset S_0$ for which f is true, the other branch contains the states $S_\bot = S_0 \setminus S_\top$ in which f is false.

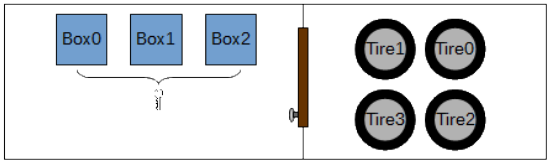
The problem representation stores these branches in a *stack*. Each time a sensing action $a_f \in A_O$ is applied, the states in $S_\bot$ are added to the *stack*, and these states are made inactive. When the goals are achieved in the active states the stack is *popped*. This means that all states in $S_\top$ are now made inactive and all states in $S_\bot$ – that were on the stack – are made active. A stack can be multiple levels deep. This happens when an observation is applied within a branch. The number of levels must be bounded when creating the model.

Included in F is the propositions *(stack number)* and *(level number)*, where *number* is a symbol representing a number. These propostions are used to encode the stack. *(stack number)* denotes that the state is stacked at the depth *number*, while *(level number)* denotes the current size of the stack. Initially the stack is empty, so *X(level 0)*.

For the full description of this translation, see Palacios et al. (Palacios, Albore, and Geffner 2014) (second translation).

A plan for P is a branched plan: every time an observation action is executed the plan branches contingently on the outcome of the observation. A solution to this planning problem requires a simple plan for all branches. Branched plans has been explored in the past, e.g. (Coles 2012) where the branches are contingent on the available resources during execution.

Example 1 *Consider a problem where we are looking for our garage key and we know that it is in one of three boxes. Also we want to fix the flat tire of our car that is in the garage, but we do not know which of the four tires is flat. We can only access the garage by opening the door with the key.*



Example 1 introduces an example problem we will use throughout the paper. There are 12 different states that we believe could be true, i.e. $|S_0| = 12$. These are the Cartesian product of the 3 possibilities for the key location and 4 possibilities for the flat tire. The solution to this problem is to find the key to the garage, open the door, then find the flat tire and fix it.

Note that after finding the key, the remaining plans are

identical. In other words, the location of the key is independent of where we can expect to find the flat tire.

Definition 1 deals with this problem as if the locations of the key and the flat tire are correlated. For example, a branch might correspond to the key in *box 0* and *tire 3* being flat, which is different from the case where we found the key in *box 1* and and *tire 3* is flat. The solution requires a simple plan for each of the 12 branches.

3 Independence Relationships

In this section we define independence relationships between observations actions. In Example 1, we are looking at two independent observations. Firstly we need to find a key, which is in three possible locations. After that we need to find and fix the flat tire of which there are four possible options, creating a possibility space of seven options.

Definition 2 — Independent Observations
Given a contingency planning problem $\langle F, S_0, A_F, A_O, G\rangle$ two observation actions $o_p, o_q \in A_O$ are *independent* if and only if:

$$|S_p|/|S_0| = |S_p \bigcap S_q|/|S_q| = |S_p \bigcap S_{\neg q}|/|S_{\neg q}|$$
$$|S_q|/|S_0| = |S_q \bigcap S_p|/|S_p| = |S_q \bigcap S_{\neg p}|/|S_{\neg p}|$$

where $S_\alpha = \{s \in S_0 \mid \alpha/s\}$.

We also observe transitive dependency relationships; for example, if o_p is not independent of o_q and o_r is not independent of o_q then o_p is not independent of o_r.

Informally, two observations are independent if the outcome of either observation does not inform about the outcome of the other observation. For example, knowing the value of q does not change the probability that p is true.

Referring to Example 1:

$$|S_{(in\,key\,box0)}|/|S_0| \neq |S_{(in\,key\,box0)} \bigcap S_{(in\,key\,box1)}|/|S_{(in\,key\,box1)}|$$

There are 4 total states in which the key is in box 0 out of the 12 states in total: ($|S_{(in\,key\,box0)}|/|S_0| = 1/3$). There are no states in which the key is in both box 0 and box 1: ($|S_{(in\,key\,box0)} \bigcap S_{(in\,key\,box0)}|/|S_{(in\,key\,box1)}|$ = 0). Hence, these observation actions that check if a key is in a box are not independent.

