Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence

Flexible and Scalable Partially Observable Planning with Linear Translations

Blai Bonet
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Hector Geffner
ICREA & DTIC Universitat Pompeu Fabra
08018 Barcelona, Spain
hector.geffner@upf.edu

Abstract

The problem of on-line planning in partially observable settings involves two problems: keeping track of beliefs about the environment and selecting actions for achieving goals. While the two problems are computationally intractable in the worst case, significant progress has been achieved in recent years through the use of suitable reductions. In particular, the state-of-the-art CLG planner is based on a *translation* that maps deterministic partially observable problems into fully observable non-deterministic ones. The translation, which is *quadratic* in the number of problem fluents and gets rid of the belief tracking problem, is adequate for most benchmarks, and it is in fact complete for problems that have width 1. The more recent K-replanner uses translations that are *linear*, one for keeping track of beliefs and the other for selecting actions using off-the-shelf classical planners. As a result, the K-replanner scales up better but it is not as general. In this work, we combine the benefits of the two approaches – the scope of the CLG planner and the efficiency of the K-replanner. The new planner, called LW1, is based on a translation that is *linear* but *complete* for width-1 problems. The scope and scalability of the new planner is evaluated experimentally by considering the existing benchmarks and new problems.

Introduction

The problem of planning with incomplete information and partial sensing has received a great deal of attention in recent years. In the logical setting, it is called contingent or partially observable planning, while in the probabilistic setting, it's known as POMDP planning. In both cases, *off-line solutions* can be regarded as policies mapping belief states into actions, with beliefs referring to subsets of states in the first case, and probability distributions over states in the second (Ghallab, Nau, and Traverso 2004; Geffner and Bonet 2013). In spite of significant progress, however, a key obstacle to scalability is the size of off-line solutions which may be exponential. Thus, one approach that has been pursued is to focus on the *on-line problem* instead. In the on-line problem two tasks must be addressed, *belief tracking* and *action selection*. While both tasks are intractable in the worst case, effective methods have been developed in recent years for dealing with them through the use of translations. In particular, the CLG planner (Albore, Palacios, and Geffner 2009), which can be used in off-line and on-line mode, is based on a *translation* that maps deterministic partially observable problems into fully observable non-deterministic ones. The translation, which is *quadratic* in the number of problem fluents and gets rid of the belief tracking problem, is adequate for practically all benchmarks, and it is in fact complete for problems characterized as having *width one* (Palacios and Geffner 2009). The more recent K-replanner (Bonet and Geffner 2011) uses two translations that are *linear* in the number of problem fluents, one for keeping track of beliefs and the other for selecting actions using off-the-shelf classical planners. As a result, the K-replanner scales up better but it is not as general. Similar ideas appear in two other recent *on-line planners*, SDR and MPSR, that use translations for action selection, but these translations, when sound, are not polynomial (Brafman and Shani 2012b; 2012a),

The goals of this work are twofold. From a theoretical point of view, we introduce a translation for belief tracking that like the one used in the K-replanner is *linear* in the number of problem fluents, and yet like the one used in CLG is *complete for width-1 problems*. From a practical point of view, we introduce a partially observable planner that appears to be practical enough: scalable as current *classical planners* and expressive enough for handling all contingent benchmarks and more. The difference between translations that are linear and quadratic is critical for this goal: in problems with 500 boolean fluents p_i, the linear translation results in a transformed problem with up to $2 \times 500 = 1000$ boolean fluents Kp_i and $K\neg p_i$, while the quadratic translation results in a problem with up to $(2 \times 500)^2 = 1,000,000$ fluents Kp_i/p_k, $K\neg p_i/p_k$, $Kp_i/\neg p_k$, $K\neg p_i/\neg p_k$. The paper is organized as follows: we review the language and model of the new planner LW1 along with the notion of width, and then consider the new translations, their formal properties, their integration into LW1, and the experimental results.

Planning with Sensing

We present the model and language associated with LW1 following mostly Bonet and Geffner (2012).

Model

The model for partially observable planning is characterized by a tuple $\mathcal{S} = \langle S, S_0, S_G, A, f, O\rangle$ where S is a finite state space, S_0 is a set of possible initial states, S_G is a set of goal states to be reached with certainty, A is a set of actions with $A(s)$ denoting the actions applicable in state $s \in S$, f is a *deterministic* state-transition function such that $s' = f(a, s)$ denotes the successor state that follows action a in state s, $a \in A(s)$, and O is a *sensor model*. We regard a sensor model as a function $o(s, a, b)$ that maps the state s resulting from an action a in a belief state b into a observation token. The addition of the belief state b of the agent as a parameter is not standard but provides additional expressive power; e.g., it is possible to express that the color of a room is observed after applying an action when the agent *knows* that it is in the room.

