A Multi-Path Compilation Approach to Contingent Planning

Ronen I. Brafman
Department of Computer Science
Ben-Gurion University of the Negev
brafman@cs.bgu.ac.il

Guy Shani
Information Systems Engineering
Ben-Gurion University of the Negev &
Deutsche Telekom Laboratories
shanigu@bgu.ac.il

Abstract

We describe a new sound and complete method for compiling contingent planning problems with sensing actions into classical planning. Our method encodes conditional plans within a linear, classical plan. This allows our planner, MPSR, to reason about multiple future outcomes of sensing actions, and makes it less susceptible to dead-ends. MPRS, however, generates very large classical planning problems. To overcome this, we use an incomplete variant of the method, based on state sampling, within an online replanner. On most current domains, MPSR finds plans faster, although its plans are often longer. But on a new challenging variant of Wumpus with dead-ends, it finds smaller plans, faster, and scales better.

Introduction

Agents acting under partial observability must acquire information about the true state of the world using sensing actions to achieve their goals. Such problems can be modeled using contingent planning, where action effects are conditioned on some unknown world features. Contingent planning is difficult because the agent plan must branch given different world states, resulting in potentially large plan trees.

The translation (or compilation) approach to conformant planning (Palacios and Geffner, 2009) compiles a problem of planning in belief-space into a classical planning problem in which the planner "reasons" about knowledge explicitly. It works by adding new "knowledge" propositions and modifying actions so that the state space is transformed into a belief space, essentially allowing a classical planner to plan in belief space. This reduction allows us to leverage advances in classical planning, such as recent, powerful heuristic generation methods, within a conformant planner. This is particularly appealing given the difficulty in generating good and fast heuristic estimates directly in belief space.

The compilation approach motivated a number of recent contingent planers that combine similar transformations with replanning or heuristic search, performing much better than previous methods (Albore, Palacios, and Geffner, 2009; Shani and Brafman, 2011; Bonet and Geffner, 2011). Classical and conformant planning differ substantially from contingent planning. The former have a single possible execution path (in state or belief space, respectively), whereas the latter has multiple possible execution paths because of the possibility of different possible sensor inputs. Depending on the sensed values, unknown to the agent at planning time, its knowledge at execution time will be different, and its behavior should be appropriate to this knowledge.

Because of this fundamental difference, existing contingent planners utilize problem compilation in a limited way. More specifically, they are based on the idea of replanning (Yoon, Fern, and Givan, 2007), where in each replanning phase, a classical planning problem that operates in a representation of the belief space is generated. To generate a deterministic, classical planning problem, planners must determinize the effects of sensing actions, somehow. Planners like CLG (Albore, Palacios, and Geffner, 2009) and K-planner (Bonet and Geffner, 2011) make optimistic assumptions about the sensed values (i.e., the planner essentially selects what values are sensed), whereas SDR (Shani and Brafman, 2011) samples an arbitrary initial state s_I and assumes that observations will be sensed as if s_I is the true initial state. Consequently, the planner does not consider future execution paths in which the observations will be different. This limited foresight is somewhat mitigated by the use of replanning, as the planner reconsiders its future plan whenever new information arrives. However, until such new information arrives, it typically executes a number of actions, that while appropriate under the assumptions it made, could be bad, or even catastrophic (i.e., lead to a dead-end) under other, possible assumptions.

The main contribution of this paper is a new sound and complete compilation method, called multi-path translation, that addresses these issues. The new compilation scheme generates a classical, deterministic planning problem whose solutions encode a true contingent plan that considers all execution paths. This method can be used offline to generate a complete, contingent plan that reaches the goal in all circumstances,

when such a plan exists.

Unfortunately, the size of such complete plans, and, more importantly, the size of the classical planning problem generated using the complete method can be linear in the number of initial states, and hence, exponential in the number of propositions. To address this problem, we resort to a replanning architecture, and use a modified, incomplete version of this translation scheme at each stage. Specifically, to control the size of the classical problem generated, much like SDR, we sample a subset of the initial states, and ignore all others. By improving the sampling techniques of SDR, we select diverse states, covering diverse future trajectories.

Two key ideas underlie the multi-path translation technique. The first idea is to use conditional knowledge, and the notion of distinguishability, both rooted strongly in the logics of knowledge (Fagin et al., 1995). More specifically, we maintain information about the knowledge of the agent at run-time, conditioned on its initial state. In deterministic environments, we can accurately know what the agent's state of knowledge will be at run-time, if we know the initial state. We keep track of which states it can distinguish between, effectively capturing its belief state. From the agent's belief state, we can deduce all of the agent's knowledge.

