Translation-Based Approaches to Automated Planning with Incomplete Information and Sensing

Alexandre Albore

TESI DOCTORAL UPF / 2011

Thesis advisor

Prof. Dr. Hector Geffner
Department of Information and Communication Technologies

UNIVERSITAT POMPEU FABRA

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy.

December 2011

The court's PhD was appointed by the rector of the Universitat Pompeu Fabra on ...., 2011.

Chairman

Member

Member

Member

Secretary

The doctoral defense was held on ...., 2012, at the Universitat Pompeu Fabra and scored as ....

PRESIDENT MEMBERS

SECRETARY

To my family.

Acknowledgements

These pages could not exist without the presence and help of many people with whom I shared pieces of my life in Barcelona.

First of all I want to thank my PhD advisor Hector Geffner, who followed me in all these years, always encouraging, with the patience of Job. Hector is one of those persons that leaves his mark, for his kindness and insight. I learnt a lot from him, he made me improve in many aspects of life, eventually making me a better scientist.

The people of the Artificial Intelligence group at UPF, Victor Dalmau, Hubie Chen, and Anders Jonsson, shared the everyday life at university, being witnesses of the hopes, the frustrations, and the rejoicings. Thanks to the personnel of the Department, for being always funny and helpful. I am grateful to my fellow PhD candidates Héctor Palacios, Nir Lipovetzky, Miguel Ramirez, and Emil Keyder: we shared so much together, from the nights in the office spent running for dead-lines, to the late rehearsals in the bedrooms at the conferences. With Héctor Palacios we walked a long way together, including the experience of preparing the tutorial on translation-based approaches to conformant and contingent planning at ICAPS'11.

Many colleagues honoured me with profitable exchanges of opinion, and very often with their friendship. I specially thank Blai Bonet, Son Thanh To, Marco Roveri, Carmel Domshlak, Jörg Hoffmann, Carlos Linarez, Eric Beaudry, for being so nice and generous at the moment of sharing their knowledge. I am deeply grateful to Piergiorgio Bertoli, he guided my first steps in the field of Automated Planning with kindness and passion. Thank you to Chiara Esposito for being so good-humoured while revising the proofs.

Friends are a great resource of love and support. Sira Ferradans has been my closest and best companion in all these years, sharing with me joys and sorrows. Ayman Moghnieh, with his enthusiasm and warmness, pushed me to carry on as the brother I never had. All my other flatmates, all nice and friendly, were my family here; Ewa, Kalle, Xiaolei, Lara, Rita, Filomena, Lisa, Nicole, and Anna, I owe you a lot. Emil and Nir, again, put up with my presence even in this aspect of life.

Fabien Girardin and Eva Fernández, Ivan Herreros and Maja Denzer, Rida Sadek and Nardine Osman, were always encouraging, challenging me with their ideas and exchanges of views. Support and friendship came also from Claudia Grossi and Mario Napolitano, fellow Neapolitan physicists that moved abroad to pursue here their research. Finally thanks to the many other friends that made Catalonia much more mine; among them Sandra Gilabert, Pep Viladomat, Mar Perez, Quimo Ribera, and Miguel Angel Carralero.

A final thanks goes to my family, that has been always supportive, despite myself. My father surely would have loved reading these pages.

Abstract

Artificial Intelligence Planning is about acting in order to achieve a desired goal. Under incomplete information, the task of finding the actions needed to achieve the goal can be modelled as a search problem in the belief space. This task is costly, as belief space is exponential in the number of states, which is exponential in the number of variables. Good belief representations and heuristics are thus critical for scaling up in this setting.

The translation-based approach to automated planning with incomplete information deals with both issues by casting the problem of search in belief space to a search problem in state space, where each node of the search space represents a belief state. We develop plan synthesis tools that use translated versions of planning problems under uncertainty, with partial or null sensing available. We show formally under which conditions the introduced translations are polynomial, and capture all and only the plans of the original problems. We study empirically the value of these translations.

Resumen

La Planificación es la disciplina de Inteligencia Artificial que estudia los procesos de razonamiento necesarios para conseguir las acciones que logren un objetivo dado.

En presencia de información incompleta, el problema de planificación puede ser modelado como una búsqueda en el espacio de estados de creencia, cada uno de ellos representando un conjunto de estados posibles. Este problema es costoso ya que el numero de estados de creencia puede ser exponencial en el número de estados, lo cual es exponencial en el número de variables del problema. El uso de buenas representaciónes de los estados y de heurísticas informadas resultan cruciales para escalar en este espacio de búsqueda.

En esta tesis se presentan traducciones para planificación con información incompleta, que transforman el problema de búsqueda en el espacio de estados de creencia, en búsqueda en espacio de estados, donde cada nodo representa un estado de creencia. Hemos desarrollado herramientas para la generación de planes para el problema traducido, ya sea con percepción parcial o nula. A su vez, demostramos formalmente bajo qué circunstancias las traducciones son polinómicas, completas y correctas. La evaluación empírica remarca el valor de dichas traducciones.

Preface

Automated Planning is concerned with the task of reasoning about action, with the objective of achieving a desired goal situation. The simplest form of automated planning, Classical Planning, involves full knowledge about the environment and deterministic operators; it is indeed the task of providing a sequence of operators, or plan, that given an initial state realises the goal. Classical planning can be seen as a path finding problem in an implicitly defined graph whose nodes represent states, specified in terms of the values of a set of variables, and where edges represent operators, specified in terms of sets of requirements and effects on the variables.

The model underlying classical planning can be extended to more general planning models, which include incomplete information and sensing on the environment where the planning agent is immersed. The main difference between planning with incomplete information and classical planning comes from the impact of uncertainty: in this setting, a planner must consider sets of states called belief states, which represent uncertainty about the world. The problem of planning with incomplete information has been then approached as a path-finding problem in belief space. Belief space is computationally exponentially bigger than state space –the belief space is the powerset of the state space– thus good belief representations and heuristics are critical for scaling up. However, the implementation of heuristic search in the belief space is more complex than in classical planning because of the difficulty of deriving good and informed heuristics, in the sense that heuristics are generally not well informed compared to heuristics used in classical planning tasks. On the other side, the belief representation and update is an intractable problem, in the worst case.

The translation-based approach to automated planning handled in this dissertation solves both issues by casting the problem of search in belief space as a search problem in state space, where each node of the search space represents a belief state. Moreover, the translation-based approach to planning under uncertainty exhibits good performance in relation to approaches that explicitly search in belief space.

In the first part of the dissertation we review the problem of automated planning, from the classical model, to the models with incomplete information and sensing. We then revise the translation-based approach to conformant planning introduced by Palacios and Geffner (2009), where conformant planning problems are translated to classical planning ones and then solved by state-of-the-art classical planners. Conformant planning is the task of finding solution plans in presence of partial information on the environment and no sensing available. A plan for a conformant problem is then a sequence of actions that drives all the states in the initial belief to the goal.

In the second part of the dissertation, we introduce new translations for Conformant Planning. New translations are needed as existent translation-based approaches need complete translations that may require exponential time and space, since incomplete translations may result in unsolvable problems. We present then a family of tractable translations for conformant planning based on sampling initial states. These translations are always complete, meaning that heuristics on the translated initial situation never evaluate it as unsolvable when it is not. We also formally describe the conditions under which such translations based on samples are also sound, and polynomial. Translations based on samples provide the backbone of the state-of-the-art conformant planner T1. In addition, to handle non-determinism in the actions' effects, we introduce different translations based on the general idea that non-deterministic effects can be compiled away by introducing new hidden artificial conditions each time a non-deterministic action is applied. This observation leads to a family of translations of conformant problems with non-deterministic effects into conformant planning problems with deterministic effects, which are themselves related to classical planning problems.

The third part of the dissertation introduces translations for Contingent Planning, which is the task of planning under incomplete information and partial observability. Unlike less complex forms of planning, in the contingent model of planning, sensing actions permit to unveil the truth values of hidden variables. Plans for contingent problems are then trees branching on observations, instead of action sequences as in conformant and classical planning. The challenge in producing general solvers for planning with partial observability add to the other difficulties of planning under incomplete information, the necessity to include sensing as a crucial element for successful plans. We propose then a translation for contingent problems, that maps them into non-deterministic but fully observable planning in state space. This compilation is linear in the number of possible initial states, which is in turn exponential in the number of fluents. We prove nonetheless that even in such cases, a sound, complete, and polynomial translation is possible, provided that the contingent problem has bounded contingent width, a parameter relevant to the difficulty of solving a planning problem with incomplete information. We show that the contingent width of almost all existing benchmarks is 1. Such translation cannot be solved by a classical planner, as the plans have different shape; we instead produce a relaxation of the problem that is a classical planning problem which provides an informed heuristic estimator that naturally embeds sensing in the relaxation. This approach has been implemented in the state-of-the-art contingent planner CLG. We then extend our translation for contingent planning problems to deal with those problems for which reaching the goal is not guaranteed *a priori*. The possible sources of dead-ends in a planning problem can come from the initial lack of information, the limits of the sensing model, or the incompleteness of the used translations. To overcome this issue, we gradually introduce automatically generated assumptions on the belief state in order to produce plans that work for as many states as possible, when finding a plan for all the states is infeasible.

Overview of Dissertation

Chapter 1 introduces Classical Planning, a deterministic and fully informed planning model.

Chapter 2 covers planning models with incomplete information and sensing.

Chapter 3 describes a translation for Conformant Planning problems into Classical Planning.

Chapter 4 introduces translations for Conformant Planning based on samples of initial states. Such translations are used in belief tracking and update, and to generate new heuristics.

Chapter 5 introduces a new family of incomplete translations for conformant planning with non-deterministic actions.

Chapter 6 presents translations that compile problems with sensing and incomplete information into non-deterministic fully observable problems. These translated problems are solved using a classical relaxation of the Contingent problem.

Chapter 7 shows how Contingent planning problems that are not always solvable can be solved by introducing automatically assumptions on the initial situation.

Chapter 8 presents a summary of the contributions of the dissertation and describes future work of planning with partial information and sensing.

The results presented in the dissertation have been published in the following articles:

- *Effective Heuristics and Belief Tracking for Planning with Incomplete Information*, by A. Albore, M. Ramirez, and H. Geffner, in 21st International Conference of Automated Planning and Scheduling (ICAPS-11), Freiburg, Germany, 2011 (Albore et al., 2011) [chapter 4].

- *Compiling Away Uncertainty in Non-Deterministic Conformant Planning Problems*, by A. Albore, H. Palacios, and H. Geffner, in 21st European Conference on Artificial Intelligence (ECAI-10), Lisbon, Portugal, 2010 (Albore et al., 2010) [chapter 5].

- *Acting in Partially Observable Environments When Achievement of the Goal Cannot be Guaranteed*, by A. Albore, H. Geffner, in Workshop on Planning and Plan Execution for Real-World Systems, at 19th International Conference on Planning and Scheduling (ICAPS-09), Thessaloniki, Greece, 2009 (Albore and Geffner, 2009) [chapter 7].

- *A Translation-based Approach to Contingent Planning*, by A. Albore, H. Palacios, and H. Geffner, in 21st International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, California, 2009 (Albore et al., 2009) [chapter 6].

- *Fast and Informed Action Selection for Planning with Sensing*, by A. Albore, H. Palacios, and H. Geffner, in Lecture Notes in Computer Science – Current Topics in Artificial Intelligence, 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA-07), Salamanca, Spain. Springer, 2007 (Albore et al., 2007) [chapter 6].

Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

Introduction

The aim of Artificial Intelligence is to program autonomous computer agents able to produce intelligent behaviour by simulating many of the higher functions of the human brain. The general problem of finding a solution to a given task has been tackled mainly following three approaches. First, *programming–based* approaches count on programmers to encode the method to solve a problem. In *learning–based* approaches, it is the program that improves itself by learning the adecuate solution, and this can be done by trial-and-error or based on the information provided by an instructor. *Model–based* methodologies rely on a general program which infers automatically a solution, starting from a suitable description of the actions, sensors, and goals.

Automated Planning is a model–based approach to autonomous behaviour, and is a quintessential discipline of Artificial Intelligence as it involves different aspects of what makes intelligent a conduct. The ability to reason well about the actions to perform, before acting, is certainly a central point to define intelligence, and is the main focus of our approach to Automated Planning.

The *classical* model for planning is a common restriction of the more general problem of selecting actions to reach a desired objective. Here, the actions are assumed to be deterministic and the information about the environment, complete. Classical planning can thus be cast as a path finding problem in a graph whose nodes are the states, and whose edges are the transitions that are possible, described in terms of actions. The purpose of classical planning solvers is to find an action sequence that reaches the target nodes from the root node of the graph.

In this chapter we introduce the classical planning model and how it is used to solve deterministic automated planning problems.

1.1 Automated Planning

Automated planning is the task of finding *plans*, i.e. solutions of a planning problem given following a specific representation. A solution plan is a path in a directed graph that drive the environment described in the problem from an initial node to a desired goal node.

Two contrasting approaches to automated planning have been followed since the '70s: domain–dependent and domain–independent. The first aims at solving tasks by ad-hoc approaches, involving high performance algorithms that exploit domain features for efficiency. Such domain–specific approaches are however not satisfying for our purpose of studying and designing machines that mimic autonomous and rational behaviour, as many aspects of reasoning are just focused on a specific problem, and not on understanding the process of planning in a whole. In contrast, domain–independent planning makes use of general languages for representing problem instances (e.g. PDDL), and general algorithms to solve all kind of tasks expressible in the language. The principal characteristic of this approach – that is also the one discussed in this dissertation – resides in the separation of the planning engine from the world model, which is given to the solver as part of the problem, together with the initial situation and the goal.

A planning task is defined by a model description provided by a language; solving an automated planning task requires then a description of the dynamic system, the *model*, whose states are considered by the planning agent. This model, also called *environment*, has to be driven by means of *actions* from an initial situation to a desired goal, or final situation. General language and models lead to high level representations that allows fast prototyping and that are often simple to modify and extend. An easy analogy can be made with the human process of representing tasks in an abstract way, that potentially involves different brain regions. Humans' and other animals'[1] remarkable capacity to switch between multiple tasks is bound to updating the many representations of the information needed to guide performance in complex tasks (Miller and Cohen, 2001; Baene, 2011).

Example 1.1 (A planning task)**.** *As an example of a planning problem consider the task of catching a train on time, while stopping to drink a coffee on the way to the train station. The initial situation would be that we are at home. Thus, reaching first the cafe and then the station would require the use of different means of transportation, a careful selection of the path leading to the locations, and use of the time–table and, eventually, gathering information about the traffic situation.*

To solve a planning task, the research in (domain–independent) automated planning has focused in the last 40 years on the development of general-purpose algorithms, called *planners*. The development of planners brought many families of algorithms and seach spaces to represent a planning problem and to solve it. We can cite planning-graph, and propositional satisfiability techniques that use powerful procedures for finding a solution. These techniques will be discussed in section 1.4. Constrain satisfaction techniques, encodes a planning problem into a constraint satisfaction problem, and uses many efficient methods to refine the plan space. These last three approaches have in common that the nodes of the search space can be viewed as a set of several partial plan, where each partial plan is a sequence of actions in the state space[2]. On the other hand, classical planning associates to every node of the search space a partial plan, such that any solution reachable from that node comprises all the actions of the associated partial plan.

[1]Just for comparison, animals with fewer than a hundred thousand neurons can approach food and avoid predators. In the human brain there are 100 billion or more neurons.

[2]When searching in the space of plans, a partial plan is a partially ordered set of actions.

Motivations

The aim of automated planning is to model real–life problems or puzzles, and to have automated systems solving them. The possible applications to such kind of problem–solving tasks are uncountable, going from automated rovers exploring afar planets to generators of dialogs for human–machine interfaces.

However, real world problems are considered hard, mainly because the dynamics of the domain are generally only partially known to a planning agent; this would involve unpredictable behaviours, unknown values of variables, and a huge number of variables describing the different aspects of the environment.

One way to limit the amount of possible contingencies of a problem, and to restrict the difficulty of finding a solution, involves algorithms for searching efficiently a simplification of the complex problem. Such simplification aims at reducing the number of variables involved, and the incomplete information about the behaviour of the environment affecting the planning task. The most common used simplification, or relaxation, of the (complex) planning task is known as the *classical planning* paradigm, and comprises full knowledge of the environment and to how a planner would affect it.

1.2 Models for Classical Planning

Methods for classical planning have had great success, and are in continual development and subject of research. The good results of classical planning techniques have cleared the path to approaches that simplify more complicated problems to classical planning problems, in order to solve them efficiently.

The paradigm of classical planning we use has a representation based on states and on transitions between them. Compared to more particular types of problems, classical planning restricts the model with some strong assumptions:

- finite set of states;
- deterministic transitions between states, caused by the agent's actions;
- full information about the initial state;

Model description

The objective of planning, and of classical planning in particular, is to find actions that drive the system from an initial state to a goal state. A classical planning problem can be translated as a directed graph whose nodes represent states, and whose edges represent actions. The change of state is then represented as a transition from a source node representing it along an edge and toward a target node representing the next state. A solution plan is then a path from the node in the graph representing the initial state to a goal node representing a state recognized as a goal state of the problem, i.e. a linearly ordered finite sequence of actions. The formal model underlying the planning problem can be described as follows:

Definition 1.1 (Classical Planning Model)**.** *The classical planning model Q is defined as the tuple $Q = \langle S, s_0, S_G, A, f\rangle$ where:*

- *S is a finite set of states,*
- *$s_0 \in S$ is the initial state,*
- *$S_G \subseteq S$ is the set of goal states,*
- *A is the set of operators (the actions),*
- *a state* transition *function $f : S \times A \rightarrow S :\ (s, a) \mapsto f(s, a) = s'$.*

s' is the state resulting from applying the operator a onto a given state s.

A fixed action a is applicable *in a state s when exists at least one target state s' such that $f(s, a) = s'$. We define the applicability domain as the subset S_a such that, $f|_{S_a}$ is injective. Executing a sequence of applicable actions $[a_0, \ldots, a_n]$ onto a given state s_0 results in a chain of states such that $f(s_0, [a_0, \ldots, a_n]) = [s_0, \ldots s_{n+1}]$, with $f(s_i, a_i) = s_{i+1}$, for $0 \leq i \leq n$, and a_i an applicable operator in s_i.*

A plan for a classical planning problem is given by a sequence of actions achieving a goal state from the initial state of the problem:

Definition 1.2 (Classical Plan)**.** *A plan π for a classical problem $P = \langle S, s_0, S_G, A, f\rangle$ is an action sequence $\pi = [a_0, \ldots, a_n]$ that, once applied on the initial state, results in a sequence of states $[s_0, \ldots, s_n]$, such that:*

- *all actions are applicable, and $s_{i+1} = f(s_i, a_i)$, for $0 \leq i \leq n$.*
- *the sequence terminates in a goal state: $s_{n+1} \in S_G$.*

We sometimes use the writing $result(\pi, s_0) = s_{n+1}$ to indicate that the state resulting from the execution of π on s_0 is s_{n+1}.

To evaluate a plan π, its length $|\pi|$ is commonly considered as a preference criterion, and it corresponds to the number of actions in the plan. This is a special case of planning with costs, where to every action is associated a positive cost:

Definition 1.3 (Classical planning model with costs)**.** *A planning model with costs P_c consists of a planning model $P_c = \langle S, s_0, S_G, A, f\rangle$ along with a function $c : A \mapsto \mathbb{R}_0^+$ that maps each operator in the model to a non-negative cost.*

Definition 1.4 (Plan cost)**.** *The cost of a plan $\pi = [a_1, \ldots, a_n]$ is given by*

$$cost(\pi) = \sum_{i=1}^{n} c(a_i)$$

The plan length corresponds with the cost of a plan when all the costs are 1. In a planning model with non–unitary costs, plans with lower cost are preferred to plans with higher cost.

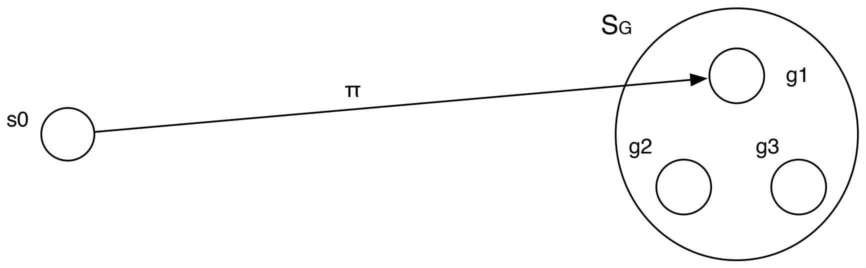


Figure 1.1: The objective of classical planning is to find a plan π, wich is a sequence of actions that provide the transitions from the initial state s_0 to a desired goal state in S_G.

Modelling language

The representation of a planning problem seen so far is general but not effective as it is often costly to represent explicitly all the states of a planning problem. Classical planning problems can be very big. We are interested in problems whose state spaces are too large to be represented explicitly, and factored representations must be used.

A factored representation represents states via a set of variables, or *fluent*[3], interpreted as a conjunction, and such that each state s is a complete assignment of the state variables. The goal states and the operators, as well as the applicability and transition functions, can also be described in terms of these state variables. In particular, the actions encoding the transitions between states are expressed in terms of preconditions and post–conditions. Action preconditions specify the conditions under which an action can be applied. The post-conditions specify the changes to variable assignments made by the effects of the applied actions.

The effects of the actions describe the changes an action a makes to the world. In the case of a factored representation that consists only of Boolean variables, these changes are commonly specified in terms of *add lists* and *delete lists.* For an action a, the add list add(a) specifies the properties that a makes true, while the delete list del(a) specifies the properties that a makes false. All other variable assignments are left unchanged by the action; we often refer to that rule as a solution to the *frame problem* (McCarthy and Hayes, 1969).

The Boolean factored representation is surely the simplest and most common, and is widely used in automated planning. In the planning language called STRIPS (Fikes and Nilsson, 1971; Nilsson, 1980) state variables are Boolean, so each such variable indicates whether a proposition about the world is true or false in a given state.

Definition 1.5 (STRIPS)**.** *A planning problem in STRIPS is defined as a 4–tuple* $\langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$*, consisting in:*

$\mathcal{F}$**:** *a set of Boolean variables (fluents),*

$\mathcal{A}$**:** *a set of operators, where each action a is a pair of preconditions and post–conditions: $\langle$ pre(a), eff(a) $\rangle$.*
The precondition pre(a) is a set of fluents over $\mathcal{F}$.
The effects are conjunction of fluents, described in terms of simple effects*, such that an effect eff(a) is either*

[3] Factored representations can use multivalued variables, but "fluents" usually refers to Boolean variables only.

- *a* simple add effect *e, where $e \in \mathcal{F}$,*
- *a* simple del effect *$\neg e$, where $e \in \mathcal{F}$, or*
- *a* conjunctive effect *$e \wedge e'$, where e and e' are effects.*

$\mathcal{I}$**:** *a set of fluents $\mathcal{I} \subseteq \mathcal{F}$, describing the initial state*

G**:** *a set of fluents $\mathcal{G} \subseteq \mathcal{F}$, describing the set of goal states.*

In STRIPS, following the Closed World Assumption, the unmentioned literals are false, so a state can be described only by literals that hold in it.

A STRIPS problem defines a state model as in definition 1.1 in the following way:

- the set of states S is defined in terms of the set $\mathcal{F}$ of fluents, s.t. $S = 2^{\mathcal{F}}$;
- the initial state s_0 is described by the assignment $\mathcal{I}$, such that in s_0 the fluents $p \in \mathcal{I}$ have the value true and all other fluents have the value false;
- the goal states are described by a (partial) assignment of the fluents in $\mathcal{G}$, such that in all the states in S_G, the fluents $p \in \mathcal{G}$ have the value true;
- the applicability is fixed by the preconditions of the actions: a is applicable in s if $\text{pre}(a) \in s$;
- the transition function is defined by the action effects, s.t. $s' = f(s, a)$ s.t. $s' = s \setminus \text{del}(a) \cup \text{add}(a)$.

Planning languages extend STRIPS in many features. In this dissertation we consider extentions of STRIPS that allows negative literals, whereas is STRIPS only conjunctions of positive literals are permitted, with a particular reference to actions' preconditions and goals. We use $\neg L$ to refer to the complement of L. Moreover, the effects of actions can be conditioned on the truth values of fluents. The difference between a *conditional effect* and a precondition resides in that the precondition *must* be satisfied in order to make the action applicable, while a condition that doesn't hold just doesn't produce the corresponding effect.

We extend the representation in definition 1.5 with operators with conditional effects:

Definition 1.6 (Conditional effect)**.** *The effects are conjunction of fluents, described in terms of* simple effects *such that an effect eff(a) is either*

- *a* simple add effect *e, where $e \in \mathcal{F}$,*
- *a* simple del effect *$\neg e$, where $e \in \mathcal{F}$,*
- *a* conditional effect *$C \rightarrow e$, where C is a conjunction of variables over $\mathcal{F}$ and e is an effect, or*
- *a* conjunctive effect *$e \wedge e'$, where e and e' are effects.*

Semantically, the consequences of conditional effects apply in the target state s' only if the condition hold in the source state s, and the action is by itself executable on that state: for a conditional effect of an action $a : C \rightarrow e$, the transition function is defined a follows:

$$s' = f(s,a) \text{ s.t. } s' = \begin{cases} s \setminus \mathrm{del}(a,s) \cup \mathrm{add}(a,s) & \text{if } pre(a) \subseteq s \\ undefined & otherwise \end{cases}$$

where:

- $\mathrm{add}(a,s) = \begin{cases} \mathrm{add}(e) & \text{if } C \subseteq s \\ \emptyset & \text{otherwise} \end{cases}$

- $\mathrm{del}(a,s) = \begin{cases} \mathrm{del}(e) & \text{if } C \subseteq s \\ \emptyset & \text{otherwise} \end{cases}$

Informally, we say that when an operator makes a set of fluents true, it *adds* these fluents, and *deletes* those that it makes false. We will refer to the set of fluents made true/false by an operator as its add/delete list, respectively.

As any planning problem described in terms of the above semantics, can be translated into an equivalent STRIPS instance, even if existing compilation techniques are worst-case exponential. This preprocessing grounding phase is commonly applied by most of the existing planners.

Example 1.2. *Consider the problem of a robot in a $1 \times N$ grid, with $N = 5$, that has to go from an extreme of the corridor (the hashed cell p_2 in Figure 1.2) to the goal cell at the other side of the corridor ("*goal*" cell in the figure). The position x of the agent is denoted by $\mathtt{at}(p_x)$. This problem can be modeled as a classical planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, with:*

Fluents $\mathcal{F}$: $\mathtt{at}(p_x)$, *for any x in* $[1, N]$,

Actions $\mathcal{A}$:

- $\mathtt{left}(p_x)$ *for any* $x \in [1, N-1]$
  Precondition: $\{\ \mathtt{at}(p_x)\ \}$ *Effect:* $\{\ \mathtt{at}(p_{x+1}),\ \neg\mathtt{at}(p_x)\ \}$
- $\mathtt{right}(p_x)$ *for any* $x \in [2, N]$
  Precondition: $\{\ \mathtt{at}(p_x\ )\}$ *Effect:* $\{\ \mathtt{at}(p_{x-1}),\ \neg\mathtt{at}(p_x)\ \}$
- $\mathtt{left}(p_5)$
  Precondition: $\{\ \mathtt{at}(p_5)\ \}$ *Effect:* $\{\ \mathtt{at}(p_5)\ \}$
- $\mathtt{right}(p_1)$
  Precondition: $\{\ \mathtt{at}(p_1)\}$ *Effect:* $\{\ \mathtt{at}(p_1)\ \}$

Initial situation $\mathcal{I}$: $\{\ \mathtt{at}(p_2)\ \}$

Goal $\mathcal{G}$: $\{\ \mathtt{at}(p_4)\ \}$

NB: Moving toward the wall has no effect, e.g. going left from cell p_5 will leave the agent in p_5.

A possible solution for the problem would be to move the agent two times to the left, as in the sequence:

$$\pi = \left[\ left(p_2), left(p_3)\ \right]$$

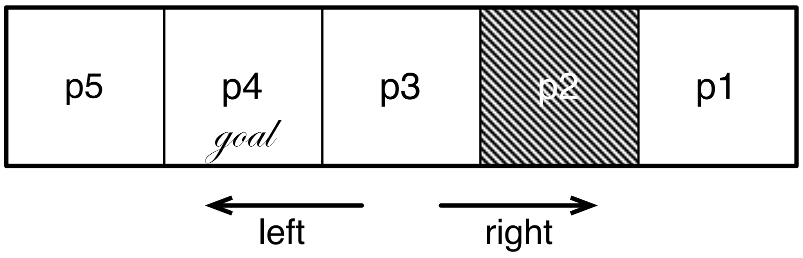


Figure 1.2: Navigation problem in a 1 × 5 grid corridor. The PDDL code of this problem in reproduced in Figure 1.3 and 1.4.

The problem can be reformulated in a different way, using conditional effects: instead of having 10 actions, we can encode 2 moving actions with 10 possible conditional effects. This change would remove the precondition, and hence would give a slightly different meaning to the action: with preconditions, the position of the agent has to be known before moving, while with conditional effects, the action can always be applied[4].

We show the example of how to change in that way the 4 `left`(p_x) action above in one action `left`:

`left`:
Precondition: $\emptyset$
Effect:
$\{\ \mathtt{at}(p_x)\ \} \longrightarrow \{\ \mathtt{at}(p_{x+1}),\ \neg\mathtt{at}(p_x)\ \},\quad \textit{for any } x \in [1, N-1].$
$\{\ \mathtt{at}(p_5)\ \} \longrightarrow \{\ \mathtt{at}(p_5)\ \}$

Using this encoding, a solution plan would be:

$$\pi = \left[\ left, left, left\ \right]$$

PDDL

Planning problems are generally expressed in the Planning Domain Description Language (PDDL). First defined in 1998 by McDermott, the PDDL language has been updated and extended through the years to match advances in planning and the evolution of the needs of the planning community. PDDL is capable of representing the semantics of both STRIPS and the ADL extension(Pednault, 1989) languages since its version 1.2, used for the IPC of 1998. Nowadays the PDDL language is widely used in the planning community and includes many specific features, as trajectory constraints for temporal reasoning or soft constraints to express preferences. Even

[4]In deterministic classical planning, this difference doesn't affect the problem and its solution, but this changes when dealing with incomplete information, as we will see in the next chapters.

complex actions with control flow blocks inspired by imperative programming languages can be compiled as ordinary PDDL actions usable with standard off-the-shelf planners (McIlraith and Fadel, 2002; Baier and McIlraith, 2006; Petrick, 2009; Claßen et al., 2007). The version that is relevant to the topics discussed in this thesis is the most basic one with Boolean state variables only. Actions in PDDL are expressed as schemata instantiated with objects, as shown in the following example:

```
(:action move
   :parameters (?s1 ?s2 - cell)
   :precondition (at ?s1)
   :effect (and (not (at ?s1))
                (at ?s2)))
```

Here the action *move*, referred to the former example, takes two parameters `s2` and `s2` that will be eventually instantiated with the possible values assumed by the *cell* objects declared in the problem description. These variables appear also in the preconditions and the body of the action effects.

Algorithms for Classical Planning

As commented above, solving classical planning problems can be cast as path-finding in a directed graph whose nodes represent states, and whose edges represent state transitions due to actions. Classical planning problems can then be solved by using graph search algorithms to find a path from the initial state to a goal state. This graph search approach is not trivial because the size of the graph may be exponential in the size of the description of the planning problem in propositional form (i.e. the number of fluents of the problem). Thus, blind search algorithms such as depth-first or Dijkstra are practically unfeasible.

An approach that has proved to be effective relies on to use heuristic search. Heuristic search uses heuristic functions to evaluate the cost-to-go from a node to a goal, or to be more general, to provide a ranking of a set of nodes in order of their relative desirability (Ghallab et al., 2004, chap.9). This estimation of the distance in the search space is then used by the search algorithm to drive the state space search, preferring to visit nodes considered more promising from their heuristic value.

Planning as Heuristic Search (Bonet and Geffner, 2001a) is sound and complete by construction, as far as the used search algorithm is complete, given that the state space contains exactly all the possible plans as paths from the initial state to any goal state.

The "best first" algorithms used for heuristic search (also called "informed" search) expand always the best state according to some evaluation function. Optimal algorithms also make use of heuristics to speed up the search; in this family we find A* (Hart et al., 1968) and IDA* (Korf, 1985). Other local search algorithms such as simulated annealing (Kirkpatrick et al., 1983; Černý, 1985), or tabu search (Glover, 1989, 1990), are little used in planning.

Many successful heuristics are obtained by solving a simpler version of the original problem relaxing its constraints (Pearl, 1983). Relaxations directly derived from the problem description are useful and efficient, such as the successful "delete relaxation",

```
(define (problem corridor-5)
  (:domain corridor)
  (:init  (x p1) )
  (:goal  (x p5) )
)
```

Figure 1.3: PDDL encoding of a navigation problem in a corridor grid. The agent in a 1×5 grid starts at position 1 and must get to position 5.

```
(define (domain corridor)
  (:requirements :typing)   (:types pos)
  (:constants p1 p2 p3 p4 p5 - pos)
  (:predicates (x ?p - pos) )
  (:action left-p5
     :precondition (x p5)
     :effect (x p5)
  )
  (:action left-p4
     :precondition (x p4)
     :effect (and (not (x p4)) (x p5))
  )
  ...
  (:action left-p1
     :precondition (x p1)
     :effect (and (not (x p1)) (x p2))
  )
  (:action right-p1
     :precondition (x p1)
     :effect (x p1)
  )
  (:action right-p2
     :precondition (x p2)
     :effect (and (not (x p2)) (x p1))
  )
  ...
  (:action right-p5
     :precondition (x p5)
     :effect (and (not (x p5)) (x p4))
  )
)
```

Figure 1.4: PDDL encoding of the actions in a navigation problem in a 1×5 corridor. The available actions are moving *left* and *right*.

obtained by dropping the negative effects of the actions. Many planners use heuristic search (McDermott, 1996; Bonet et al., 1997), which is now the most successful used approach to classical planning.

One of the first approaches making use of domain-independent heuristics is the HSP planner (Bonet and Geffner, 1999). HSP used best-first search coupled with h_{add} heuristic that approximates the distance between two states by summing the distances between the propositions in the states, ignoring the delete effects.
The Fast-Forward (FF) planner by Hoffmann and Nebel (2001) is based on the same delete relaxation as HSP but uses an explicit solution of the relaxed problem to estimate its heuristic h_{FF} and the extraction of helpful actions applied first when searching for a plan. When the FF's incomplete but effective greedy search (called "enforced hill climbing") based on helpful actions only fails, the planner launches a best-first search. The helpful actions are defined in the FF planner as those operators applicable in the current state that add some precondition of an operator in the plan. This search control technique has proved to be quite successful and effective, being the base of many developments (Hoffmann, 2002; Brafman and Hoffmann, 2006; Hoffmann and Brafman, 2005b).
LAMA planner (Richter and Westphal, 2010) makes use of a pseudo-heuristic derived from *landmarks*, i.e. propositions that must be true in every solution of a planning task (Hoffmann et al., 2004; Porteous and Cresswell, 2002). LAMA is built on top of the Fast Downward Planning System, using in particular a multi-heuristic search. The "landmark counting heuristic" (Richter et al., 2008) estimates the goal distance of a state s by counting the number of landmarks that still are needed to be achieved. Like FF's helpful actions, LAMA uses preferred operators along with the landmark heuristic.
The planner PROBE (Lipovetzky and Geffner, 2011) implements a dual search architecture for planning that is based on the idea of "probes": single action sequences computed without search from a given state for achieving a serialization of the problem subgoals that is computed dynamically along with the probe. A probe, by its construction can go quickly deep into the state space, terminating either in the goal or in failure. When the goal is not achieved, the states expanded along the probe are added to the open list, and control returns to the greedy best first search loop. PROBE is a complete planner using the standard additive heuristic h_{add}.

1.3 Complexity

The plan existence problem in the classical setting, i.e. the problem of deciding if there exists a valid plan for an arbitrary problem instance, in the propositional STRIPS planning model is decidable and has been shown to be PSPACE–complete (Bylander, 1994). This regards the fragment of STRIPS we discussed in this chapter, while the original STRIPS language (Bylander, 1994) allows for infinite state spaces and is undecidable. Planning with costs (cf. definition 1.3), which involves the optimization problem of finding a valid plan of minimal cost for an arbitrary problem instance, has the same class of complexity. The plan existence problem for ADL is also PSPACE–complete for any allowed complexity of precondition and effect formulæ (Baier, 2003). As noted above, an ADL problem can be translated into a STRIPS instance, but existing compilation techniques are worst–case exponential (Gazen and Knoblock, 1997).

1.4 Other techniques for Classical Planning

Automated planning has been strongly influenced, in its origins, by the work on automated theorem proving (Green, 1999). Logical description of the problem comes from one of the first formulations of planning problems as axiomatic description of initial state, goal, and operators. The original STRIPS representation used first order formulæ that added more expressive power. This representation has been then restricted to the actual one, where preconditions and effects are specified in terms of atoms. The STRIPS representation has been extended in many ways, while staying within the confines of the classical planning model.

The representation of the planning problem in terms of preconditions and post-conditions adapts well to state-space search algorithms. It is possible to search in both directions: from the goal to the initial state, and vice versa. It is also easy to derive heuristics automatically from the explicit goal and actions representation, in particular under the subgoal independence assumption, when interaction between goal atoms are not considered, and other relaxation of the planning problem.

Other techniques apply to the classical planning formulation. They involve the construction of planning graphs and the translation of the planning problem into propositional axioms, in order to consequently apply a satisfiability algorithm to find a model that then corresponds to a solution plan.

Planning graphs

A planning graph is constitued by the levels obtained by alternating fluents and actions layers. The first layer includes all the fluents that can obtined in the initial situation, then the second layer is made by all the actions that can be applied from the literals true in the former layer. The third layer is constitued by fluents that can be obtained from applying the actions in the precedent layer, and so on until fixpoint. fluent level including all the goal literals.

Such a planning graph is a useful structure from which information can be drawn into. The first immediate information regards *reachability*: a literal not included in the graph cannot be obtained in the problem. Such a graph can also be used as a heuristic estimate: the cost of obtaining a literal is given by the layer it appears first.

The GraphPlan algorithm (Blum and Furst, 1995; Kambhampati et al., 1997; Anderson et al., 1998; Smith and Weld, 1998) applies this approach, expanding the graph until all the goal literals are reached in a level, with no mutex links between any pair of them, i.e. if no conflict between actions prohibits two literals to be present at the same time in the same state. When such a level is reached, the algorithm intents to extract a plan from the graph.

Planning as Satisfiability

Planning as satisfiability is a powerful approach to automated planning first proposed by Kautz and Selman (1992). The approach translates a STRIPS problem and a horizon in a propositional logical formula whose satisfiability is checked. Of couse the formula includes the initial situation, the goal, and all the possible action applications, stored as propositional axioms. *If the planning problem is unsolvable, the SAT formula will be unsatisfiable.*

The main issue of planning as satisfiability comes from the encoding of the problem as a formula, being the memory required to store the propositional axioms the bottleneck of the approach. Recent work has shown that the conflict-directed clause learning algorithm (CDCL), which most the current best SAT solvers use, together with an extremely simple planning-specific scheme for selecting decision variables lead to very competitive planning, matching in efficacy other search paradigms (Rintanen, 2010b). Simple heuristics on top of the basic variable selection scheme improve the efficiency of SAT based planner even further (Rintanen, 2010a).

1.5 Thesis outline

Automated planning involves the practice of reasoning about acting. So far we have considered classical planning, the simplest form of planning which involves a deterministic action model, and complete information about the environment and the planning agent's state. In problems with incomplete information, planning is done by considering *belief states* instead of states, and a belief state is the set of all the states that are regarded as possible in a given (uncertain) situation. Thus, solving a problem involves finding a solution for all the states deemed compatible with the initial situation. Planning with incomplete information presupposes using the agent's sensing abilities to discover its environment and to reduce the uncertainty. Planning under uncertainty is mainly divided in two models: the *conformant* planning setting, where no sensing is available, which is like "moving in the dark", and the *contingent* planning setting, in which the agent can make use of limited sensing. In both settings, a solution plan is applicable in all the states of the initial belief, and drives them to the goal.

Planning problems under incomplete information, with or without sensing available, can be cast as a path-finding problem in a belief states space. There the main challenges come from deriving the heuristics to guide the search, and the representation and update of the beliefs. The translation-based approach to planning under uncertainty is elegant and exhibits good performances compared to approaches that explicitly search in belief space. This dissertation will describe new translations for conformant and contingent planning that are competitive with the state-of-the-art methods. A translation that captures all and only the solutions of the original planning problem is said to be *complete* and *sound*. In the worst case these translations are exponential, but for a large collection of problems they can be shown to be polynomial, sound and complete. We identify then a parameter for planning problems under uncertainty and sensing called the *width*, and show the conditions under wich the translations introduced are sound, complete, and polynomial. The complexity of sound and complete translations is exponential in the width parameter, that for most conformant and contingent benchmarks turns out to be bounded and equal to one.

In the first part of the dissertation, we present some background on automated planning. In the present chapter we reviewed classical planning, while in the next chapter we will describe models for planning under uncertainty. We then review the translation introduced by Palacios and Geffner (2009), that maps conformant planning problems into classical ones that are then solved by a state-of-the-art classical planner. This translation is the starting point of our research.

The second part of the dissertation describes two translation-based approaches to conformant planning. The first uses a set of states sampled from the initial belief state to provide informed heuristics, and to keep track of the belief states along the plan execution (cf. chapter 4). We will see that it is possible to identify a *basis* of sampled states for which all the solution plans are also solution plans of the whole planning problem. In chapter 5 we then define different translations for conformant problems with non-deterministic actions. Even if incomplete, such translations have been proved to be quite effective.

In the third part of the dissertation we tackle the problem of contingent planning, which is planning with incomplete information and partial observability. We delineate in chapter 6 a translation-based approach that compiles a contingent planning problem P into a non-deterministic but fully observable problem $\mathcal{X}(P)$. This planning problem is then solved using a relaxation $\mathcal{H}(P)$, that is a classical planning problem and that can be fed to a state-of-the-art classical planner that provides the next-to-apply action. This part ends with chapter 7, where we introduce an extention of contingent planning that allows us to automatically introduce assumptions on the (hidden) state of the world, in order to eventually find a solution for those planning problems that, because of limited sensing or because of the possibility of a dead-end in the search space, are not solvable by current planners.

We then summarise the contributions of this thesis, and discuss future work and applications.

1.6 Summary

The classical planning model can be cast as a search problem in a directed graph, where nodes represent states, and edges, actions. By shifting the planning problem to a factored representation, the states and the edges of the state space are implicitely represented: states are expressed as a complete assignment to a set of variables, and actions are transformations of the variables' value, in terms of pre-conditions and post-conditions. The heuristic search approach to planning uses heuristic functions, extracted automatically from the factored problem representation, for guiding the search for a plan from the initial state of the state space to a goal state.

Classical planning domains are deterministic, and complete information on the initial situation is assumed. This implies that the state of the domain is always perfectly known during a plan execution. In an uncertain environment, on the other hand, this assumption does not hold anymore, and the agent has to deal with incomplete information, because the world is partially observable, non-deterministic, or both. These kinds of representations shift the complexity of finding a solution to a planning problem from PSPACE to the more complex tasks of conformant and contingent planning, respectively EXPSPACE and 2–EXP.

Chapter **2**

Planning under uncertainty

In the precedent chapter we considered the model of classical planning, where complete information on the initial situation is available, and actions are deterministic. These two strong assumptions are certainly useful, but they also are limiting for modelling realistic domains. Complete information about the initial state and deterministic actions imply that the system can only evolve along a single completely specified sequence of states, ignoring any non-deterministic or exogenous effect. But the majority of the problems we are interested in solving are contexts where the agent does not hold *a priori* all the information necessary to solve the problem, so they fall out of the scope of classical planning.

More elaborated models of planning go beyond the limits of classical planning and involve incomplete information about the status of the environment, actions with non-deterministic effects, and the ability to sense the actual state of the system. Algorithms for planning with incomplete information and sensing provide a general enough solution that maps all the possible initial states onto the goal: due to partial information on the initial situation, it is not longer satisfactory to find a single candidate solution that works for a single and uncertain initial state.

Planning under uncertainty is motivated by realistic applications: planning agents can be immersed in an unknown environment, similarly to what happens to autonomous rovers moving on the surface of planets, or in the depths of oceans. In the case of interacting agents, the inner knowledge of either agent is not known to the others, so that their behaviour can be encoded as planning with incomplete information.

In this thesis we will tackle those extensions of classical planning that encode incomplete information about the characteristics and the dynamics of the planning domain, including the eventual deficiencies of the sensors' response and the planning agent's incomplete knowledge about its current situation.

2.1 Incomplete information and sensing

To move automated planning closer to realistic tasks, it is necessary to include the incomplete information about the planning agent's state and about the effects of the actions in the planning model. In presence of uncertainty, the agent's beliefs about the environment can be different from the actual state of the environment, on the other hand, physical actions having effects on the environment can also effect the beliefs of the agent, modifying its knowledge about the world. As this distinction is made, it is accepted that other kind of actions can modify only the mental state of the agent, i.e. its knowledge about the environment, without changing it. These actions are called *sensing actions*, or observations. In classical planning, given that the actions all have deterministic effects and that the initial situation in perfectly known, the agent does no need to sense the environment.

Consider a situation where a lever opens one of the two opposed closed exits of a room, but which one is not known initially. After pulling the lever and before moving toward a door, a sensing action should be done to check which door has been opened. This is the kind of problems that planning with incomplete information encode: where acting changes the state of the world and sensing drives the agent toward the proper solution.

In problems with sensing actions, the complete state of the system cannot always be sensed totally, and a limited amount of knowledge is generally gathered from each observation. This restriction on observability comes from different causes. First, certain aspects of the environment cannot be sensed because of a lack of proper sensors or because they are simply hidden. Second, the planning domain can be simply too large or complex to be entirely represented, and its representation should be limited to a certain level of granularity. Finally it is very possible that the sensing is subject to noise, which makes it piecemeal and approximate.

The availability of sensing defines two main models of planning with incomplete information: *conformant planning*, where no sensing is available at all, and *contingent planning*, that involves partial observability of the environment.

Planning as Heuristic Search in Belief Space

The main difference between classical planning and planning under uncertainty resides in the agent's incomplete knowledge modelled as a *belief state* (Bonet and Geffner, 2000). In a given situation, the belief state is the set of all the states deemed possible by the agent. In classical planning, the initial situation is perfectly defined and the actions are deterministic; this means that along the whole plan, the state is known and no uncertainty affects the problem. On the other hand, when the initial situation is uncertain due to incomplete information or when non-deterministic actions are elements of the problem, it is not possible to consider only a single current state, unless fully observability is assumed, yet a *set* of states that potentially represent the current situation must be considered.