Executions are sequences of action-observation pairs $a_0, o_0, a_1, o_1, \ldots$. The initial belief is $b_0 = S_0$, and if b_i is the belief when the action a_i is applied and o_i is the token that is then observed, the new belief b_{i+1} is

$$b_a^o = \{s \mid s \in b_a \text{ and } o = o(s, a, b_a)\}\,, \qquad (1)$$

where $a = a_i$, $o = o_i$, and b_a is the belief after the action a in the belief $b = b_i$:

$$b_a = \{s' \mid \text{there is } s \in b \text{ such that } s' = f(a, s)\}\,. \qquad (2)$$

An execution $a_0, o_0, a_1, o_1, \ldots$ is *possible* in model $\mathcal{S}$ if starting from the initial belief b_0, each action a_i is applicable in the belief b_i (i.e., $a_i \in A(s)$ for all $s \in b_i$), and b_{i+1} is non-empty. A policy π is a function mapping belief states b into actions. The executions $a_0, o_0, a_1, o_1, \ldots$ induced by the policy π are the possible executions in which $a_i = \pi(b_i)$. The policy solves the model if all such executions reach a *goal* belief state, i.e., a belief state $b \subseteq S_G$. Off-line methods focus on the computation of such policies, on-line methods focus on the computation of the action for the current belief.

Representation

The problems are represented in compact form through a set of *multivalued state variables*. More precisely, a problem is a tuple $P = \langle V, I, A, G, W\rangle$ where V stands for a set of state variables X, each one with a finite domain D_X, I is a set of X-literals defining the initial situation, G is a set (conjunction) of X-literals defining the goal, and A is a set of actions with preconditions $Pre(a)$ and conditional effects (rules) $C \rightarrow E$ where $Pre(a)$ and C are sets of X-literals, and E is a set of *positive* X-literals. Positive literals are expressions of the form $X = x$ for $X \in V$ and $x \in D_X$. Negative literals $\neg(X = x)$ are written as $X \neq x$, and negation on literals is taken as complementation so that $\neg\neg L$ stands for L. The *states* associated with a problem P refer to valuations over the state variables X.

The sensing component W in P is a collection of observable multivalued variables Y with domains D_Y, each one with a state formula $Pre(Y)$, called the Y-sensor precondition that encodes the conditions under the variable Y is observable, and state formulas $W(Y = y)$, one for each value of y in D_Y, that express the conditions under which the value of Y will be y. Since we assume that sensing is deterministic, the formulas $W(Y = y)$ must be mutually exclusive and jointly exhaustive in the states that make $Pre(Y)$ true. That is, in such states, one and only one value of Y is observed. An observable variable Y may be also an state variable, and in such case, $W(Y = y)$ is the formula $Y = y$. We assume that $Pre(Y)$ and $W(Y = y)$ are in Disjunctive Normal Form (DNF), and moreover, that the terms C in $W(Y = y)$ contain *positive* X-literals only. If negative literals like $X \neq x$ need to be used, they must be replaced by the disjunction $\bigvee_{x'} X = x'$ where x' ranges over the possible values of X in D_X different than x and the resulting formula must be brought into DNF.

The planning problem $P = \langle V, I, G, A, W\rangle$ defines the model $\mathcal{S}(P) = \langle S, S_0, S_G, A, f, O\rangle$, where S is the set of valuations over the variables in V, S_0 and S_G are the set of valuations that satisfy I and G, $A(s)$ is the set of actions in P whose preconditions are true in s, and $f(a, s)$ is the state-transition function determined by the conditional effects associated with a. Likewise, the sensor model O is such that $o(s, a, b)$ stands for a valuation over the variables Y in W such that $Pre(Y)$ is true in b. In this partial valuation, $Y = y$ is true iff $W(Y = y)$ is true in s. We will assume throughout a state variable in V called $LastA$ that is affected by all the actions a in the problem which make the literal $LastA = a$ true. In order to say that a variable Y is observable only after action a is executed, we just set $Pre(Y)$ to $LastA = a$.