The second idea is to enhance the set of actions so that we can encode a contingent plan, typically represented as a tree, within a linear, classical plan. To do this, we introduce multiple versions of each action for different belief states. An action a_b that corresponds to an original action a and a belief state b will have the same effect as a on states that are in b, and no effect on states outside b. A similar technique was used by Bonet, Palacios, and Geffner (2009) (BPP), the only other compilation-based planner that generates conditional plans. BPP generates finite-state controllers, which have the added advantage of generality, i.e., a plan for one maze may work for different mazes and different initial states. On the other hand BPP does not handle sensing actions, and assumes a fixed set of observable variables, sensed passively following each action. Thus, the specifics of the two methods differ substantially.

Our Multi-Path Sampling Replanner (MPSR), was evaluated empirically on old and new contingent planning domains. Our experiments show that MPSR is typically faster than SDR and CLG, although the quality of its plans is lower (i.e., longer). However, these domains do not contain dead-ends. When we include dead-ends in the Wumpus domain, we see the true strength of MPSR. SDR is completely unable to handle this domain, and while CLG can handle the smaller instances, MPSR scales much better, and generates smaller plans, faster.

Partially Observable Contingent Planning

Partially observable contingent planning problems are characterized by uncertainty about the initial state of the world, partial observability, and the existence of sensing actions. Actions may be non-deterministic, but much of the literature focuses on deterministic actions, and in this paper we will assume deterministic actions, too.

Problem Definition

A contingent planning problem is a quadruple: $\pi = \langle P, A, \varphi_I, G\rangle$. P is a set of propositions, A is a set of actions, φ_I is a formula over P that describes the set of possible initial states, and $G \subset P$ is the goal propositions. We often abuse notation, treating a set of literals as a conjunction of the literals in the set, as well as an assignment of the propositions in it. For example, $\{p, \neg q\}$ will also be treated as $p \wedge \neg q$ and as an assignment of true to p and false to q.

A state of the world, s, assigns a truth value to all elements of P. A belief-state is a set of possible states, and the initial belief state, $b_I = \{s : s \models \varphi_I\}$ defines the set of states that are possible initially. An action $a \in A$ is a three-tuple, {pre(a),effects(a),obs(a)}. pre(a) is a set of literals denoting the action's preconditions. effects(a) is a set of pairs (c, e) denoting conditional effects, where c is a set (conjunction) of literals and e is a single literal. Finally, obs(a) is a set of propositions, denoting those propositions whose value is observed when a is executed. We assume that a is well defined, that is, if $(c, e) \in$ effects(a) then $c \wedge$pre(a) is consistent, and that if both $(c, e), (c', e') \in$ effects(a) and $s \models c \wedge c'$ for some state s then $e \wedge e'$ is consistent. In current benchmark problems, either the set effects or the set obs are empty. That is, actions either alter the state of the world but provide no information, or they are pure sensing actions that do not alter the state of the world, but this is not a mandatory limitation.

We use $a(s)$ to denote the state that is obtained when a is executed in state s. If s does not satisfy all literals in pre(a), then $a(s)$ is undefined. Otherwise, $a(s)$ assigns to each proposition p the same value as s, unless there exists a pair $(c, e) \in$ effects(a) such that $s \models c$ and e assigns p a different value than s. Observations affect the agent's belief state. We assume throughout that all observations are deterministic and accurate, and reflect the state of the world prior to the execution of the action. It is possible to have observation reflect the post-action state, at the price of slightly more complicated notation. Thus, if $p \in$obs(a) then following the execution of a, the agent will observe p if p holds now, and otherwise it will observe $\neg p$. Thus, if s is the true state of the world, and b is the current belief state of the agent, then $b_{a,s}$, the belief state following the execution of a in state s — $b_{a,s} = \{a(s') | s' \in b, s'$ and s agree on obs(a)$\}$ — corresponds to the progression through a of all states in the old belief state b that assign the propositions in obs(a) the same values as s does.

Contingent Plans

A plan for a contingent planning problem is an annotated tree $\tau = (N, E)$. The nodes, N, are labeled with actions, and the edges, E, are labeled with observations. A node labeled by an action with no observations has a single child, and the edge leading to it is labeled by the null observation true. Otherwise, each node has one child for each possible observation value. The edge leading to this child is labeled by the corresponding observation.

A plan is executed as follows: starting at the root, we execute the action of the current node. If this action has no observation, we move to its (only) child. Otherwise, we move along the edge labeled by the observed value to the appropriate child. We continue recursively until a leaf has been reached. As we assume that actions and observations are deterministic, there is a single possible execution path along this tree for each initial state. We use $\tau(s)$ to denote the state obtained when τ is executed starting in state s. τ is a solution plan (or plan) for π if $\tau(s) \models G$ for every $s \in b_I$.

A belief state can be associated with each node in the tree, i.e., the set of possible states when this node is reached during run-time. b_I is the belief state associated with the root. If b is the belief state associated with node n labeled by a, and n' is a child of n connected with an edge labeled by φ, then the belief state associated with n' will be $\{a(s) : s \in b, s \models \varphi\}$.