In contrast, knowing the location of the key does not inform us about which tire is flat. For example:

$$|S_{(flat\,tire0)}|/|S_0| = |S_{(flat\,tire0)} \bigcap S_{(in\,key\,box0)}|/|S_{(in\,key\,box0)}| = |S_{(flat\,tire0)} \bigcap S_{\neg(in\,key\,box0)}|/|S_{\neg(in\,key\,box0)}| = 1/4$$

The observation actions to check the state of a tire are independent of those that observe whether a key is in a box.

We introduce the concept of a *dependent observation set* (DOS).

Definition 3 — Dependent Observation Set
Given a contingency planning problem $\langle F, S_0, A_F, A_O, G\rangle$ a *dependent observation set* for an observation action $a \in A_O$ is defined as all observations actions $a' \in A_O$, such that a is not independent of a'.

In order to find the sets of observations that are dependent we compare all pairs of observation actions. Thus, finding the sets of observations that are dependent is $O(|A_O|^2)$.

In Example 1 there are two DOSs: D_{key} ={*(*find key box0), *(*find key box1), *(*find key box2)} and D_{tire}={*(*check tire0), *(*check tire1), *(*check tire2), *(*check tire3)}.

4 Factorised Contingency Planning Problems

In this section we will discuss how we exploit the independence relationships between DOSs. Previous encodings of contingency planning problems into a classical planning problems make the implicit assumption that all observations are dependent. In this section we describe how we factorise the contingency problem such that observations that are part of one *DOS* (e.g. finding the key) are independent from observations that are part of another *DOS* (e.g. finding which tire is flat).

4.1 Partial State Sets

The encoding in Definition 1 enumerates all the possible initial states. However, such an encoding contains a lot of redundancies. Our encoding maps all possible initial states to partial state sets using the found DOSs, thereby reducing the number of states we need to encode.

Definition 4 — Partial State Set
Given the set of possible initial states S_0 and a DOS $D \subseteq A_O$, we define $F_D = \{f \mid o_f \in D\}$ as the set of all the facts that are observable by the observation actions in D. Given a state s, a partial state $s_{F_D} \subseteq s$ is the value of the facts F_D in s. A partial state set is then defined as $S_D = \{s_{F_D} \mid s \in S_0\}$.

Informally, for every DOS (D) we map each possible initial state to a partial state that only contains the facts that can be observed by the observation actions in D. Note that a partial state set does not include duplicate partial states. There are two partial states sets in Example 1:

$$S_{D_{key}} = \{\{(in\,key\,box0)\}, \{(in\,key\,box1)\}, \{(in\,key\,box2)\}\}$$
$$S_{D_{tire}} = \{\{(flat\,tire0)\}, \{(flat\,tire1)\}, \{(flat\,tire2)\}, \{(flat\,tire3)\}\}$$

The partial state $s_0 = \bigcap_{s \in S_0} s$ contains those facts that are true across all possible initial states. The initial belief state can be represented as the Cartesian product of s_0 and S_D, for each DOS D.

The number of partial state sets is usually much smaller than the original number of possible initial states. Returning to Example 1, we map the 12 possible initial states to 8 partial states: s_0, $S_{D_{key}}$, and $S_{D_{tire}}$. In the worse case scenario we find that all observations are dependent, in that case our

method fails gracefully as we fall back on a single partial state set S_0.

4.2 Factorised Problem Representation

In our factorised problem representation the initial state is the partial state set $\{s_0\}$, which is initially *active*. The observable facts are part of partial state sets.

For each pair of partial state sets S_D and S'_D, we define the actions: $(access\ S_D\ S_{D'})$ and $(exit\ S_D\ S_{D'})$, and the facts: *(accessed S_D)* and *(parent $S_D S'_D$)*.

In order to observe facts in these partial state sets, the sets need to be *accessed*. A partial state can only be accessed if and only if a single state is *active*. Accessing a partial state set $S_{D'}$ from the accessed partial state set S_D, makes the current active state $s \in S_D$ *inactive*; updates $S_{D'} = \{s\} \times S_{D'}$; and makes *active* the new set of states in $S_{D'}$. The partial state set S_D becomes the *parent* of $S_{D'}$.