Examples

If X encodes the position of an agent, and Y encodes the position of an object that can be detected by the agent when $X = Y$, we can have an observable variable $Z \in \{yes, no\}$ with formulas $Pre(Z) = true$, meaning that Z is always observable, and $W(Z = yes) = \bigvee_{l \in D}(X = l \wedge Y = l)$, meaning that value yes will be observed when $X = Y$. Since the observable variable Z has two possible values, the formula $W(Z = no)$ is given by the negation of $W(Z = yes)$, which in DNF is $\bigvee_{l,l' \in D, l \neq l'}(X = l \wedge Y = l')$. On the other hand, if the agent cannot detect the presence of the object at locations $l \in D'$, we will set $Pre(Z)$ to $\bigwedge_{l \in D \setminus D'}(X \neq l)$, meaning that the sensor is active or assumed to provide useful information when the agent knows that it is not in D'. Benchmark domains like *Medical* or *Localize* feature *one state variable only*, representing the patient disease, in the first case, and the agent location in the second. The possible test readings in *Medical*, and the presence or absence of contiguous walls in *Localize* can be encoded in terms of one observable variable in the first case, and four observable variables in the second.

Problem Structure: Width

The width of a problem refers to the maximum number of uncertain state variables that interact in a problem, either through *observations* or *conditional effects* (Palacios and Geffner 2009; Albore, Palacios, and Geffner 2009). In a domain like *Colorballs* where m balls in uncertain locations and with uncertain colors must be collected and delivered

according to their color, problems have width 1, as the $2m$ state variables do not interact. On the other hand, in a domain like *Minesweeper*, where each cell in the grid may contain a bomb or not, the problem width is given by the number of cells, as all the state variables (potentially) interact through the observations. We make the notion of width precise following the formulation of Bonet and Geffner (2012). For this, we will say that a variable X is as an *immediate cause* of X', written $X \in Ca(X')$, if $X \neq X'$, and either X occurs in the body of a conditional effect $C \rightarrow E$ and X' occurs in a head, or X occurs in $W(X' = x')$. *Causal relevance* and plain *relevance* are defined as follows:

Definition 1 *X is* causally relevant *to X' in P if $X = X'$, $X \in Ca(X')$, or X is causally relevant to a variable Z that is causally relevant to X'.*

Definition 2 *X is* evidentially relevant *to X' in P if X' is causally relevant to X and X is an observable variable.*

Definition 3 *X is* relevant *to X' if X is causally or evidentially relevant to X', or X is relevant to a variable Z that is relevant to X'.*

The *width of a problem* is then:

Definition 4 *The* width *of a variable X, $w(X)$, is the number of state variables that are relevant to X and are not* determined*. The width of the problem P, $w(P)$, is $\max_X w(X)$, where X ranges over the variables that appear in a goal, or in an action or sensor precondition.*

The variables that are *determined* in a problem refer to the state variables whose value is always known; more precisely, the set of determined variables is the largest set S of state variables X that are initially known, such that every state variable X' that is causally relevant to X belongs to the set S. This set can be easily identified in low polynomial time.

Since domains like *Medical* and *Localize* involve a single state variable, their width is 1. Other width-1 domains include *Colorballs* and *Doors*. On the other hand, the width in a domain like *Wumpus* is given by the max number of monsters or pits (the agent position is a determined variable), while in *Minesweeper*, by the total number of cells.

Linear Translation for Belief Tracking

Belief tracking is time and space exponential in the problem width (Bonet and Geffner 2012). In the planner CLG, belief tracking is achieved by means of a translation that introduces tagged literals KL/t for each literal L in the problem that expresses that L is true if it is assumed that the tag formula t is initially true. As the tags t used in the translation are single literals, the translation is *quadratic* in the number of problem fluents, and provably complete for width-1 problems. In the K-replanner, untagged KL literals are used instead, resulting into a *linear* translations that is not width-1 complete. The translation below combines the benefits of the two approaches: it is linear and complete for width 1.

Basic Translation

We abbreviate the literals $X = x$ and $X \neq x$ as x and $\bar{x}$, leaving the variable name implicit. Kx and $K\bar{x}$ then stand for $KX = x$ and $KX \neq x$. Likewise, conditional effects $C \rightarrow E$ for $E = L_1, \ldots, L_n$ where L_i is a literal, are decomposed as effects $C \rightarrow L_i$. The basis of the new translation is:

Definition 5 *For a problem $P = \langle V, I, G, A, W \rangle$, the translation $X_0(P)$ outputs a* classical problem *$P' = \langle F', I', G', A' \rangle$ and a set of* axioms *D', where*

- $F' = \{KL : L \in \{X = x, X \neq x\}, X \in V, x \in D_X\}$,
- $I' = \{KL : L \in I\}$,
- $G' = \{KL : L \in G\}$,
- $A' = A$ *but with each action precondition L replaced by KL, and each conditional effect $C \rightarrow X = x$ replaced by effects $KC \rightarrow Kx$ and $\neg K\neg C \rightarrow \neg K\bar{x}$,*
- $D' = \{Kx \Rightarrow \bigwedge_{x':x' \neq x} K\bar{x}', \bigwedge_{x':x' \neq x} K\bar{x}' \Rightarrow Kx\}$, *for all $x \in D_X$ and $X \in V$.*