The most common approach to planning under uncertainty is based on the belief state formulation. Similarly to what happens in the case of planning in a state space, under uncertainty belief states are related by an accessibility relation f, characterised as in definition 1.1. This mean that we do not tackle here other forms of planning with partial information, like probabilistic approaches. From these definitions, a planning task can be seen as a path–finding problem in a directed graph where the nodes are belief states. The source node b_0 of the graph corresponds to the belief state of the initial situation, and the target nodes are those representing belief states where the goal holds in any one of the states. In this context, every action a executable in a belief state b, maps b to another belief state b':

$$b' = \{s' \mid s' = f(s, a) \text{ for all } s \in b\} \tag{2.1}$$

Belief space planning requires, as for classical planning, verifying the action preconditions before execution: the preconditions must be true in the belief in order to apply the action effects. We say that a formula is true in a belief state b if it is true for every state $s \in b$. Consequently, an action a is executable in a belief state b if its preconditions are true in every state s in the belief b. It is in belief space that the conditional effects of actions from definition 1.6 acquire a major role. If the preconditions say when an action *possibly* has effects in a belief state, the conditions define the set of states where the effects take place. It is then also possible that an executable action has no effects, when the conditions of the effect do not hold.

We indicate with $result(\pi, b)$ the belief state resulting from applying the plan π on the belief b, given that π is applicable in b. The resulting belief will be the union of all the states of b progressed through π:

$$result(\pi, b) = \bigcup_{s \in b} result(\pi, s) \tag{2.2}$$

Strong and weak solution plans

In planning with belief states, a solution plan guarantees mapping all the possible initial states onto the goal. When such a solution plan exists, it is called a *strong solution* or simply a solution. When a solution plan does not exist for all the states of the initial belief, it might still be feasible to find a solution that applies to at least one state in the initial belief; this kind of solution plan is called *weak solution*, and is generally avoided as it does not guarantee the achievement of the goal.

Searching in belief space for solution plans is a problem exponentially larger than searching for solution plans in the classical setting, as the number of possible belief states is exponential in the number of states. This source of difficulty is tackled in this thesis by considering "translations" applied to planning problems, that may consider only a subset of initial states, possibly polynomial, even when their number is exponential. Translation-based approaches to planning under uncertainty is the subject of this thesis. Out aim is to cast the planning problem with incomplete information as a planning problem in the state space by considering, when possible, approximations to solve the original planning problem, or to guide the search by providing informed heuristics. In particular, we will prove that under certain circumstances, our translation encodes all the uncertainty of the problem.

When incomplete information is taken into account, the reasoning abilities of the system should tackle with reducing the uncertainty of the problem, which provides solution that are radically different from the one in classical planning. In fact, instead of proceeding directly to the goal in a greedy way, a solution plan would generally first remove the ambiguity about the situation by driving the domain into a known state or by performing sensing actions, and only then go to the goal.

As a motivating example, let's consider an agent in a maze, where the initial position of the agent is unknown and the goal is to find a way out of the maze. A map of the maze is given to the agent, yet it is not possible for it to initially know its location on the map. Moving directly to the exit is not possible as the path to the exit cannot be immediately known. The expected behaviour of the agent would be to look for reference points or to try to find a particular configuration that would help it determine its location with certainty.

Planning under uncertainty, as it is treated here, is divided mainly in two tasks of increasing model's complexity: *conformant planning*, where no sensing is available, and *contingent planning*, with partial observability.

2.2 Conformant Planning

The conformant planning model can be seen as classical planning with the addition of uncertainty, and no sensing available. The different sources of uncertainty may be compiled into a problem with deterministic actions only in which all the uncertainty is encoded in the initial situation (Bonet and Geffner, 2000). The task of conformant planning is to find a sequence of actions, the plan, that only requires the application of executable actions, and whose execution is guaranteed to solve the problem in spite of the incomplete information about the problem.

Conformant planning with deterministic actions is one the simplest form of planning with uncertainty. A deterministic conformant problem is like a classical problem but with many possible initial states instead of one, and a plan is conformant when it is a valid plan for each possible initial state. In spite of its simplicity, the conformant planning problem is harder than classical planning, as plan verification remains hard even under polynomial restrictions on plan length (Haslum and Jonsson, 1999; Baral et al., 2000; Rintanen, 2004; Turner, 2002). Few practical problems are purely conformant, but the ability to find conformant plans is needed in planning with sensing, of which conformant planning is a special case where no sensing is allowed. Indeed, relaxations of planning with sensing into conformant cases lies at the heart of recent methods for computing contingent plans (Hoffmann and Brafman, 2005a; Albore et al., 2009) and deriving finite-state controllers (Bonet et al., 2009).

Conformant planning can be formulated as a path-finding problem in belief space. Computational challenges faced in this formulation are the derivation of heuristics to guide the search, and belief representation and update (Bonet and Geffner, 2000). This formulation is the basis of the most recent conformant planners such as Conformant-FF (Brafman and Hoffmann, 2004), MBP (Bertoli et al., 2006), POND (Bryce et al., 2006), CNF (To et al., 2010), DNF (To et al., 2009), and T1 (Albore et al., 2011). The exception is the planner T_0 which is based on a translation of conformant problems P in classical problems $K(P)$ that are solved by off-the-shelf classical planners (Palacios and Geffner, 2009).

Model for Conformant Planning

A conformant planning problem can be described in terms of search in the belief space with deterministic actions. Solving a conformant planning problem involves selecting actions to achieve a goal state. However, unlike classical planning, such a plan should do so from each possible initial state.

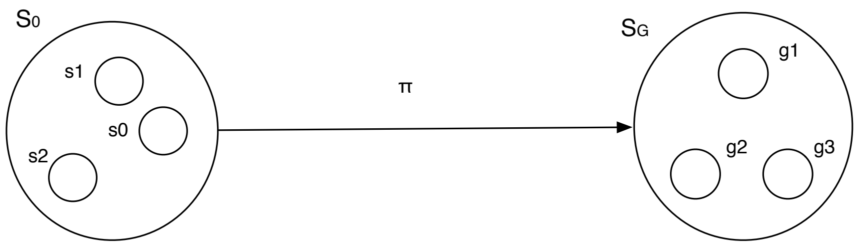


Figure 2.1: The objective of conformant planning is to produce a plan π, i.e. a sequence of actions that provide the transitions from all the initial states in S_0 to a goal belief state S_G.

As with classical planning, the model underlying conformant planning can be formally described with a state space:

Definition 2.1 (Conformant Planning)**.** *The conformant planning model Q is defined as the tuple $\langle S, S_0, S_G, A, F\rangle$, where:*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is the initial non empty belief state,*
- *$S_G \subseteq S$ is the set of goal states,*
- *A is the set of operators (the actions), with non–deterministic effects,*
- *F is the state transition function for a non deterministic action, and it is a map $F : S \times A \to 2^S : (s, a) \mapsto (s_1, ...s_n)$. We can define F in terms of deterministic transition functions as in definition 1.1, considering $F = (f_1, ...f_n)$ where we have $f_i(s, a) = s_i$, with $1 \leq i \leq n$.*

  The operator a is applicable *in a state s if F gives at least one target state, i.e. if exists at least one s' such that $F(s, a) = s'$. In terms of deterministic functions, which are injective by definition, we can rephrase this definition as follows: a is* applicable *in a state s if exists at least one f_i such that $f_i(s, a) = s_i$. The applicability domain here is the union of the applicability domains for any f_i, in other words, we consider the subset of S given by the states s such that $F(s, a) \neq \emptyset$. We remark here that the map F restricted to the applicability domain is injective.*

By extension, when considering a belief state b, a is applicable in the belief b if $F(s_i, a)$ gives at least one target state for any $s_i \in b$. Hence the set of applicable operators in the belief are those operators applicable in every state of b: $\{a \mid F(s_i, a) \neq \emptyset, \forall s_i \in b\}$.

Applying an operator a in a given belief state b results in the successor belief state b' defined when a is applicable in every state s in b. Applying a sequence of actions $[a_0, \dots, a_n]$ in a given belief state b_0 results in a chain of belief states $[b_0, \dots b_{n+1}]$, with $b_{i+1} = \{s \mid F(s_i, a_i) = s, \forall s_i \in b_i\}$, and given that a_i is applicable in b_i, for $0 \leq i \leq n$.

Syntax

Similarly to classical planning, a factored model given by a tuple of the form $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, corresponding to the state model of conformant planning, and whose semantics will be given in terms of definition 2.1.

Definition 2.2 (Non-deterministic Conformant Planning)**.** *A conformant planning problem can be defined as a tuple $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where:*

$\mathcal{F}$**:** *a set of Boolean fluents of the problem.*

$\mathcal{A}$**:** *a set of actions, where every action a is described by a tuple of preconditions and conditional effects: $\langle$ pre(a), eff(a) $\rangle$.*
The effects eff(a) are rules in the form:

$$C \rightarrow L_1 \,|\, \ldots \,|\, L_n$$

where C and L_i are conjunctions of literals in $\mathcal{F}$, for which C can be empty, and every L_i is a non-deterministic effect of the action a, for $1 \leq i \leq n$.

$\mathcal{I}$**:** *a set of clauses over $\mathcal{F}$ defining the initial situation*[1]*.*

$\mathcal{G}$**:** *a set of fluents $\mathcal{G} \subseteq \mathcal{F}$, describing the set of goal states.*

Semantics

Similarly to classical planning, we can here establish a relation between the state space model described in definition 2.1, and the factored definition 2.2 above.

Given a conformant problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, the conformant model $\langle S, S_0, S_G, A, F \rangle$ is obtained from P in the following way:

- The states s of the state space S are sets of literals that represent truth assignment over the fluents $\mathcal{F}$ in P, i.e. for every fluent L in $\mathcal{F}$, either L or $\neg L$ must belong to s.
  Notice that in the syntax of planning under uncertainty we don't assume a closed world, but we specify whether a literal is true, false, or unknown.
- The set of possible initial states S_0 are the states of S that satisfy the clauses in $\mathcal{I}$, such that $\mathcal{I} \models s_0$, for all states $s_0 \in S_0$.
- The goal states $s \in S_G$ are those such that $\mathcal{G} \models s$.
- The actions a applicable in s, are the ones in $\mathcal{A}$ such that $pre(a) \subseteq s$.
- The non–deterministic state transition function F maps the states in the belief b to a successor belief b' by applying all the possible effects of an executed action a:

$$b' = \{s' \mid F(s,a) = s', \forall s \in b, \text{ and } a \text{ applicable in } s\} \qquad (2.3)$$

[1]We will consider that $\mathcal{I}$ is encoded as a Conjunctive Normal Form (CNF).

We assume throughout that $\mathcal{I}$ is logically *consistent*, so that the set of possible initial states is not empty, and that the problem P itself is consistent, so that the bodies C_1 and C_2 of conflicting effects $a : C_1 \rightarrow L$ and $a : C_2 \rightarrow \neg L$ associated with the same action a are mutually exclusive or mutex.

For a state s, we write $\tau(s)$ to refer to the set of atoms (positive literals) that are true in s (i.e. $L \in \tau(s)$ iff L is true in s), and we write $P|_s$ to refer to the *classical planning problem* $P|_s = \langle \mathcal{F}, \mathcal{A}, \tau(s), \mathcal{G} \rangle$ which is like the conformant problem P except for the initial state that is fixed to s.

In this context, an action sequence $\pi = [a_0, a_1, \ldots, a_n]$ is a *classical plan* for $P|_s$ if the action sequence π is executable in the state s and results in a goal state. An action sequence that is a solution for at least one state in $\mathcal{I}$ is called a *weak* plan. Likewise, an action sequence π is a *conformant plan* for P iff π is a classical plan solving $P|_s$, for every possible initial state s of P.

In case of non-deterministic actions, we must consider all the state trajectories $[s_0, \ldots, s_{n+1}]$ that are possible given an action sequence $[a_0, a_1, \ldots, a_n]$: these are the ones starting in a possible initial state s_0, and with s_{i+1} a possible successor state of s_i given the action a_i. An action sequence $[a_0, a_1, \ldots, a_n]$ is a conformant plan for P if all the possible state sequences end in a goal state. When using action costs, we say that a plan π is optimal when no other plan has a smaller cost.

Deterministic conformant planning problems are the main topic of the next two chapters. As anticipated before, in chapter 5 we will describe translations of non-deterministic conformant planning problems that result in classical planning problems, solved by off–the–shelf classical planners.

Example 2.1. *We consider again the problem of example 1.2, where incomplete information about the initial situation is now part of the conformant planning problem.*

A robot in a $1 \times N$ grid, with $N = 5$, that has to go from an initial cell to the goal *cell p_4. However, the initial position of the robot is not known, and can be one of the last two cells at the opposite end of the corridor from the goal, as shown in Figure 2.2. This problem can be modelled as a deterministic conformant planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, with:*

Fluents $\mathcal{F}$: $\mathtt{at}(p_x)$, *for any x in* $[1, N]$*,*

Actions $\mathcal{A}$:

- `left`*:*
  Precondition: $\emptyset$
  Effect:
  $\{ \ \mathtt{at}(p_x) \ \} \longrightarrow \{ \ \mathtt{at}(p_{x+1}), \neg\mathtt{at}(p_x) \ \},$ *for any* $x \in [1, N-1]$.
  $\{ \ \mathtt{at}(p_5) \ \} \longrightarrow \{ \ \mathtt{at}(p_5) \ \}$
- `right`*:*
  Precondition: $\emptyset$
  Effect:
  $\{ \ \mathtt{at}(p_x) \ \} \longrightarrow \{ \ \mathtt{at}(p_{x-1}), \neg\mathtt{at}(p_x) \ \},$ *for any* $x \in [2, N]$.
  $\{ \ \mathtt{at}(p_1) \ \} \longrightarrow \{ \ \mathtt{at}(p_1) \ \}$

Initial situation $\mathcal{I}$: $oneof\big(at(p_1), at(p_2)\big)$

Goal $\mathcal{G}$: $\{ \ \mathtt{at}(p_4) \ \}$

Here the solution shown in example 1.2 cannot be applied, as we would end in an ambiguous situation: after applying the plan $\pi = \big[left, left, left\big]$*, the final belief state would be* $b = \{at(p_4), at(p_5)\}$ *(shown in light grey in the figure). Notice that the belief* b *includes the goal, but his* is not *the desired goal* $\mathcal{G}$*, as it includes the goal* $\mathtt{at}(p_4)$*, but also* $\mathtt{at}(p_5)$*. The plan* π *achieves the goal only in the case* $\mathcal{I}= \{\mathtt{at}(p_1)\}$*; it is indeed a weak plan for* P*.*

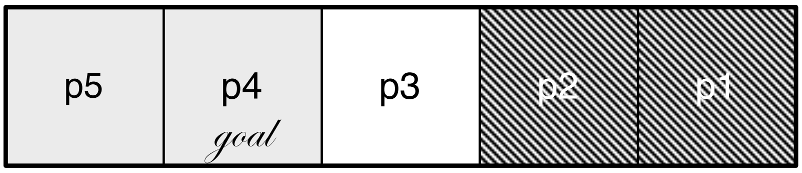


Figure 2.2: 1×5 corridor of the example 3.1. The initial possible positions of the robot are p_1 or p_2, while the goal is to reach the cell $p4$ labelled with *goal*.

One solution plan would achieve $\mathcal{G}$ *for all the (two) states in* $\mathcal{I}$*. Thus, a possible solution could be the following:*

$$\pi = \big[\ right, left, left, left\ \big]$$

After the first action, the belief size is reduced to only one state: $b = \{\mathtt{at}(p_1)\}$*, meaning that all the uncertainty of the problem has been removed. Moving right maps* $\mathtt{at}(p_1)$ *onto itself, and the other possible initial position* $\mathtt{at}(p_2)$ *in* $\mathtt{at}(p_1)$*. After applying* π*, the final belief state is just the desired goal:* $\mathtt{at}(p_4)$*.*

The conformant solution differs from the classical one also in the fact that it first drives the robot away from the goal, to an extreme of the corridor, in order to remove the uncertainty about the robot's position, and only then goes in the direction of the goal. In classical planning this in unnecessary, as the goal can be reached directly.

2.3 Contingent Planning

An important extension to conformant planning, and to planning in general, is the addition of sensing actions. When planning for real–world domains, the agent doesn't have complete knowledge of the state of the environment at each point in time, but it is allowed to sense certain features relevant to it. These domains are partially observable, as not all the features of the world are available to observe. Planning problems involving uncertainty and partial observability are particularly demanding, and are viewed as the most complex form of planning with uncertainty (Kaelbling et al., 1999; Rintanen, 2004).

In contingent planning, solution plans differ from classical and conformant plans for their shape. In classical and conformant planning, plans are given by a sequence of actions, while in contingent planning plans are *tree–shaped*, branching on observations. This difference affects the nature of the translations that can be applied to contingent planning problems, as we will see in chapter 6.

Model for Contingent Planning

We describe now the model for deterministic contingent planning problems.

Definition 2.3 (Contingent Planning). *The deterministic contingent planning model Q is a tuple $\langle S, S_0, S_G, A, O, f, o\rangle$, where:*

- *S is a finite set of states,*
- *$S_0 \subseteq S$ is the initial non empty belief state,*
- *$S_G \subseteq S$ is the set of goal states,*
- *A is the set of operators (the actions),*
- *O is the set of sensing actions (the observations),*
- *$f : S \times A \to S : \ (s, a) \mapsto f(s, a) = s'$ is a state* transition *function that maps states to states.*
  The successor state s' results from applying the executable operator a onto a given state s, and is the single successor state given by $s' = f(s, a)$,
- *an observation function $o : S \to O$ that associates to each state a possible observation.*

We will indicate with $b' = f(b, a)$ the belief state b' resulting from applying the operator a to the belief state b. Applying an observation operator o on a belief b results in the set of states compatible with observation coming from o and is denoted b^o.

The plan can be interpreted as an automaton, whose execution controls the system (that we call the planning domain) by synchronously reading the output of the system (the observations resulting from sensing) and providing step by step the input for the system (the planning actions).

The idea of performing sensing is strongly bound to solving contingent planning problems, as it provides information that consequently reduces the size of the belief state: after applying a sensing action, only the states compatible with what observed are possible. In the case of deterministic effects, performing sensing actions will monotonically increase the knowledge about the environment, as there is no uncertainty in the problem except the one encoded in the initial situation.

To describe a contingent planning problem, like for conformant planning (cf. definition 2.2), we consider a planning language that extends STRIPS with deterministic conditional effects, negation, an uncertain initial situation, and sensing actions.

Definition 2.4 (Contingent Planning). *A contingent planning problem can be defined as a tuple $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G}\rangle$, where:*

$\mathcal{F}$**:** *a set of Boolean fluents of the problem,*

$\mathcal{A}$**:** *a set of actions, where every action a is described by a tuple of preconditions and conditional effects: $\langle\ pre(a), eff(a)\ \rangle$.*
The effects $eff(a)$ are rules in the form: $C \to L$, where C and L are a conjunction of literals in $\mathcal{F}$ for which C can be empty.

$\mathcal{O}$**:** *a sensor model, or sensing actions, given by a set of pairs $< C, L >$, where C is a set of literals, and L is a positive literal in $\mathcal{F}$. The pair indicates that the (hidden) truth value of L is observable when C holds in a state. Each pair $< C, L >$ can be understood as a sensing action that activates when the precondition C is true, and that reveals the Boolean value of L.*

$\mathcal{I}$**:** *a set of clauses over $\mathcal{F}$ defining the initial situation,*

$\mathcal{G}$**:** *a set of fluents $\mathcal{G} \subseteq \mathcal{F}$, describing the set of goal states.*

We express the general solution of a contingent planning problem as a tree rooted in the initial belief, rather than an action sequence. The search space can be cast as an And-Or tree, where observations are And nodes and regular actions are the Or nodes, then a *contingent plan* is a sub-tree where all leaves are goal (belief) states. That is, the plan has to treat every possible outcome of sensing actions. Many planners consider graphs instead of trees, but the theoretical difference is nonexistent.

Example 2.2. *We now consider a new version of the example 2.1 of the robot in a corridor, to which we add sensing actions.*

As before, the initial position of the robot is not known, being one of the two possible initial poses. The difference is that now the robot has the ability to sense the presence of walls nearby. For example, if the robot is in position $\texttt{at}(p_1)$, it can sense a wall on the right.

This problem can be modelled as a deterministic contingent planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, with the same set of fluents $\mathcal{F}$ as before, but with the addition of two new literals encoding the presence of a wall on the left and on the right. $\mathcal{I}$ encodes the same uncertainty as in the former example, but it encodes also the presence of walls. The actions are, as the goal, unchanged.

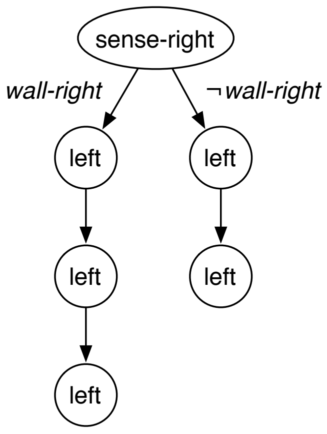


Figure 2.3: Tree showing how sensing disambiguates the initial position of the robot in example 2.2.

Fluents $\mathcal{F}$**:** $\{\ \texttt{at}(p_x)\ \} \cup \{$ *wall–left, wall–right*$\}$, *for any x in $[1, N]$,*

Actions $\mathcal{A}$**:**

- *left:*
  Precondition: $\emptyset$
  Effect:
  $\{\ \texttt{at}(p_x)\ \} \longrightarrow \{\ \texttt{at}(p_{x+1}),\ \neg\texttt{at}(p_x)\ \},$ *for any $x \in [1, N-1]$.*
  $\{\ \texttt{at}(p_5)\ \} \longrightarrow \{\ \texttt{at}(p_5)\ \}$
- *right:*
  Precondition: $\emptyset$
  Effect:
  $\{\ \texttt{at}(p_x)\ \} \longrightarrow \{\ \texttt{at}(p_{x-1}),\ \neg\texttt{at}(p_x)\ \},$ *for any $x \in [2, N]$.*
  $\{\ \texttt{at}(p_1)\ \} \longrightarrow \{\ \texttt{at}(p_1)\ \}$

Sensing Actions $\mathcal{O}$**:** *for any $x, y \in [1, N]$*

- *sense-left():*
  $\langle$ { $at(p_{1,y})$ }, *wall–left* $\rangle$
- *sense-right():*
  $\langle$ { $at(p_{N,y})$ }, *wall–right* $\rangle$

Initial situation $\mathcal{I}$: $\Big\{\big(at(p_1), at(p_2)\big), \big(\textit{wall–left}, \neg at(p_1)\big), \big(\textit{wall–right}, \neg at(p_N)\big)\Big\}$

Goal $\mathcal{G}$: { $at(p_4)$ }

Similarly to the former conformant example, the classical plan does not accommodate all the possible contingencies of the problem. However, it is now possible to produce plans taking advantage of sensing.

A solution contingent plan π would achieve $\mathcal{G}$ for all the (two) states in $\mathcal{I}$:

$$\pi = \big[\ \textit{sense–right},\ \textbf{\textit{if}}\ \textit{wall–right},\ \textbf{\textit{then}}\ \textit{left}, \textit{left}, \textit{left}.\ \textbf{\textit{else}}\ \textit{left}, \textit{left}.\ \big]$$

By only the use of sensing, it is possible to disambiguate the initial situation, and to apply the plan to drive each possible initial state in the goal.

When compared to the conformant solution in example 2.1, the contingent solution do not remove uncertainty by moving and localising the robot in a corner of the corridor, but uses its sensors to do so. The plan shown before can also be seen as a decision tree, where every sensing action helps retrieving the real position of the robot, as shown in Figure 2.3.

2.4 Complexity

PlanEx is the problem of determining if a problem has a solution plan. Outside the bounds of classical planning, different varieties of planning problems can be described, each one with its own complexity results.

For planning problems with incomplete information about the initial state, non-deterministic actions, and no sensing, PlanEx is EXPSPACE-complete (Haslum and Jonsson, 1999). For conformant planning with deterministic action effects, deciding if a plan exists is PSPACE-complete (Littman et al., 1998; Haslum and Jonsson, 1999). Planning with uncertainty about the initial state, non-deterministic actions, and full observability, where it is assumed that all belief states are singleton[2], is EXP-complete (Littman et al., 1998).

Conformant planning appears to be more difficult that classical planning in the sense that plan verification is harder: determining the existence of a classical plan with length at most k is NP–complete, if k is assumed to be polynomial in the size of

[2] The reason of considering singletons instead of sets of states is that, when the current state is ambiguous, it is always possible to observe until one state in singled out.

the problem; the same problem, for conformant planning is Σ_2^P–complete (Turner, 2002).

Planning with partial observability (or sensing as we defined it above) is the most difficult planning task. To determine if a problem has a solution plan, for problem instances with deterministic operators, uncertainty, and partial observability, is EXPSPACE–complete (Rintanen, 2004). The same decision problem for contingent problems with deterministic actions is also EXPSPACE–complete. For planning with incomplete information about the initial state, non-deterministic actions, and partial observability, the plan existence problem is 2–EXP complete (Rintanen, 2004). The intuition behind this result is easy to understand if we consider that the partially observable planning problem can be seen as a non-deterministic fully observable planning problem where the belief states are viewed as states. The belief state space has a doubly exponential size in the size of the problem instance, and hence the algorithm runs in doubly exponential time.

The following table summarises these complexity results:

		Action effects	
		deterministic	**non-deterministic**
Observability	**full**	PSPACE	EXP
	null	EXPSPACE	EXPSPACE
	partial	EXPSPACE	2–EXP

Table 2.1: Complexity of the plan existence problem for different flavours of planning with uncertainty (Rintanen, 2004).

It is interesting to note that restricting the initial belief state to only one initial state affects the complexity of only the deterministic unobservable and partially observable planning problems, bringing them down from EXPSPACE to PSPACE.

2.5 Belief representation and heuristics

In the formalisation of planning under uncertainty as a path-finding problem in belief space, we face two main computational challenges: the belief states representation and update, and the heuristics to guide the search (Bonet and Geffner, 2000).

Belief representation

As explained in section 2.1, in planning under uncertainty, the number of possible belief states of the problem is exponential in the number of states. Different approaches have been taken to deal with this potential source of inefficiency, and to represent belief states in a compact way that allows at the same time efficient belief updating after executing an action. Efficient implementations will need a compact representation of the belief state, in order to speed up the node generation rate, and reduce the memory footprint, to scale up to larger problems.

The first approach we're considering relies on representing beliefs with ordered binary decision diagrams (OBDD) (Bryant, 1992). The planner Pond has used this approach successfully (Bryce et al., 2006), and likewise has done mbp, a planner that makes use of model checking techniques to solve conformant and contingent planning tasks (Bertoli et al., 2006, 2001). The main disadvantage of the symbolic approach with OBDDs, however, is that the size of the OBDD can be very large. The structure of the diagram depends on the ordering of variables, and the manipulation of the OBDD might require intermediate OBDDs of exponential size.

The planner Conformant FF (Brafman and Hoffmann, 2004) does not employ an explicit representation of belief states, but instead represents them implicitly with the plan prefix leading to the belief state in the search from the initial belief state. In this approach, the size of the search nodes is compact because only the information given by the initial situation and the plan prefix is stored, and is thus less demanding w.r.t. OBDDs. Checking entailment in such a belief state needs to encode the initial state and the plan prefix as a propositional theory and then calling a SAT solver. The concern in this approach is the need to call a SAT solver to check –when necessary– the truth value of a fluent, trading–off space for time. The planner $T1$ (Albore et al., 2011) also uses a similar technique to keep track of belief states, as we will see in chapter 4.

In the context of contingent planning, the SDR Planner of Shani and Brafman (2011) takes the history based method of conformant–FF to an extreme, maintaining just the history of execution: the initial belief state, the actions executed, and the observations made. The belief tracking is then done in a lazy way, regressing through the executed plan the negation of the fluent to be verified.

The planner CpA uses an approximation of the belief states, considering only the literals true in all the states of the belief. This translation is called *0-approximation* (Baral and Son, 1997) and corresponds to the K_0 translation of Palacios and Geffner (2009). Information about the belief states is encoded as a formula in disjunctive normal form (DNF). As in classical planning, a successor state can be computed here in polynomial time, which makes this approach computationally attractive. But the *0-approximation* is incomplete, meaning that not all the solutions of the problem can be captured by this approximation, and a planning problem can be evaluated to be unsolvable even when it is not. Even if incomplete, for certain planning problems the size of the disjunction can be exponential, which prevents CpA from getting off the ground. The DNF planner (To et al., 2009) has been developed from a rib of CpA, from which it inherits preprocessing techniques to simplify the problem before starting the search. DNF employs an explicit representation of belief states in disjunctive normal form, using a consequent formalisation for the progression function. CpA and DNF share the overhead for computing the initial approximation state, which is sometimes quite significant. A similar approach, but with a different representation, makes use of Conjunctive Normal Form (CNF) to represent the planning problem under uncertainty. The planner CNF (To et al., 2010) represents belief states as *CNF-states*, which are a special case of CNF formulæ where no clause in a formula φ subsumes another other clause in φ. CNF-states indeed are a compact encoding of belief states. But CNF does not perform well in some domains, mainly because of the explosion in size of the CNF-state or because of the overhead in converting a belief state encoded as a CNF formula to the equivalent CNF-state.

Another recent approach to conformant planning is based on translation-based methods: the conformant planner T_0 successfully translates a conformant planning problem into a classical planning problem, and uses solutions to the latter to solve the original conformant problem. Under certain conditions, the solutions of the classical problem correspond to all the solutions of the conformant planning problem (Palacios and Geffner, 2009). The size of the initial state in the new translated problem, however, can be exponential in the size of the original problem if completeness is required, depending on the characteristics of the problem. This $K_{T,M}(P)$ translation employed by T_0 is the subject of the next chapter.

Heuristics

The ways to represent the belief states may vary in the various approaches to planning with incomplete information. The principal trade off here being between compactness and efficiency, while the heuristics remain a weak point for planning under uncertainty.

The first planner performing heuristic search in belief space is GPT (Bonet and Geffner, 2001b), which used an explicit enumeration of possible states and a heuristic function derived from a relaxation of the problem. This relaxation retains the uncertainty in the model but assumes full observability, resulting in a heuristic function that doesn't appear to be well informed for those problems where reasoning by cases is not appropriate.

MBP, and its enhanced conformant version KACMBP (Bertoli and Cimatti, 2002), two planners based on an OBDD representation of the beliefs, use the *cardinality heuristic* that measures the size of the belief state b, i.e. the number of states included in b. This heuristic estimates the "amount of uncertainty" encoded in a belief state, starting from the intuition that a belief state reduced to a singleton carries no uncertainty. Model counting for OBDDs can be performed in polytime, so evaluating the cardinality heuristic is simple and inexpensive for these planners. The actions selected aim consequently at reducing the size of the current belief state, which helps in problems where localising first is a plus, as in the example 2.1. KACMBP adjusts its heuristic estimate by alternating between the "acquire knowledge" mode, when the cardinality heuristic is considered, and the "reach–goal" mode that uses the GPT heuristic.

Translation-based approaches use heuristics coming from the planner used to solve the translation. Planner T_0 (Palacios and Geffner, 2006) uses a classical off–the–shelf planner to solve the classical problem the original conformant problem has been compiled into. There, all the information about the size of the belief seems to have been lost in translation: the classical heuristic will never consider reducing the size of the belief, which is in turn the main criterion used by KACMBP.

The cardinality heuristic is a main aspect of planning under uncertainty, but the measure of the distance from the goal can still be the main guide in the search. This is one of the main motivations that pushed us to create and implement the conformant T1 planner (Albore et al., 2011): a conformant planner that is very efficient and uses a heuristics portfolio which considers two heuristics, one dependent on the "classical" distance from the goal of the current belief, and a second second heuristic related to both cardinality heuristics, and landmark heuristics (Richter et al., 2008). The T1 planner is described in chapter 4.

It has been suggested by Funge (1998, 1999) to formalise knowledge in the situation calculus, and to keep track of the amount of uncertainty of in uncertain states with the help of epistemic interval-valued fluents. For this purpose, a special fluent I_f

is used to express the (un)certainty of a corresponding fluent f by maintaining an interval of possible values for it; for instance $I_f =< a, b >$ means that f can assume values between a and b. The idea accommodates also sensing and exogenous events.

POND (Bryce et al., 2006) uses a structure called the Labelled Uncertainty Graph (LUG) that is similar to that used in the *GraphPlan* algorithm for classical planning (Blum and Furst, 1995), and encodes information about different initial states, i.e. the mutexes, the support from preconditions to actions and from actions to effects. For example, the LUG considers multiple support actions for a literal when they are necessary across different initial states, but does not overcount when the same action is used as a support for a literal given different initial states.

On the other side, DNF makes use of a relaxation on the information about the cardinality of the belief state, the `one-of` relaxation. An exclusive disjunction on state variables is treated as a simple disjunction: the information about the amount of uncertainty carried by a belief will then appear over–estimated. For example, the uncertainty on a multi-valued variable initially described as $oneof(x_1, \dots, x_n)$ will be simplified to $(x_1 \vee \dots \vee x_n)$: the mutual exclusivity of the values is ignored, and states where more than a x_i is true will be gratuitously considered. DNF and CNF planners use also the number of satisfied subgoals to guide its search.

The Conformant-FF planner (Brafman and Hoffmann, 2004) re-uses the ideas behind the *delete relaxation* heuristic (Hoffmann and Nebel, 2001), which solves a relaxed version of the problem, which assumes that all delete lists are empty. The length of the relaxed (classical) plan is taken to be the heuristic estimate. The relaxed planning method that FF uses to compute its heuristic function is modified accordingly, with a complete but unsound form of the CNF reasoning about known propositions. The key step is to project the relevant clauses of the CNF theory of the conformant problem to one having clauses of size 2.

2.6 Probabilistic approaches to planning under uncertainty

Uncertainty about actions' outcomes can be modelled as a probability distribution function: a probability is assigned to each possible effect of an action. Consequently, the goals are represented as utility functions, i.e. numeric functions that can express preferences about the executed actions and the desired final states. In such a framework, a plan is expressed as a *policy* that details the action to execute in each state, and the planning problem reduces to an optimisation problem where a solution is an optimal policy which maximises the utility function.

The execution of a policy corresponds to a Markov Chain, an infinite sequence of states where each state depends on the previous one. Thus, each state has a different probability to be visited during the execution, such that the expected reward for being in a particular state s, following some fixed policy π in the infinite horizon case, is given by the equation below:

$$V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s'|s, \pi(s)) V_\pi(s') \tag{2.4}$$

This equation (2.4) describes the expected reward $V^{\pi}(s)$ for stating in a state s and applying the actions in some policy π, considering the immediate reward $R(s,a)$ for applying the action a in s, and a discount factor $\gamma \leq 1$.

The planning problem is to find an optimal policy that maximises the utility function, the optimal expected cost in a state s is given by the Bellman equation:

$$V^{*}(s) = R(s,a) + \max_{a} \gamma \sum_{s'} P(s'|s,a)V^{*}(s') \tag{2.5}$$

The former family of equations due to Bellman can be used to solve Markov Decision Processes (MDPs). MDPs are particularly interesting for us because it has been shown that a STRIPS planning problem can be encoded as a MDP (Koenig, 1991). Different methods exist and have been applied to MDPs's solving, the most popular being *policy iteration* and *value iteration*.

Full observable MDPs have been proved to be P-Complete by Papadimitriou and Tsitsiklis (1987); this means that even if they are solvable in polynomial time, an efficient parallel solution is unlikely: if an efficient parallel algorithm were available, then all problems in P would be solvable efficiently in parallel (Littman et al., 1995).

Contingent planning as POMDP

When the assumption of full observability that sustains MDP's approach doesn't hold, the optimal policy does not depend only on the current state anymore, but also on the information available up to that point.

For our purpose it could be useful to delineate an alternative characterisation for contingent planning, which can be cast from a dynamic programming perspective (Bellman, 1957; Bonet and Geffner, 2000). In a contingent problem, an agent starts in a given belief state $b = b_0$ and must reach a target belief b_F that only contains goal states. For this, the *physical effects* of an action a, applicable in a belief state b, deterministically map b to a new belief state b_a:

$$b_a = \{s' \,|\, s' = f(a,s)\,,\, s \in b\} \tag{2.6}$$

where $f(a,s)$ is the state-transition function determined by the effects associated with the action a in P, while the *sensing effects* of an action a non-deterministically map b_a to a belief state b_a^o:

$$b_a^o = \{s \,|s' \in b \text{ and } s \text{ compatible with } o\} \tag{2.7}$$

where o is one of the possible observations that may arise in b_a. If we write $O(b,a)$ to indicate the possible observations that may arise by doing action a in b, then the cost of reaching a target belief state b_F from a non-target belief b, in the worst case, can be obtained from the optimality equation:

$$V(b) = \min_{a \in A(b)} \left[c(a) + \max_{o \in O(b,a)} V(b_a^o) \right] \tag{2.8}$$

and the equation $V(b_F) = 0$ for goal beliefs (those that only include goal states). In this equation (2.8), $A(b)$ denotes the set of actions applicable in b; namely, those whose preconditions are true in b, and $c(a)$ is the cost of the action a, assumed to be 1 by default.

The solution to the optimality equation captures the optimal cost function $V^*(b)$ that measures the cost of reaching a goal belief from b in the worst case, providing the minimum depth of the contingent tree that solves the problem. In our setting, where actions have either physical or sensing effects but not both, $O(b,a)$ must be set to contain just one dummy observation true in all states when a is a physical action, and b_a must be set to b when a is a sensing action. For capturing expected costs rather than worst case costs, the beliefs must be set to probability distributions, and the equations for updating beliefs and the optimality equations must be adjusted, with a weighted sum over the observations $o \in O(b,a)$ replacing the max, with the weights being probability of getting each observations given b and a. The resulting model is then a POMDP (Cassandra et al., 1994; Bonet and Geffner, 2000).

POMDPs include, beside the same probabilistic transition model and reward function of MDPs, an observation model $O(s,o)$ that expresses the probability of having the observation o in the state s. Belief states are a probability distribution $b(s)$ over all possible states s such that $\sum_s b(s) = 1$, being $b(s)$ the probability assigned to the particular state s in the belief b. A policy is a function that maps belief states to actions. In a way similar to MDPs, planning problems in POMDPs can be stated as optimisation problems where an optimal policy has to be found. This is done by seeing the POMDP planning problem as a fully observable MDP planning problem on the infinite set of belief states (Åström, 1965). The optimal policy can be understood as the solution of a continuous space belief Markov Decision Process. For such continuous MDPs, the belief transition function T is expressed in terms of the probability of reaching a belief state b' from a belief b, given an action a, such that $T(b,a,b') = P(b'|b,a)$, and the reward function on belief states is given by $\rho(b,a) = \sum_s b(s)R(s,a)$.

POMDPs are much more a difficult problems than MDPs, due to the resulting belief states space, which is infinite and continuous. Thus computation of optimal policies is intractable: the problem is PSPACE–complete (Papadimitriou and Tsitsiklis, 1987), and current computing power can only solve POMDPs with a few dozen states. Instead of computing an exact solution to a POMDP, as suggested by Sondik (1971), research has focused on producing approximated solution algorithms, like Witness (Littman, 1994), which is an improved value iteration for POMDPs.

2.7 Summary

Standard planning algorithms assume complete information about a deterministic environment. However, many planning domains violate these assumptions and do not guarantee full information about the agent's situation. In these cases, a solution plan compels with all the contingencies that can arise from incomplete information, driving all the possible initial states to goal states. Contingent planning can be seen as a general case of planning under uncertainty, where sensing actions allow to "observe" the status of the environment and collect information needed to solve the problem. Contingent planning is the most difficult task of planning, since finding a plan for non-deterministic and partially observable problems is in 2-EXP, while the same problem featuring null observability is in EXPSPACE. Conformant planning is the task of finding a plan in a domain with incomplete information about the initial situation, and no sensing actions. A plan for a conformant planning problem is then a sequence of actions, rather than a tree. The ability to find conformant plans is needed in the contingent setting, of which conformant situations are a special case.

The problem of planning with incomplete information has been approached as a path-finding problem in belief space where good belief representations and heuristics are critical for scaling up. In the next chapter, we will see a different formulation for conformant problems with deterministic actions where they are automatically converted into classical ones and solved by an off-the-shelf classical planner.

Chapter 3

Conformant translations to Classical Planning

The obstacles to providing effective solutions to conformant planning are the ones of planning under incomplete information: providing a compact and efficient representation of the problem, and having informed heuristics over belief states for solving it. Translation-based approaches address the difficulty of planning under uncertainty by *compiling* planning problems to another ones, easier to solve. The aim of translations is to avoid searching the whole belief space, and to adopt a more compact representation of the beliefs themselves.

In this chapter we describe an alternative approach to conformant planning where problems are automatically compiled into classical problems. These compiled problems are then solved by an off–the–shelf classical planner. The backbone of the translation approach resides in the fact that belief states are represented as states, allowing standard classical planning heuristics to be used. We will follow the approach described in (Palacios and Geffner, 2009), where conformant planning problems are mapped into classical ones that are then solved by using a state-of-the-art classical planner. In the worst case this translation is exponential, but for a large collection of problems it can be shown to be polynomial, and complete.

Motivation

Conformant problems mapped into classical ones provide an implicit solution to the two principal issues faced by conformant planners that perform search in belief space (Bonet and Geffner, 2000), namely the belief representation and the efficacy of heuristics over beliefs. In the conformant setting, the number of possible belief states is exponential in the number of states, thus the problem of searching for conformant plans is exponentially larger than in the classical planning setting. In conformant problems translated in classical problems, belief states are represented as plain states, allowing standard classical planning heuristics to be used.

3.1 The K_0 translation

The translation-based approach to conformant planning from which we start, maps deterministic conformant problems P into classical problems $K(P)$ that can be solved by off-the-shelf planners (Palacios and Geffner, 2009). The approach is also closely related to the notion *planning at the knowledge level*, which can include sensing, and that extends STRIPS representation to model actions that make changes at the "knowledge level", rather than on the world state (Petrick and Bacchus, 2002, 2004). The main difference being that in the work of Palacios and Geffner (2009), the epistemic encoding is derived automatically.

The simplest translation, called K_0, changes the representation of the problem P by replacing all the literals L in P by literals KL and $K\neg L$ in $K_0(P)$. The literals KL and $K\neg L$ aim at capturing whether L is "known to be true" and "known to be false" respectively. A literal L is "known to be true" in a belief state b when it holds in all the states s of b, and vice versa for a literal "known to be false".

Definition 3.1 (Translation K_0)**.** *For a* deterministic *conformant problem* $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$*, the translation* $K_0(P) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ *is a classical planning problem where*

- $\mathcal{F}' = \{KL, K\neg L \mid L \in \mathcal{F}\}$,
- $\mathcal{I}' = \{KL \mid L \text{ is a unit clause in } \mathcal{I}\}$,
- $\mathcal{G}' = \{KL \mid L \in \mathcal{G}\}$, *and*
- $\mathcal{A}' = \mathcal{A}$ *with each precondition* L *for* $a \in \mathcal{A}$ *replaced by* KL*, and each conditional effect* $C \rightarrow L$ *replaced by the two rules* $KC \rightarrow KL$ *and* $\neg K\neg C \rightarrow \neg K\neg L$.

The expressions KC *and* $\neg K\neg C$ *for conjunctions* $C = C_1 \wedge \ldots \wedge C_n$ *are abbreviations for* $KC_1 \wedge \ldots \wedge KC_n$ *and* $\neg K\neg C_1 \wedge \ldots \wedge \neg K\neg C_n$ *respectively.*

The intuition behind the translation is that the only information retained in the compiled problem $K_0(P)$ is the information about $\mathcal{I}$ that is certain. In other word, the literal KL is true in $\mathcal{I}'$ if L is known to be true in $\mathcal{I}$; otherwise it is false[1]. This removes all uncertainty from $K_0(P)$, making it a classical planning problem.

Example 3.1. *Let consider again the conformant planning problem of a robot moving in a* 1×5 *corridor that we already saw in example 2.1. The initial position of the robot is not known with certainty: it can be in one of the last two cells at the end of the corridor opposite from the goal.*

Here the initial situation in the conformant problem P *is described by the formula oneof*$\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big)$*, indicating that the initial position of the robot can be either* p_1 *or* p_2 *in Figure 2.2.* $\mathcal{I}$ *is then expressed as a set of clauses:*

$$\mathcal{I} = \Big\{ \big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big), \big(\neg\mathtt{at}(p_1), \neg\mathtt{at}(p_2)\big), \big(\neg\mathtt{at}(p_5)\big), \big(\neg\mathtt{at}(p_3)\big), \big(\neg\mathtt{at}(p_4)\big) \Big\}$$
$$\mathcal{G} = \big\{\mathtt{at}(p_4)\big\}$$

[1]As said in section 2.2, a literal L is true in $\mathcal{I}$ iff $\mathcal{I} \models L$.

The only fluents that are true in all the possible situations in $\mathcal{I}$ are $\neg\mathtt{at}(p3)$, $\neg\mathtt{at}(p_4)$, and $\neg\mathtt{at}(p_5)$, thus they are the only fluents that appear in the initial state $\mathcal{I}'$ of the translation $K_0(P)$:

$$\begin{aligned}\mathcal{I}' &= \{K\neg\mathtt{at}(p_5), K\neg p3, K\neg\mathtt{at}(p_4)\}\\ \mathcal{G}' &= \{K\mathtt{at}(p_5)\}\end{aligned}$$

To preserve soundness, the actions in definition 3.1 are such that each rule $a : C \to L$ in P is mapped into *two* rules:

- a **support rule** $a : KC \to KL$, that ensures that L is known to be true when the condition C is known to be true, and
- a **cancellation rule** $a : \neg K\neg C \to \neg K\neg L$ that guarantees that $K\neg L$ is deleted (i.e. prevented to persist) when action a is applied and C is not known to be false.

The translation $K_0(P)$ is *sound* as every classical plan that solves $K_0(P)$ is a conformant plan for P, but *incomplete*, as not all conformant plans for P are classical plans for $K_0(P)$. This means that if a plan achieves KL in $K_0(P)$, then the same plan achieves L with certainty in P , but the converse is not always true, meaning that a plan may achieve L with certainty in P without making the literal KL true in $K_0(P)$.