A partial state set can be *exited*. When we *exit* the *accessed* partial state set $S_{D'}$, we make each state $s \in S_{D'}$ *inactive*, and *access* the *parent* partial state set S_D and make active $s \in S_D$. $S_{D'}$ is no longer *accessed*.

We use these new facts and actions in the construction of the Factorised Problem Representation, and describe them in detail below.

Definition 5 — Factorised Problem Representation
Given $\langle F, S_0, A_F, A_O, G\rangle$, the set of partial state sets S, a factorised contingency planning problem is defined as $\langle F', S'_0, A'_F, A_O, G\rangle$ where:

- $F' = F \cup \bigcup_{S_D \in S}((accessed\ S_D) \cup \bigcup_{S_{D'} \in S}(parent\ S_D\ S_{D'}))$
- $A'_F = A_F \cup \bigcup_{S_D, S_{D'} \in S}((access\ S_D\ S_{D'}) \cup (exit\ S_D\ S_{D'}))$
- $S'_0 = \{s_0\}$ where $s_0 = (accessed\ S'_0) \cup \bigcap_{s \in S_0} s$

Definition 5 extends Definition 1 with the *access* and *exit* actions, and associated facts.

The action $(access\ S_D\ S_{D'})$ has the precondition:

$$accessed(S_D) \\ \wedge \bigwedge_{s_i, s_j \in S_D}(\neg(active\ s_i) \vee \neg(active\ s_j))$$

and the effects:

$$\neg accessed(S_D) \wedge accessed(S_{D'}) \\ \bigwedge_{s \in S_D}((active)/s \rightarrow (parent\ S_D\ S_{D'})/s) \\ \bigwedge_{s' \in S_{D'}}(parent\ S_D\ S_{D'})/s \\ \bigwedge_{s \in S_D} \neg(active)/s \\ \bigwedge_{s' \in S_{D'}}(active)/s' \\ \bigwedge_{s \in S_D}(f/s \wedge (active)/s) \rightarrow (\bigwedge_{s' \in S_{D'}} f/s'), \quad \forall f \in F' \\ \bigwedge_{s \in S_D} f/s \rightarrow \neg f/s, \quad \forall f \in F$$

The precondition ensures that a new partial state set can only be *accessed* if a single partial state is currently *active*.

Upon accessing a partial state set, $S_{D'}$, the new partial state set becomes *accessed* and the parent S_D is no longer *accessed*. The fact $(parent\ S_D\ S_{D'})$ is set true in all partial states of $S_{D'}$ and in the single *active* state of S_D.

All facts in which are true in the single *active* partial state of S_D become true in all partial states of $S_{D'}$. Those facts are made false in the *active* state of S_D.

Finally, the single *active* state of S_D is set inactive, and all partial states of $S_{D'}$ become active.

Intuitively, the current knowledge is moved into the partial state set corresponding to one DOS. As in Definition 1, an observation action a_q can only be performed if the current belief state includes an *active* state in which the fact q is false, and some state in which q is true. Therefore, observation actions in the DOS D' can only be performed once the corresponding partial state set $S_{D'}$ has been *accessed* and its partial states made *active*.

The action $(exit\ S_D\ S_{D'})$ has the precondition

$$accessed(S_{D'}) \wedge (parent\ S_D\ S_{D'}) \wedge \\ \bigwedge_{s' \in S_{D'}} \neg\exists_{n \in \mathbb{N}}(stack\ n)/s'$$

and the effects:

$$accessed(S_D) \wedge \neg accessed(S_{D'}) \\ \bigwedge_{s \in S_D \bigcup S_{D'}} \neg(parent\ S_D\ S_{D'})/s \\ (parent\ S_D\ S_{D'})/s \rightarrow (active)/s \quad \forall s \in S_D \\ \bigwedge_{s' \in S_{D'}} \neg(active)/s' \\ ((parent\ S_D\ S_{D'})/s \\ \quad \wedge \bigwedge_{s' \in S_{D'}} f/s') \rightarrow f/s \quad \forall f \in F', \forall s \in S_D \\ (\bigwedge_{s' \in S_{D'}} f/s') \rightarrow (\bigwedge_{s' \in S_{D'}} \neg f/s') \quad \forall f \in F'$$

Intuitively, the *exit* action reverses the *access* action. Making *active* the single partial state s that was previously active in the parent partial state set. Moreover, when exiting a partial state set $S_{D'}$, each fact which is true in *all* states of $S_{D'}$ is made true in the active partial state $s \in S_D$. We can only *exit* a partial state set $S_{D'}$ if and only if there is no state $s' \in S_{D'}$ on the stack.