The expressions KC and $\neg K\neg C$, for a set C of literals $L_1, \ldots, L_n$, represent the conjunctions $KL_1, \ldots, KL_n$, and $\neg K\neg L_1, \ldots, \neg K\neg L_n$ respectively. The deductive rules or axioms (Thiébaux, Hoffmann, and Nebel 2005) in D' enforce the exclusivity (MUTEX) and exhaustivity (OR) of the values of multivalued variables. The translation $X_0(P)$ is otherwise similar to the basic incomplete translation K_0 for conformant planning (Palacios and Geffner 2006), which is also the basis for the translation used by the K-replanner. These translations are linear as they introduce no assumptions or 'tags', but are complete only when the bodies C of the conditional effects feature no uncertainty. In the partially observable setting, this translation is incomplete in another way as it ignores sensing. The extensions below address these two limitations.

Action Progression and Compilation

Consider an agent in a $1 \times n$ grid, and an action $right$ that moves the agent one cell to the right. The action can be encoded by means of conditional effects $i \rightarrow i + 1$ where i is an abbreviation of $X = i$, $1 \leq i < n$. If the initial location is completely unknown, it's clear, however, that after a single $right$ move, $X = 1$ will be false. Yet, in $X_0(P)$, the $right$ action would feature 'support' effects $Ki \rightarrow K(i+1)$ that map knowledge of the current location into knowledge of the next location, but cannot obtain knowledge from ignorance. The problem features indeed conditional effects with uncertain bodies and its width is 1. The translation used in CLG achieves completeness through the use of tagged literals KL/t for tags of size 1. It is possible however to achieve width-1 completeness without tags at all using a generalization of a 'trick' originally proposed by Palacios and Geffner (2006) for making a basic conformant translation more powerful. The idea, called *action compilation*, is to make explicit some implicit conditional effects:

Definition 6 (Action Compilation) *Let a be an action in P with conditional effect $C, x \rightarrow x'$ with $x' \neq x$. The* compiled effects *associated with this effect are all the rules of the form $KC, K\neg L_1, \ldots, K\neg L_m \rightarrow K\bar{x}$ where $C_i \rightarrow x$, for $i = 1, \ldots, m$, are all the effects for the action a that lead to x, and L_i is a literal in C_i. If $m = 0$, the compiled effects consist in just the rule $KC \rightarrow K\bar{x}$. The* compiled

effects *associated with an action a refers to the compiled effects associated which each of its original effects $C, x \to x'$ for $x, x' \in D_X$ with $x' \neq x$, and $X \in V$.*

In the above example for the $1 \times n$ grid, for $x = 1$, there is a single effect of the form $C, x \to x'$ where C is empty and $x' = 2$, and there are no effects of the form $C_i \to x$. Hence, the compilation of this effect yields the rule $true \to K\bar{1}$. The compilation of the other effects for the action yields the rules $K\bar{i} \to K\overline{i+1}$ for $i = 1, \ldots, n-1$.

This compilation is sound, and in the worst case, exponential in the *number of effects* $C \to X = x$ associated with *the same action* and *the same literal* $X = x$. This number is usually bounded and small, and hence, it doesn't appear to present any problems. The translation $X(P)$ is the basic translation $X_0(P)$ extended with these implicit effects:

Definition 7 *For a problem $P = \langle V, I, G, A, W \rangle$, the translation $X(P)$ is $X_0(P)$ but with the effects of the actions extended with their* compiled effects

The *semantics* of this extension can be expressed in terms of the relation between the progression of *beliefs* in P and the progression of *states* in $X(P)$. States in $X(P)$ are valuations over the KL literals which are represented by the literals that they make true, and are progressed as expected:

Definition 8 (Action Progression) *The state s_a that results from a state s and an action a in $X(P)$ that is applicable in s, is the state s_a obtained from s and a in the* classical problem *P', closed under the deductive rules in D'.*

The deductive closure is polynomial and fast, as it just has to ensure that if a literal Kx is in s_a so are the literals $K\bar{x'}$ for the other values x' of the same variable X, and conversely, that if these literals are in s_a, so is Kx. The completeness of this form of belief progression in width-1 problems in the absence of observations, follows from the result below and the fact that beliefs in width-1 problems can be decomposed in factors or beams, each of which contains at most one uncertain variable (Bonet and Geffner 2012):

Theorem 9 (Width-1 Action Progression) *Let b be a belief over a* width-1 problem *P with* no uncertainty about variables other than *X, and let s be the state in $X(P)$ that represents b; i.e., s makes KL or $K\neg L$ true iff b makes L true or false resp. Then, an action a is applicable in b iff it is applicable in s, and b_a makes L true iff the state s_a makes KL true.*

Adding Observations

Theorem 9 establishes a correspondence between the belief b_a that results from a belief b and an action a in P, and the state s_a that results from the compiled action a in $X(P)$ from the state s encoding the literals true in b. We extend this correspondence to the beliefs b_a^o and states s_a^o that result from observing o after executing the action a. Recall that an observation is a valuation over the observable variables whose sensor preconditions hold, and that states s are represented by the true KL literals.