Using the belief annotation, we can represent a contingent plan, τ, in linear form. We topologically sort the actions in the plan, and annotate each action instance a with the belief state, b_a, associated with a. The semantics of this plan is similar, but slightly different from that of a classical plan. Specifically, the effect and observation of action a with associated belief state b is identical to a if the current belief state is b, and is null otherwise. That is, if b is the current belief state, then upon reaching an action annotated by a belief state other than b, that action is skipped. Below we explain how a classical planner can generate such plans.

We illustrate these ideas using a 4×4 Wumpus domain (Albore, Palacios, and Geffner, 2009), which will serve as our running example. Figure 1(a) illustrates this domain, where an agent is located on a 4×4 grid. The agent can move in all four directions, and if moving into a wall, it remains in place. The agent initially is in the low-left corner and must reach the top-right corner. There are two monsters called Wumpuses hidden along the grid diagonal, the agent knows that each Wumpus is hiding in one of two possible locations, but must observe its stench, which carries to all adjacent locations, in order to deduce its whereabouts. The possible states can be characterized by the whereabouts of the Wumpuses — in both lower locations (denoted dd for down-down), in both upper locations (denoted uu for up-up), lower location first and then upper location (du), and upper location first and then lower location (ud). Figures 1(b) shows a possible plan tree for the Wumpus domain, and Figure 1(c) shows a possible linearization of this plan.

Knowledge and Its Evolution

The belief state of the agent is what, in the traditional literature, is referred to as its set of possible worlds, or knowledge state (Fagin et al., 1995). The knowledge of the agent corresponds to the facts that hold in all its possible worlds. Essentially, this is the information it can deduce from its knowledge about the set of possible initial states of the world, the actions it executed, and the observations it observed.

If some formula φ is true in all states in the agent's current belief state, we say that it knows φ, denoted $K\varphi$. Initially, the agent's belief state, the set of currently possible states, is b_I. As it attains information, the size of this set decreases, because we can rule out initial states that are inconsistent with its observations. Initial states that were ruled out due to inconsistency with some observation can never become "possible" again. If one maintains the initial belief state, and updates this representation each time a new observation is obtained (Shani and Brafman, 2011), then the uncertainty over the initial belief and hence over the current belief state, i.e., the number of possible initial and current states given the observations, can only get reduced[1].

A major difficulty of contingent planning is that we cannot predict, at planning time, what will be observed following a sensing action. Thus, we cannot predict the agent's state of knowledge at run-time. At planning time, though, useful knowledge may be gained and leveraged.

First, if we know what actions were executed, we know whether the agent will know the value of certain facts. Specifically, if the agent executes an action that observes the value of p, we know that, afterwards, it will know the value of p, even though we do not know whether p or $\neg p$ will be observed. This knowledge can be represented by propositions of the form KWp (know-whether p). Second, we can maintain specific information about the knowledge that the agent will have following the execution of an action sequence, conditioned on state s being the true initial state. If s is the initial state, in a deterministic environment, we know what observations the agent will make, and consequently, its state of information at run-time. In particular, given that s is the true world state, we know which states it will be able to distinguish from s (because they would induce different observations), and which states it will not be able to distinguish from s (because they would yield the same observations as s). Hence, we know the agent's future belief states, given that s is the true initial state.

We assume here that for each pair of initial states s_1, s_2, either there exists some plan τ resulting in an

[1] This is true only for deterministic domains. When actions have non-deterministic effects, the uncertainly may grow.

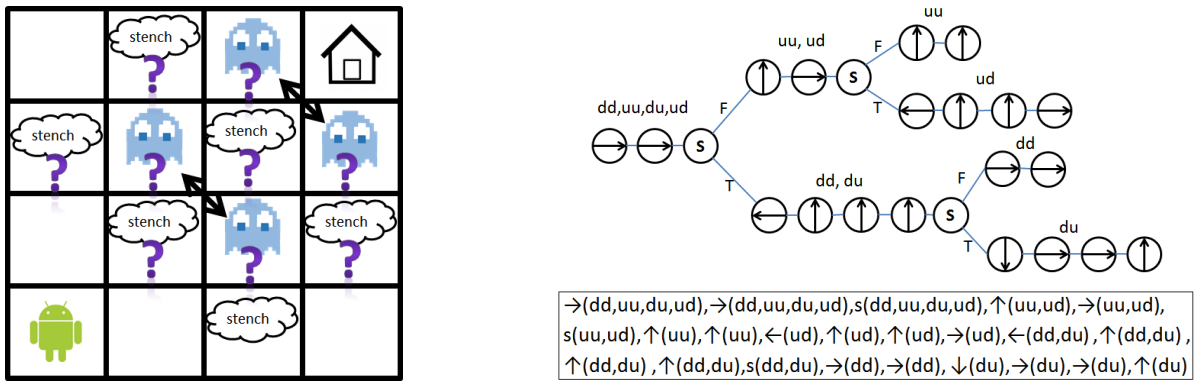

Figure 1: The 4×4 Wumpus domain. A plan tree, with arrows denoting movement actions, S denoting sensing for stench, and outgoing edges marked by T or F (stench was observed or not). Branches are associated with the current possible belief state, i.e. set of possible states. A possible linearization of the plan tree. Actions are associated with the adequate belief state.

observation o that distinguishes between the states, or there exists some plan τ' that is applicable in both initial states and obtains the goal. Otherwise, no contingent plan solves the problem.