We use Example 1 to illustrate these actions. As established earlier, there are three partial state sets: $S'_0, S_{D_{tire}}$, and $S_{D_{key}}$. Initially only S'_0 is *accessed* and s_0 is active. The only fact that is true in all possible initial states is $(door\_closed)$. Since the stack is empty, $s_0 = \{(door\_closed), (level\ 0)\}$.

First we must find the key. In order to perform the observations in DOS D_{key}, the partial state set $S_{D_{key}}$ must be *accessed*. $S_{D_{key}}$ contains three partial states. When *accessed* we end up with the following belief state:

$$S_{D_{key}} = \{ \begin{matrix} \{(in\ key\ box0), (active)\} \bigcup F'', \\ \{(in\ key\ box1), (active)\} \bigcup F'', \\ \{(in\ key\ box2), (active)\} \bigcup F'' \end{matrix} \}$$

where

$$F'' = \{ \begin{matrix} (door\_closed), (accessed\ S_{D_{key}}), \\ (parent\ S'_0\ S_{D_{key}}), (level\ 0) \end{matrix} \}$$

In this belief state we can the perform the observation actions in D_{key}. Performing the observation $a_{(in\ key\ box0)}$ yields the following belief state:

$$S_{D_{key}} = \{ \begin{matrix} \{(in\ key\ box0), (active)\} \bigcup F'', \\ \{(in\ key\ box1), (stack\ 0)\} \bigcup F'', \\ \{(in\ key\ box2), (stack\ 0)\} \bigcup F'' \end{matrix} \}$$

where

$$F'' = \{ \begin{array}{l} (door\_closed), (accessed\, S_{D_{key}}), \\ (parent\, S'_0\, S_{D_{key}}), (level\, 1) \end{array} \}$$

The observation action puts on the *stack* all partial states that conflict with the observed fact, $(in\, key\, box0)$. It makes these inactive. The remaining active state contains no uncertainty of the whereabouts of the key, so we can perform $(pickup\, key\, box0)$ to arrive at the belief state:

$$S_{D_{key}} = \{ \begin{array}{l} \{(holding\, key), (active)\} \bigcup F'', \\ \{(in\, key\, box1), (stack\, 0)\} \bigcup F'', \\ \{(in\, key\, box2), (stack\, 0)\} \bigcup F'' \end{array} \}$$

Subsequently we *pop* the stack, rendering the first state not *active* and other states active. Then, we perform the *observation* and *pickup* actions in those states states as above. We end in the following belief state:

$$S_{D_{key}} = \{ \begin{array}{l} \{(holding\, key)\} \bigcup F'', \\ \{(holding\, key)\} \bigcup F'', \\ \{(holding\, key), (active)\} \bigcup F'' \end{array} \}$$

where

$$F'' = \{ \begin{array}{l} (door\_closed), (accessed\, S_{D_{key}}), \\ (parent\, S'_0\, S_{D_{key}}), (level\, 0) \end{array} \}$$

The stack is empty, so we can now *exit* this partial state set. All facts that are true in each state in $S_{D_{key}}$ become true in the parent partial state, s_0. In our example, these facts are: $(holding\_key), (door\_closed)$, and $(level\, 0)$ (we omit the facts $(accessed\, S_{D_{key}})$ and $(parent\, S'_0\, S_{D_{key}})$ as they are deleted by the *exit* action). The current belief state becomes:

$$S'_0 = \{ \begin{array}{l} \{(holding\, key), (door\_closed), (accessed\, S'_0), \\ (level\, 0), (active)\} \end{array} \}$$

Figure 1 illustrates this process.