Definition 10 (Progression through observations) *Let s_a be the state following the execution of an action a in state s in the translation $X(P)$. Then the state s_a^o that results from obtaining the observation o is:*

$$s_a^o = \text{UNIT}(s_a \cup D' \cup K_o) \tag{3}$$

where UNIT(C') *stands for the set of* unit literals *in the* unit resolution closure *of C', D' is the set of deductive rules, and K_o stands for the codification of the observation o given the sensing model W. That is, for each term $C_i \cup \{L_i\}$ in $W(Y = y)$ such that o makes $Y = y$* false*, K_o contains the formula $KC_i \Rightarrow K\neg L_i$. If the empty clause is obtained in* (3)*, s_a^o is $\bot$.*

We establish next the correspondence between the literals L true in the beliefs b_a^o in P and the literals KL true in the states s_a^o in $X(P)$ when the width of P is 1. For this, let us say that an execution $a_0, o_0, a_1, o_1, \ldots, a_k, o_k$ is possible in $X(P)$ iff the preconditions of action a_i hold in the state s_i and the observation o_i is then possible, $i = 0, \ldots, k-1$, where s_0 is I' and s_{i+1} is s_a^o for $s = s_i$, $a = a_i$, and $o = o_i$. Likewise, observation o is possible in s_a iff o assigns a value to each observable variable Y such that KC is true in s for some term C in $Pre(Y)$, and $s_a^o \neq \bot$. Since unit resolution is very efficient, the computation of the updated state s_a^o from s is efficient too. Still, this limited form of deduction, suffices to achieve completeness over width-1 problems:

Theorem 11 (Completeness for Width-1 Problems) *Let P be a partial observable problem of width* 1*. An execution $a_0, o_0, a_1, o_1, \ldots, a_i, o_i$ is possible and achieves the goal G in P iff the same execution is possible and achieves the goal KG in $X(P)$.*

While we cannot provide a full proof due to lack of space, the idea is simple. We have the correspondence between s_a and b_a, we just need to extend it to s_a^o and b_a^o. For this, if literal $\bar{x}$ makes it into b_a^o, we need to show that literal $K\bar{x}$ makes into s_a^o. In width-1 problems, this can only happen when $C \cup \{x\}$ is a term in the DNF formula $W(Y = y')$ such that o makes $Y = y'$ false, and C is known to be true in b_a. In such a case, KC will be true in s_a, and $KC \Rightarrow K\bar{x}$ will be a deductive rule in K_o, from which unit resolution in (3) yields $K\bar{x}$. The proof of the theorem relies on the assumption that the sensor model W is made of DNF formulas whose terms contain only positive literals.

Translation $H(P)$ for Action Selection

The translation $X(P)$ provides an effective way for *tracking beliefs* over P through classical state progression and unit resolution. The classical problem P', however, cannot be used for *action selection* as the sensing and deduction are carried out outside P'. Following the idea of the K-replanner, we bring sensing and deduction into the classical problem by adding suitable actions to P': actions for making assumptions about the observations (optimism in the face of uncertainty), and actions for capturing the deductions (OR constraints) in D'. On the other hand, the exclusivity constraints in D' (mutex constraints) are captured by adding the literals $K\bar{x'}$ to all the effects that contain Kx for $x \neq x'$.

Definition 12 (Heuristic Translation $H(P)$) *For a problem $P = \langle V, I, G, A, W \rangle$, and translation $X(P)$ resulting*

into the classical problem $P' = \langle F', I', G', A' \rangle$ and deductive rules D', $H(P)$ is defined as the problem P' with the three extensions below:

1. ***D-Mutex:*** *Effects containing a literal Kx are extended to contain $K\bar{x}'$ for $x' \neq x$, $x, x' \in D_X$, $X \in V$.*
2. ***D-OR:*** *A deductive action is added with conditional effects $\bigwedge_{x':x' \neq x} K\bar{x}' \rightarrow Kx$, $X \in V$, $x, x' \in D_X$.*
3. ***Sensing:*** *Actions $a_{Y=y}$ are added with preconditions KL for each literal L in $Pre(Y)$, and effects $KC_i, \neg Kx \rightarrow K\bar{x}$ for each term $C \cup \{x\}$ in the DNF formulas $W(Y = y')$ for $y' \neq y$.*

The *heuristic translation* $H(P)$ mimics the *execution translation* $X(P)$ except that it incorporates the deductive component D', and a suitable relaxation of the sensing component into the classical problem itself so that successful executions in $X(P)$ are captured as classical plans for $H(P)$.