This translation share many similarities with the 0-approximation (Baral and Son, 1997), where belief states are represented by 3–valued fluents states where fluents can be true, false, or unknown, following Lukasiewicz's logic (Lukasiewicz, 1953). The two approximations share the same representation of the problem where incomplete information is not considered when searching for a solution plan. The result is that an action sequence π is a classical plan for $K_0(P)$ iff π is a conformant plan for P according to the 0-approximation semantics.

The translation K_0 is linear in the size of the original problem, but is generally incomplete. We are going to see in the next sections some more translations that may be exponential in the size of the problem, but that can provide completeness and applicability on a large number of problems.

3.2 General translation scheme $K_{T,M}$

The translation scheme K_0 is compact and efficient, but not sufficiently expressive when the planning problems to solve request to reason by cases. In this section we depict a more general translation scheme that is sound and also complete under certain conditions. The complexity of the complete translation is exponential in a parameter of the problem that is called the *conformant width*, that will be defined later on in this chapter (cf. definition 3.15). It turns out that the conformant width is bounded for most benchmarks, leading to a polynomial translation.

The more general translation scheme $K_{T,M}$ builds on K_0 by the mean of two parameters: a set T of *tags* t and a set M of *merges* m. The tags and the merges are used to account for conformant plans that reason by cases; indeed, the tags are used to introduce assumptions about the initial situation that are eliminated via the merges. The translation $K_{T,M}(P)$ introduces new literals KL/t to which we attribute the following semantics:

L is known to be true if t was true in the initial situation.

A tag $t \in T$ is a conjunction of literals L of P, whose truth value in the initial situation is not known. The tags t encode assumptions about the initial situation that, if they hold, they yield the literal L in the current situation, which corresponds exactly to the semantics of KL/t.

A merge m is a non-empty collection of tags t in T of which at least one is true, i.e. a merge m stands for the Disjunctive Normal Form (DNF) formula $\bigvee_{t \in m} t$. A merge m is said to be *valid* when at least one of the tags $t \in m$ is true in $\mathcal{I}$:

$$\mathcal{I} \models \bigvee_{t \in m} t \tag{3.1}$$

The translation $K_{T,M}$ is sound as long as the merges in M are valid. A merge m for a literal L in P translates into the "merge action" with single effect

$$\bigwedge_{t \in m} KL/t \quad \rightarrow \quad KL \tag{3.2}$$

The set of "merge actions" associated with the set of merges M is referred to as $\mathcal{A}_M$. The translation $K_{T,M}(P)$ is the basic translation $K_0(P)$ where the literals are "conditioned" with the tags t in T and the set of actions is extended with the $\mathcal{A}_M$ actions.

The literals written KL (without a tag) are assumed to stand for the literals KL/t where t is the *empty tag*. The empty tag expresses no assumption about the initial situation and is assumed implicitly in every set T.

Definition 3.2 (Translation $K_{T,M}(P)$)**.** *Let $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a conformant problem, then the translation $K_{T,M}(P) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is a* classical planning problem*, where*

- $\mathcal{F}' = \{KL/t, K\neg L/t \mid L \in \mathcal{F} \text{ and } t \in T\}$,
- $\mathcal{I}' = \{KL/t \mid \mathcal{I} \models (t \supset L)\}$,
- $\mathcal{G}' = \{KL \mid L \in \mathcal{G}\}$, *and*
- $\mathcal{A}' = \mathcal{A}_M \cup \mathcal{A}$ *with each precondition L for $a \in \mathcal{A}$ replaced by KL, and each conditional effect $C \rightarrow L$ replaced by the two rules:*
  $KC/t \rightarrow KL/t$ *and* $\neg K\neg C/t \rightarrow \neg K\neg L/t$.

The conditions KC/t and $\neg K\neg C/t$, for the conjunction $C = C_1 \wedge \ldots \wedge C_n$, correspond to $KC_1/t \wedge \ldots \wedge KC_n/t$, and $\neg K\neg C_1/t \wedge \ldots \wedge \neg K\neg C_n/t$, respectively.

$K_{T,M}$ can be seen as the simple translation K_0, but where the literals considered are the ones known to be true in all the initial states where a tag t holds, and not just the literals known to be true in all the states. The translation $K_{T,M}$ reduces to the basic translation K_0 when M is empty and T contains only the empty tag. For suitable choices of T and M, the translation $K_{T,M}$ is both *sound* and *complete*: sound, meaning that the classical plans for $K_{T,M}(P)$ are all conformant plans for P once merge actions are removed, and complete, meaning that all conformant plans for P yield classical plans for $K_{T,M}(P)$ once merge actions are added (Palacios and Geffner, 2009). The formal definitions are given below:

Definition 3.3 (Soundness of $K_{T,M}(P)$)**.** *A translation $K_{T,M}(P)$ is sound when for any classical plan π that solves the classical planning problem $K_{T,M}(P)$, the plan π' that results from π by dropping the merge actions is a conformant plan for P.*

Definition 3.4 (Completeness of $K_{T,M}(P)$)**.** *A translation $K_{T,M}(P)$ is complete when for any conformant plan π' that solves the conformant problem P, there is a classical plan π that solves the classical problem $K_{T,M}(P)$ such that π' is equal to π with the merge actions removed.*

The general translation scheme $K_{T,M}$ is sound, provided that all merges are valid, and that all the tags are consistent. A tag t is *consistent* if all the literals in t are true in some possible initial state. The problem $K_{T,M}(P)$ is consistent if the original problem P is consistent, i.e. for two mutex literals L and L' in the consistent problem P, the invariant $KL/t \supset K\neg L'/t$ is preserved in the translated problem $K_{T,M}(P)$ (Palacios and Geffner, 2009). We say that a clause is consistent if the set of states it implies is not empty; for example, in this thesis are only considered problems where $\mathcal{I}$ is logically consistent, meaning that at least one initial state exists.

Theorem 3.5 (Soundness of $K_{T,M}(P)$)**.** *The translation $K_{T,M}(P)$ is sound provided that all merges in M are valid and all tags in T are consistent.*

Having consistent tags ensure that the literals in a tag are true in some initial state, and valid merges that at least one of the tags is satisfied in the initial situation. This is enough to guarantee that a plan for $K_{T,M}(P)$ is also a plan for P. From now on, we will assume to have *valid translations*, i.e. translations where all the merges are valid and all the tags consistent.

3.3 The K_{S_0} Complete translation

One way to get a complete translation is by associating the tags in T with the set S_0 of possible initial states of P (in addition to the empty tag). This translation is called K_{S_0}, which is the instance of the general translation $K_{T,M}$ that results from setting T to S_0, and having one merge in M equal to S_0 *for each precondition and goal literal.* The translation K_{S_0} is complete but exponential in the worst case.

Definition 3.6 (Translation K_{S_0})**.** *For a conformant problem P, the translation $K_{S0}(P)$ is an instance of the translation $K_{T,M}(P)$ where*

- *T is set to the union of the empty tag and the set S_0 of all possible initial states of P (understood as the maximal sets of literals that are consistent with $\mathcal{I}$), and*

- *M is set to contain a single merge $m = S_0$ for each precondition and goal literal L in P.*

The translation K_{S_0} is valid and hence sound, and it is complete because of the correspondence between the tags with the possible initial states. A conformant plan π for the problem P maps all the possible initial states s in a goal state, so π is a solution for every $P|_s$. As the set of tags in K_{S_0} corresponds to the set of initial states, the same solution plan π will achieve the goal for all the tags of the problem.

Theorem 3.7 (Completeness of K_{S0})**.** *If π is a conformant plan for P, then there is a classical plan π' for $K_{S0}(P)$ such that π results from dropping the merge actions from π'.*

We saw previously that the number of possible states is exponential in the number of the uncertain fluents of the problem, from which comes that the translation K_{S_0} is exponential too in the worst case. Needless to say, planners based on the K_{S_0} translation do no scale up well, compared to other state–of–the–art conformant planners (Palacios, 2009, chap. 6), even if they appear to be efficacious on small instances of planning problems.

3.4 The compact K_i translation

The translation K_{S_0} is sound and complete, but introduces in the translated problem literals KL/t that might be exponential in the number of uncertain fluents of the problem. However it is possible to obtain from $K_{T,M}(P)$ a more compact translation that is sound and complete by just considering smaller sets of tags T and merges M. To guarantee that this reduced set of tags captures the necessary uncertainty of the problem P to solve it, we must consider the set of the clauses in $\mathcal{I}$ that are *relevant* to the literals that are helpful in solving the problem. This is not a trivial task indeed, but that can be shone upon with the help of some basic notion.

Definition 3.8 (Relevance)**.** *The conformant relevance relation $L \longrightarrow L'$ in P, read L is relevant to L', is defined inductively as*

1. *$L \longrightarrow L$*
2. *$L \longrightarrow L'$ if $a : C \rightarrow L'$ is in P with $L \in C$ for some action a in P*
3. *$L \longrightarrow L'$ if $L \longrightarrow L''$ and $L'' \longrightarrow L'$*
4. *$L \longrightarrow L'$ if $L \longrightarrow \neg L''$ and $L'' \longrightarrow \neg L'$.*

The notion of relevance traces how the uncertainty about certain literals affects others. It is important to notice that relevance only considers conditional effects and not preconditions. The reason is simple: in order to apply an action, its precondition has to be known, consequently no uncertainty can come from a precondition; however, truth value of conditions can be ambiguous.

We assume from now on that the initial situation $\mathcal{I}$ is written in Prime Implicate Form (Marquis, 2000). A formula is in Prime Implicate Form if includes only the inclusion–minimal clauses it entails, but not tautologies. We consider prime implicate form because to check whether a clause c follows logically from a formula $\mathcal{I}$ expressed in prime implicate form is a polynomial operation as it is reduced to check if c is subsumed by a clause in $\mathcal{I}$, or if it is a tautology.

The set C_I stands for the set of clauses that represent uncertainty about the initial situation, namely the non-unit clauses in $\mathcal{I}$, along with the tautologies $(L \vee \neg L)$ for complementary literals L and $\neg L$ not appearing as unit clauses in $\mathcal{I}$ (i.e. literals that are initially unknown). The notion of (conformant) relevance is extended to clauses as follows:

Definition 3.9 (Relevant Clause). *A clause $c \in C_I$ is relevant to a literal L in P when every literal c in c is relevant to L. The set of clauses in C_I relevant to a literal L is denoted as $C_I(L)$.*

The literals that have necessarily to be known in order to solve a problem are the ones present into the goal, and in the preconditions of the actions. Thus, the merges of a $K_{T,M}(P)$ translation should satisfy the clauses relevant to such literals. If we consider the CNF formula $C_I(L)$ that captures the fragment of the initial situation $\mathcal{I}$ that is relevant to a literal L, the type of merges required for completeness are then simply the valid merges m that satisfy the set of clauses $C_I(L)$. We say that a merge m satisfies a set of clauses if m satisfies each clause in the set: for every clause $c = (c_1, \ldots, c_n)$ in the set, m contains a tag t and some literal c_i in c is a consequence of t, i.e. $\mathcal{I} \models (t \supset c_i)$. This set of valid merges satisfying $C_I(L)$ is said to be *covering*:

Definition 3.10 (Covering Merges). *A valid merge m in a translation $K_{T,M}(P)$ of a conformant problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$* covers *a literal L when m satisfies $C_I(L)$.*

This yields to translations built with covering merges. A valid translation $K_{T,M}(P)$ that contains a merge m that covers each precondition and goal literal L in P is said to be a *covering translation*:

Definition 3.11 (Covering Translation). *A covering translation is a valid translation $K_{T,M}(P)$ that includes one merge that covers L, for each precondition and goal literal L in P.*

Testing whether a valid translation $K_{T,M}(P)$ is a covering translation can be done in polynomial time. Another central result is that covering translations are complete (Palacios and Geffner, 2009):

Theorem 3.12 (Completeness). *Covering translations $K_{T,M}(P)$ are complete; i.e., for any π conformant plan for P there is a classical plan π' for $K_{T,M}(P)$ such that π is π' with the merge actions removed.*

In a complete translation $K_{T,M}(P)$, the tags and merges in T and M capture the information in the initial situation relevant to each precondition and goal literal, and thus embed all the necessary information to solve the planning problem P.

A translation that is sound and complete, but generally more compact w.r.t. K_{S_0}, can be made by considering only the relevant clauses to precondition and goal literals:

Definition 3.13 (Translation K_{models})**.** *For a conformant problem P, the translation $K_{models}(P)$ is an instance of the translation $K_{T,M}(P)$ where*

- *M is set to contain one merge m for each precondition and goal literal L, given by the models of $C_I(L)$ that are consistent with $\mathcal{I}$, and*
- *T is set to contain the tags in all such merges, plus the empty tag.*

The translation K_{models} is equivalent to K_{S_0} when the set of relevant clauses for all the precondition and goal literals L equals $\mathcal{I}$, i.e. $C_I(L) = \mathcal{I}$. In other words, the translation K_{models} is equivalent to K_{S_0} when all the clauses in $\mathcal{I}$ are relevant to L; consequently K_{models} is exponential in the number of fluents appearing in the sets $C_I(L)$. Thus K_{models} is generally more compact than K_{S_0} while staying sound and complete.

The conformant width

In order to obtain a more compact translation $K_{T,M}(P)$, it is necessary to obtain the minimal set of clauses to cover the literals needed to solve the problem. For that, Palacios and Geffner defined a structural parameter, called the *conformant width* to capture the minimal size of tags necessary to have a complete translation for a conformant problem P (Palacios and Geffner, 2006). The width of a conformant problem is related to the maximum number of variables whose values are initially unknown, such that all of them are relevant to a certain precondition or goal.

A *cover* is a combination of literals close to the notion of merge:

Definition 3.14 (Cover)**.** *A cover $c(\mathcal{C})$ for a set of clauses $\mathcal{C}$, relative to a conformant problem P with initial situation $\mathcal{I}$, is the minimal set of literals $\mathcal{S}$ consistent with $\mathcal{I}$ such that $\mathcal{S}$ contains a literal of each clause in $\mathcal{C}$.*

A cover is then a DNF formula that is logically equivalent to the CNF formula $\mathcal{C}$ (in fact $c(\mathcal{C})$ satisfies $\mathcal{C}$), consistent with $\mathcal{I}$. In particular, the cover $c(C_I(L))$ of $C_I(L)$ is a valid merge that covers L. To identify the smallest set of clauses having a merge that satisfies $C_I(L)$ (like a covering merge), the set $C_I(L)$ has first to be extended with tautologies of the form $(p \vee \neg p)$ for fluents p such that either p or $\neg p$ appears in $C_I(L)$. This extended set of clauses is denoted by $C_I^*(L)$.

Definition 3.15 (Conformant Width of a Literal)**.** *The conformant width of a literal L in P, written $w(L)$, is the size of the smallest (cardinality-wise) set of clauses $\mathcal{C}$ in $C_I^*(L)$ such that $c(\mathcal{C})$ satisfies $C_I(L)$.*

The width of a problem is the maximum width of the precondition or goal literals:

Definition 3.16 (Conformant Width of a Problem)**.** *The conformant width of a problem P, written $w(P)$, is given by*

$$w(P) = \max_L w(L)$$

where L ranges over the precondition and goal literals in P.

The notion of width is important for our translation–based approaches to planning with incomplete information as it provides an important parameter on the difficulty of solving a problem. The characteristics of the translations also strongly depend on it: the width gives an idea of the structural complexity of the problem, and is related to the size of the tags needed for completeness. Having translations with bounded tags size will also bound the size of the translation itself, that will be exponential only in the width. This means that sound and complete translations can be now obtained being exponential in the size of the tags and not in the size of the problem.

The polynomial translation K_i

The translations K_{S_0} and K_{models} above are both sound and complete, but with tags of arbitrary size. An instance of the general translation scheme $K_{T,M}(P)$ can be obtained from covering translations from definition 3.11, where the covering merges are all the ones that cover all the preconditions and the goal literals of the problem that include *tags of bounded size.* More in particular, the tags of max size i provide the family of translations K_i, defined as follow:

Definition 3.17 (Translation K_i)**.** *For a conformant planning problem P, the translation $K_i(P)$ is obtained from the general scheme $K_{T,M}(P)$ where*

- *M is set to contain one merge $m = c(\mathcal{C})$ for each precondition and goal literal L in P, if there is a set $\mathcal{C}$ of at most i clauses in $C^*_I(L)$ such that m covers L. If no such set exists, one merge $m = c(\mathcal{C})$ for L is created for each set $\mathcal{C}$ of i clauses in $C^*_I(L)$, and no merges are created for L if $C^*_I(L)$ is empty;*
- *T is the collection of tags appearing in the merges above, plus the empty tag.*

When the parameter $i = 0$, then the translation $K_i(P)$ reduces to $K_0(P)$ (cf. section 3.1). In fact, K_0 has no tags but the empty tag, and no merges.

Example 3.2. *The problem P of the example 3.1 above can be translated following the terms of the $K_i(P)$ translation.*

If we take the actions of moving left and right as they are described in example 2.1, without preconditions but with conditional effects only, we obtain the relevance results below:

$$\mathtt{at}(p_i) \rightarrow \mathtt{at}(p_j), \quad \textit{for } i, j \textit{ in } [1, N]$$

Given the goal $\{\mathtt{at}(p_4)\}$, the set $C_I(\mathtt{at}(p_4))$ is given by the clauses:

$$\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big); \big(\neg\mathtt{at}(p_1), \neg\mathtt{at}(p_2)\big); \big(\mathtt{at}(p_1), \neg\mathtt{at}(p_1)\big); \big(\mathtt{at}(p_2), \neg\mathtt{at}(p_2)\big)$$

The minimal set of clauses satisfying $C_I(L)$, for L precondition of goal, is: $\big\{\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big)\big\}$, from which follows a width of the problem of 1.

The merge $\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big)$ is covering, so the problem $K_{T,M}(P)$ will have 2 tags, i.e. $\mathtt{at}(p_1)$ and $\mathtt{at}(p_2)$, plus the empty tag.

This problem can now be modelled as a classical planning problem $K_{T,M}(P) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$, with:

Fluents $\mathcal{F}$: $Kat(p_x)/at(p_1)$, $Kat(p_x)/at(p_2)$, $K\neg at(p_x)/at(p_1)$, $K\neg at(p_x)/at(p_2)$, $Kat(p_x)$, $K\neg at(p_x)$, *for any x in* $[1, N]$.

Actions $\mathcal{A}$:

left:

Precondition: $\emptyset$

Effect:
$\{Kat(p_x)\} \longrightarrow \{Kat(p_{x+1}), K\neg at(p_x)\}$
$\{Kat(p_5)\ \} \longrightarrow \{\ Kat(p_5)\}$
$\{\neg K\neg at(p_x)\} \longrightarrow \{\neg K\neg at(p_{x+1}), \neg Kat(p_x)\}$
$\{\neg K\neg at(p_5)\} \longrightarrow \{\neg K\neg at(p_5)\}$
$\{Kat(p_x)/t\} \longrightarrow \{Kat(p_{x+1})/t, K\neg at(p_x)/t\}$
$\{Kat(p_5)/t\ \} \longrightarrow \{\ Kat(p_5)/t\}$
$\{\neg K\neg at(p_x)/t\} \longrightarrow \{\neg K\neg at(p_{x+1})/t, \neg Kat(p_x)/t\}$
$\{\neg K\neg at(p_5)/t\} \longrightarrow \{\neg K\neg at(p_5)/t\}$,
for any $t \in \{\mathtt{at}(p_1), \mathtt{at}(p_2)\}$, *and for any* $x \in [1, N-1]$.

right:

Precondition: $\emptyset$

Effect:
$\{at(p_x)\ \} \longrightarrow \{\ at(p_{x-1}), \neg at(p_x)\}$ $\quad \{at(p_1)\ \} \longrightarrow \{\ at(p_1)\}$,
for any $x \in [2, N]$.

Initial situation $\mathcal{I}$: $\{\ at(p_2)\ \}$

Goal $\mathcal{G}$: $\{\ Kat(p_4)\ \}$

Here the initial situation in the conformant problem P is described by the formula oneof$\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big)$, indicating that the initial position of the robot can be either p_1 or p_2 in Figure 2.2.

$$\mathcal{I} = \Big\{\big(\mathtt{at}(p_1), \mathtt{at}(p_2)\big), \big(\neg\mathtt{at}(p_1), \neg\mathtt{at}(p_2)\big), \big(\neg\mathtt{at}(p_5)\big), \big(\neg\mathtt{at}(p_3)\big), \big(\neg\mathtt{at}(p_4)\big)\Big\}$$
$$\mathcal{G} = \big\{\mathtt{at}(p_4)\big\}$$

The only fluents that are true in all the possible situations in $\mathcal{I}$ are $\neg\mathtt{at}(p3)$, $\neg\mathtt{at}(p_4)$, and $\neg\mathtt{at}(p_5)$, thus they are the only fluents that appear in the initial state $\mathcal{I}'$ of the translation $K_0(P)$:

$$\mathcal{I}' = \{(K\neg\mathtt{at}(p_5)), (K\neg p3), (K\neg\mathtt{at}(p_4))\}$$
$$\mathcal{G}' = \{K\mathtt{at}(p_5)\}$$

The translation K_i applies to any problem P, remaining sound. If the conformant width of the problem is bound by an integer $i \geq 0$, the translation K_i is also complete (Palacios and Geffner, 2009). In all the cases, for a bounded value of the parameter i, the translation K_i is sound and polynomial in the number of fluents, actions, and clauses in the problem P.

Theorem 3.18 (Properties of K_i)**.** *Given a conformant planning problem P, for a fixed positive integer i, the translation $K_i(P)$ is sound, and polynomial in size. If $w(P) \leq i$ the translation $K_i(P)$ is also complete.*

This result is important as it has been noticed that most of the conformant planning problems have width equal to 1, meaning that a sound and complete translation K_1 can be provided to solve them with a classical planner, *with tags of size one.* Such translation has polynomial size and appears to be effective on the majority of the conformant planning problems. This is the main motivation behind the implementation of the successful planner T_0, which has been built on top of K_1. The planner T_0 has been the best performer at the international planning competition IPC 2006 (Bonet and Givan, 2006).

3.5 Summary

We saw in this chapter several sound translations for planning with incomplete information. These translations map deterministic conformant planning problems into classical problems, which solutions are plans for the conformant problem.

The general translation scheme $K_{T,M}$ is built on top of two parameters, a set of tags T encoding assumptions about the initial situation, and a set of merges M encoding valid disjunctions of tags. For T equal the total possible initial states S_0, and M a single merge upon S_0, the translation $K_{S_0}$is sound and complete, but exponential in the number of unknown fluents of the problem. Setting M to contain one merge m for each precondition and goal literal L, given by the models of $C_I(L)$ that are consistent with $\mathcal{I}$, and T to contain the tags in all such merges, plus the empty tag, the resulting K_{models}translation is sound and complete, but exponential in the number of the fluents L in $C_I(L)$. The translation K_0 includes only the empty tag and no merge; it is effective for *simple* problems[2], but complete only if $w(P) = 0$. We presented then the notion of conformant width, a structural parameter of the problem that provides an upper bound on the time and space complexity required for generating a complete translation. The sound and complete translation K_i for conformant planning is exponential in the width of the problem, meaning that for problems with high width, this translation-based approach is impractical. In those cases, adopting a translation K_i with a parameter i lower than the width of the problem, can result in an unsolvable translation for a planning problem that indeed has solutions. These results are resumed in Table 3.1.

To overcome this, and other limitations, of this translation, we introduce in the next chapter sound and complete translations that use samples, i.e. a subset of the states in the initial belief. The translation K_S built on samples, solves conformant problems effectively and compares favourably to other state-of-the-art planners.

translation	**complexity**	**completeness**
$K_{S_0}(P)$	exponential in the unknown fluents	complete
$K_{\text{models}}(P)$	exponential in unknown prec. and goals	complete
$K_0(P)$	linear	complete if $w(P) = 0$
$K_i(P)$	exponential in i	complete if $w(P) \leq i$

Table 3.1: Comparison of sound translations for conformant planning.

[2]cf. section 8.1, page 144.

Part II

New Translations for Conformant Planning

Chapter **4**

Sound and Complete translations using samples

Deterministic conformant planning is the basic form of conformant planning: it is like a classical planning problem but with many possible initial states instead of one, and a plan is conformant when it works for each one of them. The conformant planning problem is harder than classical planning (Haslum and Jonsson, 1999; Turner, 2002) and has applications like deriving finite state controllers (Hoffmann and Brafman, 2005a; Bonet et al., 2009) or generating homing sequences in electronic circuits. Moreover, both non-deterministic conformant and contingent problems can be solved employing deterministic conformant planning techniques, which make it crucial for the area of automated planning under uncertainty.

Deterministic conformant planners suffer of mainly two limitations. First, and this is a common problem of planning under uncertainty, the heuristics to guide the search are weak, in the sense that they are generally not well informed compared to heuristics for classical planning tasks. Second, the belief representation and update is an intractable problem, in the worst case. In the translation-based approach due to Palacios and Geffner, reviewed in the last chapters, the two aspects are handled together by translating conformant problems into classical ones that are solved with classical planners. But this approach suffers from a belief representation that is exponential in the width of the problems for sound and complete translations, and the resulting classical heuristics miss important aspects of planning under uncertainty, as the cardinality of the belief states.

In this chapter we introduce a new translation $K_S^i(P)$ that overcomes the limits of the translation-based approach *à la* Palacios & Geffner. We present also a new planner, T1, based on a particular case of the general translation scheme $K_S^i(P)$. T1 compares favourably with state of the art conformant planners, and scales up well on difficult planning problems.

Results from this chapter have been published in *Effective Heuristics and Belief Tracking for Planning with Incomplete Information*, by A. Albore, M. Ramirez, and H. Geffner, in 21st International Conference of Automated Planning and Scheduling (ICAPS-11), Freiburg, Germany, 2011 (Albore et al., 2011).

4.1 Motivation

The translation-based approach that we discussed in chapter 3 is competitive with belief search approaches, and is in fact an instance of them: beliefs over the conformant problem P are encoded as states over the translation $K_{T,M}(P)$, and heuristics over beliefs are reduced to classical heuristics over states. In spite of its performance, however, the translation-based approach runs into multiple problems.

First, complete translations[1] are size-exponential in the width of the problem P. This means that for problems with large conformant width, such translations are ineffective, and the planners fail to scale up completely.
The second issue, no less important, is that incomplete translations can't be used for heuristic guidance, as the heuristic values that they generate can be infinite even when the problem P is solvable. The result is that incomplete translations may end up mapping a solvable problem into an unsolvable one.
Last, important aspects that are specific to the conformant setting and that proved useful then, like the cardinality of the belief states, seem to get lost in the translation. In fact, $K_{T,M}(P)$ translations do not encode explicitly the available knowledge about the certainty of a considered belief state, which is also a fundamental factor that must be taken into account in order to control the exploration of the belief space, beside an estimate of the distance of a state to the goal.

We propose a new translation, based on top of $K_{T,M}(P)$, but not to translate the conformant problem into a classical one, solved then by an off-the-shelf classical planner; instead, to address to the problem of the representation and the heuristics, our translation uses a subset of the initial belief state, which we call a *sample*, to both provide informed heuristics for the search and to represent belief states in a compact way.

The basis of the new belief space search planner T1 is then a translation K_S^i, that unlike the translation K_i considered in section 3.4, is always *tractable* and *complete*, but not always sound. The translation K_S^i is *sound* for problems P with conformant width no greater than i.

Palacios' and Geffner's K_i translation, on the other hand, is *tractable* and *sound*, but *complete* only for problems with width bounded by i. Thus, while K_i gives up completeness for problems with width beyond i, K_S^i gives up soundness then. For conformant planning this means two things: heuristics based on the new translation K_S^i will produce infinite values only when the problem is actually unsolvable, and that the belief literals resulting from this translation must sometimes be checked for validity. This first result solves one of the main issues of the translation-based approach Ki, which was incomplete for problems with conformant width beyond i.

The conformant planner T1 uses the translation K_S^i for $i = 1$ to generate *heuristic* and *candidate belief literals*. The beliefs are then verified with a SAT engine in the way it is done by conformant-FF (Brafman and Hoffmann, 2004). Moreover, the heuristic resulting from the translation K_S^1 is extended with a second heuristic that is obtained from *invariant "oneof expressions"* derived from the problem. As we will see, this second heuristic is related to both *cardinality heuristics* (Bertoli and Cimatti, 2002), and *landmark heuristics* (Richter et al., 2008). The planner, by

[1]We recall that complete translation are for us those in which every conformant plan for P appears as a classical plan for $K_{T,M}(P)$.

using a combination of these two heuristics, has a more informed and efficient scheme to search in the beliefs space, that solves the second and third problem mentioned above, which is like killing two birds with one stone.

4.2 A translation based on samples

In order to benefit from tractable translations while avoiding the limitation of incomplete translations, we take a different approach: we formulate *a new translation that is tractable and complete, but which is not always sound.* The new translation can't be used to solve conformant problems with a classical planner, as classical plans might not be solutions of the original conformant problem, but it is effective to derive heuristics and compute beliefs inside a belief space planner.

The key idea is to sample and use a set of states S, subset of the initial set of possible states S_0, to plan from those states, and to employ the solution to drive the search in the belief space. This approach is effective, in particular when –and this is our main motivation– we sample a polynomial number of initial states, even when their total number is exponential.

This translation-based approach is similar to the translation K_{S_0} we saw in section 3.3. The translation K_{S_0} is a particular case of $K_{T,M}$, where the set of T tags is equal to the set of states in the initial belief S_0. The translation K_{S_0} is sound and complete, as all the plans for the translated problem $K_{S_0}(P)$ are all the plans for the original conformant problem P.

We introduce the translation $K_S(P)$, considering the set of states S, which is a subset of the states in the initial belief: $S \subseteq S_0$. K_S is similar to K_{S_0}, but with the set of tags $T = S$, and a single merge on S.

Definition 4.1 (Translation $K_S(P)$)**.** *Given a conformant planning problem P, and a subset S of the initial belief state S_0, the translation $K_S(P)$ is the translation $K_{S_0}(P)$ applied to the problem $P|_S$, i.e. the problem P restricted to the initial set of states S.*

A solution for $K_S(P)$, in the case $S = S_0$, is obviously a solution for the problem P, as the plan would maps all the initial states onto goal states. On the other hand, a plan π for P is necessarily also a plan for $K_S(P)$: π conforms with all the states in S_0, of which S is a subset. It comes that the translation K_S is always complete, but not always sound. Compared to the translation $K_{T,M}(P)$, the new translation trades off soundness for completeness, meaning that in $K_S(P)$ the initial situation $\mathcal{I}$ always has a finite heuristic value when P is solvable, unlike what happens for incomplete instances of $K_{T,M}(P)$.

Theorem 4.2 (Completeness of K_S)**.** *Given a conformant planning problem P, and a set of initial states S of P, the translation $K_S(P)$ is complete.*

In the general case, the translation K_S is not sound as not all the plans that map S onto the goal would map *all* the states in S_0 onto the goal. Nevertheless, under certain circumstances we will see that this translation can be sometimes sound and complete. To define the conditions under which the translation K_S, based on a set of states S, is complete we will need to recall the notion of *basis* (Palacios and Geffner, 2009). A basis, for a problem under uncertainty, is a subset S of the initial belief state S_0 such that all the plans for S are plans for S_0:

Definition 4.3 (Conformant Basis)**.** *Given S a subset of the set of all possible initial states S_0 of a conformant planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. S is a* basis *for P when any plan for $P|_S$ is a conformant plan for P.*

A direct consequence of this definition is that a problem can be solved only by considering the states in a basis S, ignoring all the other states of $S_0 \setminus S$. In fact the plans for the problem $P|_S$, which is P but taking as initial belief the set of states S, are plans for the full problem P, and the plans for P are obviously plans for $P|_S$.

If we consider the set of states $S \subseteq S_0$ such that S is a basis for the problem P, then the translation $K_S(P)$ is complete and sound, meaning that all the plans for the states in S are all and only the plans for the states in S_0:

Theorem 4.4 (Soundness of K_S)**.** *Given a conformant problem P, and a set of initial states S of P, then the translation $K_S(P)$ is sound if S is a basis for P.*

A particular interesting result from Palacios and Geffner (2009) relates the size of bases to the width of the problem (as from definition 3.15), and leads to the following:

Theorem 4.5. *Conformant planning problems P with conformant width bounded by an integer i have bases of size exponential in i.*

This theorem is useful computationally when the size $|S|$ of a basis S is exponentially smaller than $|S_0|$. For problems of bounded width, it is then possible to find a basis of size *exponential in the width*, even if the size of S_0 is *exponential in the number of fluents.*

Example 4.1. *Let's consider an example problem where the task is to defuse n bombs*[2]*. Initially the status of the bombs is not known, so the (only) uncertainty in P is due to a clause $(armed(bomb_1) \vee \ldots \vee armed(bomb_n))$.*

The available action is `defuse (bomb)`*, with one conditional effect:*
`(when (armed ?bomb) (not (armed ?bomb)))`

Here, the only uncertain literals in $\mathcal{I}$ that are relevant to a precondition or goal literal are the $armed(bomb_i)$ (NB: their negation is not relevant to a precondition or a goal). Then it is easy to check that S_0 has 2^n possible initial states, but the set of n states $S = \{s_1, \ldots, s_n\}$ that make just one of the $armed(bomb_i)$ literals true is indeed a basis for P:

$$\begin{aligned}
s_1 &: \big(armed(bomb_1), \neg armed(bomb_2) \ldots, \neg armed(bomb_n)\big)\\
s_2 &: \big(\neg armed(bomb_1), armed(bomb_2) \ldots, \neg armed(bomb_n)\big)\\
&\ldots\\
s_n &: \big(\neg armed(bomb_1), \ldots, \neg armed(bomb_{n-1}), armed(bomb_n)\big)
\end{aligned}$$

[2]This example is a vanilla version of the infamous "bomb in the toilets".

The above Theorem 4.5 depends on some results that relate bases with the the notion of relevance that has been described in definition 3.8. The relevance of a literal depends on the conditional effects and do not depend on action preconditions, that have to be known with certainty and hence do not "propagate" uncertainty. We denote with $rel(s, L)$ the set of literals in a state s that are relevant to a literal L:

$$rel(s, L) = \{L' \in s \mid L' \text{ is relevant to } L\} \tag{4.1}$$

From this notion of relevance, and the definition 4.3 of a basis S, we can derive the following result:

Proposition 4.6. *The set of states S is a basis for P if S contains a state s such that $rel(s, L) \subseteq rel(s_0, L)$, for every possible initial state $s_0 \in S_0$ and every precondition and goal literal L in P.*

The proof of this proposition is sophisticated, and we do not report it here; the enthusiastic reader can see (Palacios and Geffner, 2009) for the complete proof. Other important results on bases of this chapter can be found in the same paper.

The former result can be exploited to determine bases for conformant problems P. At least one basis can be identified for each problem P, and this basis has size exponential in a parameter i, even if the size of S_0 is exponential in the number of fluents. It is easy to check that in the example 4.1, the set S_0 contains states where multiple literals $armed(bomb_i)$ are true; yet the set S is a basis, as for any state s_0 in S_0 there is state s in S where just one of those literals $armed(bomb_i)$ is true.

A new polynomial translation K^i_S

For problems P with conformant width bounded by a value i, a basis S^i of size exponential in i exists (Palacios and Geffner, 2009). From the Theorem 4.4 follows that if we can identify such a basis S^i for a problem P with bounded width i, we guarantee that a translation based on the samples set S^i is sound and complete. This translation also has the advantage to be exponential in the parameter i, instead that in the number of unknown fluents in S_0.

The translations $K^i_S(P)$ built fixing the parameter i for the basis S^i are sound for planning problems P with width bounded by i, and complete for all problems.

The bases S^i used for such translations comprise a set of *sample states* of S_0 that are minimal models of $\mathcal{I}$ plus the literals relevant to some precondition and goal. We define S^i in terms of the samples $S^i(L)$, picked up for each precondition and goal literal L. For this we define the L–rank of s (or simply the *rank* of s) as the number of literals in s that are relevant to L:

Definition 4.7 (L–rank)**.** *Given a state s, and a literal L, the "L–rank of s" is the positive integer given by the number of literals in s that are relevant to L:*

$$\left| rel(s, L) \right|$$

For any precondition and goal literal L, and a fluent formula C consistent with $\mathcal{I}$, we define $s_L(C)$ as the set of states that satisfy $\mathcal{I}$, and C, and *a minimum number* *of literals relevant to L.* If $|rel(s, L)|$ is the L–rank of s, then $s_L(C)$ stands for the lowest ranked states that satisfies $\mathcal{I}$ and C. Of course there is always one such state, although not necessarily unique.

Definition 4.8 (Lowest ranked states s_L)**.** *Given a conformant planning problem P with initial belief state S_0, a fluent L, and a formula C consistent with $\mathcal{I}$. $s_L(C)$ is the set of initial states satisfying $\mathcal{I}$ and C, and a minimal number of literals relevant to L:*

$$s_L(C) = \{s \in S_0 \,|\, s \textit{ satisfies } \mathcal{I} \wedge C, \textit{ and } |rel(s, L)| \textit{ is minimal.}\} \tag{4.2}$$

Example 4.2. *In the example 4.1 each state $s \in S$ has rank 1: the only uncertain literals in $\mathcal{I}$ that are relevant to a precondition or goal literal are $armed(bomb_i)$, it follows that for every L precondition or goal, we have $|rel(s, L)| = 1$. Notice that a state s' making all the fluents true, such that*

$$s' = \big(armed(bomb_1), armed(bomb_2), \ldots, armed(bomb_n)\big),$$

would satisfy all the literals L, but will have also $|rel(s', L)| = n$. Given that in $\mathcal{I}$ we have $C = \big(armed(bomb_1) \vee \ldots \vee armed(bomb_n)\big)$, we thus obtain $s_L(C) = \{s_1, \ldots, s_n\}$, as the states s_i are all the minimal ranked states that satisfy $\mathcal{I}$ and the clause C.

The set S^i is chosen to depend on the width i of the problem, hence on the number of clauses relevant to preconditions and goals.

Definition 4.9 (Relevant terms of bounded size $T^i(L)$)**.** *We define $T^i(L)$ as the set of consistent conjuncts of literals initially unknown that are relevant to a precondition or a goal L, which maximum size is equal to a parameter i:*

$$T^i(L) = \Big\{\; t = L_1 \wedge \ldots \wedge L_m \;\;|\;\; m \leq i, \textit{ and for any } L_k \in t, \; L_k \textit{ relevant to } L, \\ \mathcal{I} \not\models L_k \wedge \mathcal{I} \not\models \neg L_k, \textit{ and } \quad \mathcal{I} \not\models \neg t \;\Big\}$$

If $i = 0$, then the only available term is $t = \top$.

The set of samples S^i is defined to contain the minimal ranked states that satisfy $\mathcal{I}$ and all the terms t in $T^i(L)$, with L ranging over all precondition and goal literals. In such a way, the states in S^i satisfy a minimum number of literals relevant to L.

Definition 4.10 (Set of samples S^i)**.** *For any conformant problem P, and any parameter $i > 0$, the set of samples S^i is defined as:*

$$S^i = \bigcup_L S^i(L) \tag{4.3}$$

where L ranges over the precondition and goal literals in P.

The set of states $S^i(L)$ is set of the minimal L–ranked states that satisfy the tags t in $T^i(L)$:

$$S^i(L) = \{s_L(t) \,|\, t \in T^i(L)\} \tag{4.4}$$

where $s_L(t)$ is the set of states that satisfy t, $\mathcal{I}$, and a minimum number of literals relevant to L, following definition 4.8.

The definition puts in relation the basis and the notion of tags in a covering transltation, represented by the sets $T^i(L)$, and yields to the result we were looking for:

Theorem 4.11. *Given a conformant planning problem P, the sample set S^i, as defined in 4.10, is a basis for P if the width of P is bounded by i.*

The theorem can be proved by providing an alternative equivalent characterisation of the notion of width. We are going to make a parallelism between how the basis S^i is defined and the notion of *tags*, used for the translation $K_{T,M}(P)$. In fact, for a basis, we want to capture those states s such that they satisfy a minimal set of initial terms t^* in $T^i(L)$ that are relevant to precondition or goal literals L.

For a term t in $T^i(L)$, let t^* stand for the tuple that extends t with all the literals deducible from t and $\mathcal{I}$:

Definition 4.12 (Deductive Closure t^*)**.** *Given a tuple t in $T^i(L)$, we define the deductive closure t^* of t as the set of literals entailed by t in $\mathcal{I}$:*

$$t^* = \{L \mid \mathcal{I} \wedge t \models L\} \tag{4.5}$$

Let now be $m^i(L)$ the subset of the tuples t in $T^i(L)$ such that all the literals L' in $s_L(t)$ that are relevant to L are also in t^*. In other words, $m^i(L)$ admits states $s_L(t)$ that do not make true literals relevant to L that do not belong to t^*:

Definition 4.13 (Tuples $m^i(L)$)**.** *Given a literal L of a conformant planning problem P, the tuples $m^i(L)$ can be interpreted as a DNF regrouping the tags in $T^i(L)$:*

$$m^i(L) = \left\{t \;\mid\; t \in T^i(L) \;\; and \;\; rel\,(s_L(t), L) \subseteq t^* \;\right\} \tag{4.6}$$

The tuples $m^i(L)$ can be put in correspondence with the merges of a covering translation, that satisfy the clauses encoding uncertainty that are relevant to precondition and goal literals L. The conformant width of a problem P can be then (re)defined in terms of the clauses $m^i(L)$:

Definition 4.14 (Width of a literal)**.** *The width $w(L)$ of a literal L is the minimal value of i for which the DNF formula $m^i(L)$ is entailed by $\mathcal{I}$.*

Recall that $w(P)$, the conformant width of a problem P, is just $\max_L w(L)$, where L ranges over the precondition and goal literals in P (cf. definition 3.15). Definition 4.14 thus implies that $w(P)$ is the minimum value of i for which $m^i(L)$ is a valid merge, for any precondition or goal L. Palacios' and Geffner's K_i translation is an instance of the $K_{T,M}$ translation that uses these merges along with the tags in them. The translation K^i_S below is a variation of the K_{S0} translation that uses min-ranked states $s_L(t)$ associated with the tags instead.

The set of states S^i is then a basis for a problem P if the states in S^i satisfy the terms in $T(L)$, which is the DNF of the conjuncts of any size of literals initially unknown and relevant to the preconditions or goals L. The central point here being that the states that satisfy $T^i(L)$ are the same that satisfy $T(L) = T^\infty(L)$, if $w(P) \leq i$.

In particular, if $m^i(L)$ is valid in $\mathcal{I}$, every state s_0 that satisfies $\mathcal{I}$ must also satisfy a term t in $m^i(L)$. But from the definition of $m^i(L)$, for any state $s = s_L(t)$, we will have that $rel(s, L) \subseteq rel(s_0, L)$; it derives and from Proposition 4.6, that the set $S^i(L)$ of states $s_L(t)$, for $t \in m^i(L)$, forms a basis for P, provided that L is the only precondition or goal. This proves Theorem 4.11. The tuples t can then be used as tags, with similar properties that in translation $K_{T,M}(P)$.

A very similar result has been obtained for covering merges from definition 3.10: the merges required for completeness are simply the valid merges m that satisfy the set of clauses $C_I(L)$ (the initial clauses relevant to a precondition of goal L). There, covering merges of bounded size were used for K_i conformant translation, that were sound and complete for problems of width bounded by i. Here, the states s_L – which constitute the basis S^i for a translation $K^i_S(P)$– are obtained from the tuples $m^i(L)$ such that all the literals relevant to a precondition or goal L in $s_L(t)$ are also the literals t^* entailed by the clauses t in $m^i(L)$. We define the translation $K^i_S(P)$ as follows:

Definition 4.15 (Translation $K^i_S(P)$)**.** *Given a conformant planning problem P, and a positive parameter i, the translation $K^i_S(P)$ is defined as $K_S(P)$, with $S = S^i$ defined as in 4.10.*

Defining the translation K^i_S as K_S for $S = S^i$ directly yields that:

Theorem 4.16 (Soundness of translation $K^i_S(P)$)**.** *The translation $K^i_S(P)$ has a size that is exponential in i, is complete, and is sound for problems P having width bounded by i.*

The proof is direct from Theorem 4.11: by construction, the sample set S^i is a basis for P if $w(P) \leq i$. The computational advantage of using Theorem 4.16 comes from the fact that sound and complete translations have a size linear in $m^i(L)$ and thus exponential in the parameter i only, even if the size of the initial belief is exponential in the number of unknown fluents of the problem. In particular, many conformant planning problems have with bounded by 1, which means that a sound and complete translation can be made by generating samples S^1 of polynomial size.

4.3 Computation of S^1

The conformant planner T1 uses the sample set S^1 of the $K^1_S(P)$ translation for deriving heuristics and tentative belief literals that are used in the context of a belief space search algorithm.

The first step is to select the states that form the set S^1, given by the set of initial states sampled from S_0 such that they correspond to the lowest ranked states that satisfy $\mathcal{I}$. This is done by compiling the initial situation of the problem in d-DNNF, as explained in the next section.

Sampling initial states

The set S^1 of samples is defined to contain the minimal ranked states that satisfy $\mathcal{I}$, and all the tuples t of size 1 in $T^1(L)$, with L spacing over all precondition and goal literals. S^1 is obtained from the lowest ranked states $s_L(L)$ that satisfy $\mathcal{I} \cup \{L\}$, for various literals L. These states are computed by compiling the initial situation $\mathcal{I}$ into d-DNNF (Darwiche, 2001a), a logical form that allows the computation of a number of otherwise intractable operations, such as consistency, model counting[3], and lowest ranked models, in time that is linear in the size of the compilation (which is at worst exponential in the number of variables, but not necessarily so). The main reason that attracted us to propositional theories in d-DNNF is that they are highly tractable while maintaining the same expressive power.

d-DNNF

A propositional sentence is in negation normal form (NNF) if it is constructed from atoms using only the conjoint and disjoint operators and maintaining the negation symbol only next to the atoms[4].

A d-DNNF, *deterministic* DNNF, is a normal form introduced by Darwiche (2001a) and that constitutes a particular case of DNNF, a decomposable negation normal form that supports a wide set of polynomial-time operations. For instance deciding the satisfiability of a DNNF can be done in linear time, as conjoining a DNNF formula with a set of literals (Darwiche, 1999). Another linear time operation that interests us, and that we are going to use both to calculate the conformant width of a problem and to build the sample set S_0^i, is the computation of the minimal cardinality of a DNNF. Let's recall that the cardinality of a model is the number of atoms that are set to false in the model itself. The minimal cardinality of a theory is the minimal cardinality of any of its model.

The d-DNNF compilation is also used for selecting the min-ranked states $s_L(t)$ when they are not unique so that the sets $S^i(L)$ overlap as much as possible, and hence the size of their union S^i is minimised.