The door can be opened with the key. With the door open the flat tire can be found and fixed. Note that the branching tree built to pickup the key has collapsed into a single partial state $s_0 \in S'_0$. We know with certainty that, during execution, we will have acquired the key. However, we do not know *how* this key was acquired only that we have it. The remainder of the plan is therefore not contingent of where the key was found.

5 Results

In this section we present results obtained by using our factorisation method. Our baseline is the encoding described in Section 2. We allow 30 minutes per task and 2GiB of memory. We used FF-X (Hoffmann and Nebel 2001) solve both encodings. We expect our encoding to *scale* better at domains that are highly factorisable. Domains that cannot be factorised fail gracefully as we fall back on the previous encoding. We show results for the Contingent Benchmarks CLG Suite [1].

The results are depicted in Figure 2. The number of instances solved per domain are depicted in Table 1.

[1] http://www.ai.upf.edu/software/clg-contingent-planner

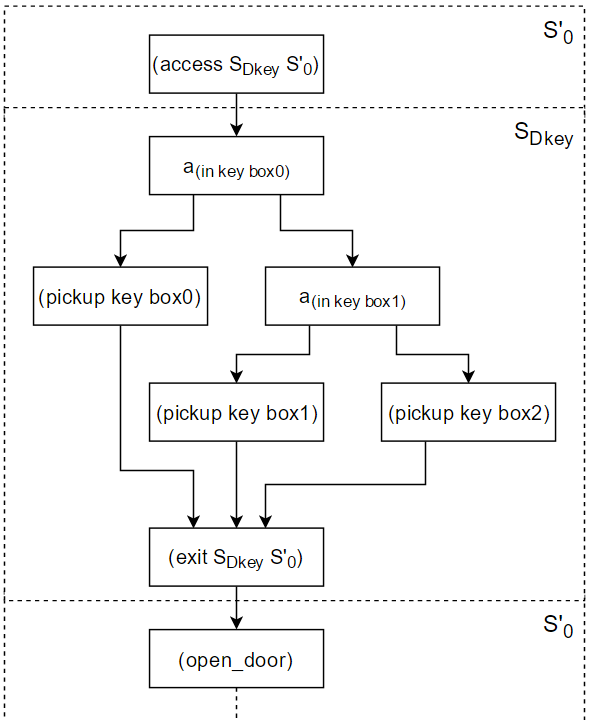


Figure 1: Solution to the example problem. The dashed box shows which partial state set is currently *accessed*.

5.1 Sliding Doors

This domain models a set of corridors where an agent needs to move from the initial corridor to the goal corridor. Every corridor has a set of doors, one of which is open. The planner needs to search for the open door in each corridor. The observation actions for every corridor are dependent, but observation actions for different corridors are independent.

This domain is highly factorisable. In every corridor a single door is open and there are no dependencies between the doors in every corridor. Our encoding creates a partial state set for each corridor that encapsulates the knowledge about which door is open. We created 16 planning instances. The number of corridors ranges from 1 to 4 and the number of doors per corridor ranges from 2 to 5.

As the number of corridors grows, our factorised approach outperforms the original encoding. Furthermore we manage to solve much larger and complex problem instances than the original encoding can deal with.

If there is only a single door in each corridor, our translation is slightly slower due to the additional overhead of our encoding. However, these problem instances are not contingency planning problems, because there is only a single door per corridor. Hence, there is no uncertainty. Our method struggles in cases where there is only a single corridor but the number of doors is very high. This set of problem instances are those for which we find a single DOS as all observation actions are mutually dependent. The original approach outperforms us in these problem instances. However, when the number of doors grow to high the encoding

	Sliding Doors	Logistics	EBTCS	Colour Balls	Find Key
Original translation	9	10	7	1	12
Factorised translation	15	13	20	17	24
Total number of problems	*16*	*18*	*20*	*32*	*25*

Table 1: Number of problems solved per domain.

becomes too large to handle and our approach can solve instances the original encoding could not cope with.

One challenge in our encoding for this problem is the *exit* action. Whenever we exit a partial state set, only those facts that are true in all active states are copied to S_0. The planner needs to make sure that it only exits a partial state set when the location of the agent is equal in all active states. The higher the number of doors, the more states that the planner needs to manage per partial state set. However, as the number of corridors increases we see that our approach scales better.