The Multi-Path Translation

We now suggest a method for translating contingent planning problems into classical planning problems whose solution is (essentially) a contingent plan in linear form. There are two elements to this translation: a representation of the knowledge state of the agent, and an enhanced actions set.

For representing knowledge we need 3 types of propositions: propositions that represent the current value of world features, conditioned upon an initial state, propositions that represent whether the agent currently knows the value of world features, given an initial state, and propositions that allow the reasoning about the belief state. For the latter, we maintain propositions that define whether two states s and s' are distinguishable, i.e., whether the agent has observed a proposition whose value is different in the two states.

We enhance the set of actions by adding, for each action a and every possible belief state b', an action $a_{b'}$, that applies only if the current belief state is b'. The number of actions thus increases substantially, and we discuss practical solutions later.

Given the input problem $\pi = \langle P, A, \varphi_I, G\rangle$ and current belief state b (at first, $b = b_I$), we generate a classical planning problem $\pi_c = \langle P_c, A_c, I_c, G_c\rangle$ as follows:

Propositions $P_c = \{p/s : p \in P, s \in b\} \cup \{KWp/s : p \in P, s \in b\} \cup \{KW\neg s/s' : s, s' \in b\}$.

1. Propositions of the form p/s capture the value at run time of p when s is the true initial state.
2. Propositions of the form KWp/s capture the knowledge at run time regarding the value of p when s is the true initial state. If KWp/s holds, then the agent knows p's value if s is the true initial state (and that value is the value of p/s).
3. When ϕ is a conjunction of literals, we write ϕ/s as a shorthand to $\bigwedge_{p\in\phi} p/s$, and we write $KW\phi/s$ as a shorthand to $\bigwedge_{p\in\phi} KWp/s$.
4. Propositions of the form $KW\neg s'/s$ denote that at run-time, if s is the true initial state, then the agent has gathered sufficient data to conclude that s' cannot be the true initial state. These propositions allow us to define the belief state during execution. $KW\neg s'/s$ is reflective, i.e. $KW\neg s'/s$ iff $KW\neg s/s'$.

Actions For every action $a \in A$, and every subset of $S' \subseteq b$, A_c contains an action $a_{S'}$. This action denotes the execution of a when the agent's belief state is S'. $a_{S'}$ has no effect on states outside S'! It is defined as follows:

$\text{pre}(a_{S'}) = \{KWp/s \wedge p/s : s \in S', p \in \text{pre}(a)\} \cup \{KW\neg s'/s : s' \in S', s \in b \setminus S'\}$. That is, the agent must know that the preconditions are true prior to applying the action in all states for which this action applies, and it must be able to distinguish between any state in S' and every state not in S'. Thus, the agent can execute $a_{S'}$ only when it knows that the current belief state is S', and all action preconditions are known to hold in S'.

For every $(c, e) \in \text{effects}(a)$, $\text{effects}(a_{S'})$ contains the following conditional effects:

1. For each $s \in S'$, $(c/s, e/s)$ — the effect applied to every state in S'.
2. $(\bigwedge_{s\in S'} KWc/s \wedge c/s, \bigwedge_{s\in S'} KWe/s)$ — if we know that the condition c holds prior to executing a in all states for which the action applies, we know whether its effect holds following a.
3. $(\bigvee_{s\in S'} \neg KWc/s, \bigwedge_{s\in S'} \neg KWe/s)$ — if we do not know whether the condition c holds prior to executing a in some of the states for which the action applies, we may not know whether its effect holds following a. This is a subtle point which we discuss later.

4. $\{KWp/s : p \in \text{obs}(a), s \in S'\}$ — for every observable p, we would know its value in all the states for which the action applies.
5. $\{(KWp/s \wedge KWp/s' \wedge p/s \wedge \neg p/s', KW\neg s/s')\}$ — for every observable $p \in obs(a)$, and every two states $s, s' \in S'$, if we know the value of p in the two states, and both states disagree on p, then we can distinguish between the states at runtime.