Definition 4.17 (DNNF)**.** *A decomposable negation normal form (*DNNF*) is a negation normal form satisfying decomposability property: for any conjunction $\bigwedge_i \alpha_i$ appearing in the form, no atoms is shared by any pair of conjuncts in $\bigwedge_i \alpha_i$.*

As an example, the NNF formula $(A \vee B) \wedge (\neg A \vee C)$ is not decomposable, since atom A is shared by the two conjuncts.

Definition 4.18 (d-DNNF)**.** *A deterministic* DNNF *(d-*DNNF*) is* DNNF *satisfying the following property: for any disjunction $\bigvee_i \alpha_i$ appearing in the form, every pair of disjuncts in $\bigwedge_i \alpha_i$ are logically disjoint.*

For a DNNF, the determinism is what makes model counting tractable. The number of models for determinist conjunctive formulæ ends up being the product of the number of models of each conjunct, while the number of models of a disjunction that satisfies determinism is the sum of the number of models of each disjunct.

The set of samples S^1 is obtained from its definition 4.10 by computing the minimal ranked states that satisfy $\mathcal{I} \cup L$ for various literals L, and in particular the ones that

[3]The *model count* of a formula stands for the number of truth assignments that satisfy the formula.

[4]A formula can be converted to the NNF by simply moving the negations across braces either using De Morgan's laws or the equivalence $(\neg\neg a = a)$.

are unknown in $\mathcal{I}$, and relevant to a precondition or goal. Thus, to compute these states, we first compile $\mathcal{I}$ into a d-DNNF formula δ, and then we extract the minimal models of δ. A model is said to be *minimal* for a satisfiable propositional formula δ if the number of atoms set to false in a truth assignment ω is minimal compared to any other truth assignment ω'. The minimal models of a formula are not necessarily unique.

If the cardinality of a model is the number of atoms that are set to false (or true) in the model, then the minimal cardinality of a theory is defined as the minimal cardinality of any of its models, i.e. the truth assignments that minimises the cardinality:

Definition 4.19 (Minimal cardinality)**.** *Let be δ a d-DNNF formula with a truth assignment ω on a d-DNNF formula. The minimal cardinality of δ, denoted* $\mathsf{mCard}(\delta)$*, is defined as follows:*

$$\mathsf{mCard}(\delta) = \begin{cases} 0 & \textit{if } \delta \textit{ is a positive literal or } \textbf{\textit{true}}, \\ 1 & \textit{if } \delta \textit{ is a negative literal}, \\ \infty & \textit{if } \delta \textit{ is } \textbf{\textit{false}}. \end{cases} \tag{4.7}$$

$$\mathsf{mCard}(\delta = \vee_{\mathsf{i}}\ \mathsf{a_i}) = min_i\ \mathsf{mCard}(\mathsf{a_i}) \tag{4.8}$$

$$\mathsf{mCard}(\delta = \wedge_{\mathsf{i}}\ \mathsf{a_i}) = \sum_i \mathsf{mCard}(\mathsf{a_i}) \tag{4.9}$$

A minimal model is such that any other assignment of variables has bigger cardinality for the same formula. The minimum cardinality of an unsatisfiable formula is defined to be ∞. The operation of computing the minimum cardinality of a d-DNNF can be done in linear time (Darwiche, 1998).

Algorithm 4.1: Generation of set of minimal samples S^1.

```
Output: set of samples $S$
1  W ⟵ 10000                    // Constant W high weight fixed to 10000.
2  $S$ ⟵ $\emptyset$
3  if $w(P) > 1$ then
4      getSamplesPlus($S$)
5  else
6      foreach $L$ goal or precondition in $P$ do
7          foreach $L'$ relevant to $L$ do
8              weight($L'$) ⟵ W
9          switch the value of $w(L)$ do
10             case 0
11                 getSamplesZero($L$,$S$)
12             case 1
13                 getSamplesOne($L$,$S$)
14 return $S$
```

Extraction of the minimal weighted models

The approximation initial belief state S^1 will be given by the set of initial states sampled from S_0 such that they correspond to the lowest ranked states satisfying $\mathcal{I}$. We will attribute a positive cost to the initially unknown literals that are relevant to a precondition or a goal literal, so to yield a set of sample corresponding to definition 4.10. In the Algorithm 4.1, the generation of the samples is differentiated following the width of each literal. The key idea is to extract a minimal model that "covers" as much literals as possible, i.e. that makes true as much literals L' relevant to a precondition or a goal. To obtain sample states as much different as possible, we use a high weight for literals relevant to a precondition or goal (the constant W), and a low weight (set to 1) for those literals made true in a model. This shrewdness makes the algorithm prefer models where are made true those literals that are false in other models: this is applied in Algorithm 4.2 at line 9, in Algorithm 4.3 at line 16, and in Algorithm 4.4 at line 9.

Algorithm 4.2: getSamplesZero – selects samples when $w(L) = 0$

```
   Input: literal $L$, set of samples $S$
1  foreach $L_0$ goal or precondition in $P$ do
2      if $L_0 == L$ || $w(L_0) \neq 0$ then
3          continue
4      foreach $L'$ relevant to $L_0$ do
5          weight($L'$) $\longleftarrow$ 1
6  s $\longleftarrow$ minModel($\mathcal{I}$)
7  if $s \notin S$ then
8      foreach $L'$ in $s$ do
9          ++weight($L'$)
10     $S := S \cup s$
```

All the optimizations aimed at minimising the size of the sample set S^i while retaining its properties are implemented by exploiting the capabilities of the d-DNNF compilation. By taking advantage of the fact that samples are computed incrementally, samples that make true literals relevant to other precondition or goal L' are considered if the rank of s in minimal for L', when $w(L') > 0$. Other optimizations, like removing states $s_L(t)$ from $S^i(L)$, when the tag t can be removed from the merge $m^i(L)$ without affecting its validity, are accommodated as well.

Example 4.3. *We can consider as an example the domain illustrated in Figure 4.1, an instance of the square-center domain, described in section 6.5. Here, positions in a grid 7×7 are denoted by a couple of coordinates (x, y). The initial position is completely unknown (marked in coloured areas in the figure), and the goal is to reach a position in the center of the grid. The initial situation $\mathcal{I}$ is given by the disjunctions $x_1 \vee \cdots \vee x_7$ and $y_1 \vee \cdots \vee y_7$. The movement is from an area to an adjacent one, so all the (coordinates of the) areas are relevant between themselves. In this class of problems, then arbitrary choices of the sets $s_L(x_k)$ and $s_{L'}(y_k)$ may result in a sample set S^1 with $2n$ states. On the other hand, it is possible to choose* *the states $s_L(x_k)$ to be equal to the states $s_L(y_k)$, e.g. by making all other x's and y's false, so that the size of S^1 is just n.*

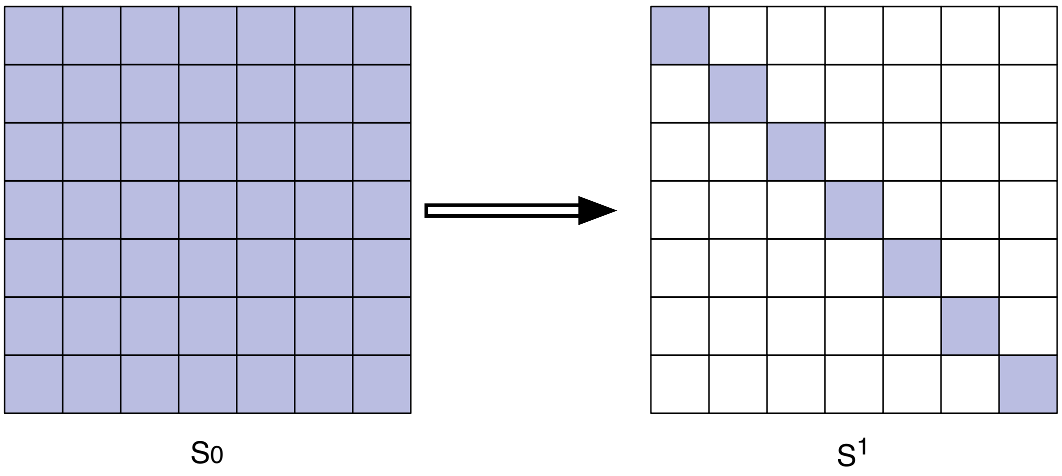


Figure 4.1: A grid domain. On the left S_0, the original set of states in $\mathcal{I}$. On the right the sampled set of states $S_1 \subseteq S_0$.

For problems with width bigger than one, the function `getSamplesPlus` is invoked (cf. Algorithm 4.4). The aim is not to produce a covering set of states, as the belief approximation is potentially exponential, but a set S^1 as varied as possible of fixed size; this size is set to 20 in Algorithm (line 1). Then, from Theorem 4.16, the approximation S^1 we use is unsound in any case, for problems with high width.

The d-DNNF compilation has also been used to determine the conformant width of each literal of the problems, from the results in previous section, in the following way: for all unknown literal relevant to a precondition or a goal, we set its weight to 1. Then the minimal weighted model is computed for the initial state and the negation of the literal, as described in Algorithm 4.5.

4.4 The T1 planner

The planner T1 is built on top of the K_S^1 translation. The algorithm is then sound for problems with width 1, and complete for all problems. We perform a forward search in belief space. Each node of the search space represents a belief state. For each belief state expanded, we perform a verification procedure to establish the literals that are positively or negatively known. This procedure is specifically executed in problems with width bigger than 1, as the translation in those cases is unsound. We then discuss how belief states are represented and updated in the search space.

Verification

For problems P where the translation $K_S(P)$ is unsound, not all the plans for the translated problem are plans for the original one. Thus, if for an actions sequence a the literal L is true in the belief state resulting from applying a to S_0, then L can be true in certain states of P but not in all of them. This mean that the set of

Algorithm 4.3: getSamplesOne – selects samples when $w(L) = 1$

```
Input: literal $L$, set of samples $S$
1  foreach $L_0$ goal or precondition in $P$ do
2      if $L_0 == L$ then
3          continue
4      foreach $L'$ relevant to $L_0$ do
5          if $L'$ relevant to $L$ then
6              continue
7          weight($L'$) $\longleftarrow 1$
8  foreach $L'$ relevant to $L$ do
9      if $L'$ is not unknown initially then
10         continue
11     s $\longleftarrow$ minModel($\mathcal{I} + L'$)
12     if mCard(s) $== \infty$ then
13         continue
14     if $s \notin S$ then
15         foreach $L_i$ in $s$ do
16             ++weight($L_i$)
17         $S := S \cup s$
```

Algorithm 4.4: getSamplesPlus – selects samples when $w(L) > 1$

```
Input: set of samples $S$
1  $N \longleftarrow 20$                // Constant N arbitrarily fixed to 20 samples.
2  foreach $L'$ relevant to $L$ goal or precondition in $P$ do
3      weight($L'$) $\longleftarrow 1$
4  while tot $\leq N$ do
5      s $\longleftarrow$ minModel($\mathcal{I}$)
6      if $s \in S$ then
7          break
8      foreach $L'$ in $s$ do
9          ++weight($L'$)
10     $S := S \cup s$
11     ++tot
```

Algorithm 4.5: Minimal weighted model to determine the conformant width of a literal L goal or precondition in P

```
Input: literal $L$
Output: width of $L$
1  foreach $L'$ relevant to L goal or precondition in $P$ do
2      if $L'$ is not unknown initially then
3          continue
4      $rel_L \longleftarrow L'$
5      weight($L'$) $\longleftarrow$ 1
6  if mCard(minModel($\mathcal{I}$)) $== 0$ || $rel_L == \emptyset$ then
7      width($L$) $\longleftarrow$ 0
8      return
9  foreach $L'$ in $rel_L$ do
10     if mCard(minModel($\mathcal{I}+$ $L'$)) $== 1$ then
11         $\mathcal{S} \longleftarrow L'$
12 if $\mathcal{S} == \emptyset$ then
13     width($L$) $> 1$
14     return
15 if mCard( minModel($\mathcal{I}+\bigwedge_{L' \in \mathcal{S}} \neg L'$) ) $\neq \infty$ then
16     width($L$) $> 1$
17 else
18     width($L$) $\longleftarrow$ 1
```

known and negatively known propositions in a belief state has to be computed. A proposition L is said to be known in a belief b if $s \models L$, for all states $s \in b$. Similarly, a proposition L is negatively known in b if the proposition L does not hold in any of the states s in b. Following these definition, we call *unknown* a proposition that in neither known nor negatively known. Deciding whether a proposition is known or not is co–NP–complete.[5] This verification can be done by making use of an implicit representation of the belief state b, given by the initial situation $\mathcal{I}$ encoded as a CNF formula together with the action sequence π that leads from $\mathcal{I}$ to b.

We check if a proposition is known, or negatively known, in a belief state b by testing if it is contained in the intersection of all the states of b. This can be done by taking advantage of the implicit representation of the belief states, by calling a SAT solver over a formula representing the current belief b in the propositional encoding $\Gamma(P)$ of the planning problem P. In other words, we check if applying the action sequence π to $\mathcal{I}$ ends up into a belief state where the proposition holds, given that the propositional encoding $\Gamma(P)$ of the problem P is such that the models of $\Gamma(P)$ are in direct correspondence with the plans the solve P.

For a conformant problem P, a plan π is a solution for all possible initial state $s \in \mathcal{I}$ if and only if the formula

$$\big(\Gamma(P) \wedge \pi \wedge s\big) \text{ is satisfiable}^4. \tag{4.10}$$

[5]The proof can be found in (Brafman and Hoffmann, 2006).

[4]This case applies only if the actions in π are deterministic.

The test we perform to check if a proposition L holds in the belief b resulting of applying π on $\mathcal{I}$, is to verify if the formula

$$\big(\Gamma(P) \wedge \pi \wedge \neg L\big) \text{ is not satisfiable} \tag{4.11}$$

For this, we used the approach of (Brafman and Hoffmann, 2006) that computes the sets of known and negatively known propositions in a belief state from a CNF formula built to correspond to the semantics of the action sequence π to be tested, and executed from the initial situation $\mathcal{I}$. As from eq. (4.11), a literal L is true in a belief b if the resulting CNF formula $\Phi(\pi)$ is denoted as *unsat* from a SAT solver call.

In $\Phi(\pi)$, a time index is used to differentiate between the values of the propositions in each step of the action sequence π, e.g. a proposition p true at time step i will be denoted by $p(i)$. The time indexes range from 0 to $|\pi|$. The encoding of the formula $\Phi(\pi)$ starts with all the clauses in $\mathcal{I}$, indexed with time 0:

- for each clause $(l_1, \ldots, l_n)$ in $\mathcal{I}$, we add to $\Phi(\pi)$ the disjunction $\big(l_1(0), \ldots, l_n(0)\big)$.

Then, let be $\pi = [a_1, a_2 \ldots, a_n]$, we extend the formula $\Phi(\pi)$ adding, for each action $a_i \in \pi$ at timestep i, the following effect, and frame axioms:

Effect Axioms

- For every conditional effect with positive effect $C \rightarrow P$, with $C = c_1 \wedge \ldots \wedge c_k$, and every proposition $p \in P$ of a_i, we add to $\Phi(\pi)$ the *add* effect axiom $\neg c_1(i-1) \vee \neg c_k(i-1) \vee p(i)$;
- For every conditional effect with negative effect $C \rightarrow P$, with $C = c_1 \wedge \ldots \wedge c_k$ (and $P \in del(eff)$), and every proposition $p \in P$ of a_i, we add to $\Phi(\pi)$ the *delete* effect axiom $\neg c_1(i-1) \vee \neg c_k(i-1) \vee \neg p(i)$.

Frame Axioms

- For every conditional effect with negative effect $C \rightarrow P$, with $C = c_1 \wedge \ldots \wedge c_k$ (and $P \in del(eff)$), and every proposition $p \in P$ of a_i, we add to $\Phi(\pi)$ the *positive* frame axiom $\neg p(i-1) \vee c_1(i-1) \vee c_k(i-1) \vee p(i)$;
- For every conditional effect with positive effect $C \rightarrow P$, with $C = c_1 \wedge \ldots \wedge c_k$ (and $P \in add(eff)$), and every proposition $p \in P$ of a_i, we add to $\Phi(\pi)$ the *negative* frame axiom $p(i-1) \vee c_1(i-1) \vee c_k(i-1) \vee \neg p(i)$.

The effect axiom encodes the effects (positive and negative) of each action a_i: if the condition holds, then the effect holds. The frame axiom encodes that if a proposition was true (resp. false) before applying the action a_i, and the effects of a_i do not delete (resp. add) the proposition, then it is still true (resp. false) in the next frame, i.e. after executing a_i.

If an action is executable, its preconditions are not encoded into $\Phi(\pi)$; this is done for all propositions p that are unknown. Several SAT calls must be performed during the search of a plan, however we limit them depending of the width of the problem, even if modern SAT solvers can hold problem instances with ten thousands variables. When the problem width is minor or equal to the translation parameter i in K_S^i, the set of known and negatively known propositions is given by simple progressing the initial known propositions through the action sequence π. This procedure is reflected in our implementation, and in particular in the way search nodes are encoded.

Search space

A node in the search graph represents a belief state and corresponds to a tuple $n = \langle \pi, S, R \rangle$, where:

- π is the plan prefix used to reach n from the root node of the search,
- S is the sample set S^1 progressed through π,
- R is a set of literals [6].

The belief state b_n represented by the node n corresponds to the whole set S_0 of initial states of P progressed through π. This means that belief states are not computed explicitly, but represented implicitly in the plan prefix π. The set of literals in R contains literals all of which are known to be true in b_n, and is sound with respect to b_n but not necessarily complete; i.e. even if R contains all literals known to be true in b_n, it does not contain in general all such literals. Likewise, S is complete with respect to b_n but not necessarily sound; i.e., all literals true in b_n are true in all the states in S, but not the other way around:

$$\text{Literals in } R \;\subseteq\; \text{Literals true in } b_n \;\subseteq\; \text{Literals true in } S \tag{4.12}$$

Still, if the problem has width bounded by 1, the set of true precondition and goal literals coincide in b_n and S, and such literals are added to R.

The root node in the search is $n_0 = \langle \pi_0, S^1, R_0 \rangle$, where π_0 is the empty plan, S^1 is the approximation of the initial belief in the $K_S^1(P)$ translation, and R_0 is the set of literals known to be true in the initial situation.

The goal nodes are the nodes $n = \langle \pi, S, R \rangle$ where R contains the literals goal of the problem.

The edges in the graph correspond to the applicable actions. An action is applicable in $n = \langle \pi, S, R \rangle$ when all the action preconditions are in R. The node n' that results from applying the action a in the node n is given by $n' = \langle \pi', S', R' \rangle$, where π' is π with the action a appended, S' is the result of progressing each of the states $s \in S$ through a, and R' is the set of literals obtained by progressing the set of literals R in n through the action a in the following manner:

$L \in R'$ iff
1) there is an effect $a : C \to L$ such that $C \subseteq R$ (L is added), or
2) $L \in R$ and for every effect $a : C' \to \neg L$, R contains the complement of a literal in C' (L persists).

[6] In the implementation, the set S of progressed samples is not stored in the nodes; rather the progression is done when required.

The computation of the atoms in R' from R corresponds to the application of the support and cancellation rules in the translation $K_0(P)$ that involves no tags, as we can remember from section 3.1, and which corresponds in turn with the 0-approximation semantics (Baral and Son, 1997).

While there is no need for the set of literals R in $n = \langle \pi, S, R \rangle$ to include all literals true in the belief state b_n for the planner to be complete, R must be complete with respect to action preconditions and goals. This means that if a goal is true in b_n or an action is applicable in b_n, then the goal and the action preconditions must be in R. For problems having conformant width no larger than 1, the above property is guaranteed as the sample set S is not only complete but also sound, and hence the literals true in S can be added to R. For problems with higher width, on the other hand, a verification operation is carried out after the node has been generated. This is done by looking at the samples in S: a precondition or goal literal L is added to R when L is true in S, and this belief is certified by calling a SAT solver over a suitable CNF formula, as in Conformant-FF (Brafman and Hoffmann, 2006) and similarly as described in the previous section (p. 60). The formula corresponds to the semantics of the action sequence π, executed from the initial situation: $\mathcal{I}$ is encoded at time 0, the conditional effects of the i-th action in the plan prefix π are encoded at each time slice $[i, i+1]$, and the *negation* of the literal L to be tested is encoded at time $|\pi|$. The literal L is true in b_n if the resulting CNF formula is unsatisfiable. This procedure is sound, but not complete in a general sense, since the test is performed only on precondition and goal literals.

A last operation is performed in T1 to test whether two nodes n and n' represent the same belief state, i.e. if $b_n = b_{n'}$. This test is not needed for completeness, but for saving duplicate work, ruling out nodes that have already been expanded. When the problem width is 0 or 1, the two nodes $n = \langle \pi, S, R \rangle$ and $n' = \langle \pi', S', R' \rangle$ can be safely collapsed when $S = S'$ and $R = R'$. For problems with higher width, an additional test is needed: n and n' are collapsed if, in addition to the former property, for each possible initial state s of the problem, the plans π and π' result in the same state s'. This test, which is sound, is performed through a single SAT call over two formulæ like the CNF above to verify if L follows from a plan prefix π from $\mathcal{I}$, except that now two disjoint formulæ are built to encode the plan prefixes π and π'. The first formula involves fluent variables $x(i)$ for atoms x in the problem, $0 \leq i \leq |\pi|$, and the second uses primed variables $x'(k)$ for atoms x in the problem, $0 \leq k \leq |\pi'|$. Then these two formulæ are joined along with two other formulæ built as follows. The first, establishing the equivalence between the x variables at the beginning of the plans, $\bigwedge_x (x(0) \equiv x'(0))$, the second postulating a non-equivalence at the end of the plans, $\bigvee_x \neg\big(x(|\pi|) \equiv x'(|\pi'|)\big)$, where x ranges in both cases over the fluents of the problem. If there is a model that satisfies this formula, it means that there is a possible initial state s_0 and an atom x such that one of the plans π or π' achieves x from s_0 and the other does not. The test is not complete because b_n and $b_{n'}$ may contain the same states, even if π and π' do not result in the same state from some possible initial state (e.g., as when there are two possible initial states s and s' such that π maps s into s', and s' into s, while π' is empty). Conformant-FF performs a similar test for the same purpose of ruling out duplicate nodes.

4.5 Effective heuristics for planning with incomplete information

The difficulties in finding and using heuristics in the belief space come from the belief state formulation of the different aspects of the planning problem. When planning under uncertainty, not only the distance from the goal has to be used in order to find a solution, but other aspects too, related to the uncertainty in the problem. One of these aspects is the size of the belief, i.e. the number of states in it, which is an estimate of the amount of uncertainty of the situation it encodes. This information is being mainly used by the KACMBP planner, for instance. However, reducing the size of the belief state has no direct relation with reducing the goal distance.

As a motivational example, let's consider a 9×9 grid–like domain, where a robot must reach a goal cell (the central cell outlined in red in the leftmost grid in Figure 4.2) starting from an ambiguous initial position (one of the four blue cells). Four actions are available to the robot: East, West, South, and North. A wall can't be passed, so moving in its direction would not change the current position. Reducing the

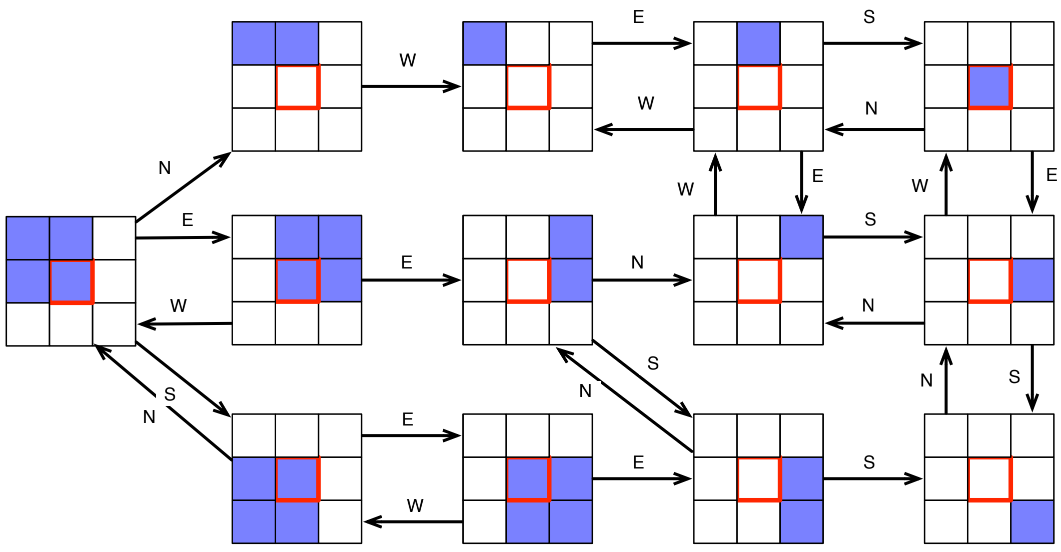


Figure 4.2: Portion of the Belief space of the square room localisation domain.

distance from one of the possible initial states by moving East would not guarantee that the goal is reached for sure, as many possible states are currently possible. A good solution plan would be to move in the top–left corner [North, West] to remove the ambiguity about the position of the robot and, once localised, to move deterministically through the goal [East, South]. With this simple example we have a glimpse of the main difference between finding a solution in state space and finding a solution in the belief space: clearly, in both cases an estimate of the distance of a state to the goal is essential to drive the search and obtain plan of good quality, however the amount of available knowledge is also a fundamental factor that must be taken into account in order to control the exploration of the belief space.

We aim, with the new translation $K^i_S(P)$, at deriving heuristics, based on the states in S, that take into account these two major aspects of automated planning under uncertainty: the cardinality of the belief state, and its distance from the goal. The result planner T1 uses two heuristics in combination, that we call, the *classical heuristic* and the *certainty heuristic* for reasons that will be clear below.

Classical Heuristic h_C

The classical heuristic $h_C(n)$ for a node $n = \{\pi, S, R\}$ is defined as the estimated cost of the *classical planning problem* $K_S(P)$. For the root node, this is the estimated cost of the translation $K_S^1(P)$, as S is then S^1.

The estimated cost of the classical problem $K_S(P)$ is obtained from the combination of the additive and relaxed plan heuristics (Bonet and Geffner, 2001a; Hoffmann and Nebel, 2001), as formulated by Keyder and Geffner (2008), where a relaxed plan is constructed backward from the goal by collecting the best supporters of the atoms in the goal, and recursively, the best supporters of the preconditions of those supporters. The estimated cost is the number of actions in the relaxed plan, while the *helpful actions* (to be used in the search) are as defined in FF (Hoffmann and Nebel, 2001): the applicable actions that add a precondition or a goal in the relaxed plan.

The computation of the heuristic has been done by introducing new fluents $L|_s$, to keep track of the truth values of literals L in each state s of the set S used for the translation. The border condition for the Dijkstra algorithm used is given by $h(L|_s) = 0$ if L is true in s, while all the other heuristic values are set to ∞ (Dijkstra, 1959). In the heuristic computation, merge rules are implicitly encoded as rules with zero cost: $\bigwedge_{s \in S} L|_s \rightarrow L|_S$.

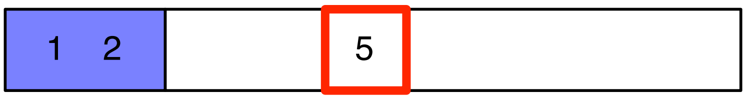


Figure 4.3: Corridor domain. Initially the agent can be either in position 1 or 2; the goal is to reach the cell marked “5”. Allowed actions: to move left or right.

Example 4.4. *In a* 1×10 *corridor like the one illustrated in Figure 4.3, where a robot can move either left or right, and* 1 *and* 10 *are the left and rightmost positions, the* $oneof(x_1, \dots, x_{10})$ *expression is an invariant encoding the possible positions of the robot.*

If initially the robot is either at 1 *or* 2 *on the left, and the goal is to reach position* 5*, then* $h_C(n_0)$ *for the root node will be* 7*: the distance from position* 1 *to position* 5*, plus distance from position* 2 *to* 5*. In this case we consider that* S *is made by two states corresponding to the two initial poses. The best action according to the classical heuristic would be to move right, towards the goal, which would lower the heuristic of the node by* 2*. We will see next that this result is different when considering the certainty heuristic* $h_K(n_0)$*.*

Certainty Heuristic h_K

The *certainty heuristic* $h_K(n)$ is defined in terms of a set of “oneof invariants” explicitly given in the problem or derived from it. A given oneof invariant is an exclusive disjunction $oneof(x_1, \dots, x_n)$ in the initial situation $\mathcal{I}$ that is maintained by the actions in the problem, i.e. after applying any action of the problem, a true oneof formula is still true.

More precisely, a $oneof(x_1, \ldots, x_m)$ expression is invariant if all pairs x_i, x_j in it are mutex for $i \neq j$, and every action a with an effect $C \rightarrow \neg x_i$, has an effect $C \rightarrow x_j$, for $1 \leq i, j \leq m$. When a $oneof(x_1, \ldots, x_m)$ expression in $\mathcal{I}$ is not invariant, an attempt is made to find a set of literals $y_1, \ldots, y_k$ that "extend" the original formula so that the expression $oneof(x_1, \ldots, x_m, y_1, \ldots, y_k)$ is true in $\mathcal{I}$ *and* invariant. To do so, we must make certain that each y_i is mutex with the other literals in the formula. The computation of these completions is simple and follows the calculation of similar invariants in classical planning [7] (Helmert, 2009).

For a "oneof invariants" $oneof(x_1, \ldots, x_n)$, the clause $(x_1 \vee \ldots \vee x_n)$ expresses that the set $\{x_i\}_{i=1}^{n}$ of (invariant) literals is *exhaustive*, meaning that at least one of them is true in every reachable state, while the set of clauses $(x_i \vee x_j)$ and $(\neg x_i \vee \neg x_j)$ capture the *mutual exclusivity* of such literals, meaning that only one of them can be true in a reachable state, for $i \neq j$. Generally, if the non–unary clauses of $\mathcal{I}$ are invariant, they can be associated to sets of multivalued variables which value is initially unknown, and the uncertainty in the problem corresponds to the uncertainty on such variables.

To illustrate the use of these oneof invariants, we can consider a version of the *gripper* problem: an object o is placed in an unknown position and has to be picked up by a mechanical arm to finally place it in a box. The problem can be encoded with a $oneof(at(o, p_1), \ldots, at(o, p_n))$ expression in $\mathcal{I}$, with the p_i fluents encoding the position. However this expression is not invariant, but can be completed into the invariant $oneof\big(at(o, c_1), \ldots, at(o, c_n), hold(o), at(o, box)\big)$: the last two propositions encode that the object o can be in the box or being hold by the arm.

The *certainty heuristic* $h_K(n)$ for a node $n = \langle \pi, S, R \rangle$ is then defined as the number of literals x_i in oneof invariants with a literal in the goal, such that $\neg x_i$ is not in R. In other words, if we take the literals x_i in the invariants as representing the values $X = X_i$ of some *multi-valued variable* X that is mentioned in the goal, $h_K(n)$ simply counts the possible values of such variables that have not yet been knocked out of the current belief. Thus, if x_i is in the goal, then at the end of any conformant plan x_i must be known, and so must the negation of the literals x_k that are mutex with x_i.

Example 4.5. *Let's consider again the 1×10 corridor illustrated in Figure 4.3. If initially the robot is either at 1 or 2 on the left, and the goal is to reach position 5, then $h_K(n_0)$ for the root node will be 2. The best action according to the certainty heuristic would be to move left, away from the goal, reducing the heuristic and the uncertainty by 1. On the other hand, we saw in example 4.4 that the best action according to the classical heuristic would be to move right, towards the goal, which however cannot be achieved if the uncertainty is not first removed.*

The certainty heuristic has relation to two heuristics used in planning that appeared to be completely orthogonal up to now. One is the *cardinality heuristic* that simply

[7] Roughly, if there is an action with effect $C \rightarrow \neg x_i$ and no effect $C \rightarrow x_j$, the candidate invariant appears to be refuted; in such a case we look for an effect $C \rightarrow y_1$, for y_1 mutex with $(x_1, \ldots, x_m)$. If so, the new candidate invariant is $oneof(x_1, \ldots, x_m, y_1)$, which is tested and extended in the same way, until no refutation is found, producing eventually the invariant $oneof(x_1, \ldots, x_m, y_1, \ldots, y_k)$. On the other hand, if the current candidate invariant is refuted and there is no mutex y_1 as above to extend it, the candidate invariant is dropped.

counts the number of states in a belief state Bertoli and Cimatti (2002). The other is the *landmark heuristic*, popularised by the classical planner LAMA(Richter et al., 2008), that counts the number of unachieved landmarks, where a landmark is an atom that is made true by all plans (Hoffmann et al., 2004). The relation to the cardinality heuristic is immediate in problems where all the uncertainty comes from a set of multi-valued variables that appear in the goal and whose initial value is initially unknown, like in our example 4.5. In such cases, the target belief states have cardinality one, and hence it makes sense to establish a preference for belief states with lower cardinality. However, although the cardinality heuristic takes the *product* of the cardinalities of the beliefs over the variables, the certainty heuristic takes the *sum*. This sum corresponds to the number of literals that must be achieved in all the plans; namely, the goal literals x_i, along with the negation of all literals x_k that are mutex with x_i. These literals are like *epistemic landmarks*, meaning that any conformant plan must achieve all these literals with certainty.[8]

Search Algorithm

The two heuristics h_C and h_K are used in the context of a multi–heuristic best first search algorithm following the classical planners FD and LAMA(Helmert, 2006; Richter et al., 2008). Multi–heuristic best first search is a variant of the well known best first search algorithm that combines several heuristic evaluators, which are in our case the classical and the certainty heuristics.

In T1, the search is implemented as a multi–queue best first search making use of three open lists that we call Q1, Q2, and Q3. Children nodes obtained using helpful actions or actions that decrease the value of the certainty heuristic are placed in the first two queues, while those obtained with other actions are placed in the last queue. Nodes in Q1 and Q3 are ordered with classical heuristic h_C, while nodes in Q2 are ordered with certainty heuristic h_K. Ties are broken in each of the queues using first the other heuristic (i.e. h_C for Q2, h_K for Q1 and Q3), and then the accumulated cost to the node. The expansions of the first two queues alternate, and every tenth iteration, the last queue is expanded. Expanding a queue means picking up the best node in the queue, checking if it is a goal node, and if not, producing all of its children.

4.6 Empirical evaluation

The conformant planner T1 involves five parts: parsing, sampling (computing S^1), search, computation of the heuristics, and verification of beliefs (for widths greater than 1). The parser is implemented on top of CONFORMANT–FF sources, the computation of the samples uses the CNF to d–DNNF compiler `c2d` (Darwiche, 2004), and the verification process is done with MINISAT-2.2 (Een and Sorensson, 2004). T1 is written in `C++`.

We compared T1 with DNF (To et al., 2009) and T_0 (Palacios and Geffner, 2009). Conformant-FF (Brafman and Hoffmann, 2004), POND (Bryce et al., 2006), and

[8] Notice that the certainty heuristic, as defined here, overestimates the true count by the number of oneof invariants with a literal in the goal, yet this difference is a constant that has no effect in the search.

MBP (Bertoli et al., 2006) perform well on several domains but have smaller coverage, so we stuck to the former planners for our tests. We wanted to test also the recent CNF planner (To et al., 2010), but the author have not made it available at the time we are writing. For DNF preprocessing we used SWI PROLOG rather than the proprietary SICSTUS PROLOG used by the authors. T_0 has been ran with classical planner FF (Hoffmann and Nebel, 2001).

The results of the comparison are shown in Table 4.1. The problems are from past IPC competitions and various planner's distributions. For each of the 17 domains considered, three instances are shown, from easy to hard, when at least one planner solved them. The experiments have been executed on a cluster of multi–core, multi–CPU machines with a clock speed of 2.33GHz running Linux, with 2 hours and 2GB for time and memory outs.

T1 solves 44 problems out of 47 (93%), while T_0 solves 38 (80%), and DNF 35 (74%). In general, however, T_0 is fastest, solving most of the (translated problems) with the effective EHC search in FF (such instances appear with a number of expanded nodes $x + 0$, meaning that x nodes where expanded by EHC and 0 by the greedy best-first search). DNF does comparatively best on the "corner version" of the Square and Cube domains (Corners-Cube and Corners-Square), where neither of the two heuristics used in T1 appears to be useful (this can be seen in the number of expanded nodes). T_0 doesn't well either in this domain, as the heuristic used by FF over the K_1 translation is similar to one of the heuristics used in T1 (the classical heuristic h_C). DNF solves these instances with much less expansions, meaning that the heuristic it's using is the most informative for these domains. In most other domains, both T1 and T_0 expand much less nodes than DNF, which nonetheless, expands nodes very fast. For example, in the second Bomb problem, DNF expands 175k nodes in 23 secs, while T_0 and T1expand 1.1k nodes in 3.5 secs, and 140 nodes in 21 secs, respectively. This means, that in this instance, DNF, T_0, and T1 expand roughly 7k nodes per second, 300 nodes per second, and 70 nodes per second respectively. In general, T1 expands fewer nodes than T_0, but not necessarily faster. Part of the explanation for the slow node expansion rate of T1 vs. T_0 is what expansion means in each of the two planners. In T_0, while FF is running in EHC mode, an expansion is the application of the helpful actions only. This search is incomplete, but as it can be seen in the table, it's often quite effective. In T1, on the other hand, an expansion is a full expansion: all the children nodes are generated, and if they are "helpful" they are placed in the right queue. This, however, is a problem in instances with high branching factors where the number of nodes *generated* can be much larger than the number of nodes *expanded.* This is the reason that T1 runs out of memory in domains such as Bomb and Logistics, where it expands few nodes that lead however to the generation of hundred of thousands of nodes. This problem could avoided by delaying the generation and evaluation of such nodes. Such techniques are used in recent classical planners such as FD and LAMA.

Table 4.2 compares the performance of T1 when restricted to use just one heuristic, either the classical heuristic h_C or the certainty heuristic h_K, as opposed to the two heuristics combined. When running T1 with a single heuristic, the "helpful" queue corresponding to the other heuristic is removed. As it can be seen, there are some domains where the classical heuristic is better than the certainty heuristic, and other domains, where the opposite is true. At the same time, the combination in T1 seems

	T_0			DNF			T1			
Problem	T	L	E	T	L	E	T	L	E	$\|S_0^1\|$ / $\|b_0\|$
1-Dispose-8-1	**79**	1316	119+0	OM	–	–	142	**560**	3.6k	64/64
Blocks-1*	U	–	7+872	0.1	7	5	**0.05**	**6**	23	2/2
Blocks-2*	0.2	23	86+0	**0.1**	38	20	0.3	**20**	72	6/6
Blocks-3*	71	**80**	5.3k+0	**4.9**	307	125	TO	–	6.3k	12/125
Bomb-b100-t10	7.8	290	4.4k+0	**2.6**	**190**	17k	4.8	**190**	190	$101/2^{100}$
Bomb-b100-t60	**3.5**	**140**	1.1k+0	23	**140**	175k	21	**140**	140	$101/2^{100}$
Bomb-b100-t100	**7.0**	**100**	201+0	50	**100**	348k	OM	–	42	$101/2^{100}$
Coins-18	**0.1**	**97**	679+0	1.0	100	2.4k	2.8	191	635	$10/2{\cdot}10^6$
Coins-17	**0.1**	**96**	382+0	1.0	110	3.1k	1.0	257	1.1k	$10/2{\cdot}10^6$
Coins-21	OM	–	–	OM	–	–	**193**	**904**	11.9k	$10/1{\cdot}10^{16}$
Comm-20	**0.3**	**278**	1.1k+0	171.6	296	1.5k	43	295	616	41 / 2^{20}
Comm-25	**1.2**	**453**	1.8k+0	1857	501	2.5k	339	478	1k	$66/2^{25}$
Comm-30	**2.0**	**593**	2.3k+0	OM	–	–	30	1090	1.3k	$86/2^{30}$
Cube-67	33	**294**	7.6k+0	1895	2019	71k	**10**	303	308	$67/67^3$
Cube-91	OM	–	–	2200	2271	73k	**31**	**408**	408	$91/91^3$
Cube-139	OM	–	–	TO	–	–	**151**	**622**	623	$139/139^3$
Corners-Cube-15	**0.6**	147	5.8k+10k	0.7	**117**	1.4k	2.7	159	5.7k	2/8
Corners-Cube-20	2.2	258	13k+29k	**1.8**	**217**	2511	8.6	248	12k	2/8
Corners-Cube-55	OM	–	–	**18.6**	**806**	10k	836	1644	182k	2/8
Corners-Sqre-36	1.0	412	2.6k+16k	**0.7**	**138**	451	12	412	8.7k	2/4
Corners-Sqre-72	20	1474	10k+141k	**12**	**615**	3.3k	241	1474	44k	2/4
Corners-Sqre-120	190	3898	29k+681k	**75**	**1870**	10k	2694	4014	136k	2/4
Dispose-12-1	**55**	1274	267k+0	5590	**330**	13k	488	786	2.5k	144/144
Dispose-12-2	2037	1437	3922k+0	5810	**567**	18k	**1690**	1196	4.5k	$146/144^2$
Dispose-12-3	OM	–	–	6305	**1131**	34k	**6257**	1385	5.6k	$146/144^3$
Logistics-4-3-3	**0.01**	**24**	53+0	7.5	160	16k	0.1	41	96	$8/4^3$
Logistics-2-10-10	**0.4**	**84**	414+0	OM	–	–	47	94	859	$3/2^{10}$
Logistics-4-10-10	**0.7**	**125**	774+0	OM	–	–	121	167	1.7k	$5/4^{10}$
Look-Grab-4-2-1*	U	–	3+16	OM	–	–	**1.8**	**42**	138	16/256
Look-Grab-8-1-1	59	242	6.4k+44	OM	–	–	**16**	**106**	377	64/64
Look-Grab-8-3-2*	OM	–	–	OM	–	–	**3957**	**56**	3.1k	$20/64^3$
Push-To-8-1	**83**	464	74k+0	134	**163**	3.9k	271	538	6.5k	64/64
Push-To-8-2	817	423	131k+0	195	**162**	124k	**53**	336	1.5k	$66/64^2$
Push-To-8-3	1213	597	132k+0	OM	–	–	**59**	**300**	1.3k	$66/64^3$
Raos-Keys-2*	**0.02**	**16**	22+0	0.5	22	70	0.7	21	130	2/4
Ring-10	**0.1**	55	530+0	1546	39	107	0.6	**41**	484	10/590k
Ring-20	**2.0**	95	2.3k+0	OM	–	–	29	**91**	3.2k	$20/6{\cdot}10^{10}$
Ring-30	**24**	**121**	8.1k+0	OM	–	–	393	133	11.9k	$30/6{\cdot}10^{15}$
Safe-30	**0.06**	**30**	30+0	0.2	**30**	465	0.2	**30**	30	30/30
Safe-70	**0.5**	**70**	70+0	2	**70**	2.5k	3.0	**70**	70	70/70
Safe-100	**1.0**	**100**	100+0	5	**100**	5k	13	**100**	100	100/100
Square-24	0.4	**69**	172+0	3	351	2.6k	**0.2**	70	71	$24/24^2$
Square-92	36	**273**	1413+0	1236	2444	36k	**14**	274	275	$92/92^2$
Square-120	OM	–	–	2376	2813	45k	**36**	**358**	358	$120/120^2$
Uts-k 30	**4.0**	**89**	92+0	8.5	101	13k	8.3	112	691	$30/2^{30}$
Uts-k 40	**13.4**	**119**	122+0	26.5	136	31k	42	143	1.1k	$40/2^{40}$
Uts-k 50	**33.8**	**149**	152+0	63.9	171	61k	OM	–	–	$50/2^{50}$
#Solved/#Total	**38/47**			**35/47**			**44/47**			

Table 4.1: Performance of T_0, DNF and T1 over 47 conformant problems: T is total time, L is plan length, and E is number of expanded nodes

		h_C				h_K				T1			
Domain	I	S	T	E	L	S	T	E	L	S	T	E	L
Bomb	9	7	71	4k	101	7	11	773	101	**8**	2	100	101
Coins	9	**8**	1	888	78	**8**	8	7k	74	**8**	3	425	166
Comm	9	**9**	2	1k	176	**9**	21	30k	175	**9**	5	440	214
Square(Ctr)	10	4	18	5k	186	**10**	0.2	224	44	**10**	0.1	43	44
Square(Cor)	11	10	604	212k	659	**11**	51	81k	119	**11**	100	18k	661
Cube(Ctr)	12	6	84	32k	188	10	1	890	61	**12**	0.1	61	58
Cube(Cor)	11	8	92	219k	271	10	4	26k	88	**11**	12	15k	269
Dispose	11	7	664	8k	349	**9**	57	2k	190	8	134	1k	491
Logistics	4	2	0.2	546	30	2	544	1613k	30	**4**	0.1	554	78
Look-Grab	6	**6**	0.1	96	10	3	41	92k	7	**6**	0.1	20	11
Push-To	8	**6**	657	21k	247	**6**	238	12k	116	**6**	83	1k	237
Ring	7	6	1	1k	17	5	571	58k	17	**8**	0.2	214	31
Safe	5	**5**	0.05	40	10	1	5	10k	10	**5**	0.04	9	10
UTS-k	15	**15**	0.06	26	7	2	0.05	154	7	13	0.04	10	9
Coverage	127	99				93				119			

Table 4.2: Comparison of T1 with one heuristic (either h_C or h_K) and the two heuristics combined. I is number of instances, S is number of instances solved; T, E, and L are avg. time, avg. number of expanded nodes, and avg plan length, respectively Averages taken over instances solved by the three configurations.

to get the best of both, in certain cases solving instances that cannot be solved with one of the heuristics alone. This doesn't mean, however, that the synergies between the two heuristics cannot be exploited further.

4.7 Discussion

The experimental results show that T1 is very competitive with state-of-the-art conformant planners in number of problems solved and quality of solutions. Additional progress on scalability seems feasible by exploiting further the synergies between the two heuristics, and by dealing in a more effective manner with the huge number of nodes that are generated in problems with large branching factors.

The individual width of precondition and goal literals, as opposed to the width of the problem given by the maximum of such width, could be used to avoid some verifications and, in principle, to simplify the detection of duplicate nodes.

Impact of the representation

The way to represent beliefs has an impact on the belief tracking and update, and consequently, on the performances of the planner.

The OBDD encoding of belief states, of which d-DNNF is a superset, is implemented in MBP and KACMBP planners (Bertoli et al., 2001; Bertoli and Cimatti, 2002). d-DNNFis more space efficient (Darwiche, 2001b) the OBDD, and provides many useful functions for planning that can be computed in linear time, like model counting.