The LW1 Planner

The LW1 planner (Linear translations for Width-1 problems) is an on-line partially observable planner that works like the K-replanner: it computes a classical plan from $H(P)$ and executes the plan until the first (relaxed) sensing action $a_{Y=y}$. At that point, $Pre(Y)$ must hold, and the true value y' of Y is observed. If $y' = y$, the execution proceeds, skipping the non-executable action $a_{Y=y}$ until the next sensing action is encountered . On the other hand, if $y' \neq y$, a new classical plan is computed using $H(P)$ from the current state and the process iterates. The observations that are obtained in the execution come from a *hidden initial state* that is given as part of the problem, and is progressed through the actions executed.

The classical plans obtained from the heuristic translation $H(P)$ are *sound*, and hence are executable in $X(P)$ until the first (relaxed) sensing action. For *completeness*, it's possible to prove a key *explore-exploit* property similar to the one achieved by the K-replanner but in the broader class of *width-1* problems:

Theorem 13 (Goal Achievement for Width-1 Problems) *In* width-1 *problems P with* no dead-ends*, LW1 reaches the goal in a number of replanning iterations that is bounded by the sum of the cardinalities of the beliefs over each of the state variables.*

The no dead-end condition has nothing to do with partial observability, as the same applies to *on-line* algorithms in the classical setting, where the only way to ensure not being trapped in states that are not connected to the goal is by computing a full plan. The theorem ensures that the replanning algorithm will not be trapped into loops in width-1 problems. This is because in these problems, beliefs over each of the variables cannot increase in cardinality, and must decrease when the execution of a plan from $H(P)$ reaches a relaxed sensing action $a_{Y=y}$ whose assumption $Y = y$ is refuted by the actual observation.

Extensions

The LW1 planner accommodates two extensions that, while not needed in width-1 problems, are useful in more challenging domains like *Minesweeper* or *Wumpus*. The first involves adding the literals $KY = y$ to the problem for observable variables Y. Notice that such literals do not appear in either $X(P)$ or $H(P)$ unless Y is a state variable. These literals, however, can be important in problems where some of the observations have no immediate effect on state literals. These literals are added to $X(P)$ and $H(P)$ by enforcing the invariants $W(Y = y) \Rightarrow Y = y$ over the K-literals. In addition, deductive rules and actions are added to $X(P)$ and $H(P)$ so that $KY = y$ implies KC when C is one of the terms in the formula $W(Y = y)$ and the other terms are known to be false. This is a polynomial operation that involves adding variables for each of the terms.

Experiments

We implemented LW1 on top of the K-replanner. For comparing the two planners, we also added a *front-end* to the K-replanner so that it can handle the syntax of contingent benchmarks. For LW1, we converted these benchmarks by hand into the syntax based on multivalued state and observable variables. The experiments were performed using the classical planner FF (Hoffmann and Nebel 2001) on a cluster made of AMD Opteron 6378 CPUs with 4Gb of memory, running at 2.4 Ghz. The third on-line planner considered in the experiments is HCP; we took the data from Shani et al. (2014), where HCP is compared with CLG, SDR, and MPSR. HCP can be understood as extending the K-replanner with a subgoaling mechanism: in each replanning episode, rather than planning for the top goal, the planner looks for a closer subgoal; namely, the preconditions of a first sensing action that is selected heuristically.

Table 1 compares LW1, the K-replanner with the front end, and HCP. For the first two planners, the table shows the number of randomly generated hidden initial states considered for each problem (#sim), the number of instances solved, the average number of calls to the classical planner, the average length of executions, and the average times. The times are expressed as total time, time taken by FF loading the PDDL files in each invocation (preprocessing), and execution time (the difference between the previous two). The execution time is the key to scalability. The preprocessing time can actually be cut down substantially by loading the PDDL files once per problem. Shani et al. don't report the number of simulations (#sim) nor whether the reported times include preprocessing, but from the fact that HCP also uses FF as its underlying (re)planner, most likely, such time is not included. It is easy to see from the table that, in terms of execution times, LW1 scales up as well as the K-replanner, producing in general shorter executions. LW1 also appears faster than HCP in most of the domains, producing also shorter executions, with the exceptions of *Rocksample* and *Unix*. In terms of coverage, the K-replanner is unable to deal with two of the domains, *Localize* and *Rocksample*, while HCP is unable to deal with the former.