The number of actions in a plan cannot be compared directly, as we have additional actions in our encoding (*access* and *exit*). A plan for the original encoding is a complete tree where each traversal is a complete plan for one of the states. Whereas in our factorised encoding we have a more compressed plan as we force the planner to converge to a single state whenever we exit the partial state set. For problem instances that are not factorisable, we pay for the overhead and our plans are longer. However, for domains that factorise well, we find shorter plans in less time.

5.2 Logistics

The logistics domain models a number of cities which have a number of locations and an airport. Each city has a number of trucks that can move packages around in the city. For a package to reach another city it needs to be moved to an airport, and transported via an aeroplane to another city.

In this experiment we limit the domain to a single aeroplane and each city has a single airport and a single truck. The number of cities ranges from 1 to 2, the number of packages ranges from 1 to 3, and the number of locations per city ranges from 1 to 3. Creating a total of 18 problem instances.

Each package is hidden in a single city at one of its possible locations. The planner needs to locate each package and deliver it to a destination in a different city. Packages can be located by trucks and aeroplanes, both trucks and aeroplanes can perform sensing actions to check if a package is at the same location the vehicle.

If there is only a single location per city, the problem is not a contingency problem as each packages can only be at a single location. In the original encoding this problem collapses in a single state, making it very easy to solve. However, in our encoding each package is part of its own *partial state set* which makes the problem much harder to solve. For example, a problem instance where there are two cities, each consisting of a single locations, and three packages per city (six total) our encoding has seven partial state sets (one for each package in addition to the initial state), whilst the original encoding collapses the entire problem in a single state.

For problem instances with more locations, our encoding solves more problem instances and does it quicker. However, we do not observe the improvement we expected. Whilst the problem is highly factorisable the planner struggles to exploits our encoding. Like the previous problem, the *exit* and *access* partial state sets seem to be the bottleneck. We will return to this discussion in our conclusions.

5.3 EBTCS

This domain encodes a bomb diffuse problem. To diffuse a bomb, we must find in which package it is hidden and subsequently flush it down the toilet.

In order to test whether our method scales better we have varied the number of bombs and the number of packages each bomb can be hidden in. The number of bombs ranges from 1 to 5 and the number of packages ranges from 2 to 5. For a total of 20 problem instances. We have limited the number of toilets in each problem instance to one.

Our method clearly scales better as the number of bombs and packages increase. In fact we manage to solve all the benchmark problems. The largest problem instance the previous method manages to solve contains five bombs that can be hidden in two different packages. This problem instance has $2^5 = 32$ possible initial states. In contrast, our method manages to solve the problem instance that has five bombs hidden in five packages which has $5^5 = 3125$ possible initial states. In our encoding, we encapsulate the whereabouts of each bomb in a partial state set, this results in 5 partial state sets, each consisting of 5 partial states. This reduces our encoding to $25 + 1$ states.

5.4 Colour Balls

The colour balls domain is an interesting domain as there are two hidden facts per ball, its location and its colour. Both needs to be discovered before we can dispose the ball in the appropriate garbage container. In our encoding we have as many garbage containers as there are colours, each garbage container accepts one unique colour of balls. Furthermore, we made sure that all locations are fully connected.

The number of balls ranges from 2 to 3, the number of locations ranges from 2 to 5, and the number of colours ranges from 2 to 5. In total there are 32 problem instances.

Our method does exceptionally well, solving over double the number of problem instances compared to the previous method. For most problem instances the number of states is too large for the previous method to cope with. For example, the problem instances where there are three balls, three locations, and three colours yields $3^{(3^3)} = 19683$ states. Our encoding reduces the number of states that need to be encoded in the partial state sets down to $3 * (3 + 3) + 1 = 19$. Two