Petrick and Bacchus (2002) represent beliefs by formulæ, and actions are modeled as updates to these sets of formulæ. However a set of formulæ is more compact than a

BDD, but this representation compiles away the single states information from the beliefs, and thus some planning problems that require reasoning about single states cannot be modelled by this approach although they can be handled with BDDs.

The planners DNF and CNF (To et al., 2009, 2010) use homonymous representations. Both appear to be adequate, but some difference are evidenced by empirical results: the CNF encoding of belief states is more compact than DNF encoding in the general case, but the CNF compilation and update has a cost which is paid by a time overhead (To et al., 2011a). On the other hand, the DNF encoding of disjunctive formulæ is larger than the CNF equivalent, but the transition function if exponential only in the number of unknown antecedents of conditional effects for DNF, while the transition function for CNF is exponential also in the number of clauses representing the (belief) state in the CNF encoding (To et al., 2011a).

Heuristics issues

Generally the two heuristics combined together, in an alternation mechanism, appear to work better than alone. This is shown in Table 4.6. However, in certain domains better solutions are found using either the cardinality heuristic alone or the classical heuristic alone. When does it happen? Consider a classical domain, where solving the uncertainty on $\mathcal{I}$ in not necessary, or a domain where the goal can be reached regardless the size of the belief state: in those cases the classical heuristic alone should guide the search efficiently. En example of such a case is illustrated in Figure 4.4. In the domain in the figure, with similar characteristics as the problem illustrated in Figure 4.2 but with the difference that the cardinality does not affect a solution. This domain can indeed be solved by T1 without considering the certainty heuristic; a solution would be: {East, South} without the need of localising first in the corner as in the former example, so alternating between the two heuristics is simply useless.

This example shows another aspect of the certainty heuristic that improves on the cardinality heuristic: here, the certainty heuristic would never have a value < 4, as the atoms in the goal are considered. The cardinality heuristic, however, would simply aim at reducing the size of the belief state possibly moving the agent away from the goal, which potentially ends into a longer search and a longer plan. Driving the search in order to reach the same cardinality as the goal, and no more, is what the certainty heuristic does in these cases. A very similar mechanism is implemented in in the KACMBP (Bertoli and Cimatti, 2002) planner, making advantage of the OBDD representation to reduce the uncertainty of the agent's belief to the same size of the goal belief state.

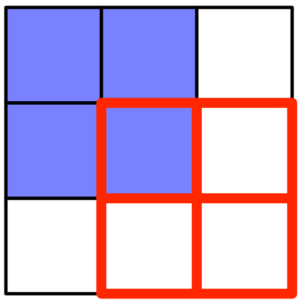


Figure 4.4: Square room domain. Initial belief is the 4 top left cells, the goal is in the lower right outlined cells.

Now, if we consider the other case of a domain where the goal is to reach either end of a corridor, and the initial position in unknown, i.e. the agent can be in any pose in the corridor as shown in Figure 4.5. Here, reaching an end of the corridor means removing the ambiguity about the position of the agent, so only the certainty heuristic is relevant to the solution of the problem: the classical heuristic wouldn't bring additional information.

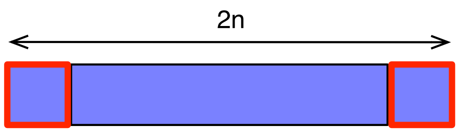


Figure 4.5: Corridor domain of length $2n$. The initial position is unknown, goal is reaching either end of the corridor. Actions: move left or move right.

In cases like the ones we just saw, alternating between the two heuristics is simply of no use, as one of them does not add helpful information to solve the problem. But these are not the only case of inefficiency: occurrences where the values of one heuristic decreases where the other rises, and vice–versa, have negative repercussions on the plan quality. This happens in the planning domain shown in Figure 4.6: a corridor domain where a robot has to reach the goal is at one end, and with the initial possible positions of the robot at the other end. The closest position where to reduce the cardinality of the initial belief state is in the opposite direction from the goal. Here the two heuristics drive the search literally in opposite directions. Reducing quickly the certainty heuristic, would end up going all way to the left, and only then moving through the goal, in the right direction. This plan is inefficient as it is also possible to remove the uncertainty about the position on the right, in the goal cell. Moreover, depending on the initial situation, alternating between the two heuristics could end in this example in alternating from one direction to the other. Here a *boosting*

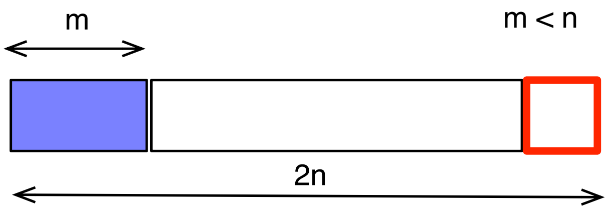


Figure 4.6: Corridor domain of length $2n$. The initial position is unknown and the goal is to reach the right end of the corridor. Actions: move left or move right.

mechanism could be advantageous; increasing the expansion rate of the queue that improves most the global heuristic goal distances. Another possibility would be the use of a lookahead mechansim, identifying when a lower value of a heuristic will also lead to lower values for the other heuristic functions. Such technique would indirectly avoid search plateaux, a situation that arises when goals are in “conflict”, meaning that approaching one presupposes to move away from the others, from an heuristic point of view. This can be done exploiting techniques from classical planning, where both lookahead in search (Vidal, 2004), or automated generation of *probes*[9] (Langley, 1992; Nakhost and Müller, 2009; Lipovetzky and Geffner, 2011) have been employed to solve the problems with more efficiency.

4.8 Summary

Conformant planning is the task of finding a solution plan in problems where there is partial information on the domain and where no sensing is available. Conformant planning can be formulated as a path-finding problem in belief space, where the main

[9]A probe is an action sequence computed from a state, which can be done greedily or randomly, in the former case the aim is achieving a serialisation of the problem subgoals, dynamically generated along with the probe.

challenges come from the heuristics to guide the search, and the representation and update of the beliefs.

The translation-based approach to planning under uncertainty is elegant and exhibits good performances in relation to approaches that explicitly search in belief space. While competitive with the state-of-the-art methods, translation-based approaches shock against the difficulty that complete translations may require exponential time and space, since incomplete translations may result in unsolvable problems.

We presented a translation-based planner for deterministic conformant planning that tries to combine the two approaches: the flexibility of planners that search explicitly in belief space, with the heuristics and beliefs that arise from translations. For this we have formulated a novel translation K_S^i that is always tractable and complete, and sound for problems with width bounded by i. The T1 planner uses the set of samples that result from the K_S^1 translation in the context of a belief search planner. We take advantage of the samples not for solving the problem with a classical planner, but for deriving heuristics and computing belief states. We incorporated a heuristic derived from the "oneof invariants" in the problem, related to the cardinality and the landmarks heuristics, and using a multi-queue best first search algorithm patterned after the classical planners FD and LAMA. The experimental results show that T1 is competitive with the state-of-the-art conformant planners in number of problems solved, and quality of solutions.

Non-determinism is a feature that seems to be ignored by the majority of conformant planners, even if it's one of the principal sources of uncertainty for planning in realistic environments. Translation-based approaches can handle actions with non-deterministic effects. Non-deterministic effects can be compiled away by introducing hidden conditions afresh each time a non-deterministic action is executed. This method mean that many "copies" of the action must be added to the problem. The way these actions are introduced, and the limitation on the number of their copies, will determine the properties of the translation, and in particular its completeness. These transformations are the subject of the next chapter.

Chapter **5**

Translations for Non-Deterministic Conformant Planning

In the previous chapters, we saw that conformant planning problems where the actions have *deterministic effects* can be translated into classical problems that can be then solved by off-the-shelf classical planners. This is the case of the planner T_0, and in part of the approach behind the planner T1. When planning problems have actions with *non-deterministic effects*, these have to be adapted to the translations, in order to find a solution. Moreover, the non-deterministic effects can appear as many "copies" of the action, which will determine the properties of the translation, in particular its completeness.

In this chapter we will introduce different translations to handle non-determinism in the effects of the actions. Very few planners, however, deal with such an important feature of planning problems modeling, the most successful of them are the ones developed in Trento; we named MBP and KACMBP (Bertoli et al., 2001; Bertoli and Cimatti, 2002), to which we compare to.

Results from this chapter have been published in *Compiling Away Uncertainty in Non-Deterministic Conformant Planning Problems*, by A. Albore, H. Palacios, and H. Geffner, in 21st European Conference on Artificial Intelligence (ECAI-10), Lisbon, Portugal, 2010 (Albore et al., 2010).

5.1 Motivation

Non-determinism is an important feature of planning for real world. Exogenous events can be common in a realistic domain, while the outcome of a performed action is not always certain. Nevertheless, this important feature seems to have been ignored by the majority of existent planners.

In this chapter, we extend the formulation for deterministic conformant problems to non-deterministic conformant planning. We will start with the well known observation that non-deterministic effects can be eliminated by using hidden artificial conditions that must be introduced afresh each time a non-deterministic action is applied (cf. section 2.2). This observation leads to an original translation of conformant problems with non-deterministic effects into conformant planning problems with deterministic effects, which are themselves related to classical planning problems. However, this translation has to be recomputed as the search for plans proceeds.

We will then introduce other translations that provide sound –but not necessary complete– solutions to the original problem by solving a relaxed conformant deterministic problem. These translations, while incomplete, appear to be quite effective and result in classical planning problems that need to be solved only once.

5.2 Translations for Non-Deterministic actions

In chapter 2 we introduced non-deterministic conformant planning (cf. definition 2.2). Conformant planning problems P are represented as tuples of the form $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where $\mathcal{F}$ stands for the fluent symbols in the problem, $\mathcal{I}$ is a set of clauses over $\mathcal{F}$ defining the initial situation, $\mathcal{A}$ stands for a set of (ground) operators or actions a, and $\mathcal{G}$ is a set of literals over $\mathcal{F}$ defining the goal. Every action a has a precondition pre(a) given by a set of fluent literals, and a set of conditional effects or rules

$$a : C \to L_1 \mid L_2 \mid \cdots \mid L_n, \quad n \geq 1, \tag{5.1}$$

where C and L_i stand for sets (conjunctions) of literals, and C can be empty. The effect is *deterministic* if $n = 1$, and *non-deterministic* otherwise. When convenient, we take a deterministic effect $C \to C'$ as the set of effects $C \to L$ for each $L \in C'$.

The semantics of a non-deterministic problem P is defined in terms of the state trajectories that are possible. A state s, as we saw, is a set of literals representing a truth assignment over the fluents in $\mathcal{F}$. An action $a \in \mathcal{A}$ is applicable in s if $pre(a) \subseteq s$, and s' is a *possible successor state* of s if for each of the $n_i(a)$ conditional effects associated with the action a

$$C^i \to L_1^i \mid \cdots \mid L_{n_i(a)}^i \tag{5.2}$$

we have s', single successor state of s given an action a' that is like a but with the *deterministic* conditional effects $C^i \to L_{f(i)}^i$, where $1 \leq f(i) \leq n_i(a)$ is a function that selects one effect $L_{f(i)}^i$ from the set of possible effects. We assume that this successor state is always well defined, and hence, that if two different outcomes L_k^i and L_l^i are complementary, the bodies of these effects can't be reached jointly from any possible initial state. The state trajectories $[s_0, \ldots, s_{n+1}]$ that are possible given an action sequence $[a_0, \ldots, a_n]$ are the ones that start in an initial state s_0 and that include the states s_{i+1}, which are possible successors of s_i given an action a_i.

An action sequence $[a_0, \ldots, a_n]$ is a *conformant plan* for P if each action a_i, with $1 \leq i \leq n$, is applicable in the state s_i of all the state trajectories $[s_0, \ldots, s_i]$ that are possible given the preceding action sequence $[a_0, \ldots, a_{i-1}]$, and s_{n+1} is a goal state. Alternatively, if b_0 is the set of initial states deemed possible, and b_{i+1} is the set of states that are possible given an action a_i applicable in each state in b_i, then the sequence $[a_0, \ldots, a_n]$ is a conformant plan for P if it maps the initial set b_0 to a final set b_{n+1} of goal states.

Deterministic Relaxation

We aim at applying the $K_{T,M}(P)$ translation to non-deterministic conformant planning, but the translations we discussed in chapter 3 are pertinent to deterministic conformant planning and can't be applied directly to the cases we are treating here. We present so a deterministic relaxation that introduces new hidden variables for the problem to encode non-deterministic effects, with the limitation that, one executed, the action will always have a fixed outcome.

To compile away non-deterministic effects, one possibility is to take advantage of a well known transformation that makes use of auxiliary *hidden variables* h^i_j. In general, an hidden variable is a fluent p such that neither p not $\neg p$ are known initially. Hidden variables are used here to map each one of the *non-deterministic* effects in eq. (5.2) to $n_i(a)$ *deterministic* effects

$$C^i \;\wedge\; h^i_j(a) \;\rightarrow\; L^i_j \tag{5.3}$$

with $1 \leq j \leq n_i(a)$. The following "oneof" expressions are then added to the initial belief state $\mathcal{I}$:

$$\textit{oneof}\,\big(h^i_1(a), \ldots, h^i_{n_i}(a)\big) \tag{5.4}$$

In this transformation, the uncertainty in the state transitions becomes converted in clauses with the $h^i_j(a)$ literals in the initial situation.

Definition 5.1 (Deterministic transformation P_d)**.** *Let be $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ a non-deterministic conformant problem, then the transformed problem $P_d = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is a* deterministic conformant planning problem*, where*

- $\mathcal{F}' = \mathcal{F} \cup \{\, h^i_j(a) \mid$ *for each non-deterministic outcome j of conditional effect i of action $a \in \mathcal{A}$* $\}$,
- $\mathcal{A}' = \mathcal{A}$ *with each conditional effect of action a: $C^i \rightarrow L^i_1 \mid \cdots \mid L^i_{n_i(a)}$ replaced by the $n_i(a)$ rules: $C^i \;\wedge\; h^i_j(a) \;\rightarrow\; L^i_j$*
- $\mathcal{I}' = \mathcal{I} \;\cup\;$ *oneof* $\big(h^i_1(a), \ldots, h^i_{n_i}(a)\big)$, *for each action $a \in \mathcal{A}$, with conditional effect i,*
- $\mathcal{G}' = \mathcal{G}$

By changing the non-deterministic rules in deterministic rules and introducing new hidden conditions following eq. (5.4), the original non-deterministic conformant problem P is now translated in a deterministic conformant problem P_d. The new problem P_d is equivalent to P when non-deterministic actions are applied only once. In fact, for a non-deterministic action a, the *oneof* constraint in eq. (5.4) can be seen as a multivalued variable which value is initially hidden and that informs on the effect of the action a. Once its value unveiled, the action a is bound a specific deterministic effect.

Proposition 5.2. *Let π be an action sequence that involves each non-deterministic action from P at most once. Then π is a plan for the* non-deterministic *conformant problem P if π is a plan for the* deterministic *conformant plan P_d.*

The difference between P and P_d is that the hidden h conditions in P_d establish correlations among the possible outcomes of the same action during the execution of a plan. When non-deterministic actions are applied more than once, P_d would not consider other effects than the one corresponding to the hidden condition h, while a different effect can take place in the original problem P. In particular, the possible outcome of an action a in P can be L^i_j the first time, and L^i_l the second time, with $l \neq j$; this is not possible in the deterministic relaxation P_d.

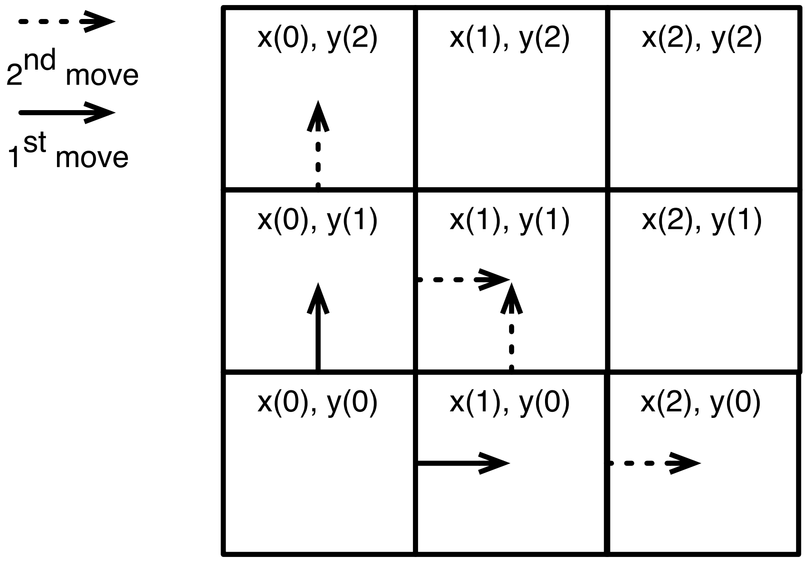


Figure 5.1: Grid from example 5.1. The first move is shown in continuous line, the second move in dotted line.

Example 5.1. *Let's consider a moving agent in a grid. The action "move" has two non-deterministic effects in P:*

$$x(i) \wedge y(j) \;\rightarrow\; x(i+1) \wedge \neg x(i) \;\Big|\; y(j+1) \wedge \neg y(j)$$

meaning that the movement can be either in vertical or in horizontal.

This action "move" is translated in a problem P_d as follows:

$$x(i) \wedge y(j) \wedge h^1_1(move) \;\rightarrow\; x(i+1) \wedge \neg x(i)$$
$$x(i) \wedge y(j) \wedge h^1_2(move) \;\rightarrow\; y(j+1) \wedge \neg y(j)$$

where the two new fluents $h^1_1(move)$ and $h^1_2(move)$ are added, along with in the initial situation a clause

$$oneof\big(h^1_1(move), h^1_2(move)\big)$$

A sequence of two move actions starting at $\big(x(0), y(0)\big)$ may end in P in one of three possible locations: $\big(x(2), y(0)\big)$, $\big(x(0), y(2)\big)$, and $\big(x(1), y(1)\big)$.

On the other hand, only the first two locations are possibly reachable in P_d: the first follows from states where the first hidden condition $h^1_1(move)$ is true; the second, from states where the second hidden condition $h^1_2(move)$ is true.

Even if not all the plans for P are plans for P_d, the transformation of P to P_d can be used to obtain an incomplete non-deterministic conformant planner in a simple manner. A traslation $K = K_{T,M}$ can be applied on the deterministic conformant problem P_d. Then, a *classical* planner is called over the translation $K(P_d)$, and if no non-deterministic action from P appears twice in the plan returned, from the soundness of the translation and Proposition Theorem 5.2, the plan with the merge actions removed must be a plan for P.

This simple translation allows only to represent non-deterministic actions that are executed once. It is of course possible to extend the above deterministic translation to obtain problems where non-deterministic actions can be applied more than once. We introduce then a generalization of the deterministic relaxation P_d that works on a variant of P that we call P^m. The conformant problem P^m is exactly like P but with each non-deterministic action a in P copied m times, with different names $a^1, \ldots, a^m$. These copies make no difference to P, as the problems P and P^m are equivalent, but make a difference in the relaxations P_d and P_d^m, of P and P^m respectively. The two relaxations P_d and P_d^m are not equivalent: while the relaxation P_d can capture plans for P that include each non-deterministic action of P *at most once*, the relaxation P_d^m can capture plans for P where each non-deterministic action is done *at most m times.* Indeed, for a sufficiently large m, P_d^m will necessarily account for a plan that solves P, and even for a plan that solves P optimally. Useless to say, the relaxation P_d is P_d^m with the parameter m set to 1.

The relaxation P_d^m we just saw can be slightly modified so that the translation $K(P_d^m)$ generates only sound plans. The translation $K(P_d^m)$ would produce plans where each non-deterministic action is applied up to m times and no more.

The change is very simple:

1. we create new fluents $\mathit{blocked}(a^k)$, for each copy a^k of a non-deterministic action a in P_d^m (with $1 \leq k \leq m$),
2. we set all the new atoms true initially, except for $\mathit{blocked}(a^1)$, that is false initially,
3. we add the literal $\neg\mathit{blocked}(a^k)$ to the precondition of the action a^j, and add the literals $\neg\mathit{blocked}(a^{k+1})$ and $\mathit{blocked}(a^k)$ to its effects.

The objective of using the *blocked* fluents in the copy of the actions is to enable the n^{th} copy of the action only after executing the action $n-1$ times.

Definition 5.3 (Deterministic relaxation P_d^m). *Let be $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ a non-deterministic conformant problem, and a positive parameter m. The deterministic relaxation $P_d^m = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is a* deterministic conformant planning problem*, where:*

- $\mathcal{F}' = \mathcal{F} \cup \{\, h_j^i(a^k) \mid$ *for each possible outcome j of the i-th conditional effect of the action $a \in \mathcal{A}$*$\}$*, with* $0 < k \leq m$.
- $\mathcal{A}' = \mathcal{A}$ *with for each action a^k having*
  - *each precondition L replaced by* $\big(L \wedge \neg\mathit{blocked}(a^k)\big)$*, and*
  - *each conditional effect $C^i \rightarrow L_1^i \mid \cdots \mid L_{n_i(a)}^i$ replaced by the $n_i(a)$ rules:* $C^i \wedge h_j^i(a^k) \rightarrow L_j^i \wedge \mathit{blocked}(a^k) \wedge \neg\mathit{blocked}(a^{k+1})$

- $\mathcal{I}' = \mathcal{I} \cup$ *oneof* $\big(h^i_1(a), \ldots, h^i_{n_i}(a)\big) \cup$ *blocked*(a^k), *for each action* $a \in \mathcal{A}$, *with conditional effect* i, *and for* $2 \leq k \leq m$.
- $\mathcal{G}' = \mathcal{G}$

The naive scheme for eliminating non-determinism can be used to solve a non-deterministic problem P by applying any of the $K_{T,M}$ translations to the deterministic problem P^m_d, for a parameter m that is large enough. The difficulty with this method, however, is that many problems involve the execution of non-deterministic actions many times, and moreover, it is not easy to come up with an upper bound m in general. Following that idea, it is possible to implement a planner that *incrementally* searches for solutions of P by incrementing the value of the parameter m until a solution is found. Such a planner would be sound, as a relaxation with a sufficient number of copies of the actions would have a solution for the original problem P. Another benefit is that the bound m does not have to be known. This will be the topic of the next section.

5.3 The K-replanner

This approach, for solving non-deterministic problems P, makes use of the ideas behind the P^m_d relaxation without using it explicitly. It can be understood as a lazy implementation where the new copies a^k of the non-deterministic actions are added incrementally, along with its associated fluents $h^i_j(a^k)$ and *blocked*(a^k), and their corresponding initial conditions.

The K-Replanner is a *lazy* implementation of a planner based on the classical translation K of the deterministic relaxation P^m_d of P, for an arbitrary m, where the last copy a^k of each non-deterministic action a does not get blocked and can be used more than once. It exploits the property that if such actions a^k are not used more than once in the classical plan π returned for $K(P^m_d)$, then π is a conformant plan for P (once the merge actions are dropped). On the other hand, a plan $\pi = [a_0, a_1, \ldots, a_n]$ returned for $K(P^m_d)$ such that $\pi_{i+1} = [a_0, \ldots, a_{i+1}]$ is the first prefix that violates this condition, constituting a *flaw*, has to be repaired. This flaw is "repaired" by preserving the "flawless" prefix π_i and merging it with a plan tail π'. The plan tail π' is obtained recursively from the resulting state s_{i+1} before the flaw and over an encoding that is like $K(P^m_d)$ except that the "faulty" repeated action copy a^k is split in two: 1) the action a^k itself, that is blocked in s_{i+1}, and 2) a new copy a^{k+1} with its own fresh hidden h variables, that is not. The resulting planning algorithm is *dynamic* in the sense that each time a classical plan with a repeated non-deterministic action is returned, a plan tail is computed (recursively) over a slightly different classical problem that includes more fluents (the new hidden h variables and the *blocked* fluents), more initial conditions (involving the new *blocked* fluents and the clauses coming from the *one-of* over the h-fluents expressions), and more actions (the merges resulting from the new *one-of* expressions). If we call P_i the deterministic relaxation of P before the flaw, and P_{i+1} the deterministic relaxation after the flaw, the classical problems before and after the flaw are $K(P_i)$ and $K(P_{i+1})$ respectively. Notice that for all translation schema $K = K_{T,M}$, the translation $K(P_{i+1})$ can be computed incrementally with just minor modifications from the translation $K(P_i)$ of the previous deterministic relaxation P_i.

Example 5.2. *Let's consider the example 5.1. The hidden fluents bound to the two non-deterministic effects of the move action are h_0 and h_1. The K-replanner starts from a problem P_d^m where the parameter m is fixed to 1. So, from definition 5.3, the move action has in P_d^1 two deterministic conditional effects:*

$$\begin{aligned} &\mathit{move}\big(x(i), y(j)\big) : \\ &\qquad x(i) \wedge y(j) \wedge h_0 \rightarrow x(i+1) \wedge \neg x(i) \\ &\qquad x(i) \wedge y(j) \wedge h_1 \rightarrow y(j+1) \wedge \neg y(j) \end{aligned}$$

Where the initial situation includes the constraints on the hidden fluents:

$$\mathcal{I} = \big\{(x(0), y(0), \mathit{oneof}(h_0, h_1)\big\}$$

A second execution of the action move would increment the parameter m; so that the transformation $P_2 = P_d^2$ of the original problem would include a second copy of the action and new hidden variables. The initial situation of P_2 would include the blocked constraint on the copy ($\mathit{blocked}(m^1)$), and the conditions on the new hidden variables h_0^1 and h_1^1:

$$\mathcal{I} = \big\{(x(0)), (y(0)), \mathit{oneof}(h_0^1, h_1^1), \mathit{oneof}(h_0, h_1), (\mathit{blocked}(m^1))\big\}$$

Then the action move is then present in two copies, the first of which is unblocked initially and that corresponds to the action present in P_1, but which execution is limited to one time:

$$\begin{aligned} &\mathit{move}^0\big(x(i), y(j)\big) : \\ &\qquad \text{precondition}: \neg \mathit{blocked}(m^0) \\ &\qquad \text{effect}: \\ &x(i) \wedge y(j) \wedge h_0 \rightarrow x(i+1) \wedge \neg x(i) \wedge \mathit{blocked}(m^0) \wedge \neg \mathit{blocked}(m^1) \\ &x(i) \wedge y(j) \wedge h_1 \rightarrow y(j+1) \wedge \neg y(j) \wedge \mathit{blocked}(m^0) \wedge \neg \mathit{blocked}(m^1) \end{aligned}$$

The action copy move^1 is unblocked only after the execution of move^0:

$$\begin{aligned} &\mathit{move}^1\big(x(i), y(j)\big) : \\ &\qquad \text{precondition}: \neg \mathit{blocked}(m^1) \\ &\qquad \text{effect}: x(i) \wedge y(j) \wedge h_0^1 \rightarrow x(i+1) \wedge \neg x(i) \\ &\qquad \qquad x(i) \wedge y(j) \wedge h_1^1 \rightarrow y(j+1) \wedge \neg y(j) \end{aligned}$$

If move^1 is executed twice, the planner is called again on a new version P_3 of the problem P.

We call this incremental planning schema able to handle non-deterministic actions and that starting with $P_1 = P_d$, the *K-replanner*. The K-replanner is incomplete even if the translation K is complete for deterministic problems. The incompleteness is a result of the commitment to the “flawless” plan prefixes that are extended after each iteration, and which may render a solvable problem P, unsolvable. The schema, however, is sound:

Proposition 5.4. *If the K-replanner returns an action sequence π, then π with the merge actions removed is a plan for the non-deterministic conformant problem P.*

However, the incompleteness is more a result deriving from the search algorithm that doesn't backtrack than from the translation itself, if we put aside the limit on the number of copies of the non-deterministic actions.

The K-replanner requires the classical planner to be called more than once: the algorithm starts with a deterministic conformant problem P_0 that is similar to the relaxation P^m above for $m = 1$, the empty plan prefix π_0, and the initial state s_0 of $K(P_0)$, and if the obtained plan π contains a "flaw", a classical planner is called over the classical problem $K(P_n)$ obtained from the problem P_n that extends P_{n-1} with an extra copy of the non-deterministic action repeated in π, with initial situation s_n that is the state achieved in $K(P_{n-1})$ just before the "flaw". The resulting conformant plan is the concatenation of all the action sequences obtained at each step, and it is a solution plan for P (once the merge actions are dropped). The search terminates with failure if the classical problem $K(P_{n+1})$ from the state s_{n+1} is unsolvable.

The planning schemas that we will introduce next are simpler and require calling a classical planner only once. The classical plan returned is a solution to the original non-deterministic conformant problem: both schemas are thus sound, and while neither one is complete, they turn out to be more effective than the K-replanner.

5.4 The K-Reset Planner

The *K-reset* planner uses the translation $K(P_d)$ of the deterministic relaxation P_d extended with the *blocked* fluents that prevent a non-deterministic action from being applied more than once. The classical encoding $K(P_d)$ is extended with reset actions for each non-deterministic action a, denoted *reset*(a). Resets are used to unblock the non-deterministic actions and to apply them multiple times *in a sound manner.*

The definition of the *reset*(a) action takes advantage of the structure of the translations $K = K_{T,M}$ seen in chapter 3, and it allows multiple occurrences of non-deterministic actions without having to introduce various action copies. Assume a belief space planner that represents belief states as sets (conjunctions) of formulæ. Actions in such a setting, deterministic or not, map one set of formulæ F_i to another set F_{i+1} (i.e. map belief states to belief states, cf. equation (2.3)). Likewise, an action sequence $[a_0, \ldots, a_n]$ is a solution plan if it maps the initial set of formulæ F_0 to a final set F_{n+1} that implies the goal. Now consider a version of such a belief space planner that drops some of the formulæ in F_{i+1} and thus maps a set of formulæ F_i to a weaker set F'_{i+1}. Such a planner is still sound but possibly incomplete, as some states in the beliefs are ignored. The relevant observation here is that such incomplete planner over the deterministic relaxation P_d can accommodate plans with multiple occurrences of non-deterministic actions a provided that, before new occurrences of the same action a are applied in P_d, *all belief states involving hidden conditions $h^i_j(a)$ are dropped.* This is equivalent to ignore the information about the number of executions of non-deterministic actions, and their consequences.

This is precisely what the *reset*(a) action does: it unblocks the action a while deleting all beliefs involving the hidden conditions $h^i_j(a)$ associated with a. This is achieved by having the literal $blocked(a)$ as a precondition of $reset(a)$, and the literals $\neg blocked(a)$ and $\neg KL/t$ as its effects, for all L and tags t that include a hidden condition $h^i_j(a)$.[1]

Definition 5.5 (Translation $K(P_d)$)**.** *Let $P_d = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ be a conformant problem as in definition 5.1, then the translation $K(P_d) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is a* classical planning problem*, where*

- $\mathcal{F}' = \{KL/t, K\neg L/t \mid L \in \mathcal{F} \textit{ and } t \in T\} \cup \{blocked(a) | a \textit{ is a non-deterministic action in } \mathcal{A}\}$,
- $\mathcal{A}' = \mathcal{A} \cup \{\, reset(a) \mid \; a \textit{ is a non-deterministic action in } \mathcal{A}\,\}$*, and with each precondition L for $a \in \mathcal{A}$ replaced by $KL \wedge \neg blocked(a)$, and each conditional effect $C \rightarrow L$ replaced by the two rules:*
  $KC/t \rightarrow KL/t$ *and* $\neg K\neg C/t \rightarrow \neg K \neg L/t$.

  The $reset(a)$ action, for all non-deterministic action $a \in \mathcal{A}$, is such that:

  - $pre\big(reset(a)\big) : blocked(a)$
  - $eff\big(reset(a)\big) : \neg blocked(a) \wedge \neg KL/t$, *for all tag t such that $h^i_j(a) \in t$ or $h^i_j(a) \in L$, for some integer i, j,*
- $\mathcal{I}' = \{KL/t \mid \mathcal{I} \wedge t \models L\}$,
- $\mathcal{G}' = \{KL \mid L \in \mathcal{G}\}$.

If P_d is the deterministic relaxation of P extended with the $blocked(a)$ fluents, and $K(P_d)$ is the translation extended with the $reset(a)$ actions, it can be proved that the classical plans for $K(P_d)$ are all plans for P:

Proposition 5.6. *Any plan π returned by a classical planner from the translation $K(P_d)$ extended with the reset actions, is a plan for the non-deterministic conformant problem P, once the merge and reset actions are removed.*

5.5 The $\langle K, K_0 \rangle$ Planner

The third non-deterministic planner is a special case of the K-reset planner; it uses a particular type of K translation for getting rid of the $reset(a)$ actions and the $blocked(a)$ fluents. This family of translations are motivated by the fact that the reset actions and the blocking fluents are not needed in the K-reset planner when the translation K does not generate tags involving the hidden conditions $h^i_j(a)$. Not generating tags simply means that the hidden conditions on actions execution are not relevant to the goal or to the preconditions involved in the planning problem.

For example, the K-reset planner, for the special case of $K = K_0$, does not require blocking fluents and reset actions as it does not generate any tags at all, except for the empty tag. In any case, the K_0-reset planner is just too weak.

[1] It is actually not necessary to delete all literals KL/t involving tags featuring the hidden conditions $h^i_j(a)$ before applying the action a a new time; it suffices to delete all such literals when KL does not hold. Otherwise, all literals KL/t can be maintained. This refinement is often convenient and is part of the K-reset planner tested below.

A much larger family of translations that do not generate tags involving the hidden conditions $h^i_j(a)$ can be defined in analogy to the family of deterministic translations K_{models}, and K_i for $i \geq 0$ in chapter 3. Recall that these translations are instances of the general $K_{T,M}$ translation defined by the manner in which they map subsets of clauses of $C_I(L)$ to merges, for literals L. Given a non negative parameter x, we denote for a translation K_x its set of merges for each literal L as $m_x(C_I(L))$. The translation K_x follows the schema described in section 3.4. A class of translations $\langle K_x, K_0\rangle$ can then be defined for the deterministic relaxation P_d of P by considering the set of clauses in $C_I(L)$ split in two sets: the clauses $C^h_I(L)$ that involve hidden conditions $h^i_j(a)$ for some action a, and the clauses $C^o_I(L)$ that do not. The translation $\langle K, K_0\rangle$ for $K = K_x$ is defined by discarding the clauses $C^h_I(L)$ that involve the hidden conditions, and hence by applying the K_x translation to the remaining set of clauses $C^o_I(L)$ only. Namely, the merges in the translations $\langle K, K_0\rangle$ are simply the merges $m_x(C^o_I(L))$ for the goal and precondition literals L, and the resulting set of tags is the set of tags in all such merges (along with the empty tag). The translation $\langle K, K_0\rangle$ does not generate tags involving the hidden conditions $h^i_j(a)$, nor beliefs that are conditional over such conditions. Thus it is not required to block or to reset (non-deterministic) actions a whose effects depend on hidden condition. It can be shown that:

Proposition 5.7. *Any plan π returned by a classical planner from the translation $\langle K, K_0\rangle(P_d)$ is a plan for the non-deterministic conformant problem P once the merge actions are dropped.*

Since the $\langle K, K_0\rangle$ translation does not capture disjunctive reasoning over the hidden h-conditions, we extend it with two types of general inference rules from Palacios and Geffner (2006), implemented as additional actions in the classical encoding that capture some of those patterns.

The first is the *static-or* rule, that is based on the disjunctions $L_1 \vee \cdots \vee L_n$ in the problem P that are *invariant*, meaning that these clauses are true in all reachable belief states.

For instance, consider the example 3.1, where the initial position is completely unknown, being possibly one in the set $S = \{p1, p2, p3, p4, p5\}$. The clause $(p1 \vee p2 \vee p3 \vee p4 \vee p5)$ expresses that the set S of (invariant) literals is *exhaustive*, meaning that at least one of them must be true in every reachable state, while the set of clauses (pi, pj) and $(\neg pi, \neg pj)$ capture the *mutual exclusivity* of such literals, meaning that only one of them can be true in a reachable state, for $i \neq j$. Given that, we can say that the set S of literals are invariant for this planning problem. Generally, if the non–unary clauses of $\mathcal{I}$ are invariant, they can be associated to sets of multivalued variables which value is initially unknown, and the uncertainty in the problem corresponds to the uncertainty on such variables.

The action associated to an invariant has n conditional effects $\bigwedge_i K\neg L_k \rightarrow KL_i$, for $1 \leq i \leq n$ and $k \neq i$. For example, in a grid $n \times m$, the disjunctions $x_1 \vee \cdots \vee x_n$ and $y_1 \vee \cdots \vee y_m$ encoding the possible x and y locations are invariant and therefore result in two actions of this type.

The second rule, called *action compilation* (Palacios and Geffner, 2006), makes explicit effects that are otherwise implicit. Consider an action a that under certain conditions C' can force a literal L to make the transition from false to true, while preventing the opposite transition. In this case it can be inferred that a makes L true, even if it is initially unknown. This type of inference is captured in the translation as follows:

Definition 5.8 (Action Compilation)**.** *Let be P a conformant planning problem and $K(P)$ a classical translation over P. The action compilation rules are added to $K(P)$ for each rule a in P of the form $a : C \wedge \neg L \rightarrow L$, and the effects for the same action a that delete L are $C_i \rightarrow \neg L$, with $1 \leq i \leq n$. The action compilation rules are of the form: $KC \wedge K\neg c_1 \wedge \cdots \wedge K\neg c_n \rightarrow KL$, where c_i is a literal in C_i.*

This is a modular translation rule in which the C' above is the formula $(C \wedge \neg c_1 \wedge \cdots \wedge \neg c_n)$, for any combination of literals c_i chosen to preempt the rules $C_i \rightarrow \neg L$ of the same action a that can add L. The action compilation rule basically inhibit the condition of effects that would delete a literal L, added by another effect of the same action. Action compilation obtains such implicits effects in polynomial time as each action is considered in isolation, as a preprocessing step. This traslation rule preserves soundness, as so do the approximation of the rule that we implemented by using in the head of the rule only one c_i for each effect C_i. This approximation is also polinomial, as here only of of the possible combination of literals c_i is used.

The family of translations $\langle K, K_0 \rangle$ adapt to different family of problems, and have the vantage of needing a single call to the classical planner. They constitute a particular case of translations, where case base reasoning ignores the hidden conditions on the non-deterministic effects, but where it is applied to the other unknown variables of the problem. In the experimental section below, the $\langle K, K_0 \rangle$ implemented planner tests the translations incrementally. Over a planning problem, first the $\langle K_0, K_0 \rangle$ translation is tried, then $\langle K_1, K_0 \rangle$, and finally $\langle K_{models}, K_0 \rangle$. Of course, the first translation that solves a problem ends up the planning process.

5.6 Experimental evaluation

The different translation-based strategies discussed in this chapter have been implemented in different planners that make use of the $K_{T,M}(P)$ translation. The source code, written in OCaml, has been built as an extension of the translator from conformant to classical of the planner T_0 (Palacios and Geffner, 2009).

The performance of the $\langle K, K_0 \rangle$ and K-reset planners are evaluated using LAMA (Richter et al., 2008) and FF (Hoffmann and Nebel, 2001) as the base classical planners over the translated problem. The results for the K-Replanner are not included: the experiments using a preliminary implementation suggest that it does not scale up due to the multiple calls over incremental versions of the planning problems: adding a copy of a non-deterministic actions implies a polynomial increasing in the size of the problem, making it to degrades fast as action copies are incrementally added. The results are compared with MBP and KACMBP (Bertoli et al., 2006; Bertoli and Cimatti, 2002), which to the best of our knowledge are the only other (qualitative) conformant planners that deal with non-deterministic actions.

	$\langle K, K_0\rangle$		K_1-reset		MBP		KACMBP	
	time	#acts	time	#acts	time	#acts	time	#acts
bmtuc-10-10	0.0	20	0.0	20	65.9	29	0.2	20
bmtuc-20-10	0.1	40	0.1	40	> 2h		0.6	40
bmtuc-100-10	12.0	200	12.7	200	> 2h		25.1	200
bmtuc-100-50	5.6	200	6.1	200	> 2h		> 2h	
bmtuc-100-100	**10.6**	200	**11.9**	200	> 2h		> 2h	
btuc-100	2.7	200	2.8	200	> 2h		2.0	200
btuc-200	20.3	400	21.4	400	> 2h		16.9	400
btuc-300	**70.6**	600	**72.8**	600	> 2h		**62.1**	600
nondet-ring-30	430.1	206	440.9	206	> 2h		21.1	349
nondet-ring-40	1698.4	276	1729.4	276	> 2h		67.6	469
nondet-ring-50	SMF, PTL		SMF, PTL		> 2h		**603.1**	2552
nondet-ring-1key-05	0.4	30	unsolvable		0.1	33	0.2	42
nondet-ring-1key-10	12.6	77	unsolvable		11.2	122	4.0	197
nondet-ring-1key-15	101.9	272	unsolvable		> 2h		33.7	375
nondet-ring-1key-20	SM		unsolvable		> 2.1 GB		**246.5**	1104
sgripper-10	0.1	47	0.9	56	> 2h		0.6	68
sgripper-30	2.5	147	34.7	176	> 2h		23.3	228
sgripper-50	**16.0**	247	255.1	296	> 2h		155.6	388
mouse-and-cat-10	0.3	17	11.8	17	0.0	17	0.0	17
mouse-and-cat-20	5.2	37	1031.7	37	1.8	37	0.2	37
mouse-and-cat-30	23.3	57	KT		38.8	57	0.9	57
mouse-and-cat-40	KT		KT		49.2	77	**2.2**	77
nd-coins-06	0.0	24	0.0	24	165.9	25	1.6	45
nd-coins-08	0.0	26	0.0	26	882.1	24	2.4	52
nd-coins-10	0.0	21	0.0	21	> 2h		3.8	106
nd-coins-12	0.1	68	0.1	68	> 2h		> 2h	
nd-coins-20	**0.1**	88	**0.1**	88	> 2h		> 2h	
nd-uts-04	0.0	23	0.1	27	12.2	40	18.8	42
nd-uts-05	0.1	29	0.2	30	> 2h		735.2	61
nd-uts-06	0.1	35	0.4	40	> 2h		> 2h	
nd-uts-07	**0.2**	41	**0.6**	44	> 2h		> 2h	
trail-follow-100	0.8	198	PMF, PTL		0.2	198	0.1	198
trail-follow-150	1.3	298	PMF, PTL		0.4	298	0.1	298
trail-follow-200	1.9	398	PMF, PTL		0.7	398	**0.2**	398
move-pkgs-nd-3-3	0.1	22	51.7	22	8.4	28	0.3	22
move-pkgs-nd-4-1	0.0	8	34.4	8	0.0	8	0.0	8
move-pkgs-nd-4-2	0.1	35	PMF, PTL		0.1	18	0.1	28
move-pkgs-nd-4-3	0.2	28	PMF, PTL		48.3	27	1797.0	37
move-pkgs-nd-5-1	0.2	19	PMF, PTL		0.0	25	0.1	19
move-pkgs-nd-5-2	0.2	49	PMF, PTL		0.4	35	1.6	73
move-pkgs-nd-5-3	**0.4**	22	PMF, PTL		> 2h		398.6	26

Table 5.1: Performance of the non-deterministic conformant planners based on the $\langle K, K_0\rangle$ and K_1-reset translations, and comparison with MBP and KACMBP.

The results are shown in Table 5.1. The table shows times in seconds, and plan quality. Times including preprocessing, translation, and search, and are rounded to the closest decimal. Plan quality is expressed as number of actions in plan. The data has been generated on PCs running Linux at 2.33GHz with 8GB of RAM, with a cutoff time of 2 hours, and a memory bound of 2.1GB. The best times for each domain shown in **bold**. In legends, KT means translation time out, PMF means FF preprocessor memory-out, PTL means preprocessor times out in LAMA, SMF means that search memory-out in FF. "Unsolvable" means that the translation results in classical planning problem with an unreachable goal ($h(s_0) = \infty$).

Non-deterministic benchmarks

Many of the domains used in the tests are taken from the MBP, KACMBP and T_0 distributions (Bertoli et al., 2006; Bertoli and Cimatti, 2002; Palacios and Geffner, 2009). These include *bmtuc*, *btuc*, *nondet-ring* and *nondet-ring-1key*, from the former, and *sgripper* from the latter. The first two are non-deterministic variations of the *bomb-in-the-toilet* problem where the action to dunk the the bomb can non-deterministically clog the toilet. The *nondet-ring* is a variation of the deterministic *ring* domain, where all the windows of a ring-shaped corridor have to be closed and, when not locked, the windows can open again). The last one is a variation of the classical *gripper domain*: a movement can end non-deterministically in one of the target rooms).

The other domains are new. *Mouse-and-cat-n* is about a mouse that must collect one of m cheeses in known locations over a $n \times n$ grid. The initial position of the cat is known, but every time the mouse moves, the cat moves non-deterministically in one of the four possible directions. The mouse can move or grab a cheese only if the cat is not in that position. An instance has a solution if the mouse can get to a cheese, reaching each position before the cat does.

The domains *nd-coins* and *nd-uts* are non-deterministic extensions of the *coins* and *uts* domains used in the conformant track of previous IPCs. In nd-coins, a certain amount of coins is hidden and has to be collected from different floors of a building. In this particular benchmark, the lift non-deterministically closes its doors when the agent steps in or out. The lift can't move if a door is left open, and an action is available to shut the doors. In nd-uts, a traveller has to visit different destinations in order to fulfil the goal, not knowing initially its own position. The traveller can forget his passport in the plane after each leg of the trip and, before leaving for another destination, there is an action to recover the passport that is necessary to travel.

Trail-follow-n is a about an agent moving in a $n \times n$ grid from $x = 0$, $y = n/2$ to $x = n$, $y = n/2$. There are actions for moving 1 unit along the x-axis with noise over the y coordinate that can be $+1$, -1, or 0. In addition, there is an action "back-to-trail" that moves the agent 1 unit up or down, or none, according to whether the agent is below, above, or at the $y = n/2$ row (the trail).

Last, *move-pkgs-n-m* is about moving m objects from their initial locations to their final locations over a $n \times n$ grid. The possible actions involve picking-up or putting-down an object, and moving from a location to an adjacent one. The action "move" has the non-deterministic effect that the object being held may drop at the target location.

Empirical results

The best results in Table 5.1 are for the KACMBP and $\langle K, K_0\rangle$ planners. The $\langle K, K_0\rangle$ planners produce much shorter plans then KACMBP or MBP. The $\langle K, K_0\rangle$ planner used here tries different translations, in order: first the $\langle K_0, K_0\rangle$ translation, then $\langle K_1, K_0\rangle$, and finally $\langle K_{models}, K_0\rangle$, until a solution is found. A translation is assumed to fail when the classical planner reports an infinite heuristic for the initial state, meaning that no solution can be captured by the translation.