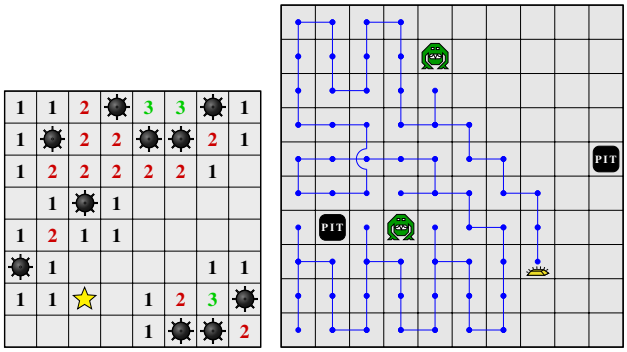
We also ran LW1 on two more challenging domains: a fuller version of *Wumpus*, and *Minesweeper*. In *Wumpus*, the number of (hidden) monsters and pits per increasing board size are 1+1, 2+2, and 4+4 respectively, while in *Minsweeper*, the number of mines per increasing board size are

			LW1						K-Replanner with Front End						HCP	
				average		avg. time in seconds				average		avg. time in seconds				
domain	problem	#sim	solved	calls	length	total	prep	exec	solved	calls	length	total	prep	exec	length	time
clog	7	12	12	2.2	17.8	0.0	0.0	0.0	12	10.0	39.3	0.2	0.1	0.1	nd	nd
clog	huge	3125	3125	7.0	45.6	3.5	2.8	0.7	3125	44.6	156.6	6.7	3.6	3.0	53.5	1.8
colorballs	9-5	1000	1000	65.6	126.8	468.2	454.0	14.2	1000	210.4	481.2	725.0	687.9	37.0	320	57.7
colorballs	9-7	1000	1000	69.8	146.1	632.7	615.5	17.1	1000	292.4	613.3	1719.0	1645.9	73.1	425	161.5
doors	17	1000	1000	54.2	114.1	495.3	490.1	5.1	1000	65.0	213.6	88.3	77.1	11.2	143	17.7
doors	19	1000	1000	67.2	140.1	928.2	920.5	7.6	1000	82.7	269.2	143.5	128.5	14.9	184	46.1
ebtcs	50	50	50	25.5	26.5	2.5	1.7	0.7	50	25.5	27.5	1.3	0.9	0.4	nd	nd
ebtcs	70	70	70	35.5	36.5	5.2	4.2	1.0	70	35.5	37.5	3.2	2.4	0.7	34.5	0.3
elog	5	8	8	1.9	19.5	0.0	0.0	0.0	8	14.5	67.6	0.4	0.2	0.2	nd	nd
elog	7	12	12	2.2	17.8	0.0	0.0	0.0	12	14.0	66.8	0.4	0.2	0.1	19.9	0.0
localize	15	134	134	9.3	15.2	21.8	5.5	16.3	—	—	—	—	—	—	—	—
localize	17	169	169	10.7	17.2	69.9	20.1	49.7	—	—	—	—	—	—	—	—
medpks	150	151	151	2.0	2.0	10.9	10.0	0.9	151	2.0	2.0	1.3	1.2	0.0	nd	nd
medpks	199	200	200	2.0	2.0	26.0	23.5	2.4	200	2.0	2.0	3.2	3.1	0.1	nd	nd
rocksample	8-12	1000	1000	6.9	191.5	124.2	1.4	122.7	—	—	—	—	—	—	115	0.5
rocksample	8-14	1000	1000	10.2	272.3	22.5	2.7	19.7	—	—	—	—	—	—	135	0.6
unix	3	28	28	17.0	46.5	1.9	1.4	0.4	28	17.0	46.5	1.2	1.0	0.2	42.0	0.6
unix	4	60	60	33.0	93.7	23.0	21.6	1.4	60	33.0	93.7	16.4	15.3	1.1	76.5	7.2
wumpus	5d	8	8	2.2	16.2	0.1	0.0	0.0	8	3.8	25.0	0.2	0.1	0.0	nd	nd
wumpus	10d	256	256	4.4	33.8	2.2	2.0	0.2	256	5.3	46.2	1.6	1.0	0.6	nd	nd
wumpus	15d	1000	1000	5.3	47.2	27.2	26.4	0.8	1000	6.2	61.0	7.9	6.2	1.6	65.0	2.3
wumpus	20d	1000	1000	5.3	57.2	162.6	160.5	2.0	1000	5.8	69.2	28.9	25.8	3.0	90	5.1
wumpus	25d	1000	1000	5.4	67.3	729.7	724.5	5.1	1000	6.1	80.9	73.5	68.4	5.1	nd	nd