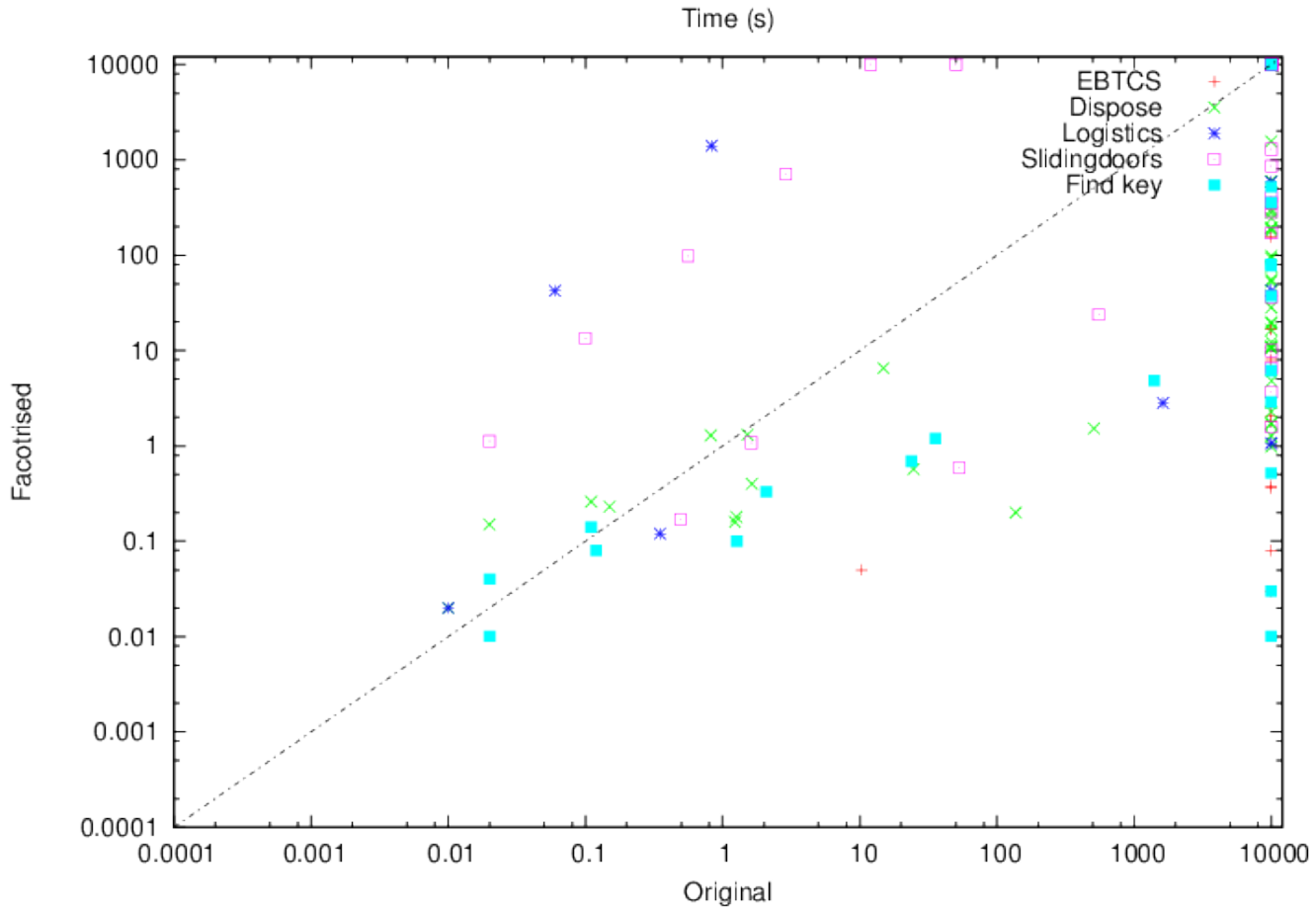


Figure 2: Scatterplot of the time it took (in seconds) to solve problem instances. Values of 10.000 denote instances that were not solved. We omitted results neither translation could find a solution to.

partial state sets per ball, one that encapsulates the colour a ball has and the other encapsulates the location of where the ball is plus one for the initial state.

5.5 Find Key

The final problem we explore is the *find key* problem that we have used as a rolling example througout the paper. The number of boxes and tires range from 1 to 5 for a total of 25 problem instances.