In the table, the classical planner used is LAMA, except for *move-pkgs* ($\langle K, K_0 \rangle$), *btuc*, *trail-follow*, *sgripper*, where results with FF have been reported. The classical planners FF and LAMA provide roughly a similar coverage over these domains, with most failures arising not during the search but during preprocessing, as neither planner has been designed to handle the huge grounded PDDL files that result from the translations. For instance, in the K_1-reset translation, where tags are added for each of the hidden h conditions, LAMA times out in the translation into SAS in domains like *trail-follow* and *move-pkgs*, while in the same two domains, FF's parser breaks down. Likewise, in *nondet-ring*, FF runs out of memory in the search, while LAMA times out while processing the landmarks. Last, in *mouse-and-cat*, the problem is in our translators, that time out. This problem, however, should be fixable with a better implementation.

The $\langle K, K_0 \rangle$ translation with $K = K_0$ produces solutions for *mouse-and-cat*, *sgripper*, *trail-follow*, and *move-pkgs*, and with $K = K_1$, solutions for all the other domains except for *nondet-ring-1key*. For the K-reset planner, $K = K_1$ was used in all cases, reporting the *nondet-ring-1key* instances as unsolvable. Even leaving the non-deterministic actions aside, this problem has width higher than 1 and the K_1 translation does not render it solvable. Hence, the K_{models} translation ends up being used in the $\langle K, K_0 \rangle$ planner, but the size of this translation grows exponentially with the number of rooms. On the other hand, in the version without the key that has width equal to 1, the difficulties arise in the classical planners: LAMA times out while ordering the landmarks, and FF gets lost in the search. KACMBP is best on the two *nondet-ring* domains.

Many solutions come from the K_0 or from the K_1 translation, without considering hidden condition h^i_j. Thus, $\langle K, K_0 \rangle$ is able to deal with the non-determinism involved in many planning problems by considering them like exogenous events from which the plan can be recovered into a known state by deterministic actions.

In principle, MBP and KACMBP can deal with a broader sets of problems, they are provably complete, but they tend to get lost more easily in the search in problems that have a significant planning component. The reason of that behaviour has to be searched in the heuristics used by these two planners. The accent is put to the *cardinality* of the belief state, i.e. the number of states that it includes. In certain domains, this is an useful guide to the search, but the classical planners make use of more powerful search estimates on top of the classical translation $K_{T,M}(P)$.

5.7 Discussion

The translation-based approach to conformant planning for settings where some of the actions have non-deterministic effects makes use of a deterministic relaxation that is correct as long as the non-deterministic actions are executed at most once. One theoretical issue for the future, involves studying the conditions under which some of the incomplete translations described before are either strongly or weakly complete. A translation $K(P)$ is strongly complete if it captures all plans for P, and is weakly complete if it captures some plans. In the latter case, the translation is useful too, as it can be used to obtain a plan for P. The K_1 translation for deterministic conformant planning is strongly complete for problems with width bounded by 1, and yet it is often effective (weakly complete) for problems with higher widths. These characterizations are still to be worked out for the incomplete translations proposed.

The problems that cannot be solved by the $\langle K, K_0 \rangle$ and K-reset planners are problems that involve non-trivial disjunctive reasoning patterns over the hidden conditions h. For example, consider a problem where a goal $x = n + 1$ is to be achieved starting from $x = 0$ and $y = 0$ with an action that increases x one by one up to $x = n$, and increase y non-deterministically by either 1 or 0. If there are then n actions $enter(i)$, with $1 \leq i \leq n$, to move from $x = n$ to $x = n + 1$, each with condition $y = i$. The plan that increases x for n times, followed by the actions $enter(1), \ldots, enter(n)$, solves the problem, but can't be captured by the $\langle K, K_0 \rangle$ and K-reset planners for any translation K if $n > 2$.

While the results show that the translation-based approach is feasible and competitive in the non-deterministic setting, they also suggest that scalability could be improved by integrating the classical planner and the translators more tightly. Moreover, tags in the translation play two roles: keeping track of the "conditional beliefs", and producing the heuristic for guiding the search over beliefs. It seems also that scalability could be improved by separating these two roles, and implementing them in different ways.

Related work

The planners MBP and KACMBP allow to represent non-deterministic effects in actions' outcome as these two planners are based on model checking technique that relies on non-deterministic state transition systems (Bertoli et al., 2001; Bertoli and Cimatti, 2002).

Many other planner that deal with incomplete information, like POND (Bryce et al., 2006), do not encode non-deterministic effects. The reason seem to lie in a preference for probabilistic effects rather than non-deterministic effects; the difference being that –in theory– non-deterministic effects imply that no particular effect is preferred over the others, meaning simply that the action model is completely unknown to the planning agent. Conformant planning with non-deterministic action effects can be indeed done as probabilistic planning and asking for a 1.0 probability plan. the use of probabilities appear to us to be justified only in those cases where the (non-deterministic) action model is already partially known, which doesn't appear to be the case in the general setting of planning under incomplete information as conformant and contingent planning.

5.8 Summary

We have considered extensions of the translation-based approach to conformant planning problems where some of the actions have non-deterministic effects. We employed a deterministic relaxation that is sound as long as the non-deterministic actions are executed at most once. We then considered several incomplete translation schemas and planners that use this relaxation, some of which appear to be quite effective and map non-deterministic conformant problems into classical ones. Two of these planners, based on the K-reset and $\langle K, K_0 \rangle$ translations, are compatible with any translation K, and in particular, the $\langle K, K_0 \rangle$ translation applied successively for $K = K_0$ and $K = K_1$ appears to be quite effective. The empirical results of these translations are encouraging, even if the resulting planners do not always perform better than existing ones.

The non-deterministic effects in actions are a typical but non commonly used source of incomplete information in planning. Typically actions with multiple possible effects encode exogenous events, or partial knowledge of actions outcome. When the action model is not well known, the actions outcomes are not previsible. Our encoding allow us to use assumptions on the initial situation to model such kind of planning problems. This topic will be discussed deeply in chapter 7.

Conformant planning is fundamental in dealing with problems with incomplete information, as the ability to find conformant plans is needed in contingent settings where conformant situations are an special case. The contingent planning model extends conformant planning with the addition of sensing actions: actions that allow the planning agent to observe some features of the world. Contingent planning is one of the most computationally difficult tasks of automated planning and is well adapted to model real world problems. We use results obtained from conformant planning to characterise and express families of translations adapted to contingent planning.

Part III

Translations for Contingent Planning

Chapter 6

A translation-based approach to Contingent Planning

In this chapter, we introduce an approach to contingent planning that uses and extends ideas advanced recently for compiling conformant problems into classical planning problems (Palacios and Geffner, 2007).

Here, a contingent problem P, which is a non-deterministic search problem in belief space, is compiled into a non-deterministic problem $\mathcal{X}(P)$ in state space whose literals represent the beliefs over P. We assume that the problem P involves uncertainty in the initial situation only, and that actions are all deterministic. As we will see, a straightforward sound and complete compilation is feasible by tagging each of the fluents L in P with the possible initial states of P.

This compilation, however, is linear in the number of possible initial states, which is in turn exponential in the number of fluents. We show nonetheless that even in such cases, a sound, complete, and polynomial translation $\mathcal{X}(P)$ is possible, provided that the problem P has bounded contingent width, and we show that the contingent width of almost all existing benchmarks is 1; a result that parallels the one for conformant planning reported by Palacios and Geffner (2009). We then show how the non-deterministic but fully observable problem $\mathcal{X}(P)$ can be solved using a suitable relaxation $\mathcal{X}^+(P)$ that is a classical planning problem. The resulting contingent planner CLG (Closed-Loop Greedy Planner) accepts then a contingent problem P with deterministic actions, and solves it by using the translation $\mathcal{X}(P)$ for keeping track of the beliefs, and a suitable strengthening of the relaxation $\mathcal{X}^+(P)$ for selecting the actions to apply next. We finally present empirical results and a summary.

Results from this chapter have been published in

- *A Translation-based Approach to Contingent Planning*, by A. Albore, H. Palacios, and H. Geffner, in 21st International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, California, 2009 (Albore et al., 2009).

- *Fast and Informed Action Selection for Planning with Sensing*, by A. Albore, H. Palacios, and H. Geffner, in Lecture Notes in Computer Science – Current Topics in Artificial Intelligence, 12th Conference of the Spanish Association for

Artificial Intelligence (CAEPIA), Salamanca, Spain. Springer, 2007 (Albore et al., 2007).

6.1 Motivation

Contingent planning is concerned with the problem of achieving goals in the presence of incomplete information and sensing actions (Peot and Smith, 1992; Pryor and Collins, 1996). This is one of the most general problems considered in the area of planning and one of the hardest (Haslum and Jonsson, 1999; Rintanen, 2004). In the last few years, significant progress has been achieved resulting in a variety of contingent planners that can solve large and non-trivial problems, usually by casting the contingent planning problem as a search problem in belief space (Bonet and Geffner, 2000).

In spite of this progress, however, a large obstacle remains: many problems involving incomplete information and sensing actions have solutions of exponential size. This is different than in classical or conformant planning where exponential length solutions are the exception. Contingent plans of exponential size follow naturally from situations where the number of observations required is linear in the size of the problem[1].

Domain-independent planning techniques can be exploited for dealing with these problems. However, rather than aiming at constructing full contingent plans, we aim at an effective *action selection mechanism* that chooses the action to do next in closed-loop fashion. For this to work, the action selection mechanism must be fast and informed. Indeed, while it is not possible to consider *explicitly* all possible combination of contingencies that may arise, these contingencies cannot be ignored. So, an action selection mechanism that *implicitly* takes all contingencies into account, is the greedy policy $\pi_h(b)$ obtained with a sufficiently informed heuristic function $h(b)$ over the beliefs b. For suitable heuristic functions (e.g., the optimal value function), the resulting policies are optimal. In practice this approach faces two problems: the difficulty of getting fast and informed heuristics over belief space, and the time and space complexity of carrying out the beliefs.

As we will see, a contingent planning problem P can be converted into a suitable planning problem $\mathcal{X}(P)$ at the knowledge-level whose states represent belief states over P. The problem $\mathcal{X}(P)$ is thus a *fully observable but non-deterministic planning problem*, whose solutions represent solutions to the *contingent planning problem P*. These problems cannot be directly fed into classical planners, but an appropriate relaxation $\mathcal{H}(P)$ can. Classical heuristics for $\mathcal{H}(P)$ hence estimate the length of plans that implicitly consider all the contingencies while taking sensing actions into account. With these transformations performed as preprocessing, the *action selection problem in planning with sensing* is mapped into an *action selection problem in classical planning.* The resulting *Closed-Loop Greedy planner (CLG)* scales up well: constructing contingent plans faster than other planners over a broader range of scenarios, and producing also meaningful executions in problems where the construction of full contingent plans is not feasible.

[1] Compact solutions to these problems are often possible in languages closer to the ones used in programming that accommodate loops and subroutines, yet planning with such constructs appears to be very hard (as hard as automatic programming), and as shown here, not always necessary.

6.2 The translation $\mathcal{X}(P)$ for Contingent Planning

Our intention is to use the results and advances obtained for conformant planning to solve contingent planning problems. Contingent planning is more complex than conformant planning, as conformant planning can be seen as a special case of contingent planning. In fact, contingent planning adds the possibility to perform sensing to a conformant planning problem with uncertainty, but no sensing available. Removing sensing actions from a contingent problem P yields a conformant planning problem, where uncertainty is embedded in the initial situation only.

However, the direct translation of a contingent problem P into an equivalent classical problem P' is not possible as the problems have different solution form. The solution to a conformant problem is an action sequence, while the solution to a contingent problem P is a contingent plan that can be regarded as a *policy tree* Π: a tree[2] with a function mapping the internal nodes n of the tree into actions $a(n)$ in P. Roughly, a node n in the tree has many children when the action $a = a(n)$ is a sensing action[3], so that the different (complete) branches stand for the different possible executions. A policy tree Π solves the problem P when the executions associated with each of the branches in Π are all feasible (i.e. the action preconditions hold when actions are applied), and end up in belief states where the goals are true.

Since contingent plans are not just sequences of actions, we cannot derive (interesting) contingent plans from a classical planner following similar technique as in chapter 3. Rather, here we aim at obtaining *heuristics* from a translation of a contingent planning problem P into a classical planning problem $K(P)$, and an efficient and compact *belief representation* that, while not always complete[4], will be sufficiently flexible and practical to enable the solutions of interesting problems.

Execution model: the $\mathcal{X}(P)$ translation

Contingent problems P cannot be translated into classical problems but can be translated into *non-deterministic fully observable problems*: these are problems where the states are observable and some actions have non-deterministic effects. (Strong) solutions to such problems are also policy trees: in fact a solution must hold for any possible branch of the tree Π; the difference being that belief states $b(n)$ in P associated to each node n in Π can be replaced by plain states $s(n)$ over a non-deterministic fully observable problem $\mathcal{X}(P)$. The benefit of this translation, that is common to the translations of conformant problems into classical ones, is that beliefs are compiled away in the target problem. Of course, the translation is fully efficient when the mapping from P to $\mathcal{X}(P)$ can be done in polynomial time.

To describe contingent planning problems, we consider a planning language that extends STRIPS with conditional effects, negation, an uncertain initial situation, and sensing actions. We use the definition 2.3, where a contingent problem P is a

[2] Most contingent planners build graphs rather than trees, yet this distinction is irrelevant from a theoretical point of view.

[3]Generally it is considered that the children nodes of a sensing action a are two: one of each possible truth value of the observed (Boolean) variable. Of course, in case of multivalued variables, the potential outcomes of an observation are as many as the possible values of the variable can be.

[4]The approach is sound but incomplete, meaning that some solutions of P (contingent plans) are not solutions of $K(P)$.

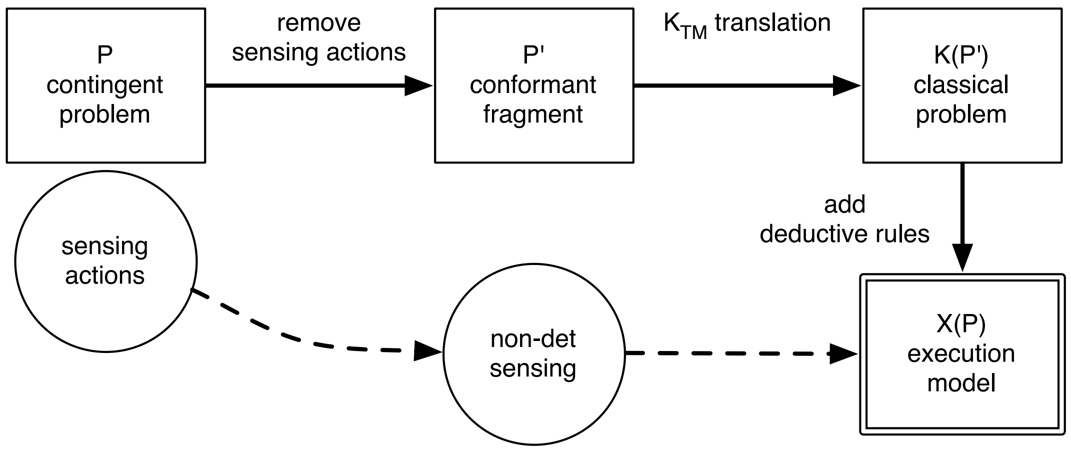


Figure 6.1: Steps for the $\mathcal{X}(P)$ translation.

5-tuple $< \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$, where $\mathcal{O}$ is the set of observations or sensing actions[5]. Each sensing action is described by a pair $< C, L >$, meaning that when C holds, then the truth value of fluent L can be observed. An observation that uncovers the truth value of a positive literal L and is also denoted as $obs(L)$. Observations may have preconditions, but for simplicity we assume that they do not have any other effects. Sensing actions have a different scope and meaning than regular actions: while an action *modifies* the environment by acting on it, a sensing action does not change the status of the environment, but *reveals* it, when initially hidden to the planning agent. A normal action $a \in \mathcal{A}$ has preconditions given by a set of fluent literals, and a set of conditional effects $C \to L$ where C is a set of literals and L is a single literal.

We start by defining the *conformant fragment* P' of the contingent problem P without the sensing actions. So if P is described by the tuple $\langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, then $P' = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. On top of P', we apply the $K_{T,M}$ translation as seen in definition 3.2, to obtain a fully observable problem $K_{T,M}(P')$ which is a classical planning problem.

To obtain the translation $\mathcal{X}(P) = X_{T,M}(P)$, we extend the former translation $K_{T,M}(P')$ with two components: an encoding of the *sensing actions*, expressed as non-deterministic actions, and two *deductive rules* expressed as actions that extend the merge rules used for conformant planning, reflecting that tags may be inferred to be false when observations are gathered. We will see that the translation $X_{T,M}(P)$ is sound, and that for suitable choices of tags T and merges M, it can be shown to be complete too.

Example 6.1. *We can consider, as a motivating example, the following contingent planning problem: a robot in a T-shaped corridor has the goal of reaching one of the two extremes of the T (top left or right end). A sensor at the bottommost cell of the central corridor indicates whether the goal is on the left or the right.*

This problem can be modelled with the standard up, down, left, and right move actions, that move the agent one cell in the relative direction, and with the goal

[5] No closed world assumption about $\mathcal{I}$ is made throughout the presentation, yet as it is standard, the actual planner assumes $\neg L$ in $\mathcal{I}$ if the positive literal L is not mentioned at all in $\mathcal{I}$.

$(p \land X_1) \lor (\neg p \land X_2)$ where X_1 and X_2 stand for the possible goal locations. Notice that this goal is made by two disjuncts, but it can be brought back to a conjunction by simply adding a "end" action with each term as precondition and a single effect literal g set to be the goal of the problem. The goal also depends on the value of the fluent p that encodes the information available in the bottommost cell, that can be obtained from an information gathering action $obs(p) =< X_5, p >$:

$$obs(p):\ X_5 \to p \mid \neg p$$

where X_5 is the location of the bottommost cell, and p corresponds to the (hidden) domain variable encoding the goal location.

Without the presence of the information gathering action, this problem would be unsolvable because the truth value of the fluent p could never be uncovered.

The sensing actions $obs(L) :< \top, L >$ in the contingent problem P become in $\mathcal{X}(P)$ the *non-deterministic actions*

$$obs(L) : \neg KL \land \neg K\neg L \ \to\ KL \mid K\neg L\ . \tag{6.1}$$

These are *physical actions* that result in either KL or $K\neg L$ when neither KL nor $K\neg L$ hold before[6], thus capturing the effect of observing the truth value of L at the knowledge-level (Petrick and Bacchus, 2002). If either L or $\neg L$ is known before applying $obs(L)$, the action has no effect.

Additional deductive rules are needed in $\mathcal{X}(P)$, as the tags t that are initially unknown may become known due to observations. For example, if the disjunction $x_1 \lor x_2$ is true in the initial situation, an action a with conditional effect $x_1 \to y$ is applied, and y is then observed to be false, it should be possible to conclude both $K\neg x_1$ and Kx_2 (provided that a doesn't affect x_2). Moreover, if this is followed by an action b with conditional effect $x_2 \to L$, it should be inferred KL as well.

The additional deductive rules are encoded as actions with single conditional effects:

1. **Contingent Merge:** $$\bigwedge_{\substack{m\in M\\ t\in m}} (KL/t \lor K\neg t) \ \to\ KL \tag{6.2}$$
2. **Tag Refutation:** $$KL/t \land K\neg L \ \to\ K\neg t \tag{6.3}$$

Contingent Merge (CM) is a generalization that subsumes the *Merge* actions in the conformant translation by replacing every literal KL/t by the disjunction $KL/t \lor K\neg t$. This is because in the contingent setting it suffices to have L true given the tags t in a merge m that have not been refuted by the observations for L to be known.

Tag Refutation (TR) yields the tags t that are refuted by the observations: these are the tags that predict a literal L which is known to be false. In these rules, Kt and $K\neg t$ refer to new atoms added to $\mathcal{X}(P) = X_{T,M}(P)$ for all $t \in T$ (except the empty tag).

[6]Recall that in the $K_{T,M}(P)$ translation the couple of new fluents KL and $K\neg L$ indicate that the fluent L in the original problem is "known to be true" and, respectively, "know to be false".

The use of these rules is combined with a simple transformation that make all tag literals static. A *static literal* is never added or deleted by some action effect. So, in case a tag in the initial situation equals to a non static L (and L can be a conjunction of literals), then the new atom Kt is added in the translation to refer to the tag, instead of using KL. The tags considered for the translation are then limited to such static literals only.

Definition 6.1 (Non-deterministic fully observable translation $X_{T,M}(P)$)**.** *For a contingent problem P, $X_{T,M}(P)$ is the non-deterministic, fully observable problem given by translation $K_{T,M}(P')$ of the conformant fragment P' of P, extended with the non-deterministic actions in eq.(6.1), and the deductive actions **CM** and **TR**.*

For a contingent problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, the problem equivalent to the conformant fragment P' is given by $P' = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$. Finally the classical problem *$X_{T,M}(P) = \langle \mathcal{F}', \mathcal{A}', \mathcal{I}', \mathcal{G}' \rangle$ is:*

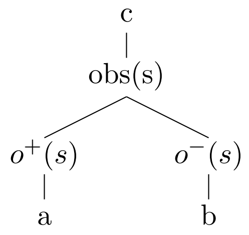
$$\begin{aligned}
\mathcal{F}' &= \{KL/t, K\neg L/t \mid L \in F\} \\
\mathcal{A}' &= \{\, a : KC/t \rightarrow KL/t,\ \ a : \neg K\neg C/t \rightarrow \neg K\neg L/t \ \ \mid a : C \rightarrow L \ in\ \mathcal{A} \,\} \cup \\
&\quad \left\{ \bigwedge_{t \in m} KL/t \vee K\neg t \rightarrow KL \mid m \in M \right\} \cup \\
&\quad \{\ KL/t \wedge K\neg L \ \rightarrow\ K\neg t \mid t \in m, m \in M\} \ \cup \\
&\quad \{\, KC \wedge \neg KL \wedge \neg K\neg L \ \rightarrow\ KL \ \vee\ K\neg L \mid \mathit{obs}(L) =< C, L > \in \mathcal{O} \,\} \\
\mathcal{I}' &= \{KL/t \mid \mathit{if}\, \mathcal{I} \models t \supset L\} \\
\mathcal{G}' &= \{\ KL \mid L \in \mathcal{G}\}
\end{aligned}$$

with the preconditions of the actions a in $K_{T,M}(P)$ including the literal KL if the preconditions of a in P include the literal L, and t ranging over T.

Example 6.2. *Consider a contingent problem P with $\mathcal{I} = \{\neg s\}$, $G = \{h\}$, actions a and b with effect h and preconditions d and $\neg d$ respectively, action c with conditional effect $d \rightarrow s$ and no precondition, and the sensing action $obs(s) =< \top, s >$. The objective is to obtain h without knowing if d is initially true or false. A contingent plan for this problem is*

$$\Pi = \{c,\ obs(s),\ \textit{if true a else b}\}$$

that can be understood as a policy tree with two complete branches with action/observation sequences $\pi_1 = [\,c, o^+(s), a\,]$ and $\pi_2 = [\,c, o^-(s), b\,]$, where $o^+(x)$ and $o^-(x)$ stand for observing x and $\neg x$ respectively.



In the translation $X_{T,M}(P)$ with tags $t_1 = \{d\}$ and $t_2 = \{\neg d\}$, and merge $m = \{t_1, t_2\}$ for d and $\neg d$, this plan would work as follows:

1. *The action c in the first branch yields the literal $K\neg s/t_2$, while $o^+(s)$ yields the literal Ks.*
2. *From TR, it follows $K\neg t_2$, which along with Kd/t_1 (that is initially true and persists) permits to obtain Kd from CM for $L = d$, so that action a can be applied and the goal Kh obtained.*
3. *The second branch works in a similar way, but with $o^-(s)$ yielding $K\neg s$.*

The former example shows that the solution tree Π for P solves also the translation $X_{T,M}(P)$ for suitable tags and merges, provided that *deductive actions* (merges and tag refutations) are interleaved with plan execution. This is the basis for the notions of soundness and completeness below.

6.3 Properties of $\mathcal{X}(P)$

The proofs of all the theorems in this chapter are detailed in Appendix A.

The first result that can be established is that the translation $X_{T,M}$ preserves consistency, i.e. if P is a consistent contingent planning problem, then the translated problem $X_{T,M}(P)$ is consistent as well[7].

Theorem 6.2 (Consistency of $\mathcal{X}(P)$)**.** *Let be a contingent problem P. A valid translation $\mathcal{X}(P)$ is consistent if P is consistent.*

To prove the soundness and completeness of the translation $\mathcal{X}(P)$ we consider for convenience policy trees Π^* for $\mathcal{X}(P)$, where the labels $a(n)$ associated with the internal nodes of Π^* can stand for an action a in P followed by a (possibly empty) sequence of deductive actions (merges and tag refutations). We write then $a(n) = a^*$ and prevent deductive actions from appearing anywhere else in Π^*. As in the execution in $\mathcal{X}(P)$ actions are followed by the application of deductive actions, any policy tree for $\mathcal{X}(P)$ can be written in this way.

With this notation, two transformations on policies can be defined: *dropping* the deductive actions from Π^* replaces the "actions" $a(n) = a^*$ by the actions $a(n) = a$ over all nodes in Π^*, and vice versa, *adding* deductive actions in a policy Π for P, replaces the actions $a(n) = a$ by $a(n) = a^*$ for suitable sequences starting with a. The soundness and completeness of $\mathcal{X}(P)$ can be then defined as follows:

Definition 6.3 (Soundness)**.** *A translation $\mathcal{X}(P)$ is* sound *if for any policy tree Π^* that solves $\mathcal{X}(P)$, the policy tree Π obtained from Π^* by dropping the deductive actions solves P.*

Definition 6.4 (Completeness)**.** *A translation $\mathcal{X}(P)$ is* complete *if any policy tree Π that solves P can be extended into a policy tree Π^* that solves $\mathcal{X}(P)$ by adding some deductive actions to Π.*

The first result that can be established is soundness:

[7]We recall that a contingent problem P is consistent if the initial situation $\mathcal{I}$ is logically consistent, and every pair of complementary literals L and $\neg L$ is mutex in P.

Theorem 6.5 (Soundness)**.** *The translation $\mathcal{X}(P)$ is sound.*

The proof is immediate, considering that the deductive actions in eq.(6.2) and (6.3) are not present in problem P.

The conditions that ensure completeness are more subtle and are considered below. A simple complete translation however can be obtained from the general $X_{T,M}(P)$ scheme as in conformant planning, by letting the set of tags T represent the set S_0 of all the possible initial states of P and by having a single merge m for each precondition and goal literal in P s.t. $m = S_0$. We call the resulting translation $X_{S_0}(P)$:

Theorem 6.6 (Completeness of X_{S0})**.** *The translation $X_{S_0}(P)$ is sound and complete.*

This translation $X_{S_0}(P)$ is exponential in the number of uncertain fluents in the initial situation of P, in the worst case, as S_0 stands for the set of all the possible initial states of P. Still, where this number is small enough, the translation can be quite effective. Before addressing how these non-deterministic but fully observable problems $\mathcal{X}(P)$ are solved, we consider complete translations that may be compact. We assume from now that all merges are valid.

Covering translations

Similarly to what happens in conformant planning (Palacios and Geffner, 2006), the merges $m = \{t_1, \ldots, t_n\}$ that are needed for completeness in $\mathcal{X}(P)$ are the ones in which each tag t_i subsumes the clauses in $\mathcal{I}$ that are relevant to the precondition and goal literals of P. Clearly, the merge $m = S_0$ achieves this, yet more compact merges will often do as well. We will say that a merge $m = \{t_1, \ldots, t_n\}$ with tags t_i all consistent with $\mathcal{I}$, *satisfies* a set of clauses $\mathcal{C}$ if for each t_i in m, and each clause c in $\mathcal{C}$, there is at least one literal c_i in c entailed by t_i and $\mathcal{I}$. For characterizing the set of clauses $C_{I,O}(L)$ that are relevant to a given literal L, we assume that $\mathcal{I}$ is in prime implicate form (Marquis, 2000), meaning that $\mathcal{I}$ includes only the inclusion-minimal clauses it entails. $\mathcal{I}$ is then extended with the tautologies $(L \vee \neg L)$ for complementary literals L and $\neg L$ that do not appear as unit clause in $\mathcal{I}$.

Definition 6.7 (Set of relevant clauses $C_{I,O}(L)$)**.** *For a literal L in P, with $\mathcal{I}$ in prime implicate form, $C_{I,O}(L)$ is the set of non-unary clauses $c \in \mathcal{I}$ such that each $c_i \in c$ is relevant*[8] *to an observable literal L' in O or to L.*

O stands for a set of observables*: literals L such that either obs(L) or obs$(\neg L)$ is a sensing action in P.*

The implication of the set O of observables in definition 6.7 is the main difference with the corresponding result for conformant planning. This set can be taken to comprise the set of all observables, yet a smaller set does as well: it suffices to take $O = O(L)$ to be the observables that may affect L. This reduced set $O(L)$ corresponds to the unique minimal set of observable fluents that obey the following condition:

[8] The notion of relevance used here is the same than the one from definition 3.15.

Definition 6.8 (Set of relevant observables $O(L)$)**.** *Given a literal L in a contingent problem P, the set of observables $O(L)$ is the set of literals $\{L_i\}_{i=0}^n$ such that, given a clause $c = (c_1, \ldots, c_m)$ in $\mathcal{I}$, the literals L_i verify:*

- *c_1 is relevant to L_i, for a fixed i, and*
- *exists a c_k in c such that c_k is relevant to L_j, where $L_j \in O(L)$ or $L_j = L$.*

If $i = j$, the membership of L_i in $O(L)$ is trivial, as L_i already belongs to $O(L)$.

Again, in the contingent setting the literals that affect L are not the same as the literals that may affect L in the conformant setting, which are fully characterized by the notion of relevance. Until now relevance expresses "causal relevance", but in the contingent setting there is "evidential relevance" as well, which expresses how possible observations affect information about the current belief.

The condition for completeness can then be expressed in terms of the clauses $C_{I,O}(L)$ for each precondition or goal literal L, as these clauses are the ones needed for solving the problem. We call this set of clauses $C_{I,O}$.

Definition 6.9 (Covering translation)**.** *$X_{T,M}(P)$ is a covering translation for a contingent problem P when for each precondition and goal literal L in P such that $C_{I,O}(L)$ is non-empty, the set M contains a merge m that satisfies $C_{I,O}(L)$.*

Theorem 6.10. *Covering translations $X_{T,M}(P)$ are complete.*

As an illustration, we can consider the following example:

Example 6.3. *Let's consider the contingent problem P that has initial situation $\mathcal{I} = \{(x_1 \vee \ldots \vee x_n)\}$, goal $\mathcal{G}= \{y\}$, actions a_i with precondition x_i and effect y, and sensing actions $obs(x_i) =< \top, x_i >$, with $1 \leq i \leq n$.*

For this problem, a translation $X_{S_0}(P)$ would be exponential in size, as in $\mathcal{I}$ the x_i are not mutually exclusive.

On the other hand, the translation $X_{T,M}(P)$ with tags $t_i = \{x_i\}$ and merge $m = \{t_1, \ldots, t_n\}$ for each precondition x_i can be shown to be covering, and hence complete. For the goal y, the set $C_{I,O}(y)$ is empty because $O(y)$ is empty, as there are no observations relevant to a literal in a clause with y. For the preconditions $L = x_i$, the set $C_{I,O}(L)$ contains all the prime implicates that follow from $(x_1 \vee \ldots \vee x_n)$ –in this case $O(x_i)$ includes all the observables–, all of which are satisfied by each t_i in m.

The completeness results presented here are important for both testing whether a translation is complete and to generate complete translations. Indeed, for generating a covering translation we just need to identify the set $C_{I,O}(L)$ and compute a (valid) merge m that satisfies $C_{I,O}(L)$. This computation is polynomial if the size of this set is bounded.

6.4 Contingent width and the X_i translation

The CNF formula $C_{I,O}(L)$ encodes the uncertainty in $\mathcal{I}$ relevant to the precondition and goals L. Covering translations "require" the translation of the CNF formula $C_{I,O}(L)$ in a DNF formula entailed by $\mathcal{I}$ (the merge for L) and that satisfies $C_{I,O}(L)$. Such DNF formula can be characterized by considering the cover $c(\mathcal{C})$ of a set of clauses $\mathcal{C}$ (cf. definition 3.14), i.e. the collection of all minimal sets of literals S that contain a literal of each clause in $\mathcal{C}$, and that are consistent with the initial situation $\mathcal{I}$. The cover $c(\mathcal{C})$ satisfies $\mathcal{C}$, and can be computed in polynomial time if $|\mathcal{C}|$ is bounded. From the completeness of covering translations, it follows that a complete translation $X_{T,M}(P)$ can be constructed in polynomial time if the size $|C_{I,O}(L)|$ is bounded, for all preconditions and goals L in P. Unfortunately this condition rarely holds, yet there is a weaker sufficient condition that does: it is often possible to find a subset $\mathcal{C}$ of clauses that are either in $C_{I,O}(L)$ or are tautologies $(p \vee \neg p)$, for any p or $\neg p$ mentioned in $C_{I,O}(L)$, such that $c(\mathcal{C})$ satisfies $C_{I,O}(L)$. The *contingent width* of a literal L is defined in terms of the cardinality of such sets:

Definition 6.11 (Contingent Width of a Literal). *The contingent width of a literal L in P, written $w(L)$, is the size of the smallest (cardinality-wise) set of clauses $\mathcal{C}$ in $C_{I,O}(L) \cup Taut$ such that $c(\mathcal{C})$ satisfies $C_{I,O}(L)$, with Taut denoting the set of tautologies $(p \vee \neg p)$, for any p or $\neg p$ in $C_{I,O}(L)$ (if both appear in $C_{I,O}(L)$, then $(p \vee \neg p)$ is in $C_{I,O}(L)$ from its definition). We will refer with $C^*_{I,O}(L)$ to this extended set of clauses, i.e. $C^*_{I,O}(L) = C_{I,O}(L) \cup Taut$.*

A consequence of this definition is that the width of a literal lies in the interval $0 \leq w(L) \leq n$, where n is the number of fluents in P whose status in the initial situation is not known. The width of a problem is the width of the precondition or the goal literal with maximum width:

Definition 6.12 (Contingent Width of a Problem). *The contingent width of a contingent problem P, written as $w(P)$, is $w(P) = \max_L w(L)$, where L ranges over the precondition and goal literals in P.*

Like for the (tree)width of graphical models, computing the width of a problem P is exponential in $w(P)$, so the recognition of problems with small width can be carried out quite efficiently (Palacios and Geffner, 2009):

Proposition 6.13 (Time complexity of contingent width). *The width $w(P)$ of P can be determined in time that is exponential in $w(P)$.*

In particular, we can test if $w(P) = 1$ by considering one by one each of the sets $\mathcal{C}$ that includes a single clause from $C^*_{I,O}$(L), verifying whether $c(\mathcal{C})$ satisfies $C_I(L)$ or not. If $w(P) \not\leq 1$, then the same verification must be carried out by setting $\mathcal{C}$ to each set of i clauses in $C^*_{I,O}$(L) for increasing values of i. For a fixed value of i, there is a polynomial number of such clause sets $\mathcal{C}$ and the verification of each one can be done in polynomial time. Moreover, from the arguments above regarding $w(L)$, the width of the problem $w(P)$ can never exceed the number of unknown fluents in the problem.

The translation $X_i(P)$, for a non-negative integer i, is a special case of the general translation $X_{T,M}(P)$. For a fixed i, the translation is sound, polynomial in i, and complete if $w(P) \leq i$.

Definition 6.14. *The set of merges for a precondition or goal literal L in $X_i(P)$*

1. *is empty, if $C_{I,O}(L)$ is empty,*
2. *contains a single merge $m = c(\mathcal{C})$, if $\mathcal{C}$ is a set of at most i clauses in $C^*_{I,O}(L)$ such that $c(\mathcal{C})$ satisfies $C_{I,O}(L)$,*
3. *is the collection of all merges $m = c(\mathcal{C})$, for each set $\mathcal{C}$ of i clauses in $C^*_{I,O}(L)$, otherwise.*

The tags in $X_i(P)$ are that ones appearing in the merges along with the empty tag.

Theorem 6.15. *For a fixed i, the translation $X_i(P)$ is sound, polynomial, and if $w(P) \leq i$, covering and complete.*

The translation $\mathcal{X}(P)$ in the example 6.3 above, with $\mathcal{I} = \{xor(x_1, \ldots, x_n)\}$, is evidently an instance of the $X_1(P)$ translation, and is then polynomial in size when a translation $X_{S_0}(P)$ would be not.

Like in conformant planning, the *contingent width of almost all contingent benchmarks* turns out to be 1, and hence for these problems, the $X_1(P)$ translation is sound and complete. Still, it is not necessary for $X_1(P)$ to be complete for being useful: if the translation $X_1(P)$ is solvable, it can produce a solution to P. The incompleteness of $X_1(P)$ just means that there is no *guarantee* that *all* solutions to P can be obtained in this way.

6.5 The CLG planner

The computational pay-off of the translation $\mathcal{X}(P)$ is that it allow us to compute plans with states represented by sets of literals, rather than with beliefs represented by sets of states. Nonetheless, the solution of non-deterministic fully observable problem like $\mathcal{X}(P)$ is not trivial. Fortunately, however, $\mathcal{X}(P)$ is a problem of a special type that can be solved using classical planning techniques. The central point resides in solving a relaxation $\mathcal{X}^+(P)$ which is a classical planning problem, and that will provide an informed heuristic to solve $\mathcal{X}(P)$ efficiently. These translations are the base of the CLG contingent planner.

Heuristic model: the $\mathcal{H}(P)$ relaxation

Hoffmann and Brafman (2005b) shown that removing the preconditions and moving them into the conditions of the actions would end up in having a conformant planning problem. We want to apply such a transformation to contingent problems, deprived of sensing actions, to derive an informed and effective heuristic that can be used to solve $\mathcal{X}(P)$ by selecting the action to do next without having to construct the full plan first.

The relaxation is based first on a transformation of the contingent problem to a conformant problem, then to a classical problem. The key result is based on the *classical planning problem* $\mathcal{X}^+(P)$, which stands for the relaxation of $\mathcal{X}(P)$ where *deletes*, *preconditions*, and actions with *non-deterministic* effects are dropped.

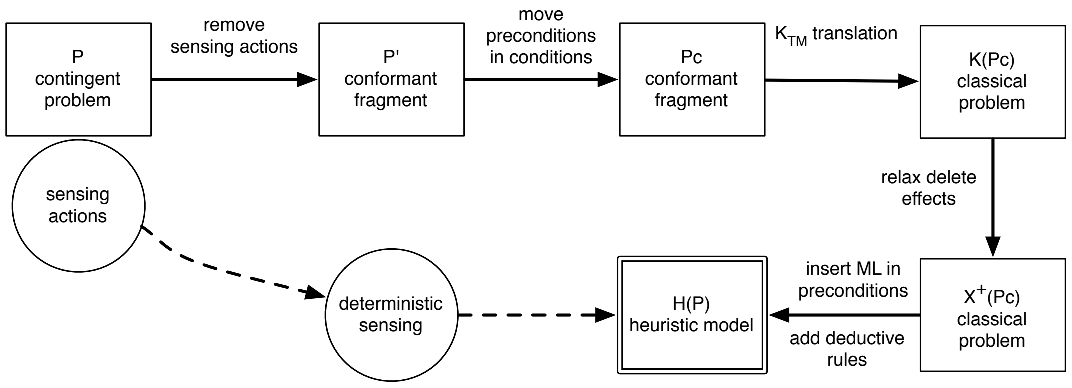


Figure 6.2: Steps for the $\mathcal{H}(P)$ translation.

Theorem 6.16. *If $\mathcal{X}(P)$ is a covering translation and Π^* is a policy tree that solves $\mathcal{X}(P)$, then there is a classical plan π for the relaxation $\mathcal{X}^+(P)$ that only uses the deterministic action in Π^*.*

The proof comes directly from the applicability of the policy tree in $\mathcal{X}^+(P)$, and the fact that for Π^* to be a solution, the goal is reached regardless of the sensing results. A very similar results applies to generous execution semantics (Hoffmann and Brafman, 2005b), where there is a sequence of the non-sensing actions in a contingent plan for a problem P that can be linearised to yield a conformant solution to relaxation P^+ obtained from P by assuming empty preconditions and delete lists. The difference being here that the positive thinking relaxes the preconditions only respect to the observations outcomes.

Like in classical planning, we can use the relaxation $\mathcal{X}^+(P)$, that can be solved in polynomial time, to get estimates of the size of the plans that are needed for solving $\mathcal{X}(P)$. The result applies to covering translations $\mathcal{X}(P)$, but not to arbitrary non-deterministic problems, which can be rendered unsolvable when the non-deterministic actions are dropped.

In the CLG planner, the relaxation $\mathcal{X}^+(P)$ is strengthened in two ways. First, rather than using $\mathcal{X}^+(P)$, we use a *stronger relaxation* $\mathcal{X}^+(P_c)$ obtained from a problem P_c that is *equivalent* to P but with each precondition L of an action a in P copied as a condition of all the effects associated with a. The result is that "wishful thinking" in $\mathcal{X}^+(P_c)$ about the applicability of actions does not translate into "wishful thinking" about the action effects. This is crucial for the heuristics obtained from $\mathcal{X}^+(P_c)$ to be well informed. In Contingent-FF, a similar transformation is used, where preconditions in P are moved in as conditions (Hoffmann and Brafman, 2005b), the difference being here that preconditions are relaxed only with regard to the sensing actions' effects, maintaining so the central role of sensing in the contingent setting.

Second, rather than making the preconditions of an action a in $\mathcal{X}^+(P_c)$ empty, we add literals ML for each precondition L of a in P expressing roughly that L must be known in *some* branch of the contingent plan. The ML literals are added by a suitable relaxation of the sensing actions $obs(L)$

$$obs(L) : \neg KL \wedge \neg K\neg L \;\rightarrow\; ML \wedge M\neg L \wedge\; o(L) \tag{6.4}$$

that makes *both ML and $M\neg L$ true* after observing the truth value of L. The model is enriched by deductive actions similar to the ones used for the KL literals in equations (6.2) and (6.3) :

$$\mathbf{M-CM}: \quad \bigwedge_{\substack{t,t' \in m \\ t' \neq t}} M\neg t' \rightarrow Mt \tag{6.5}$$

$$\mathbf{M-TR}: \quad KL/t \wedge o(L) \rightarrow M\neg t \tag{6.6}$$

$$\mathbf{M-K}: \quad KL \rightarrow ML \tag{6.7}$$

with $o(L)$ representing an atom that is true when the action $obs(L)$ has been done. Moreover, for each action a in P with effects $a : L_1 \wedge \ldots L_n \rightarrow L$, the M-action $a : ML_1 \wedge \ldots \wedge ML_n \rightarrow ML$ is added as well in $\mathcal{X}^+(P_c)$.

The classical planning problem that results from these extensions is what we call the *heuristic model* $\mathcal{H}(P)$:

Definition 6.17 (Heuristic Model). *$\mathcal{H}(P)$ is the classical planning problem given by the relaxation $\mathcal{X}^+(P_c)$ extended with the ML literals into the preconditions for each precondition L in P, the encoding in eq.(6.4) of the sensing actions, the actions **M-CM**, **M-TR**, and **M-K**, and the rules $a : MC \rightarrow ML$ for each rule $a : C \rightarrow L$ in P.*

The sensing actions $obs(L)$ in the form (6.4) are part of this classical model $\mathcal{H}(P)$ as they are needed for achieving the ML preconditions of other actions. The ML literals in the model $\mathcal{H}(P)$ arise from the observations of the KL literals, and get propagated by the M-actions and deductive rules.

Example 6.4. *To illustrate the heuristic model we can consider an action a with precondition L and effect $L_1 \rightarrow L_2$ in P. In $\mathcal{H}(P)$ then a will be an action with precondition ML and effects:*
$ML_1 \rightarrow ML_2$, *and*
$KL/t \wedge KL_1/t \rightarrow KL_2/t$, *for all tags t in $\mathcal{H}(P)$.*

The CLG action selection mechanism

The *Closed-Loop Greedy (CLG)* planner uses the $X_1(P)$ translation, called now the *execution model*, for keeping track of beliefs, and the *heuristic model* $\mathcal{H}(P)$ for selecting the actions to do next in a closed-loop fashion. Starting with the initial state s of the problem $\mathcal{X}(P) = X_1(P)$, an action sequence π is selected for application in s, and the loop resumes from the resulting state s', until s' is a goal state in $\mathcal{X}(P)$. The action sequence π is obtained using a modified version of the classical FF planner. In FF, a single enforced hill climbing (EHC) step is a local search that results in an action sequence π that maps a state s to a state s' with a better heuristic value h_{FF}. In this local search, the classical model is used for doing the state progression, and its delete-relaxation for computing the relaxed plans. In CLG, the same search strategy is adopted, with the execution model $\mathcal{X}(P)$ used for the progression, and the heuristic model $\mathcal{H}(P)$ used for computing the relaxed plans. In addition, in order to avoid the consideration of non-deterministic actions in the local search, whenever a "local plan" π that ends in a sensing action $obs(L)$ is being considered, the action sequence π is returned without further evaluation. Notice that for an action to be considered into the local plan, the action must have been found to be "helpful" according to FF's criterion.

The CLG planner can be used *on-line* or *off-line*. In the first case, the non-deterministic actions $obs(L)$ in $\mathcal{X}(P)$ are applied by selecting one of outcomes randomly[9]; in the second, both outcomes are considered. In both cases, CLG is invoked recursively on the resulting states. In on-line mode is used for capturing single executions: the planner behaves as a reactive platform, interleaving sensing with planning actions. In off-line mode, the planner is used for constructing *full contingent plans.* While an off-line plan is a tree-shaped sequence of actions and observations, forking on sensing outcomes, an on-line solution is simple a sequence of actions and observations as the observation outcome is provided directly by sensing on the environment.