In addition, for each literal l (w.r.t. P) and each subset of states $S' \subseteq b$ we have a merge action that allows us to gain knowledge:

- $\text{pre}(\text{merge}(l, S')) = \{\bigwedge_{s \in S'} l/s\} \wedge \{\bigwedge_{s' \in S', s \in b \setminus S'} KW\neg s/s'\}$ — the merge can be used when all states in S' agree on the value of l, and we can distinguish at run time between the states in S' and the rest of the states.
- $\text{effects}(\text{merge}(l, S')) = \{KWl/s : s \in S'\}$ — the effect is that we now know whether l holds in all these states.

Initial State $I_c = \bigwedge_{s \in b, s \models l} l/s$ — for every literal we specify its value in all states.

Goal $G_c = \{\bigwedge_{s \in b} G/s\}$ — we require that the goal will be achieved in all states.

The reader may have observed that the merge actions are powerful. If we correctly keep track of the value of every proposition given every possible initial state, and the value of propositions of the form $KW\neg s'/s$, we can infer any propositions of the form KWp/s that is currently true. Keeping track of basic (p/s) propositions is easy, and keeping track of $KW\neg s'/s$ propositions is also easy: they never become false — new observations can only cause us to distinguish between more states. This implies that, in principle, we can offer an even simpler, sound and complete translation, in which none of the non-merge actions adds knowledge about the value of the propositions (i.e., propositions of the form KWp/s), and all such knowledge is removed after every action. However, this would require much longer plans that contain many merge actions, in order to recover the lost knowledge. Instead, we conservatively update the agent's knowledge via regular actions. Any missing information, can be deduced with the merges. These observations underlie the soundness and completeness result below.

It is always beneficial to apply merge actions when possible, because merge actions are internal actions which can be considered to be costless for the agent. Thus, it is better to acquire information through merge actions rather than through sensing actions. In fact, many planners support axioms, which are automatically executed whenever possible, which is suitable for the merge actions.

Theorem 1. π has a contingent plan IFF π_c has a plan.

Proof. (outline) Having established the relation between plan trees and their linearizations, the main observations upon which this result builds are (1) The classical plan correctly maintains the actual state of the world for every possible initial state (using the propositions p/s). (2) The classical plan correctly and completely tracks the set of distinguishable state (i.e., propositions of the form $KW\neg s'/s$). (3) One can use the merge actions to deduce any valid proposition of the form KWp/s. (4) All conclusions of the form KWp/s added by a non-merge action are sound. □

Algorithm 1 MPSR

```
Input: Contingent Planning Problem: π = ⟨P, A, φ_I, G⟩, Integer: size
  b :=initial belief state
  changed := false
  while G is not known at the current belief state do
    Sample a set of states S'_I consistent with b s.t. |S'_I| ≤size
    ρ := (linear) solution to ⟨P, A, S'_I, G⟩ using the multi-path translation.
    if ρ is empty then
      return failure
    end if
    while ρ ≠ ∅ and not changed and pre(first(ρ)) is known at the current belief state do
      a :=first(ρ)
      execute a, observe o
      b_new := {a(s) : s ∈ b and s ⊨ o}
      changed := [b ≠ b_new]
      Remove a from ρ
    end while
  end while
```

Translations Linear in $|b|$ The above translation adds a number of actions exponential in the number of initial states, and potentially, doubly exponential in the number of propositions. However, there exists an alternative translation that, like previous complete translations for conformant planning, is linear in the number of initial states, and hence only worst-case exponential in the number of propositions. We define one action a_s for each original action a and possible initial state s. This action a_s applies a to all states that are currently indistinguishable from state s', where s' is the state obtained by applying the current plan prefix to initial state s. This translation exploits the fact that the number of possible belief states after each plan prefix is at most $|b_I|$. Yet, although this translation introduces a linear number of new actions, these actions require many conditional effects. Such actions are challenging for current classical planners. Thus, given the fact that after sampling (Section) we consider only a small number of initial states, we use the former translation, which is more effective in practice.

The Multi-Path Planning Algorithm

The multi-path translation allows us, in principle, to solve any contingent planning problem with a classical planner. However, this approach is impractical for two reasons. First, complete plans (trees) are very large. Generating them offline is, thus, difficult, and recent

work has moved towards online planning, in which the planner generates a single execution sequence. Second, our translation is linear in the number of possible initial states, and hence exponential in the number of propositions — due to the need to maintain propositions for each possible world state. One can slightly improve the situation by using tags, denoting sets of states, rather than actual states, as introduced by Palacios and Geffner (2009), but even with tags the size of the translation can be unmanageable (Shani and Brafman, 2011). Instead, we take a sampling and replanning approach in the spirit of the SDR planner (Shani and Brafman, 2011).