Given the set of keys k and the set of locations l the number of states that are created in the original encoding is equal to $\mid l \mid^{\mid k \mid}$. In our encoding the number of partial states is equal to $\mid l \mid + \mid k \mid +1$. As the number of locations and keys increases our encoding to performs better as we have to deal with less partial states.

Our encoding solves all problem instances, except the problem instance with 5 keys and 5 tires.

6 Discussion and Future Work

In this paper we have presented a novel translation method that translates a contingency planning problem into a classical planning problem. Our method factorises the contingency planning problem by finding dependent observation sets, each set consists of actions whose observations are dependent. We presented an efficient algorithm to find these sets and presented how these sets are turned into partial state sets and how they are exploited in our translation.

Our translation is based on a previous encoding that enumerates all possible states and encodes a stack mechanism in the planning domain to cope with uncertainty. The mapping we use to construct the partial state sets greatly reduces the number of states we need to encode for problem instances that are highly factorisable. For example, consider a problem instance for the *find key* domain where we have five locations and four keys. The previous method encodes 625 different states, whereas we only encode 21 partial states.

The previous encoding creates a plan that branches every time an observation is made. These plans can be exponential in the number of observations. This is one of the reasons why this method does not scale very well. In contrast we reduce the encoding by splitting the possible initial states into partial state sets. Every time we exit *exit* a partial state set, the number of states the planner need to consider collapses

into a single state. Branches only occur in the context of a partial state set.

Our method manages to scale better and can solve larger problem instances faster. However, the encoding of a contingency planning domain into a classical planning problem is not flawless. We rely on the planner to find a branched plan in the context of partial state sets such that – when we exit that partial state set – all the facts that we require are identical for all states in the active states. That is why, as the number of states in a partial state set becomes too large, we struggle to find plans or take longer than the original approach.

In future work we will explore the possibility of improving performance by moving the machinery that tackles the contingent nature of the problem inside the planner. We hypothesise that such a planner can find solutions faster, and allow us to construct better heuristics. Whilst the RPG heuristic proved sufficient for the domains explored in this paper we want heursitics that are not agnostic to the contingent nature of the planning problem. For example, when FF calculates the heuristic values of successor states to the current state, it is unaware that inactive states on the stack have no influence on the current branch.

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*.

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three*, 1936–1941. AAAI Press.

Bonet, B., and Geffner, H. 2014. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res. (JAIR)* 50:923–970.

Bonet, B. 2009. Deterministic pomdps revisited. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, 59–66. Arlington, Virginia, United States: AUAI Press.

Brafman, R. I., and Shani, G. 2012a. A multi-path compilation approach to contingent planning. In Hoffmann, J., and Selman, B., eds., *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press.

Brafman, R. I., and Shani, G. 2012b. Replanning in domains with partial information and sensing actions. *Journal of Artificial Intelligence Research* 45:565–600.

Bryce, D.; Kambhampati, S.; and Smith, D. E. 2006. Planning graph heuristics for belief space search. *J. Artif. Int. Res.* 26(1):35–99.

Coles, A. J. 2012. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI-10)*.

Ghallab, M.; Isi, C. K.; Penberthy, S.; Smith, D. E.; Sun, Y.; and Weld, D. 1998. Pddl - the planning domain definition language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Hoffmann, J. 2005. Contingent planning via heuristic forward search with implicit belief states. In *In Proceedings of ICAPS05*, 71–80. AAAI.

Muise, C. J.; Belle, V.; and McIlraith, S. A. 2014. Computing contingent plans via fully observable non-deterministic planning. In Brodley, C. E., and Stone, P., eds., *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 2322–2329. AAAI Press.

Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. Improved non-deterministic planning by exploiting state relevance. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*. AAAI.

Palacios, H.; Albore, A.; and Geffner, H. 2014. Compiling contingent planning into classical planning: New translations and results. *Models and Paradigms for Planning under Uncertainty: a Broad Perspective* 65.

Rintanen, J. 2004. Complexity of planning with partial observability. In *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 345–354. AAAI Press.

To, S. T.; Pontelli, E.; and Son, T. C. 2011. On the effectiveness of cnf and dnf representations in contingent planning. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, 2033–2038. AAAI Press.