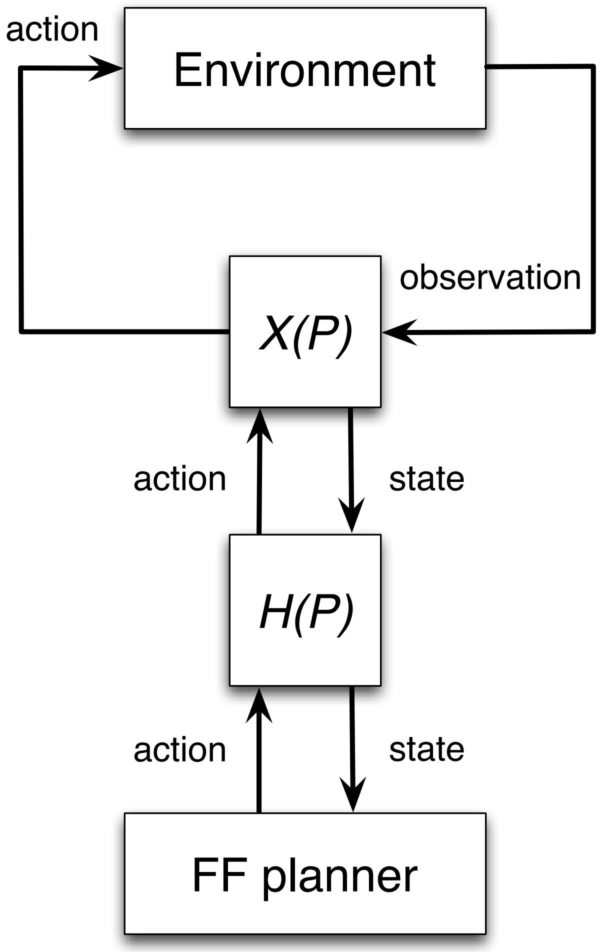


Figure 6.3: The closed-loop greedy action selection mechanism.

Implementation

CLG is implemented on top of a revised version of FF v2.3 (Hoffmann and Nebel, 2001) that loads a single PDDL file where convenient flags are used for distinguishing the $\mathcal{X}(P)$ and $\mathcal{H}(P)$ models.

The translations used in CLG accommodate certain simplifications and some additional actions that are in line with general encoding, and make explicit certain types of deductions for efficiency purposes:

1. for each static disjunctions $c = (L_1 \vee \cdots \vee L_n)$ in P, KL_i is derived in $\mathcal{X}(P)$ from the following deductive rule:

$$\bigwedge_{\substack{i \neq j \\ L_i, L_j \in c}} KL_i \rightarrow K\neg L_j$$

2. A generalization rule subsumed by contingent merge and the static merge above, for all tags t we add the rule:

$$KL/t \wedge Kt \rightarrow KL$$

The translations considered are all *uniform* in the sense that all literals L in P and all rules $C \rightarrow L$ are "conditioned" by each of the tags t in T. From a practical point

[9] To run the tests shown in Table 6.2 the random selection of the sensing outcome has been applied, however the implementation allows manual selection of the outcome, or sensing accordingly to a given initial hidden state.

of view, however, this is not needed. The literals KL/t encoding the belief in P are then only generated when the closure t^* contains[10] literals relevant to L, or if t is in the set $O(L)$. In this doesn't happen, the tag t cannot "affect" the literal L, i.e. the invariance $KL/t \supset KL$ holds for any t, and thus every occurrence of the literal KL/t in $K_{T,M}(P)$ can be replaced by KL.

After applying each action in a sequence π in $\mathcal{X}(P)$, the resulting state s' is closed under the deductive K-actions above, the contingent merge, and the tag refutation. The closure of the state is performed by translating the fluents in the state and the deduction rules in SAT clauses, and their application until fixpoint is performed by a single call to the `MiniSAT` SAT solver (Een and Sorensson, 2004). This implementation shrewdness that boosted the empirical results of the CLG planner, is possible because all the deducing rules *only add new literals without deleting any.*

From the implementation of T_0 (Palacios and Geffner, 2006), on which we based the translator for $\mathcal{X}(P)$ and $\mathcal{H}(P)$, we inherited other simplifications, namely:

- The support rules $a: \;\; KC/t \rightarrow KL/t$ for non-empty tags t are not created when L is not relevant to a literal L' within a merge that contains t: in such a case, the literal KL/t cannot contribute to establish a precondition or goal. Similarly, cancellation rules $a: \; \neg K\neg C/t \rightarrow \neg K\neg L/t$ for non-empty tags t are not created when $\neg L$ is not relevant to a literal L' in a merge that contains t.

- The support and cancellation rules
$a: \; KC/t \rightarrow KL/t$, and $\quad a: \; \neg K\neg C/t \rightarrow \neg K\neg L/t$
are grouped in a single rule $a: \;\; KC/t \rightarrow KL/t \wedge \neg K\neg L/t$ every fluent L' relevant to L is such that either L' or $\neg L'$ belongs to t^*. In such a case, there is no incomplete information about L given t in the initial situation, and thus the invariant $KL/t \vee K\neg L/t$ always holds, bringing that to write $\neg K\neg C/t$ is equivalent to KC/t, given that C is by construction relevant to L.

These rules are computationally sound, and appear to help in certain domains without hurting in others.

Empirical evaluation

CLG has been tested over a broad range of problems comparing it with Contingent-FF (Hoffmann and Brafman, 2005a) and Pond 2.2 (Bryce et al., 2006). The planner Contingent-FF appears to scale up better than other contingent planners such as POND, while planners such as MBP use a different modelling language. We used Contingent-FF with two options (with or without helpful actions), reporting the best option for each instance. Pond was run with the A^* search algorithm. The experiments are obtained on a Linux machine running at 2.33 GHz with 2Gb of RAM with a cutoff of 45 mn or 1.8Gb of memory.

Some of the benchmarks are taken from the Contingent-FF distribution, like *ebtcs*, *elog*, *medpks*, and *unix*; others are more challenging problems from our own: *cballs-n-x*, about disposing of x balls with unknown location and colour into boxes according to their colour; *doors-n*, about moving in a $n \times n$ grid with hidden doors, *localize-n*,

[10]To recall the notion of closure, cf. to definition 4.12.

about robot localization in a known map, and *wumpus-n*, a variation of the Wumpus world (Russell and Norvig, 2002) where a monster hides in a cave and it has to be found from its stench that can be smelled from the adjacent caves. Last, *clog* is a variation of *elog* where all action conditions have been moved out as preconditions. In this suite of problems, only two domains have contingent width greater than 1: *cballs*, and *wumpus*.

	CLG in Off-line Mode					
	Contingent FF		POND		CLG	
problem	**time**	**#acts**	**time**	**#acts**	**time**	**#acts**
ebtcs-50	11,96	99	6,02	99	4,93	149
ebtcs-70	69,66	139	29,82	139	21,32	209
elog-7	0,06	223	1,10	212	0,07	210
eloghuge	M		M		240,14	38894
medpks-50	164,94	51	100	3,23	2,32	101
medpks-70	1098,44	140	T		8,12	141
medpks-99	T		T		44,28	199
unix-2	0,09	48	2,18	48	0,23	50
unix-3	4,10	111	M		3,56	113
unix-4	222,65	238	M		156,47	240
cballs-4-1	0,27	277	0,98	102	0,20	295
cballs-4-2	35,88	18739	40,92	1897	18,03	20050
cballs-4-3	T		1063,11	28008	1603,63	1136920
cballs-10-1	T		M		415,73	4445
cballs-10-2	T		M		T	
cballs-11-1	T		M		1172,96	5884
cballs-11-2	T		M		T	
localize-5	9,8	188	T		0,72	137
localize-7	MC		T		3,80	314
localize-11	MC		T		58,88	577
localize-13	MC		T		M	
clog-7	E		1,12	212	0,08	210
clog-huge	E		M		248,44	37718
doors-7	E		21,48	2159	6,32	2153
doors-9	E		1432,34	44082	1036,18	46024
doors-11	E		T		T	
wumpus-5	E		5,58	587	0,39	754
wumpus-7	E		703,54	11673	7,90	6552
wumpus-10	E		M		2125,65	321268

Table 6.1: Full Contingent Plans: Contingent-FF vs POND vs CLG. Figures shown are total times in seconds, and total number of actions in solution. *M, T, MC* and *E* refer to memory out, time out, too many clauses, and buggy response, respectively.

Table 6.1 displays the ability of CLG to build *full contingent plans* which compares favourably with the other two planners. Times reported stand for total time, and in the case of CLG, they include the translation time. The quality of the plans appears to be comparable (except for *cballs*). Domains marked with an 'E', are reported as unsolvable by Contingent-FF without any search. After checking with Hoffmann and Brafman, it appears that this is due to a bug that results from simplifications made in Contingent-FF that are not easy to fix.

Table 6.2 shows average results for CLG used in execution mode over a sample of 50 random executions for each problem. The first two columns of Table 6.2 show the

	CLG in Execution Mode				
	Translation		Search Time	#acts	ratio
problem	**time**	**size**	**avg / max**	**avg / max**	**n ÷ a**
ebtcs-70	7,7	12,9	0,62 / 3,48	12,5 / 71	1,0
elog-huge	1,1	1,7	0,63 / 1,43	44,25 / 59	1,8
medpks-99	13,7	21,6	0,54 / 3,30	15,1 / 100	1,0
medpks-150	48,7	65,0	4,26 / 28,60	20,8 / 150	1,0
unix-4	27,9	87,2	10,88 / 70,03	27,0 / 181	1,9
cballs-9-1	20,9	16,5	1,21 / 7,80	33,7 / 197	1,6
cballs-9-2	56,4	33,7	4,84 / 25,70	57,1 / 288	1,6
cballs-9-3	113,7	51,4	46,26 / 122,19	76,3 / 367	1,7
clog-huge	1,0	1,6	0,44 / 0,71	48,2 / 69	2,6
doors-9	6,0	8,1	0,72 / 1,46	34,2 / 80	1,1
doors-11	19,4	20,0	2,58 / 6,37	42,8 / 120	1,1
doors-13	52,4	44,7	6,78 / 20,68	53,0 / 178	1,1
doors-15	127,5	90,2	MP		
localize-9	4,6	8,9	0,37 / 0,60	21,3 / 34	1,0
localize-11	12,1	20,4	M		
wumpus-7	2,4	4,5	0,42 / 0,56	38,5 / 46	1,6
wumpus-10	12,1	16,8	3,26 / 5,59	57,5 / 86	1,8
wumpus-15	101,1	80,7	M		

Table 6.2: CLG in Execution Mode: Averages over 50 samples. Figures shown are time and size of translation, avg and max search time for execution, avg and max number of actions in execution, and ratio of nodes per actions executed in local EHC search. *M* stands for memory out, and *MP* for too many predicates. Times are in seconds.

time consumed in the translation from P to $\mathcal{X}(P)$ and $\mathcal{H}(P)$, as well as the size of the resulting PDDL file in MBytes. All the executions end up in the goal with the number of actions shown in the fourth column. The last column shows how focused is the search for the next action to apply, and more precisely, the number of nodes expanded in the local EHC search vs. the number of actions executed. Indeed, a ratio of one means a very focused search. The results show that CLG in execution mode can solve problems for which building a full contingent plan is not feasible due to its size. For example, the largest *door* instance solved in off-line mode is *doors-9* that results in a policy tree with more than 46 000 actions; in on-line mode CLG solves *doors-13* that is much larger. The same is true for other domains like *cballs* and *wumpus*.

6.6 Discussion

Contingent planning has been addressed by using many different ways to represent belief states, as their explicit representation is impractical due to their possible exponential size. The way belief states are encoded have an impact on the performances of the planner, in contingent planning as in conformant planning. We discussed about the representation of belief states in section 2.5 and section 4.7, but we will broaden here the analysis to the context of contingent planning.

The Contingent-FF planner by Hoffmann and Brafman (2005a) adopts an implicit representation of the beliefs, borrowed from (Brafman and Hoffmann, 2004). Belief states are represented through the action sequences leading to them from the initial belief state. The fluents that are true in a belief state are then computed by the verification procedure that we used for the T1 planner in chapter 4. This lazy approach is enough to perform conformant planning, as to apply actions and check the goal state only the precondition and goal literals need to be verified to be true in all the states of the belief. This procedure is extended to contingent planning by considering action-observation sequences, and the construction of the formulæ encoding that capture the semantics of the action sequence applied to the initial situation needs only to be slightly modified: the verification algorithm is still co-NP complete, and is still done by CNF reasoning. This representation uses very little memory, but the trade-off is made with the computation time expended to compute the known literals in each belief. in particular, this approach suffers when the actions' structure or the unknown fluents make the verification problem harder.

MBP (Bertoli et al., 2001) and POND (Bryce et al., 2006) use a OBDD-based representation for belief states. OBDDs are generally compact, and each visited belief state is represented by a unique OBDD. The transitions between belief states can be described in logical terms, and are encoded in MBP and POND by means of BDD-based transformations, e.g. union, projection and intersection. OBDDs provide for an efficient implementation of these operations needed for manipulating belief state (Cimatti et al., 1998). Still, the size of an OBDD can be exponentially large in the number of variables, and intermediate formulæ of exponential size used to compute successors states can be required during the search, making the operation expensive.

A different approach has been recently used for the DNF_{ct} contingent planner (To et al., 2011b). There, a DNF representation is used to encode belief states, along with a variant of the AND/OR search algorithms which includes some techniques to prune the search space. The DNF encoding in DNF_{ct} is compact and permits the planner to scale up well. However, as in many other contingent planners, the heuristic function based mostly on the number of achieved subgoals does not help much DNF_{ct} in the search for a solution. The DNF representation can be very large in those domain that include a large number of clauses in $\mathcal{I}$. The scalability of the planner is obviously affected by these contingencies; for example in the *wumpus-10* instance, the initial DNF contains 2 567 504 partial states.

The assumption in most of related works is that the hidden preconditions become observable, and hence known to be either true or false, in states where the non-hidden preconditions hold. Under this assumption, a greedy strategy can solve the planning problem by first making the most convenient assumption about the values of the hidden variables, and then execute the plan that is obtained from the resulting classical planning problem, and revise the assumptions and replan, if during the execution the observations gathered refute the assumptions made. This method works for on-line planning, and has been proposed recently by Shani and Brafman (2011). In their SDR planner, a sample is taken from the initial belief state is and a single initial state s_i is selected, then a plan is generated as if s_i was the real initial state. This approach has some points in common with the sampling used for conformant planning in chapter 4, where a subset of states in the initial belief was selected and used for planning, but in SDR nothing guarantees that the encountered plan is a correct solution for all the states in the initial belief. In fact the technique applies only to produce single branches of the complete plan, in what we called *on-line mode* for CLG.

Another limitation of many approaches to contingent planning comes from the heuristics used to guide the search. These heuristics are generally poorly informed, a deficiency that partially comes from the belief representation employed, that do not allow to extract efficiently informed heuristics. We overcame this issue with the heuristics model $\mathcal{H}(P)$, used for computing a classical plan relaxation that provides useful heuristics.

Sensing actions $obs(L)$ are naturally integrated into CLG, via the relaxation about action preconditions in the heuristic model, and the introduction of the M-literals in the preconditions. The ML literals are added in the relaxed problem by a suitable relaxation of the sensing actions $obs(L)$ which allow the sensing actions to be considered, and included in the relaxed plan when they appear to be necessary to trigger the preconditions of an action. In Contingent-FF, the heuristic is derived from the delete relaxation of the classical planner FF (Hoffmann and Nebel, 2001) sensing actions, and sensing actions are not directly considered by the heuristic, which is a major drawback for solving contingent planning problems. In the SDR planner, sensing will never by used purposely, as considering a single initial state end up in classical planning, where full information is assumed. Actions are then artificially forced into the plan: at each step, if it is possible to sense the value of an unknown proposition without affecting the state, the correspondent observation is performed.

The translation K_0, even if normally incomplete, has been proved to be effective in *simple problems*, defined as deterministic planning problems where the non unary clauses in $\mathcal{I}$ are all invariant, and no hidden fluent appears in the body of a conditional effect. The extension K_0' of the translation K_0 has been applied to simple partially observable problems P, where policies for the fully–observable deterministic problem $K_0'(P)$ can be mapped into actions applicable in P. $K_0'(P)$ is always sound, and complete is P is simple and connected (Bonet and Geffner, 2011). In partial observable multi-agents planning, a translation based on K_0 has been used to search for plans involving two agents. We will discuss more about this application in chapter 8.

6.7 Summary

We have extended the translation-based approached to conformant planning introduced by Palacios and Geffner, to planning with sensing. In both settings, the translation maps search problems in belief space into search problems in state space with complete translations being exponential in a width parameter that is 1 for most benchmarks. We have also tested these ideas empirically by formulating a contingent planner CLG that uses the new translation along with a suitable relaxation, and have found that the planner scales up better than existing contingent planners, and that when used in on-line mode, it can scale up to problems for which the construction of full contingent plans is not feasible. Two advantages of the translation-based approach are that it results in compact belief representations that are often complete, and classical plan relaxations that provide useful heuristics.

The $X_i(P)$ translation is incomplete for problems P with high contingent width: this means that to guarantee that a solution is always found for $X_i(P)$, the width of the problem should bound the parameter i, related to the size of the tags used in the translation. The CLG planner implements only tags of size one, a choice made because the translation $X_i(P)$ for $i > 1$ is exponential in the parameter i, which makes a planner built on this translation unable to scale up. It is thus possible that CLG outputs an infinite heuristic value for the initial state in the problems with width bigger than 1, even when the planning problem does have a solution.

But there are other classes of problems that actual contingent planners cannot solve. These are the planning problems that have a dead-end, i.e. the initial belief state includes some states for which no solution is possible. This can happen when certain configurations of the domain are not solvable, and no information is available initially to shine light on the incomplete information of the domain, so the goal might be effectively reachable even if the information is hidden. In other cases no policy can deal with all the possible states in the initial belief state, even though none of these states is a dead-end: any taken policy will work for some initial states, but not for others.

Such problems are not solved neither by contingent planners (they answer that no solution is reachable), nor by POMDP-based solvers (the expected cost of reaching the desired state is infinite as there is a non-zero probability that the paths to the goal are actually blocked). However, a contingent planner should not freeze in such situations, but plan in order to gather information to eventually solve the problem. We will see that it is easy to adapt our model for contingent planning to solve these classes of "unsolvable" problems. *Assumptions* can be automatically cast to make the heuristic finite, and to plan toward the goal, gathering information where sensing is available.

Chapter 7

Planning when goal reachability is not guaranteed

In the translations we developed in the previous chapters, the search of a plan ends without a result when the classical planner called on the translated problem answers that a solution is not satisfiable.

In planning with incomplete information and sensing, it might be impossible to find a strong solution simply because the sensors do not capture the information needed to reach the goal with certainty, or because this information is not available initially. Consider for instance, a robot that has to cross a room, without knowing the position or the nature of the obstacles it can find on its path. In this case, no solution can be provided in initially, as the path to the goal might be blocked. However, it is desirable for the robot to move anyway and gather information in order to find a way to the goal.

In this chapter we address the problem of planning in partially observable environments, with uncertainty encoded in the initial situation, when no plan guarantees to reach the goal from any possible initial state. We attempt to produce a solution even in these cases by automatically generating *assumptions* on the planning domain that allow the planner to produce a potential solution anyway. These assumptions are derived from the very structure that constitute the basis of the $K_{T,M}(P)$ translations: the set of tags and merges.

Results from this chapter have been published in *Acting in Partially Observable Environments When Achievement of the Goal Cannot be Guaranteed*, by A. Albore, H. Geffner, in Workshop on Planning and Plan Execution for Real-World Systems, at 19^{th} International Conference on Planning and Scheduling (ICAPS-09), Thessaloniki, Greece, 2009 (Albore and Geffner, 2009).

7.1 Motivation

Planning in partially observable environments can be regarded as a search problem in belief space where beliefs express the collection of states that are deemed possible. We address the problem that arises when the goal cannot be reached at least from some of the possible initial states.

A large number of relevant planning problems require the ability to deal with partially observable domain models. Planning under these conditions is an extremely hard task, and often strong plans that guarantee reaching the goal in spite of incomplete run-time information do not exist. This extreme case of planning under uncertainty is sometimes called “hard partial observability”, to express the difficulty to plan when (partial) observability on the domain’s variables do not permit to achieve the goal; in other words, the hidden state of the environment is not observable. In such situations, no contingent plan exists, neither when considering the introduction of probabilities. Moreover, no contingent planner provides any mean to carry on, gathering observations in order to eventually solve the problem. Nevertheless, in many cases it is possible to express reasonable assumptions over the planning domain, so to ease the planning task, allowing to build assumption-based solutions that achieve the goal under certain conditions.

One option in such cases is to find contingent plans or policies that maximise “coverage”, i.e. the set of possible states for which the solution under assumptions works. To do so, we extend the action-selection mechanism of CLG for contingent planning (seen in the last chapter) to generate in an automated way the assumptions under which a solution exists. The assumptions are selected to plan for as many initial states as possible, when finding a plan for every possible initial state is not doable.

We will show that scenarios where the planning agent “freezes” without finding any solution by lack of information are common, and that the proposed mechanism helps overcoming this difficulty, having at the same time other applications as well: we will use the machinery for planning under assumptions to learn action models in those domains where actions outcomes are not specified.

7.2 Planning when missing information is not always observable

Consider the problem of a robot that has to move from one position to another position in a grid, not knowing which of the cells in the grid are free. The robot can move into an adjacent cell, and can also sense whether it is free or not. The problem can be expressed as a contingent planning task and fed easily into a state-of-the-art contingent planner like CLG (Albore et al., 2009). But the planner will not help, because the problem has no contingent solution. Indeed there are *contingencies* in the problem that prevent to have a plan that guarantees the robot to reach its destination with certainty; namely, all the possible configurations where every single path to the goal is blocked by no-free cells. Still, in the absence of contingent plans, it is not desirable that the robot freezes without being able to search for a solution plan. What it should do instead, is to move toward the target and give up only when one of these possibilities turns out to be true, i.e. when effectively no path leads to its destination. Contingent planners do not help in producing such behaviour, answering that no solution exists.

This limitation does not apply only to contingent planning, but to many other models of action selection that require the goal being achievable with certainty. The problem can be cast also as a POMDP by filling in the probabilities that each cell is free and maintaining the goal of reaching the target with certainty (Cassandra et al., 1994). Even then, the expected cost of reaching this target belief will be infinity as there is a non-zero probability that the paths to the goal are actually blocked [1]. So probabilistic POMDP solvers will not help in this problem either.

Dead-ends

Since contingent and POMDP problems can be cast as search problems in belief space (Åström, 1965; Bonet and Geffner, 2000), the hard partial observable problems above correspond to situations in which certain belief states are *dead-ends*, i.e. belief states from which the goal beliefs cannot be reached with certainty. In the robot navigation example, the initial belief state is obviously a dead-end. In the contingent setting, the belief states represent the set of all the states deemed possible at one point (cf. section 2.1), so the states in the example encode the position of the agent and the status of the cells (which can be free or not). The initial belief state is a dead-end because it contains states where the non-empty cells block the paths to the goal. Those states are dead-ends in themselves as the goal cannot be reached from any of them, although the state is assumed to be completely observable. In contingent planning or in POMDPs, a belief state containing a dead-end state is itself a dead-end. Even in these cases, it is possible that the problem is intrinsically solvable, but the information initially available doesn't allow the planner to provide a solution, which is surely a weakness of those formulations, even if such situations are very common.

When planning with incomplete information, dead-end belief states do not always arise only from dead-end states. A problem can be unsolvable if no policy reaches the goal from any of the states in the initial belief, even though none of the initial states is a dead-end. This is the case of the medical domain in the example below.

Example 7.1. *Let's consider the medical domain, first introduced by Weld et al. (1998), where a patient may suffer of one of several illnesses. Every illness has a cure, but the cure is deadly if applied on an illness different from the one it has been designed for. It is possible to diagnose the illness by exploiting diagnosis actions: measuring the white cells in the blood, or using a litmus paper.*

Here, if the observations available discard all but two possible diseases, and no therapy works for both diseases at the same time, the belief is a dead-end even if none of the states in it are, as each disease can be treated separately.

Another possible cause of dead–end can arise from translation-based approaches to planning under uncertainty: we saw that some translations are complete only under certain circumstances. When these circumstances are not full-filled, it is possible that the classical planner evaluates as unsolvable a translation of a solvable problem.

When the achievement of the goal cannot be guaranteed, one option is to find contingent plans or policies that maximise "coverage", i.e. the set of possible states for which the solution works. These policies are well-defined except in the very unfortunate situations where each of the states that are deemed possible is a dead-end. This

[1] Costs and rewards are often discounted in POMDPs: this results in bounded expected costs and rewards. Yet discounting, which is a convenient mathematical mechanism, is not always a meaningful one: for example it can produce finite costs in reachability problems with no solutions. In any case, in our example, even if discounting can be used to bound the expected costs, it doesn't result in meaningful policies.

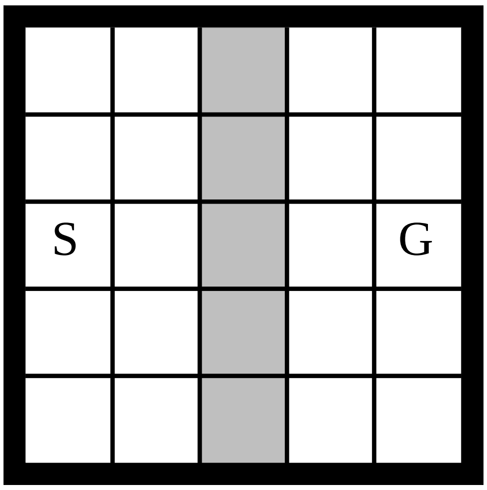


Figure 7.1: A 5×5 grid.

is the approach that we take here. We extend the action selection mechanism for contingent problems described in section 6.5 to work in such settings and producing an action selection mechanism for on-line planning. Through a number of examples we will see that this kind of scenarios are frequent in planning, and the proposed extension of our translation for contingent problems has other applications as well. In particular, it can be used to generate meaningful, goal-oriented behaviour in solvable but complex contingent settings where state-of-the-art contingent planners fail.

Planning under assumptions

From the POMDP perspective (cf. section 2.6), a dead-end belief state is just a belief state b with infinite cost; this reflects that a goal belief state can't be reached with certainty from b, even supposing full state observability.

Dead-end states s have infinite optimal costs $V^*(s)$ and they induce an infinite optimal cost on any belief state b that include them: it is easy to show then that $V(b) \geq V^*(s) = \infty$. But these results do not reflect whether the problem is effectively solvable or not, in fact many other states s' in b can have a finite optimal cost, in spite that the whole belief appear as unsolvable.

Yet acting should not be stopped when one of the possible scenarios is a dead-end but rather only when these possibilities are the only ones left, i.e. when there is the certainty that no plan can solve the problem. A possible principle for guiding the search of a solution is to follow a policy obtained from a relaxation, where the current belief is augmented with some assumptions aiming at taking away those states that are actually dead-ends. These assumptions can then be confirmed or denied as a result of following such a policy. This process can be iterated until the goal has been reached or it has been found out to be effectively unreachable. This is actually the approach adopted by Albore and Bertoli (2004), but in our framework those assumptions will arise naturally from the planning process.

Example 7.2. *Figure 7.1 shows an instance of the example used in the introduction: a 5×5 grid with cells marked in grey that may be free or not, and with the agent initially to the left of this line of cells, while the goal is on its right side. The status* *of those cells is not known to the agent, but the agent can move one unit in each of the four directions, if the corresponding adjacent cell is free, and it can sense the status of the adjacent cells.*

The problem involves in the order of $5^2 \times 2^5$ *states:* 5^2 *possible positions for the agent, and* 2^5 *possible configurations of the cells in grey. Some of these states are not reachable, like those where the agent is at one of the cells in grey and this cell is not free. The dead-end states for this problem are quite few indeed and correspond to those in which the agent is to the left of the line of grey cells, and none of these cells is free. These are* 10 *states, and one of these* 10 *states are part of the initial belief state* b_0*, where the agent knows its position but does not know the status of the cells in grey. This yields* b_0 *to be a dead-end belief state, so that the problem has no solution.*

We fed this problem into some recent contingent planners. Contingent-FF, for example, notices right away and without search that the problem has no solution and immediately quits. POND*, on the other hand, starts searching for a solution and keeps searching for quite a while, without obviously finding it. CLG behaves like Contingent-FF, producing an infinite heuristic value for the initial belief state and quitting without doing any search.*

In this case, the responses of both Contingent-FF and CLG are adequate if the objective is to generate a plan that considers all the possibilities. However, this is not a reasonable objective nor a requisite for an agent acting in partially observable environments.

In the absence of a policy that will work in all cases, the strategy that makes sense is to follow the policy that will work in most cases, revising the policy as new observations are gathered. This means actually to compute a policy assuming the current belief not to be b, if b is a dead-end, but a belief b' that differs minimally from b and is no longer a dead-end. In the example 7.2, this "alternative" initial belief state b'_0 can be possibly obtained in two ways. One is by excluding from b_0 the states where all the grey cells are blocked. Another non-minimal change that will work as well, is to keep in b'_0 only the states in which one specific grey cell is free: this change amounts to making the assumption, rather than the observation, that such a cell is free. Planning considering this assumption will result in a policy that, if followed, might turn out to prove the assumption wrong, which can then be revised and replaced by another one if the resulting belief state still remains a dead-end. For this to work, a reactive on-line planning platform must be used.

Formally, an assumption is a formula that restrict the initial belief state; this is done to obtain a solution from the resulting planning problem under the assumption.

Definition 7.1 (Assumption). *Given a contingent planning problem* $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, an* assumption t *is a valid formula over* $\mathcal{I}$ *such that*

$$\mathcal{I} \wedge t \not\models \emptyset \tag{7.1}$$

Definition 7.2 (Assumption-based solution). *Given a contingent planning problem* $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$*, and an assumption* t*, we say that a contingent plan* π *is an* assumption-based solution *for* P *if* π *is a solution for the problem* $P|_s$*, which is the problem* P *with initial state equal to* s*, for all state* s *compatible with the assumption, i.e.* $\mathcal{I} \wedge t \models S_0$ *and* $s \in S_0$*.*

Otherwise stated, the plan π is a solution for the contingent problem P' that is equal to P but with the initial belief state restricted by t, i.e. $P' = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I} \wedge t, \mathcal{G} \rangle$.

Our scope is to permit to cast assumptions on the initial situation such that an assumption-based solution works for a maximum number of states. These assumptions will be included in our translated problem as *assumptive actions*. The assumptive actions will be introduced incrementally in the problem's model to avoid to execute them as first instance, solving so the problem by "assuming that it is solved". These actions have a high cost, which, coupled with a cost-sensitive heuristic, will provide solution plans that make a use as limited as possible of the assumptive actions.

In particular, we will see that not always an assumption is a "bet" on the domain behaviour: when enough sensing is available at execution-time, the validity of the assumption can be verified and, if it is necessary, replanning episodes can be triggered. An assumption is *monitorable*, or verifiable, if at execution time it is possible to observe its truthfulness.

7.3 The CLG+ Planner

To solve contingent planning problems where the goal is initially not reachable, we will introduce assumptions under which a solution might exist. Assumptions will be encoded as actions with high cost in the CLG planner, used in on-line mode. The consequence is that plans including actions that code assumptions will be produced as last option when generating relaxed plans. Thus, cost optimisation will result in plan strategies that are as strong as possible.

The advantages of building on CLG are three.

First, in many of the problems the bottleneck for contingent planners is not the lack of solutions, but the *size of the solutions*, that is exponential in the number of possible observations. For example, the solution to the robot navigation problem above, once fixed to ensure the reachability of the goal, will have a size that is exponential in the number of cells that sit between the initial and the goal locations. The ability of CLG to produce goal-oriented observation–action sequences without having to build a complete contingent plan is thus a plus, and CLG does not fail scaling up in these situations.
Second, in on-line mode the CLG planner does not build a full contingent plan, but becomes an action selection mechanism that outputs the next action to do given the past history of actions and observations. Even if on-line mode generates a single branch of the full plan, the heuristic implicitly considers all the possibilities: that makes perfect sense if the choice is between an action that works in all cases, and one that works only on some, but fails when the choice is between actions none of which works in all cases.
Third, the use of tags t in the underlying translation, that denote assumptions about the initial situation, can be used to render an unsolvable problem into a solvable one, and to find contingent plans or policies that maximise "coverage". This maximisation will be done heuristically, by extending the translation $\mathcal{X}(P)$ with actions that can manipulate these assumptions, at a high cost. The selection of actions that are

compatible with the plans that maximise coverage will be obtained from the relaxation $\mathcal{H}(P) = \mathcal{X}^+(P)$ using standard heuristic functions that are sensitive to costs, ending by selecting assumption-based actions only when necessary to minimise the cost of the plan.

Our adaptation of CLG action selection mechanism exploits the translation that compiles beliefs away while translating contingent problems P into non-deterministic but fully observable problems $\mathcal{X}(P)$, whose literals encode the beliefs over P. In these translations, a tag is a set of literals in P whose status in the initial situation $\mathcal{I}$ is not known. Tags are assumed to represent consistent assumptions about $\mathcal{I}$, i.e. $\mathcal{I} \not\models \neg t$. We recall that the fluents in both $X_{T,M}(P)$ and $K_{T,M}(P)$, for a conformant fragment $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ of a contingent problem, are of the form KL/t for each fluent $L \in \mathcal{F}$ and tag $t \in T$, meaning that "if t is true in the initial situation, then L is true in the current situation".

These changes result in an action selection mechanism that we call CLG+, and whose difference with CLG is that it works in contingent problems that can have a priori no solution. Thus CLG+, like CLG, does not "freeze" due to the size of the contingent solutions, but unlike CLG, it does not "freeze" either due to their absence. As long as there is the possibility of reaching the goal, CLG+ will go for it, making it a quite robust, persistent, and fast action selection mechanism for dealing with partially observable environments.

The planner CLG+ uses the same execution and heuristic models of CLG (cf. chapter 6), with the addition of actions to introduce assumptions on the environment behaviour, and the algorithms needed to handle them.

Assumptive actions

The key idea in CLG+ is to introduce assumptions in the form of new *assumptive actions* in the translated contingent problem. The available assumptions are the tags of the problem, which are also the conjunction of literals relevant to the preconditions and the goal. So, in CLG+, to introduce an assumption is to make available to the planner a conjunction of literals needed to solve the problem.

Definition 7.3 (Assumptive action)**.** *Given a contingent planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, and a translation $X_{T,M}(P)$, the assumptive actions a_t of the problem are such that*

$$\begin{aligned} & a_t : \\ & \quad \text{precondition} : \neg K\neg t \wedge \neg Kt \\ & \quad \text{effect} : \quad \rightarrow K\neg t \end{aligned}$$

for all tag $t \in T$.

These actions a_t form part of both the execution model $\mathcal{X}(P)$ and the heuristic model $\mathcal{H}(P)$ of the planner. The reason for having assumptions adding the negation of a tag $K\neg t$, rather than directly "knowing that the tag itself is true" Kt, is that the former are finer grained. For example, in a problem involving an initial situation with an $xor(x_1, \ldots, x_n)$, where the tags are $t_i = x_i$, we will be able to discard some x_i without necessarily committing to another one.

The result of having such actions is evident: if no plan exists given an initial situation $\mathcal{I}$, maybe a plan exists for another initial belief $\mathcal{I}' = \mathcal{I} \wedge t$. There is however a difference in the use of assumptive actions in the execution model $\mathcal{X}(P)$ and the same assumptions in the heuristic model $\mathcal{H}(P)$. Recall, that the heuristic model is used to compute heuristic values and relaxed plans, and to indicate the actions that are helpful. On the other hand, states are progressed using the execution model, both when they are finally selected and applied, and when they are applied in the enforced hill climbed search (EHC). The difference between an assumption $K\neg t$ in $\mathcal{H}(P)$ and the same assumption in $\mathcal{X}(P)$ is that the former is just "thinking" (considering the possible situation given the assumption in order to search efficiently for a plan) whereas the latter is "acting" (i.e. executing an action under the presumption that the assumption holds in the real world). In different words, the first just extends the relaxation with an extra assumption, while the second is a commitment that affects the current beliefs.

The asymmetry between assumptions $K\neg t$ in the execution and heuristic models is captured by integrating the assumption in three steps.

1. The actions to be done in the current state s are computed by mean of an EHC search from s, with the helpful actions but *excluding the assumptive actions*. This first step doesn't use at all the actions from definition 7.3

2. If this fails, then this EHC search is repeated, but considering *all* actions and not only the helpful ones, and still without executing any assumption. This mean that no assumptive action will be executed (they are *de facto* not introduced in $\mathcal{X}(P)$).

3. If this fails too, then this same EHC search is carried out with all the actions enabled, including the assumptions in the execution model. It is only then that assumptions may be chosen for execution.

In all cases, as in CLG, if a sensing action is selected for application in the EHC search, the path leading to the sensing action is selected for application. As in CLG, this is done in order to avoid the simulation of the non-deterministic effects of sensing in the EHC search.

From the integration process, it is evident that executing an assumptive action has to be done as last resort: these action modify the belief state, arbitrarily excluding states from it. It is like a "bet" on the domain behaviour, letting the planner to choose which is the subset of the initial belief that will be driven to the goal by the solution plan. As we will see, sometimes the current belief needs to be changed. This can occur when there is a single possible relevant action to apply, but there still is uncertainty about the status of its preconditions, and no observation can help disambiguating it. If the agent is supposed to head for the goal no matter what, as CLG+ does, then it must then take risks: if it applies an action assuming that its preconditions hold, and this is not the case, then the execution will fail.

An essential element in the move from CLG to CLG+ is a device for preventing inconsistent assumptions from producing arbitrary conclusions. Since the deductive actions in CLG are not deductively closed, the merge rules in the heuristic model may result in unwarranted conclusions.

Consider a merge m for a literal L: the contingent merge action $\bigwedge_{t \in m}(KL/t \vee K\neg t) \rightarrow KL$ can be used to derive KL from the assumptions $K\neg t$ applied for all $t \in m$. These assumptions are jointly inconsistent, as it is not guaranteed that L is known to be true under any tags, but nothing prevents them from being made in a "relaxed plan" with a sufficiently high cost. In order to avoid this, we introduce a new fluent in the language, $ok(m, L)$ that we keep true only when *some* literal KL/t is true, and introduce this new literal as an extra condition in the merge action. This prevents the merge m for L to trigger if none of the literals KL/t are true.

A cost sensitive heuristic

The second element needed in CLG+ is the handling of action costs. In order to establish a preference for solutions and relaxed solutions that work in most cases, actions that involve assumptions are penalised with a high cost. This is needed to prefer solutions that make an use of assumptive actions as limited as possible. In particular, in our implementation, the cost of assumptive actions if fixed to $100 \times |T|$, i.e. 100 times the total number of tags[2]. The following step is to integrate a heuristic sensitive to costs, to properly select the actions.

In order to preserve as much of the architecture of CLG as possible, we achieve this in CLG+ by moving the underlying classical planner from FFto FF(h_a^s), a planner that retains from FF everything except the definition and computation of the relaxed plans, that makes use of the *set-additive heuristic* h_a^s (Keyder and Geffner, 2008). We moved to FF(h_a^s) because in FFthe relaxed plans that serve to provide the heuristics and the helpful actions are computed using the relaxed plan graph, which is not sensitive to costs.

In FF(h_a^s), the relaxed plans are computed recursively, very much as the heuristic estimates in the additive heuristic. The difference between the additive heuristic h_{add} and the set-additive heuristic h_a^s, is that the former propagates numbers $h(p; s)$ that estimate the cost of achieving the fluent p from a state s, while the latter propagates labels $\pi_a^s(p; s)$ that capture the relaxed plans for computing p from s. To avoid over counting of operators, the set-additive heuristic h_a^s approximates the optimal relaxed plan for a set of fluents as the union of the relaxed plans for each fluent. The heuristic estimate for a given state s is then the cost of the set-additive relaxed plan π_a^s to achieve the goal fluents $\mathcal{G}$ from s:

$$h_a^s(\mathcal{G}; s) = cost\big(\pi_a^s(\mathcal{G}; s)\big) \tag{7.2}$$

which is given by the following equations:

$$\pi_a^s(p; s) = \begin{cases} \{\} & \text{if } p \in s \\ \pi_a^s(a_p^{sa}; s) & \text{otherwise} \end{cases} \tag{7.3}$$

[2] This number is set to a high value to use assumptions as a last resort. The value of such an action could possibly be accommodated to depend on the number of tags in a merge it refers, but we left this fine tuning of the algorithm to future work.

where

$$a_p^{sa} = \operatorname{argmin}_{a \in ADD(p)} cost(\pi_a^s(a; s)) \tag{7.4}$$

$$\pi_a^s(a; s) = \{a\} \cup \left\{ \bigcup_{q \in pre(a)} \pi_a^s(q; s) \right\} \tag{7.5}$$

and with $ADD(p)$ denotes the actions in the problem that add p, such that $ADD(p) = \{a \mid p \in add(a)\}$. The cost of a relaxed plan is the sum of the costs of the operators it contains:

$$cost(\pi) = \sum_{a \in \pi} cost(a)$$

Although cost-sensitive relaxed plans could be obtained from using the least expensive additive heuristic, the set-additive heuristic has benefits when dealing with conditional effects, that are numerous in the translation $\mathcal{H}(P)$ and are not treated well by most heuristics. Indeed, the h_{FF} heuristic treats conditional effects as independent actions, except when the conditional effects appear in the same level of the relaxed plan graph. This choice is rather arbitrary, as sometimes the heuristic value of a state can be reduced by moving a conditional effect to a successive layer.

The set-additive heuristic combines a sensitivity to costs with the ability to deal with conditional effects in a more principled manner. When a conditional effects of an action a are present in the support plan for preconditions of the same action a, then the count is increased. On the other hand, if some conditional effect of the same action appear in the relaxed plans for different preconditions, then the action is counted only once. Another way to think about it is to consider the relaxed plan represented as a tree: if two independent sub-trees have conditional effects of the same action, then they are combined and counted as the same action. But if two conditional effects of the same action lie along a single path to the root, then they are considered as different actions and counted separately.

Taking advantage of a cost-sensitive heuristic, the cost of all the deductive actions in $\mathcal{X}(P)$ and $\mathcal{H}(P)$ is set to 0.

Doing the union of the plans, as in eq.(7.5), sometimes the result ends up in undercounting the total cost of the plan. As an example we can consider a problem with two actions a and b, each with two conditional effects, and the goal of achieving both g_1 and g_2, such that

$$\begin{array}{ll} a: & b: \\ \quad \rightarrow p & \quad \rightarrow q \\ q \rightarrow g_1 & p \rightarrow g_2 \end{array}$$

The relaxed plan that achieves g_1 from $\mathcal{I}$ is: $\pi_a^s(g_1; \mathcal{I}) = \{a, b\}$, while the relaxed plan that achieves g_2 is: $\pi_a^s(g_2; \mathcal{I}) = \{b, a\}$. From this, the relaxed plan that achieves both g_1 and g_2 is the union of the two plans, i.e. $\pi_a^s(g_1 \wedge g_2; s) = \{a, b\}$ when this is clearly unsound. The union doesn't consider the *order* of the actions it includes. What should be done is to consider, instead of the regular union, the minimal superset of unions, which will provide, once the plans have been represented as graphs, as minimal superset of edges in the plans such that every path in the plans has to appear in the transitive closure. In that way, we could obtain for this problem $\pi_a^s(g_1 \wedge g_2; s) = \{a, b, a\}$, for instance, by considering instead of (7.5), the following $\pi_a^s(a; s) = \{a\} \cup \biguplus_{q \in pre(a)} \pi_a^s(q; s)$.

7.4 Learning action models

A direct application of the model brought by planning under assumptions concerns the task of learning partially observable deterministic action models. When actions' models are not known in a planning problem, a solution plan cannot be directly derived because of the incompleteness of the model. The approach we tackle here is to encode unknown action's outcomes as uncertainty in the initial situation, where hidden variables encode the unknown effect of an action. In this case, the possible effects of an unknown action cannot be encoded as non-deterministic effects because the effects of the action are in fact deterministic: the result of applying the action will always be the same, it is just initially unknown.

Definition 7.4 (Action with unknown effects)**.** *An action a with preconditions p, and whose possible effect e_i is unknown ($1 \leq i \leq n$), is written:*

$$\begin{aligned} \text{precondition}: &\ p \\ \text{effect}: &\ \rightarrow\ e_1 \\ &\ \dots \\ &\ \rightarrow\ e_n \end{aligned} \tag{7.6}$$

In case the condition C of the effect is known, the action a can be written similarly as in eq. (7.6):

$$\begin{aligned} \text{precondition}: &\ p \\ \text{effect}: &\ C\ \rightarrow\ e_1 \\ &\ \dots \\ &\ C\ \rightarrow\ e_n \end{aligned} \tag{7.7}$$

For a contingent planning problem P, we call the set of actions with unknown effects $\mathcal{A}_U$. Of course we have that $\mathcal{A}_U$ is a subset of $\mathcal{A}$.

The contingent planning problem P with unknown action effects can be encoded as a contingent planning problem $\bar{K}(P)$ where all the actions having an unknown outcome are modelled as in definition 7.4, and where hidden fluents bound to each effect are introduced to learn the real result of applying the action:

Definition 7.5 (Contingent translation $\bar{K}(P)$)**.** *Let be $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, a contingent planning problem with actions with unknown effect. The contingent problem $\bar{K}(P) = \langle \mathcal{F}', \mathcal{A}', \mathcal{O}', \mathcal{I}', \mathcal{G}' \rangle$ with no unknown action effects is defined as follows:*

- $\mathcal{F}' = \mathcal{F}\ \cup \{h_i(a) \mid$ *for each unknown effect e_i of an action $a \in \mathcal{A}_U$*$\}$,
- $\mathcal{A}' = \mathcal{A}$ *where every effect $C \rightarrow e_i$ of the unknown action a in $\mathcal{A}_U$ has been replaced by $C \wedge h_i(a) \rightarrow e_i$,*

- *$\mathcal{I}' = \mathcal{I} \cup$ oneof $\big(h_1(a), \ldots, h_{n(a)}(a)\big)$, for every action $a \in \mathcal{A}_U$, with $n(a)$ that indicates the total number of possible unknown effects of a,*
- *$\mathcal{G}' = \mathcal{G}$.*

The problem $\bar{K}(P)$ with all unknown actions replaced by the mechanism above is only solvable by using assumptions, as it is not guaranteed that all the effects bring to the goal; in other words, the goal is not reachable in all the contingencies. However, thanks to the use of assumptions, the CLG+ planner doesn't freeze and provides a plan that unveils the effect unknown actions by observing the changes in the domain at execution-time. From deductive rules like the contingent merge and the tag refutation, the h_i's truth value can be learnt and thus also the actions' effects [3].

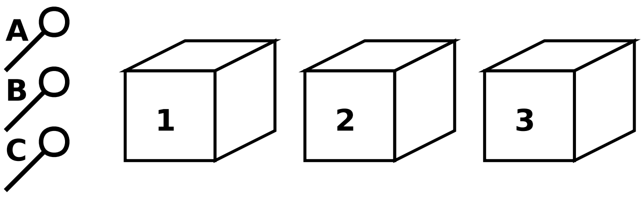


Figure 7.2: A boxes domain.