Replanning with MPSR

MPSR (for multi-path, sampling, replanner) is an online contingent planner that uses a replanning approach. MPSR maintains the agent's current belief state using the lazy method introduced in SDR. At each iteration MPSR generates a multi-path translation using only a small subset of b_I — typically of size no greater than 4. The resulting plan is typically not a solution plan because some possible initial states were ignored. However, it is much more informed than plans generated using single-execution path methods, such as SDR. MPSR executes the plan until either a new observation is made, which alters the belief state, or the next action is not safe to execute because its preconditions do not hold in one of the possible states. Then, it replans again. The high level scheme of MPSR is described in Algorithm 1. size controls the size of the set of world states S'_I.

For belief maintenance and update we use the lazy regression technique suggested by Shani and Brafman (2011), where only the initial belief state is maintained, and queries concerning possible current states are resolved by regressing the query through the action-observation history.

Sampling and Resampling

To improve the value of the plan generated by multi-path translation, we would like to sample a diverse set of states. There are different ways to define diversity in this context. Our sampling algorithm (Algorithm 2) seeks to find a set S of possible initial states of predefined size, such that the number of propositions p that attain different values on state in S is maximized (i.e., there exists states $s', s'' \in S$ such that $s' \models p$ and $s'' \models \neg p$).

We use the following technique to obtain a diverse sample (Algorithm 2): We sample an unknown proposition, assign it a value, and propagate this assignment through the belief state constraints, setting the values of other dependent propositions. We continue to sample this way until all propositions have been assigned. For the rest of the states, whenever we need to pick a proposition to assign, we seek a proposition that was assigned uniformly in all samples, so far. We assign it the complementary value, propagate its value, and continue. If all remaining literals have diverse values in the states assigned so far, we continue sampling arbitrary unassigned propositions, as before. Diverse sampling is an important component of MPSR, but given the small sample size that we use, we may not be able to cover all aspects of the initial state. To get a more informed plan, we use resampling. That is, we generate m initial state sets $S_1, S_2, ..., S_m$, all of which contain only a small number of states. We then plan for each S_i independently, resulting in a plan π_i. As computing m plans with a sample size n is (much) faster than solving a single plan with sample size $n+1$, this approach scales better than increasing the sample size. Then, we select the plan that applies a sensing action the earliest. This usually indicates an awareness to conflicting needs that can be resolved by sensing.

Algorithm 2 Diverse-Sampling

```
Input: φ — the constraints over the belief state, size
  P ← all unknown predicates in φ
  S ← ∅, P' ← P, φ' ← φ, s ← ∅
  while P' ≠ ∅ do
     Choose p ∈ P', l ∈ {p, ¬p} uniformly
     Propagate l through φ'
     if φ' is solvable then add l to s else backtrack
     P' ← all unknown predicates in φ'
  end while
  Add s to S, s ← ∅
  while |S| < size do
     P' ← P, φ' ← φ
     while P' ≠ Φ do
        Choose p ∈ P', l ∈ {p, ¬p} s.t. l does not appear in S. If no
        such l exists, pick p, l uniformly.
        Propagate l through φ'
        if φ' is solvable then add l to s else backtrack
        P' ← all unknown predicates in φ'
     end while
     Add s to S
  end while
  return S
```

Empirical Evaluation

We now compare MPSR to CLG (Albore, Palacios, and Geffner, 2009) and SDR* — the best variation of SDR (Shani and Brafman, 2011), on benchmarks from the SDR paper. These are currently the best contingent planners. We evaluate MPSR using a single sample, denoted MPSR, and MPSR using resampling with 5 samples, denote MPSR ×5. MPSR always uses two sampled states. The experiments were conducted on a Windows Server 2008 machine with 24 2.66GHz cores (although each experiment uses only a single core) and 32GB of RAM. The underlying planner is FF (Hoffmann and Nebel, 2001).

Table 1 shows that MPSR is typically faster than all other planners. When resampling is used, planning time increases, but plan quality (number of actions) typically improves. In localize, MPSR does not offer

Table 1: Comparing MPSR to CLG (execution mode) and SDR* — the best SDR variation in each domain. For domains with conditional actions (localize) CLG execution cannot be simulated. We denote TF when the CLG translation failed, CSU when CLG cannot run a simulation with a uniform distribution, and PF where the CLG planner failed, either due to too many predicates or due to timeout. Domains were FF was unable to solve the translation are denoted by FFF. Values in parentheses denote standard error. Maximum time allowed was 30 minutes. Best time and plan length for each domain are bolded.