Table 1: Comparison of LW1, K-replanner with front end, and HCP on range of contingent benchmarks. Dash (—) means that the planner cannot solve a domain, and 'nd' means that no data is reported for the instance. Key columns are highlighted in gray.

				average		avg. time in seconds		
domain	prob.	#sim	solved	calls	length	total	prep	exec
mines	3x4	100	11	3.5	14.0	1.0	0.8	0.1
mines	3x5	100	15	4.0	17.0	2.0	1.8	0.2
mines	4x4	100	35	5.1	18.0	11.3	10.7	0.6
mines	5x5	100	48	6.5	27.0	93.4	90.1	3.3
mines	6x6	100	37	9.6	38.0	522.4	506.6	15.8
mines	7x7	100	45	11.0	51.0	1320.7	1278.3	42.3
mines	8x8	100	43	13.1	66.0	3488.2	3365.4	122.7
wumpus	5x5	100	100	12.2	15.2	1.4	0.9	0.4
wumpus	10x10	100	100	54.1	60.5	182.5	173.2	9.2
wumpus	15x15	100	100	109.7	121.0	3210.3	3140.3	70.0

Table 2: LW1 on Minesweeper and richer version of Wumpus.



8×8 minesweeper 10×10 wumpus

Figure 1: Example of instances solved by LW1. *Left:* an 8×8 *Minesweeper* instance where the star marks the first cell opened. *Right:* a 10×10 *Wumpus* instance with 2 monsters, 2 pits, and unknown position of gold. The trace shows the path walked by the agent when looking for the gold, beginning at the lower left corner.

2, 2, 3, 4, 6, 8 and 10. We are not aware of other planners able to deal with these domains. Minesweeper is NP-hard (Kaye 2000). We used the problem generator by Bonet and Geffner (2013), keeping the instances that their belief tracking algorithm, wrapped in a hand-crafted policy, was able to solve. Table 2 shows the results of LW1 on 100 random hidden initial states for instances of different size. Figure 1 shows two solved instances for *Minesweeper* and *Wumpus*. While LW1 manages to solve many of the large, non-trivial instances, it doesn't solve all of them. This is because some of these problems require forms of inference that are more sophisticated than unit resolution.

Conclusions

We have developed a new on-line planner for deterministic partially observable domains, LW1, that combines the flexibility of CLG with the scalability of the K-replanner. This is achieved by using two linear translations: one for keeping track of beliefs while ensuring completeness for width-1 problems; the other for selecting actions using classical planners. We have also shown that LW1 manages to solve the problems solved by other planners, and more challenging problems as well. Width-1 completeness is important for two reasons: it ensures that the simplest problems where actions propagate uncertainty will be handled, and it provides a solid basis for dealing with more complex problems. For example, more powerful forms of deduction can be accommodated, and in certain cases, subsets of variables may be aggregated into one variable if required. Regarding deduction, the belief tracking algorithm called *beam tracking* performs much better than LW1 in *Minesweeper* and it is also polynomial (Bonet and Geffner 2013). The reason is that it uses *arc-consistency* rather than *unit resolution*. Yet nothing prevents us from replacing one by the other in the inner loop of the LW1 planner.

Acknowledgments

This work was partially supported by EU FP7 Grant # 270019 (Spacebook) and MICINN CSD2010-00034 (Simulpast).

References

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

Bonet, B., and Geffner, H. 2011. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. IJCAI*, 1936–1941.

Bonet, B., and Geffner, H. 2012. Width and complexity of belief tracking in non-deterministic conformant and contingent planning. In *Proc. AAAI*, 1756–1762.

Bonet, B., and Geffner, H. 2013. Causal belief decomposition for planning with sensing: Completeness results and practical approximation. In *Proc. IJCAI*, 2275–2281.

Brafman, R. I., and Shani, G. 2012a. A multi-path compilation approach to contingent planning. In *Proc. AAAI*, 9–15.

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

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: theory and practice*. Morgan Kaufmann.

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

Kaye, R. 2000. Minesweeper is NP-Complete. *Mathematical Intelligencer* 22(2):9–15.

Palacios, H., and Geffner, H. 2006. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proc. AAAI*, 900–905.

Palacios, H., and Geffner, H. 2009. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *Journal of Artificial Intelligence Research* 35:623–675.

Shani, G.; Karpas, E.; Brafman, R. I.; and Maliah, S. 2014. Partially observable online contingent planning using landmark heuristics. In *Proc. ICAPS*.

Thiébaux, S.; Hoffmann, J.; and Nebel, B. 2005. In defense of PDDL axioms. *Artificial Intelligence* 168(1-2):38–69.