	MPSR		MPSR × 5		SDR*		CLG	
Name	#Actions	Time(secs)	#Actions	Time(secs)	#Actions	Time(secs)	#Actions	Time(secs)
cloghuge	FFF		FFF		61.17 (0.44)	117.13 (4.19)	51.76 (0.33)	8.25 (0.08)
ebtcs-70	44.5 (0.7)	22.4 (0.3)	37.2 (0.8)	12.8 (0.3)	35.52 (0.75)	3.18 (0.07)	36.52 (0.86)	73.96 (0.14)
elog7	23.5 (0.1)	1.4 (0.1)	22.4 (0.1)	3.8 (0.1)	21.76 (0.07)	0.85 (0.01)	20.12 (0.05)	1.4 (0.08)
CB-9-1	359.1 (3.9)	61.1 (1.3)	188.9 (3.9)	61.1 (1.3)	124.56 (2.49)	71.02 (1.57)	94.36 (1.83)	129.3 (0.26)
CB-9-3	FFF		FFF		247.28 (2.91)	245.87 (4.03)	252.76 (2.66)	819.52 (0.47)
CB-9-5	FFF		FFF		392.16 (2.81)	505.48 (8.82)	PF	
CB-9-7	FFF		FFF		487.04 (2.95)	833.52 (15.82)	PF	
doors5	17.24 (0.2)	2 (0.1)	15.8 (0.2)	5.8 (0.1)	18.04 (0.18)	2.14 (0.03)	16.44 (0.18)	2.4 (0.1)
doors7	40 (0.4)	6.5 (0.1)	34.2 (0.3)	16.4 (0.2)	35.36 (0.41)	9.29 (0.1)	30.4 (0.24)	20.44 (0.02)
doors9	67.12 (0.7)	17.8 (0.2)	53.92 (0.5)	42.7 (0.4)	51.84 (0.55)	28 (0.31)	50.48 (0.5)	38.52 (0.06)
doors11	117.8 (1)	48.8 (1.4)	75.68 (0.6)	99.2 (0.8)	88.04 (0.91)	79.75 (1.04)	71.68 (0.79)	126.59 (0.1)
doors13	197.92 (1.2)	105.5 (2.1)	140.12 (1)	249.1 (3.2)	120.8 (0.93)	158.54 (2.01)	105.48 (0.89)	330.73 (0.21)
doors15	262.2 (1.9)	190 (3.3)	167.8 (1.6)	418.5 (6.1)	143.24 (1.36)	268.16 (3.78)	PF	
doors17	368.25 (3.4)	335.3 (5.3)	221.4 (2.2)	686.6 (11.4)	188 (1.64)	416.88 (6.16)	PF	
localize3	8.1 (0.1)	1.2 (0.1)	7.2 (0.1)	2.1 (0)	8 (0.12)	1.77 (0.03)	CSU	
localize5	16 (0.3)	1.5 (0.1)	15.5 (0.3)	4.1 (0.1)	14.56 (0.24)	7.12 (0.1)	CSU	
localize9	30.7 (0.4)	11.1 (0.2)	33.1 (0.5)	25.6 (0.5)	28.52 (0.42)	72.69 (1.43)	CSU	
localize11	38.5 (0.6)	31.1 (1.4)	39.9 (0.6)	71.3 (1.4)	34.67 (0.61)	155.6 (3.87)	PF	
localize13	42.4 (0.8)	73.4 (1.9)	47.2 (0.8)	185.4 (5)	37.52 (0.62)	396.76 (10.72)	PF	
localize15	45 (0.9)	130.6 (4.1)	59.8 (1.1)	384.9 (8)	40.08 (0.61)	667.22 (19.7)	PF	
localize17	59.8 (0.9)	230.4 (7.7)	71.8 (1.1)	687.4 (15.3)	45.00 (0.86)	928.56 (33.2)	PF	
unix1	9.8 (0.2)	0.6 (0.1)	10.8 (0.1)	0.9 (0)	12.2 (0.16)	0.48 (0.01)	11.68 (0.23)	0.35 (0.01)
unix2	30.6 (0.6)	1.9 (0.1)	24.9 (0.4)	3.9 (0.1)	26.44 (0.72)	1.41 (0.03)	19.88 (0.47)	2.69 (0.01)
unix3	69.7 (1.7)	5.2 (0.1)	51.7 (1.1)	13 (0.2)	56.32 (1.72)	5.47 (0.18)	51.32 (0.97)	18.56 (0.05)
unix4	158.6 (4.3)	30.4 (1.1)	138.1 (3.6)	61.8 (1.2)	151.72 (4.12)	35.22 (0.94)	90.8 (2.12)	189.41 (0.6)
Wumpus05	23.4 (0.2)	4.5 (0)	23.0 (0.2)	7.1 (0.1)	34.72 (0.3)	6.51 (0.07)	24.12 (0.1)	2.38 (0.09)
Wumpus10	47.2 (1)	12.4 (0.2)	44.2 (0.4)	23.1 (0.2)	70.64 (1.13)	65.89 (1.13)	40.44 (0.18)	36.29 (0.04)
Wumpus15	65 (1.6)	126.6 (3.1)	67.2 (0.8)	262.7 (5)	120.14 (2.4)	324.32 (7.14)	101.12 (0.67)	330.54 (0.25)
Wumpus20	71.6 (1.2)	261.1 (7)	79.9 (0.9)	773.6 (7.9)	173.21 (3.4)	773.01 (20.78)	155.32 (0.95)	1432 (0.47)

better plan quality than SDR because in this domain the best behavior is to guess a possible state and plan for it. In colorballs (CB) and the larger logistics domain (cloghuge), the underlying FF planner failed to solve the translated domains, possibly because the resulting plan trees are huge. In Wumpus, MPSR generates much better plans, much faster than either SDR or CLG.

To demonstrate where MPSR overcomes the disadvantages of SDR, we experiment on a Wumpus variation with deadends. Wumpus requires a smart exploration and sensing policy, and is thus one of the more challenging benchmarks. Originally, the agent requires a cell to be "safe" before entering it. We removed this precondition, and changed the move action so that the agent is dead if it enters a cell containing a wumpus or a pit, creating deadends. As expected SDR and all its variations fail utterly. CLG, however, solves these domains without failing. MPSR can rapidly solve this domain with good quality plans. MPSR uses 3 initial states, that given our diverse sampling technique, cover all possible safe configurations. We note that this is a fairly simple type of deadend, that is easily detected and handled. In general, MPSR sampling technique can still be trapped by more sophisticated deadends.

Table 2: Wumpus domains with deadends. TF denotes that the CLG translation did not complete in 30 minutes.

	MPSR		CLG	
Name	Time (secs)	#Actions	Time (secs)	#Actions
Wumpus 4	1.3 (0.01)	17.5 (0.1)	0.17 (0.001)	17.7 (0.04)
Wumpus 8	16.9 (1.5)	27.5 (1.1)	2.8 (0.01)	40.5 (0.31)
Wumpus 16	93.5 (4.5)	38.3 (1.2)	182.5 (1.73)	119.7 (0.91)
Wumpus 20	216.8 (6.7)	48.28 (1.2)	TF	
Wumpus 24	285.5 (5.5)	71.5 (0.7)	TF	

Conclusion and Future Work

We introduced a new translation scheme from contingent planning into classical planning. The method is sound and complete, but comes at substantial cost — the resulting plans can be very large. This cost can be controlled by selective sampling of the initial states combined with replanning, as we did here, or by us-

ing the multi-path translation as a method for generating informed heuristic estimates. Our empirical evaluation shows that MPSR is typically faster than SDR and CLG, although its plans are longer, but that on the new, more challenging domains with dead-ends, it produces better plans, faster, and scales-up better.

In this paper we assume that actions have deterministic effects. The multi-path formulation is partly motivated by the difficulty of current methods to deal with non-deterministic effects. This difficulty has two sources; The inherent rise in uncertainty due to non-determinism, which makes it harder for current transformation based planners to generate good choices. Their choices are greedy, and often need to be undone (when possible) in later stages when effects are not as expected. Here, we hope the multi-path method will help. The second difficulty is technical. All translation methods require some way of capturing the set of possible paths. In deterministic domains, these correspond to the set of initial states. In non-deterministic domains, this set depends on the actual plan executed, and cannot be bounded a-priori, and it is impossible to capture all eventualities. Workarounds exist, such as focusing on the first time each action is performed, as suggested by Palacios and Geffner (2009), which is well suited for translation based methods. Nevertheless, building a contingent planner that handles non-deterministic domains well remains an important challenge.

Another issue that requires a closer investigation is our diverse sampling approach. We intend to analyze the effectiveness of using more states, and the importance of diverse sampling. It is also possible in domains with a small number of possible state to exhaustively check all possible subsets, analyzing the importance of the sampling on the resulting plan.

Acknowledgement: Ronen Brafman is partially supported by ISF grant 8254320, the Paul Ivanier Center for Robotics Research and Production Management, and the Lynn and William Frankel Center for Computer Science.

We consider a multi-level jury problem in which experts are We consider a multi-level jury problem in which experts

are We consider a multi-level jury problem in which experts
are We consider a multi-level jury problem in which experts
are We consider a multi-level jury problem in which experts
are We consider a multi-level jury problem in which experts
are We consider a multi-level jury problem in which experts
are We consider a multi-level jury problem in which experts
are

References

Albore, A.; Palacios, H.; and Geffner, H. 2009. A translation-based approach to contingent planning. In IJCAI, 1623–1628.

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In IJCAI'11.

Bonet, B.; Palacios, H.; and Geffner, H. 2009. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In ICAPS.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. Reasoning about Knowledge. MIT Press.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. JAIR 14:253–302.

Palacios, H., and Geffner, H. 2009. Compiling uncertainty away in conformant planning problems with bounded width. JAIR 35:623–675.

Shani, G., and Brafman, R. I. 2011. Replanning in domains with partial information and sensing actions. In IJCAI, 2021–2026.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. Ff-replan: A baseline for probabilistic planning. In ICAPS.