Example 7.3. *We consider a Boxes domain, illustrated in Figure 7.2, where a treasure is hidden in one of three closed boxes. There are three levers, each one controlling the opening or the closure of one box, but the relation between a lever and the box it opens is initially unknown. The goal is to retrieve the treasure, and to close all the opened boxes again.*

The problem is modelled with an uncertain initial situation where an XOR is used to state that each lever controls one box, but nothing implies that each box is controlled by one lever. In particular, the `push_lever` and `pull_lever` actions have conditional effects depending on the truth value of the fluents `(controls ?lever ?box)`; the result of pushing opens a box, pulling will close a box.

The problem where the agent has to get the hidden treasure has no contingent solution. The `push_lever` is shown here as example:

```
(:action push_lever
 :parameters (?l - lever)
 :effect (and (when (controls ?l box1)
                    (opened box1))
              (when (controls ?l box2)
                    (opened box2))
              (when (controls ?l box3)
                    (opened box3))  ))
```

A similar action are used to close them by pulling the lever. The agent can check if the treasure is in a box, if the box is open. The boxes should all be closed at the end: this tests whether the agent is learning the action model; namely, which

[3] The values of the newly introduced literals h_i does not change by the effect of the actions, and can only be "learnt" from observing the actions' outcomes.

lever controls which box. Passive learning is assumed in this case, meaning that all applicable sensing actions are automatically applied after each normal action. The observable fluents include whether a box is opened or closed, and whether the treasure is or is not in an open box.

The aim is to make the agent able to learn the action model from the gathered sensing, in order to open and subsequently close the boxes acting on the levers of the example 7.3. Another issue of the domain resides in the absence of a contingent solution (a strong plan). In fact, it is very possible that no lever opens the treasure box. Nevertheless, it should be possible to check whether the treasure is hidden in one of the boxes that can be opened.

Example 7.4. *Continuing on the example 7.3, let's notice that the fluents `(opened boxX)` (for $X \in \{1, 2, 3\}$) are effects that can be (automatically, if passive sensing is enabled) observed after manoeuvring a lever. The initial relaxed plan is based on identifying a box X containing the treasure relying on assumptions, pushing a lever, observing that the box X is open, and picking up the treasure.*

"Pushing a lever" and "observing" are the first two actions selected by the planner. This is reflected by the following execution obtained by CLG+ for the hidden initial state where the treasure is in box 2, and the levers A, B, and C control the boxes 1, 2, and 3 respectively:

```
1: PUSH_LEVER-A *
2: PUSH_LEVER-B **
3: PULL_LEVER-A
4: PICKUP_BOX3
5: PULL_LEVER-B
```

To illustrate the reasoning rules applied after an observation, we propose to analyse the result of the first action (marked with $*$). The sensing after `PUSH_LEVER-A` carries that only the box 1 is opened. This result is carried out by reasoning rules like Tag Refutation and Contingent Merge, bringing that lever-A does not control neither box 2 nor box 3, but controls (by exclusion) box 1. In that way the action model for lever A is learn, i.e. the atom $Kcontrols\_leverA\_box1$ is true in the state after the first sensing.

After `PUSH_LEVER-B`, a similar mechanism infers that the lever B controls box 2. From the same sensing episode, the treasure is observed in box 2. The plan then closes box 1, picks up the treasure, and closes box 2, applying the knowledge gained about the levers' mechanism.

7.5 Cases studies and experiments

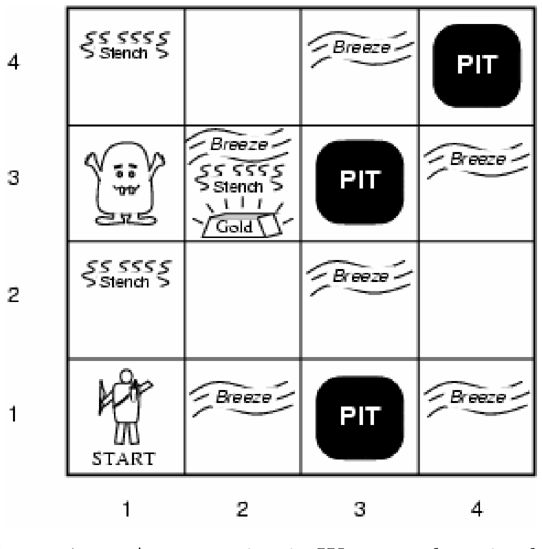
The approach taken by CLG+, and by planning under assumptions in particular, is extremely powerful as it allows to go out from the scope of contingent planning. As we saw in the previous section, assumptions can be cast to describe planning problems for which the model is not fully known, and to plan in such domains. Assumptions are not simply preferences on the domain hidden state: when automatically generated, they allow to solve by planning problems that are not solvable by any other planner, and this by finding plans that maximise the coverage.

To illustrate the behaviour of CLG+, we run the planner over a number of different problems, most of which are meaningful contingent problems without solutions. We will see also complex problems with contingent widths greater than 1 which are not solved by CLG because of the incompleteness of the translation for such problems, but that CLG+ manages to solve. The problems are then divided into those with dead-ends that arise from dead-end states, those with dead-ends that arise due to the absence of policies able to handle all possibilities, and those that are in theory solvable, but for which the incompleteness of the translation does not allow to capture a solution.

Problems with Dead-End States

Problems with dead-end states are intrinsically unsolvable as a contingent plan doesn't guarantee to solve the problem for all the contingencies. This is the case seen in example 7.2, where certain configurations of the domain are not solvable, and where the heuristic evaluation points to an unreachable goal. But the goal might be effectively reachable even if the information is hidden. It should be then possible for the planning agent to execute a plan under assumptions, and use sensing to find out whether the goal is achievable, whereas this information is not available initially.

Wumpus



```
1: MOVE_P1-1_P2-1
2: MOVE_P2-1_P1-1
3: MOVE_P1-1_P1-2
4: MOVE_P1-2_P2-2
5: MOVE_P2-2_P3-2
6: MOVE_P3-2_P2-2
7: MOVE_P2-2_P2-3
8: GRAB_P2-3
```

Figure 7.3: An execution in Wumpus domain: hidden state shown on the left and execution (with passive sensing) shown on the right.

The Figure 7.3 shows a version of the Wumpus problem from (Russell and Norvig, 2002), which is actually solvable, although the initial belief has an infinite heuristic value. In this problem, the agent shown in the Start state must get the Gold by sorting dangerous wumpuses and pits. We assume that the agent can move deterministically one unit in each of the four directions, provided that in the target cell there is no wall, wumpus, or pit.[4]

Initially, the agent knows its location, but doesn't know where the gold is, nor where the wumpuses or pits are. The uncertainty in the initial situation is modelled by having uncertainty in each cell, regarding whether it contains a wumpus, a pit, none, or both. The uncertainty on the gold location is an XOR over all the grid cells. This is a natural encoding of the problem, yet it makes the problem unsolvable, as the gold may be in a cell with a wumpus or pit, or it may be blocked by them. The agent can sense the presence of a wumpus in *some* adjacent cell by the stench, the presence of a pit in *some* adjacent cell by the breeze, and the presence of the gold in its own cell by the glitter. The stench, the breeze, and the glitter are all observable.

On the right side of the figure is reproduced the execution obtained from CLG+ by running it on the hidden state shown in the picture, where there are 3 pits and 1 wumpus, and the gold is located at cell 2-3. For simplicity, the sensing is assumed to be passive, meaning that all applicable sensing actions are applied after every step. Thus, the actions shown in the execution all refer to non-sensing actions.

In the execution, the agent starts at (1,1) knowing that there is no wumpus or pit in the two adjacent locations. It then moves to (2,1), where it senses no stench, yielding that there is not wumpus around, and a breeze, from which it concludes that the only safe cell to move is (1,1). It moves there, and then to the other known safe cell (1,2), where it senses no breeze, and thus no pit around, but it senses a stench. From the observations gathered so far it concludes that there is neither a wumpus nor a pit at (2,2): this cell is safe. Its moves there, and senses no stench and no breeze. It moves then to (3,2), but sensing a breeze there it backs up to (2,2), heading up now into (2,3), where it sees the glitter and grabs the gold.

In this computation, CLG+ never commits to an assumptive action $K\neg t$ in the execution, but uses many of those actions in the construction of relaxed plans, and in the computation of the heuristic. In every relaxed plan, it is assumed that the gold is not at a number of locations, from which it is inferred that it must be at particular location. This is not enough though. All the relaxed plans involve also assumptions about cells that are safe to be traversed in order to reach to the assumed gold position. These assumptions are all handled automatically as actions in the heuristic model, and no assumptive action from the execution model is applied.

Navigation in Unknown Map

We consider a version of a navigation problem, where an agent must move from an initial location I to a target location G in a grid where the status of all other cells is unknown (cf. Fig. 7.4). Such cells can be free or not, the agent can sense the status of adjacent cells, and can move into them if free. The problem is not solvable, as it contains dead-end beliefs that result from dead-end states: those in which the cells that are not free block the path to the goal.

Like before, the hidden status of the environment is solvable, as a path to the goal cell G exists; it is just unknown initially to the agent. The assumptions concern possible free cells, but none of these assumptive action is actually executed, as sensing always unveil the preconditions of moving actions.

[4]In the book, if the agent moves into a wumpus or pit cell, it dies. Also agent is there armed with an arrow which can kill the wumpus if fired properly.

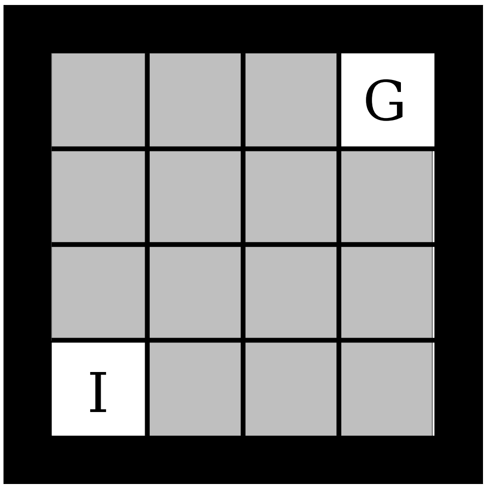


Figure 7.4: Navigation. Initially the status of all cells other than I and G is unknown and therefore marked in grey.

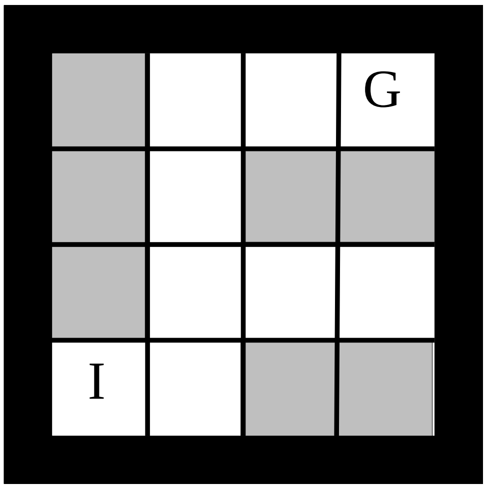


Figure 7.5: Navigation. The hidden state of the navigation problem: the free cells are in white.

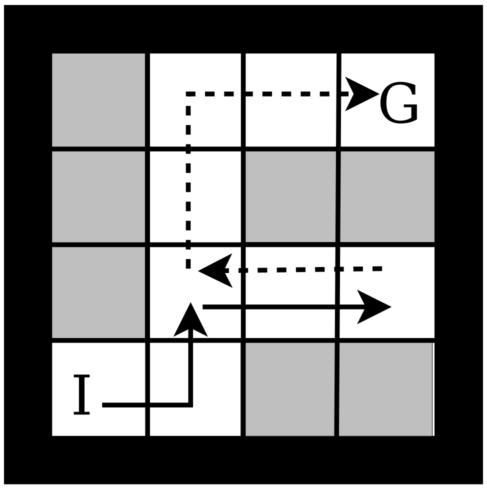


Figure 7.6: Navigation. Initially the status of all cells other than I and G is unknown. The hidden state is shown by indicating the free cells in white.

The Figure 7.6 shows the execution of CLG+ in this 4×4 grid example. Arrows in the figure indicate the execution that results from a particular hidden state where the cells that are free are the ones shown in white. In such a case, by moving and sensing, the agent follows the path shown. Not knowing that the two cells under the goal are blocked, the agent heads initially to the right, but having learnt that, it backtracks, going around those two cells to eventually reach the goal.

Problems with Pure Dead-ends Beliefs

Problems with pure dead-ends beliefs are unsolvable problem that result from the absence of a policy that can deal with all the possible states in the initial belief, even though none of these states is a dead-end. Any taken policy will work for some initial states, but not for others. Thus, in these cases, the adoption of a policy involves "betting" on a possible contingency: if we are lucky, we solve the problem, if we are not, we fail. This is different from the problems above where the actions of the agent did not affect the solubility of the problem; the question was then whether the hidden state was a dead-end or not. If it wasn't, sensing would lead to the goal at execution-time.

Minesweeper

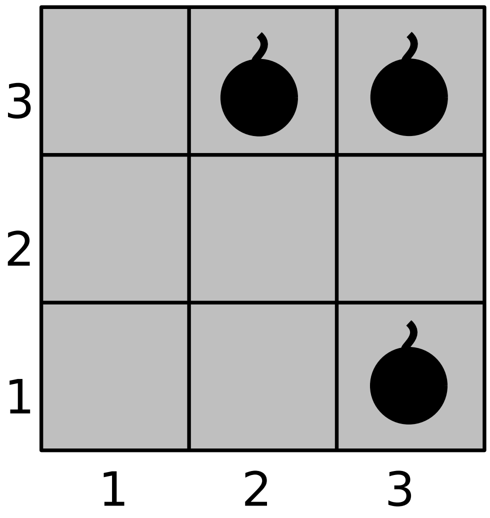


Figure 7.7: Minesweeper. Distribution of the bombs in the hidden state.

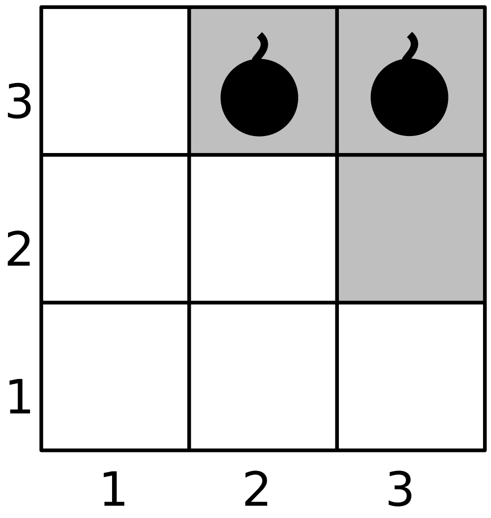


Figure 7.8: Minesweeper. Deductions of the agent on the (hidden) state of the world: white cells have a known status.

We consider a version of the well known Minesweeper problem. This problem involves a grid where each cell may contain a mine or not. Cells without a mine must be cleared with the *clear* action, while cells with a mine must be cleared with a *sweepmine* action. The goal is to have all the cells cleared. Knowledge about the location of the mines is obtained through a *check* action on a cell that yields a failed execution if the cell contains a mine. We model this by setting *no mine at cell* as a precondition of *check cell*, and *mine at cell* as precondition of *sweepmine cell.* Given the initial complete lack of knowledge about the mine locations, the problem is unsolvable, and indeed, it can lead to failure after the first *check.* One thus must "bet" on the first cell to clear. The initial belief state is thus a dead-end despite the fact that it does not contain any dead-end states. Indeed, the problem would be solvable for any initial state if the state was observable.

The hidden state of the example has mines at (2,3), (3,1), and (3,3), as shown in Figure 7.7. The first action is to assume that there is no mine at (1,1), and to check this position. This is the first domain where assumptive actions need to be executed outside of the heuristic model, in order to obtain the knowledge about the action preconditions. This could lead to execution failures in case the executed assumption does not yield a "desired" precondition for the next-to-apply action. Here this could happen if there were a bomb in (1,1). In this example (1,1) is free of bombs, and thus the action is successful. After the check at (1,1), for the hidden state shown, the planner infers that there is no bomb at either (1,2) or (2,1). It thus checks these two positions, and infers that (1,3) and (2,2) are free of mines. On the other hand, from the check at (2,1) it infers that there must be a bomb at (3,1), since it is known by then that there is no bomb at (2,2), and minesweeps that bomb. The belief at that point is shown in Figure 7.8, where the white cells are the ones whose status is known. It then chooses to do a check at (3,1), that is now cleared too, to find out that (2,3) is clear, and from the resulting observations there and in (2,2), finds out the status of the two last cells with bombs.

Problems with High Contingent Width

Binary Tree

As a last example, we show a solvable problem having contingent width higher than 1, out of the scope of those translation-based approaches that are sound and complete for contingent problems with width ≤ 1. These problems are difficult for state-of-the-art contingent planners, and the translation at the basis of CLG is not complete, which means that the planner can indicate an infinite heuristic in the initial state even if the problem is completely solvable (which is the case in the example below). This example is not solved by both CLG and Contingent-FF, while POND solves only the small instances. CLG+, on the other hand, turns out to scale up better in this example as a full contingent planner.

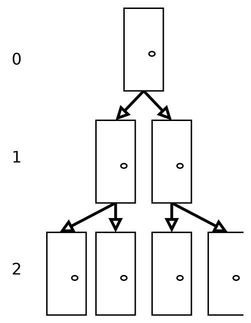


Figure 7.9: Binary tree domain of depth 3

The problem involves a binary tree with depth n. Starting in the root, the agent has to move to a leaf of the tree. The possible actions are to move from a node n to its left or to its right son, if the corresponding edge is not blocked. One of the two edges, however, is always blocked, and the agent can find out which by doing a sensing operation. There are also actions for moving up in the tree. The problem has width n, meaning that a complete $X_i(P)$ translation would be exponential in n. As a result, CLG that uses the $X_1(P)$ translation does not solve the problem, and reports an infinite heuristic value. The problem, however, is simple, just requiring sensing at each node, and moving to the son along the edge that is open.

Table 7.1 shows some results on different instances of the binary tree domain. CLG does not find a solution, as well as Contingent-FF, to this solvable problem, CLG+ does for all the instances tested, while POND solves the first two but not the last one.

	POND		CLG+	
depth of the tree	**time**	**# actions**	**time**	**# actions**
3	0.11	6	0.24	6
4	0.87	8	0.99	8
5	T	T	3.60	10

Table 7.1: CLG+ compared to POND on different instances of the binary tree. Figures shown are total time in seconds and total number of actions in solution. T stands for time out (cut-off of 45mn or 1.8Gb of memory).

Scalability and Overhead

	CLG		**CLG+**	
problem	time	# actions	time	# actions
ebtcs-50	11.96	149	12.88	149
ebtcs-70	34.37	209	34.13	209
medpks-50	3.23	101	3.17	101
medpks-70	9.89	141	9.10	141
medpks-99	28.77	199	27.61	199
unix-3	9.26	113	52.17	111
unix-4	120.72	240	1748.84	238
cballs-4-1	0.35	295	0.71	282
cballs-4-2	18.83	20050	56.44	20203
cballs-4-3	1537.99	1136920	T	
cballs-9-1	192.16	3385	234.88	3497
cballs-9-2	T		T	
clog-7	0.17	210	1.12	215
clog-huge	157.94	37718	T	
doors-7	10.60	2153	64.28	2145
doors-9	1042.96	46024	T	
wumpus-5	1.76	732	14.31	753
wumpus-7	89.32	10681	1217.39	17256
wumpus-10	T		T	

Table 7.2: Examples to calibrate the performance and scalability of CLG+ in relation to plain CLG. Figures shown are total time and total number of actions in solution. `T` stands for time out (cut-off of 45mn or 1.8Gb of memory).

Table 7.2 displays the ability of CLG+ to build *full contingent plans* over standard benchmarks, in comparison with CLG, shown in (Albore et al., 2009) to scale up better than Contingent-FF and POND. All the tests are obtained on a Linux machine running at 2.33GHz with 2Gb of RAM.

In these problems, all the additional machinery in CLG+ is not needed, and thus the difference in performance between CLG and CLG+ shows the overhead resulting from these changes. As it can be seen, CLG+ is slower than CLG, but it manages to solve most of the problems that CLG can solve. These figures give an idea of how well CLG+ scales, an idea that cannot be obtained from the examples discussed above, that are aimed at describing the functionality of CLG+ rather than its scalability.

7.6 Discussion

Planning with preferences on domain uncertainty

Assumptions are modelled as actions with high cost in our model. However, as the problem has been tackled until now, all such actions have the same cost. Now we ask what would be the behaviour of the planner in case different costs were to be attributed to different actions.

Of course, certain assumption $K\neg t$ with lower cost than other assumptive actions would be selected first during the plan search, obtaining as side effect of this formulation with different costs the generation of a solution plan under preferences.

Preferences encoded in this form do not aim at expressing preferences over states achieved by the plans as in (Brewka, 2004), nor to express importance of achieving certain variables over others like in TCP–nets (Brafman et al., 2006), but it is used to express the "strength" of assumptions used to solve a problem otherwise without solution.

Say that in the example 7.2, the possibly blocked central wall is encoded as a disjunction $(free(x_1) \vee \ldots \vee free(x_5))$, where $x_1, \ldots, x_5$ are the central grey cells. Expressing preferences would result, for instance, in giving to the assumptive action $K\neg free(x_3)$ relative to the central cell, a cost higher than to the other actions to express that this cell is more likely to be free. In this case, to direct the search directly to that cell x_3, the cost would be proportional to the elements in the disjunct, i.e. more than 4 times the cost of each other assumption.

Actually differentiating the encoding of the assumptions is not feasible in an automatic way, given the actual implementation. This means that exploring the capacities of expressing preferences over domain uncertainty is left as a future work. Such an encoding would surely appear to be a powerful mean to deal with complex problems, like the ones common in the field of robotics. Then, these preferences would subsume the assumptions hand coded as a formula describing the expected domain behaviour (Albore and Bertoli, 2004, 2006), a technique that could also be used to express preferences over plan trajectories.

Related Work

The approach tackled with CLG+ makes use of the paradigm of interleaving planning and execution, as has been done in many robotic frameworks. Interleaving planning and execution has sometimes been considered a necessary approach to deal with real world problems (Nyblom, 2005). In (Genesereth and Nourbakhsh, 1993), planning and execution is interleaved to reduce the search space for a conditional planner. A similar approach has been applied to the conditional planner MBP, modified and reused in a planning/execution algorithm (Bertoli et al., 2004). The main differences between this approach and ours are that we use optimistic plans to guide the search instead of information gain and we use a cost-aware planner instead of a conditional planner. Efficient heuristics from classical planning help the search in CLG+, since they assume that a path exists from a well defined initial state, regardless the non-deterministic actions' possible outcome, then they optimistically forecast the outcome of each action, but checking the result before triggering the next planning episode.

The common assumption in most of related works is that the hidden preconditions become observable, and hence known to be either true or false before applying an action. This includes the absence of dead-end in so called *connected* search space.Under this assumption, a greedy strategy can solve the planning problem by first making the most convenient assumption about the values of the hidden variables, and then execute the plan that is obtained from the classical planning problem obtained by selecting the most promising initial state. If during the execution the observations gathered refute the assumptions made, it will always be possible to revise the assumptions and replan. The replanning technique is popular and effective when the search space is small and connected (Yoon et al., 2007). However, this assumption might not hold, and such planners just freeze without being able to find a solution or fail during execution.

An instance of this general idea, used in robot navigation tasks, is the "freespace assumption" (Koenig and Smirnov, 1997), where an optimistic assumption is made about the crossability of a maze, i.e. that the path to the goal is free.

Albore and Bertoli (2004) restrict the belief search space by adopting assumptions on the initial belief state. The search space is in that way reduced, but the plans generated are such that a mistake in the assumptions can always be detected by the available sensing before acting, triggering a replanning episode that considers the whole search space if necessary. This *safe* assumption-based planning has been extended to use more complex formulæ to describe expected effects of non-deterministic actions and exogenous events over time: there safe plans are still generated by assumptions, but described via an LTL encoding (Albore and Bertoli, 2006). However, in their approach, the assumptions are generated by hand, while CLG+ selects them in an automatic way in order to maximise the "coverage", i.e. number of states to which a solution under assumptions applies.

Learning action models in partially observable domains, i.e. how actions affect the world's states change, has been approached with a learning algorithm that updates a formula representing the initial transition belief state with the sequence of executed actions and sensing (Chang and Amir, 2006; Amir, 2005). A transition belief state is constituted by a belief state and action-schemas, which are propositions that represent the possible transition relations. The actions to apply are selected by a SAT-based planning subroutine, which searches for plans that could possibly work. The plan is executed and the transition belief updated, so that the formula returned includes all consistent models, which can be retrieved then with additional processing. The approach scales up well, and guarantees to reach the goal within a bounded number of steps, linear in the length of the longest plan needed for reaching the goal, and exponential in the complexity of the action model learnt. On the other hand, the size of the formula grows linearly with the sequence length, but does not depend on the domains size.

Other approaches to learning action models differ from CLG+ because they assume full observability (Gil, 1994; Pasula and Zettlemoyer, 2004; Sutton and Barto, 1998). Reinforcement learning in POMDPs approaches, on the other side, fail in scaling up (Even-Dar et al., 2005).

Do probabilities solve this problem?
Actually not. In undiscounted goal MDPs and POMDPs, the agent is supposed to reach the goal with certainty, and in the presence of such dead-ends this is not possible. At the same time, discounting makes all costs (rewards) bounded . . . but meaningless. Take the following example: a treasure is at distance d (which encodes its cost), then for any discount factor $\gamma < 1$, it is possible to select d large enough so that the agent would prefer *not* to go for the goal, even though it can get the treasure with certainty.

With deterministic actions, optimal (probabilistic) POMDP policies are acyclic, like in contingent plans. The difference resides in that the former are aimed at minimising expected cost, and the latter at minimising cost in the worst case. Both costs are infinite if the goal is not reachable from one of the possible initial states.

The cost of the assumptive actions do not take into account the cardinality these assumptions restrict the belief states. Indeed, costs help the search algorithm to provide solutions with as less assumptions as possible, but this minimisation is not calibrated on the actual model count of the resulting belief. This optimisation can be obtained by a preprocessing computation: performing a model counting operation on the initial belief state $\mathcal{I}' = \mathcal{I} \wedge t$, for all assumption t. This would give a weight to each assumption/tag that would provide a consequent cost to the relative assumptive action a_t.

7.7 Summary

We introduced a translation-based model for planning with incomplete information and sensing, and always generates a plan to execute, even when contingent planners or POMDPs solvers fail because of the potential unreachability of the goal. Instead, we produce on-line plans under assumptions for which the goal is reachable, and plan consequently on a reactive platform. The resulting planner CLG+ never freezes in front of lack of initial information of the problem, replans when the automatically generated assumption is observed to be false, and fails only when the execution fails. The assumptions are generated in order to maximise the coverage, and the method provides also many other applications, one of them being learning incomplete action models.

Part IV

Conclusions

Chapter 8

Conclusions

In this dissertation we have described translation-based methods to deal with conformant and contingent planning problems. Planning with incomplete information is a search problem in belief space. Our encoding reduces the belief space search to state space search. We have identified the conditions under which the proposed translations are sound, complete, polynomial, and consistent.

We now summarise the contributions of the thesis, and briefly discuss some ways in which the work presented may be extended.

8.1 Contributions

We enumerate the contributions that we feel to be of greatest interest to the AI planning community.

Conformant planning

Conformant planning is the task of planning with incomplete information and no sensing. In this dissertation we built on and extended the work by Palacios and Geffner (2009) by providing:

1. **A formal description of tractable translations to conformant planning based on sampling initial states K_S.** For a planning problem P, the translation $K_S(P)$ is always complete. We also characterise the translation K^i_S, that is always complete and is sound for problems having width bounded by i. e used for the state–of–the–art conformant planner T1. For a problem P with conformant width bounded by i, the translation $K^i_S(P)$ is sound, polynomial, and complete (chapter 4, section 4.2).

2. **Heuristics that take into account two major aspects of automated planning under uncertainty**, namely the cardinality of the belief state, and its distance from the goal. The planner T1 uses these two heuristics, called the *classical* and *certainty heuristics*, in combination. The certainty heuristic h_K is given in terms of a set of "oneof invariants" derived from the problems,

and exploits these invariants to obtain more accurate estimates (chapter 4, section 4.5).

3. **A family of translations for conformant planning problems with non-deterministic actions** that compile away the non-determinism using newly-introduced state variables. The starting point for this family is a deterministic relaxation that is sound as long as the non-deterministic actions are executed at most once. We provided completeness results for these translations (chapter 5).

4. **Several incomplete translation schemas for non-deterministic planning**, some of which appear to be quite effective, mapping non-deterministic conformant problems into classical ones. In particular, the $\langle K, K_0 \rangle$ translation applied successively for $K = K_0$ and $K = K_1$ appears to be quite effective (chapter 5, section 5.6).

Contingent planning

Contingent planning is the task of planning with incomplete information and sensing. In this dissertation we approached the task by providing:

5. **The translation-based approach for contingent planning $X_{T,M}$.** The computational pay-off of the translation $X_{T,M}$ is that it allows us to compute plans with states represented by sets of literals, rather than with beliefs represented by sets of states. The translation $X_{T,M}$ preserves consistency for contingent problems. We also characterise soundness and completeness properties of the translation X_i we used for the state–of–the–art contingent planner CLG that depends on a parameter of the problem, the *contingent width*. We have also shown that for a problem P with contingent width bounded by i, the translation $X_i(P)$ is sound, complete, and polynomial (chapter 6, section 6.3).

6. **The relaxation $\mathcal{H}(P)$ for contingent problems P that is a classical planning problem**. We use $\mathcal{H}(P)$ in conjunction with a classical planner to provide an informed heuristic to efficiently drive the search of the state–of–the–art contingent planner CLG in a closed loop fashion. The planner CLG can produce off-line and on-line plans. (chapter 6, section 6.5).

7. **A modification to the translation-based approach used in CLG to deal with problems that do not guarantee goal reachability**. In such cases, instead of "freezing", our CLG+ planner provides anyhow a solution under assumptions, automatically generated from the problem description. Depending on the results of sensing actions, these assumptions can be either confirmed or shown to be false during execution, triggering a replanning episode when necessary. Assumptions are introduced in such a way that actions depending on them, are executed only as a last resort.

8.2 Future & ongoing work

Translation-based approaches are an efficient and elegant way to represent planning problems with incomplete information. The flexibility of the translations used throughout this thesis allowed us to extend the original compilation of conformant problems to classical ones of (Palacios and Geffner, 2009) to non-deterministic actions, to contingent planning, and also to capture those plans that do not guarantee goal reachability, including some application to learning unknown action models. We will see now some future lines of research bound to extending or re-using the translations described in this dissertation.

Using samples to translate contingent planning problems

The translations based on samples are efficient and compact. We presented in chapter 4 the translation-based planner T1 for deterministic conformant planning that uses the K^1_S translation, and that tries to combine the flexibility of planners that search explicitly in belief space, with the heuristics and belief representation that arise from translations. The novel general translation K^i_S selects sets of initial states depending on the width i of the conformant problem. The translated problem $K^i_S(P)$ is always tractable and complete, and sound for problems with width bounded by i.

We believe that similar translation techniques based on sampling states in the initial belief can be used for contingent planning problems, and more accurately if the initial samples form a basis for the problem. For this, the notion of basis has to be extended to planning with incomplete information and sensing. We can consider that a set of states S is a *basis* for a contingent planning problem $P = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ if S is a subset of the set of all possible initial states S_0 of P, and if any plan for $P|_s = \langle \mathcal{F}, \mathcal{A}, \tau(s), \mathcal{G} \rangle$ is a contingent plan for P, for any state $s \in S$.

Given a contingent basis for a problem P, a solution plan π for $P|_S$ (which is P restricted to the initial subset of states S of S_0) is a solution plan for the whole contingent problem P. The translation-based approach using samples exploits contingent bases to generate a contingent plan that maps the belief S into the goal, and that conforms for all the states in S_0. In particular, we can relate contingent bases to the covering merges used in translations $X_i(P)$ that are sound and complete for problems having contingent width bounded by i (as seen in section 6.4). The set of states that satisfy a covering merge is *de facto* a basis by construction. If we consider covering merges, we can identify a subset of the initial set of states that satisfy merges covering every precondition or goal L, allowing so to track relevant beliefs along the plan execution.

A basis S can be built from the covering merges of a translation $X_i(P) = X_{T,M}(P)$. We start by considering a merge $m = \{t_k\}_{k=1}^n$ that covers a precondition or goal literal L, and we select the set $S_{t_k}(L)$ of possible initial states s of P such that $rel(s, L) \subseteq t^*_k$. This result mirrors a similar one obtained in the conformant planning setting: the set $S_{t_k}(L)$ contains the possible initial states of P that make false all the literals L' that are relevant to L, except for those in the closure t^*_k of t_k. Finally, the set $S = \bigcup_{t_k \in m, L} S_{t_i}(L)$ is a basis for the contingent problem P, for all merges $m \in M$ and all precondition or goal literals L.

We can give a sketch of the proof for this result. First we notice that, by induction on the length of a plan π, it can be proved that S is a basis for P if for every possible initial state s_0 of P, and given a precondition and goal literal L in P, S contains a state s such that $rel(s, L) \subseteq rel(s_0, L)$. In fact, if S is a basis for P, then a plan π that achieves the goal for $P|_S$, achieves the goal also for $P|_{S_0}$. Then every state $s_0 \in S_0$

must satisfy a tag t_k in a valid merge m for a precondition or goal literal L. From Theorem A.25 in the appendix, we know that for each initial state s of P exists a tag t_k in a valid merge m such that $rel(s, L) \subseteq t_k^*$, and therefore $rel(s, L) \subseteq rel(s_0, L)$, since s must satisfy t_k^* and possibly other literals L' that are relevant to L.

Thus, from a valid set of merges of a covering translation, a contingent basis can be extracted. This result provides us a method to extract samples to build a basis for P, given the tags t_k in a covering translation $X_{T,M}(P)$. Once a contingent basis is identified, we can exploit the "oneof" invariants of the problem to produce a heuristic sensitive to the "amount of uncertainty" of the belief states. Similarly to what has been done in T1, such a heuristic would be related to the cardinality and the landmarks heuristics, and improve the efficiency of solving the translated problem.

Sentence generation as Multi-agent planning

Dialogue between two agents can be viewed as actions interchanging information or knowledge about the environment, in order to reach a final situation, where both agents achieve their own goal. So sentence generation problems can be cast as planning problems under incomplete information, and with some sensing available.

We then consider a two-agent collaborative planning model, where here *collaborative* means that both agents share the same goal, to tackle the dialogue generation problem. A multi-agent planning problem, and in particular a two-agent problem, can be cast as a single agent planning problem with incomplete information and sensing (Weerdt et al., 2005), as agents act independently. The "state of knowledge" of each agent is encoded by the mean of epistemic modal operators K_1 and K_2 to denote what each agent "knows", e.g. $K_1 K_2 L$ indicates that the agent 1 is aware that the agent 2 knows that L is true.

In practice, however, single-agent planning problems that feature incomplete information and sensing are solved using classical planners and replanning. As we saw in chapter 6, problems with incomplete information and sensing can be cast as non-deterministic fully observable planning problems.

The translation $K_0'(P) = \langle \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ is based on the K_0 translation for conformant problems (cf. section 3.1), and extends it to capture partially observable problem P at the "knowledge-level", in a fashion similar to what $\mathcal{X}(P)$ does (cf. chapter 6). $K_0'(P)$ represents only the literals that are known and negatively known in the original problem, and is a more compact translation compared to $\mathcal{X}(P)$, since K_0' is linear rather than quadratic.

This translation is the base for an on-line planner for partially observable domains, where a reactive platform allows getting feedback from the environment and replanning. This approach can be extended to the general multi-agent case.

The translation K_0' requires a connected state space, which guarantees that no dead-ends are present. Another assumption is considered in approaching partially observable problem with replanning techniques: the hidden preconditions of actions become observable at execution, and thus are known to be either true or false. Under these optimistic assumptions, but that are easily achieved in multi-agent planning problems, the $K_0'(P)$ model can be cast as a *classical planning* problem by fixing –when needed– the values of all the unknown variables. This method follows the scheme of an effective greedy on-line algorithm:

1. to consider the most convenient assumption about the values of the unknown variables;
2. to execute the plan obtained from the resulting classical planning problem fed to a classical planner;
3. to revise the assumptions made, from the observations gathered at execution time, and eventually replan if the observations refute the assumptions made.

This method is similar in spirit to the assumption-based translation we saw in chapter 7 for problems where reaching the goal is not *a priori* guaranteed. The difference here is that assumptive actions are not on tags, as –similarly to the translation K_0– the beliefs are compiled away, but the assumptions are only made about the truth value of a single fluent.

Another central aspect of multi-agent planning is the coordination between the agents. Coordination between agents aims at avoiding conflicts during the execution of plans. In the former on-line planning algorithm, coordination is taken into account very lazily, by letting the agents observe the outcome of their actions, or letting them sense the inner knowledge state of the other agent. But multi-agent planning needs a more elaborate form of coordination, like messages interchanged between the agents, or sensing of agent's knowledge state that includes the next-to-apply actions. This modelisation of a coordination algorithm is left as future work. We have, however, applied and tested a simplification of the two-agent planning model to the dialogue generation problem that avoids any coordination problem.

A dialogue session between the computer agent and a user is an actual conversation, consisting of a specific series of dialogue turns[1]. On this benchmark, we formulated the two-agents planning model as a Master–Slave multi-agent problem, where *the slave acts as decided by the master*. In our case, the user acts like a slave of the computer agent. This abstraction allows the master to communicate to the slave all the actions in the environment, that it then performs while the master can sense its mental state to check the outcome of the actions, and the results of the sensing. Another important simplification is that the slave does not have false beliefs. Consequently if the master knows a fact to be true, then the slave knows it as well.

As for single agent planning, a solution to a Master–Slave two-agent problem P can also be regarded as a policy Π that maps belief states b into actions, or action sequences π. The implementation we carried out produced an on-line replanner, involving the $K_0'(P)$ translation, and featuring the master's actions, the master's and the slave's physical sensors of the environment, and the master's mental sensor of the slave's internal state. For the simulated executions on a reactive platform, the executor keeps track of the hidden state of the world, and the hidden belief state of the slave agent (captured by means of invariants and epistemic literals).

We obtained preliminary results of this simplified model applied to language generation, reproducing the dialogue between a human user and an automatic agent to

[1]We had the occasion to approach dialogue generation in the context of the SpaceBook European project SP7, where an automated agent drives verbally a user toward a map to a desired location. The dialogue is also needed to establish the user's goal.

solve troubleshooting problems, that replicate the "modem restarting" task proposed by Boye (2007). The tests shows the ability of the replanner to propose solutions based on the knowledge of the master's agent, that is initially null. Via a question-answering procedure, the master identifies the correct sequence of actions the slave has to apply.

The Master-Slave is indeed a big simplification that avoids the coordination problem and that does not feature an independent slave agent (the slave does not act by itself, and does not need to sense the master's internal state of knowledge). The master-slave model for two-agent planning can thus be encoded as a single agent planning problem. However, these first results are encouraging for developing a more complex two-agent planning model, based on classical translations and replanning.

Ongoing Work

The translation-based approach to planning with incomplete information has inspired some new line of research in automated planning recently. In these pages we focused on finding efficient translations that put together the capacity of scaling up with the properties of soundness and completeness. For this reason we identified and used a parameter called the *width*, to specify families of conformant and contingent problems for which a translation is sound and complete. As the size of many of the translations we introduced here is exponential in the width, there is a trade–off between the capacity of scaling up and the width of the family of problems that a translation can solve. Notice that even an incomplete translation can nevertheless provide solutions for a planning problem, yet this is not always guaranteed.

One can then consider a translation that maps the problem with incomplete information to a deterministic problem; this is done by considering the best assumption s on the uncertain literals, and planning for the original problem P restricted to the initial situation $P|_s$. This is indeed a classical planning problem, and is the starting point of two successful translations for planning with incomplete information and sensing.

Bonet and Geffner (2011) extended the K_0 translation to include reasoning rules on the invariants of the problem K_0' for *simple problems*. A problem is said to be simple when (i) the non-unary clauses representing the uncertainty about the initial situation are invariant (i.e. left unchanged by any action), and (ii) the fluents that are unknown in the initial situation do not appear in the body of conditional effects, all of which are assumed to be deterministic.

For simple problems P, and for connected state spaces, the translation $K_0'(P)$ is shown to be both sound and complete.

Shani and Brafman (2011) use replanning and a translation to classical planning in their SDR planner. The translation considers only a single initial state is considered initially: SDR samples an initial state s and plans for the restricted problem $P|_s$, replanning when necessary. This technique is fast and efficient, in particular for problems with high contingent width, where the size of a complete $X_{T,M}$ translation would affects the performances negatively.

Both the K-replanner of Bonet and Geffner (2011), and SDR of Shani and Brafman (2011) use replanning while solving a classical planning problem, in a fashion that is not dissimilar to CLG in on-line mode. While SDR uses a sample state of the initial belief, K-replanner considers only the literals that are known initially. The mechanism allows the planner to make optimistic assumptions on the preconditions, while focusing on sensing, as these assumptions might be then confirmed by observation.

Sensing actions are handled by the K-replanner when "assuming" the truth value of a literal: a confirmation by sensing is then needed. In SDR, observations have to be forced by a bias, that can be deactivated when necessary. Observing is a key characteristic for contingent planning to succeed, as it is often beneficial to sense the environment to limit the amount of uncertainty of the problem.

The simplicity of these translations illustrate well the flexibility of the approach. These planners have good performances, scaling up well on several benchmark domains, where the machinery of CLG in on-line mode is not adapted to the task. However, the simplicity of the translation techniques also fail in more complex domains, or just in domains featuring dead-ends, due to their incompleteness. But the challenge here is to identify the planning problems that can benefit from translations, to use them proficiently. The ongoing work in translations follows this path, identifying and applying successfully new translations for different families of planning problems with incomplete information and sensing.

Part V

Appendix

Appendix **A**

Proofs of Soundness and Completeness for $\mathcal{X}(P)$

In this appendix we provide formal results used in chapter 6.

1 Basic Definitions

Definition A.1. *A policy tree* Π *for a contingent planning problem* P *is a pair* $< T, A >$ *where* T *is a rooted directed tree, and* A *is a function mapping the* internal nodes n *of* T *to actions a(n) in* P*.*

A similar definition can be cast for $\mathcal{X}(P)$.

For the fully-observable non-deterministic problem $\mathcal{X}(P)$ and for the contingent problem P, we consider only policy trees $\Pi =< T, A >$ where each node n can have no child (a tip node), one child, denoted as $c(n)$, or two children, denoted $c^+(n)$ and $c^-(n)$. The only nodes n that can have two children are those in which a non-deterministic action $set(L)$ is performed in a state[1] that results in the two possible outcomes KL and $K\neg L$. This is formalised below:

Definition A.2 (Policy Trees applicable to $\mathcal{X}(P)$)**.** *A policy tree* Π *for* $\mathcal{X}(P)$ *is* applicable *if there is a state function* $s(n)$ *over the nodes in* Π *such that:*

1. $s(n_0) = s_0$ *where* s_0 *is the initial state of* $\mathcal{X}(P)$ *and* n_0 *is the root node of* Π,
2. $a(n)$ *is executable in* $s(n)$ *and* n *has a single child* $c(n)$ *if* $a(n)$ *is a deterministic action in* $s(n)$*, and two children* $c^+(n)$ *and* $c^-(n)$ *otherwise, with states:*
   a) $s(c(n)) = f\,(s(n), a(n))$
   b) $s(c^+(n)) = f\,(s(n), set^+(L))$ *if* $a(n) = set(L)$
   c) $s(c^-(n)) = f\,(s(n), set^+(\neg L))$ *if* $a(n) = set(L)$

[1] In this appendix we use the notations *set* and *obs*, instead of simply *obs* as in chapter 6, to differentiate the sensing actions encoding in $\mathcal{X}(P)$ and P respectively.

with f the transition function associating successors states $f(s,a) = s'$ to states s after applying the action a in P.

In this definition, $set(L)$ is the non-deterministic action $set(L) = C \rightarrow KL \mid K\neg L$ in $\mathcal{X}(P)$. We also use $set^+(L')$ and $set^-(L')$, where $L' \in \{L, \neg L\}$, to identify two deterministic actions with the same preconditions as $set(L)$ but with effect KL' and $K\neg L'$ respectively. Of course $set^+(\neg L) = set^-(L)$. The solutions of the fully observable, non-deterministic problem $\mathcal{X}(P)$ are then given in terms of the unique state function $s(n)$ that the definition above associates with any applicable policy tree.

Definition A.3 (Achievement in $\mathcal{X}(P)$)**.** *A policy tree* Π *achieves a literal L in $\mathcal{X}(P)$ if* Π *is applicable in $\mathcal{X}(P)$ and L is true in all the states $s(n)$ associated with the tip nodes n of* Π.

Definition A.4 (Solutions to $\mathcal{X}(P)$)**.** *A policy tree* Π *solves $\mathcal{X}(P)$ if* Π *is applicable in $\mathcal{X}(P)$ and it achieves each goal literal L_G in $\mathcal{X}(P)$.*

The corresponding definitions for the contingent problem P follow.

Definition A.5 (Policy Trees applicable to P)**.** *A policy tree* Π *for a contingent problem P is* applicable *if there is a belief state function $b(n)$ over the nodes n in* Π *such that:*

1. *$b(n_0) = b_0$, where b_0 is the initial state of $\mathcal{X}(P)$ and n_0 is the root node of* Π,
2. *$a(n)$ is executable in $b(n)$ and n has a single child $c(n)$ if $a(n) \neq obs(L)$, and has two children $c^+(n)$ and $c^-(n)$ if $a(n) = obs(L)$, and neither L nor $\neg L$ are in $b(n)$. The belief state for these children is:*
3. *$b(c(n)) = f(b(n), a(n))$ if $a(n) \neq obs(L)$*
4. *$b(c(n)) = b(n)$ if $a(n) = obs(L)$, and either $L \in b(n)$ or $\neg L \in b(n)$*
5. *$b(c^+(n)) = b(n) \cap |L|$ if $a(n) = obs(L)$*
6. *$b(c^-(n)) = b(n) \cap |\neg L|$ if $a(n) = obs(L)$*

In this definition, a belief state $b(n)$ stands for a set of states, $L \in b(n)$ means that L is true in all the states in $b(n)$, and $a(n)$ is applicable in $b(n)$ if for each precondition L of a it occurs that $L \in b(n)$. Likewise, $|L|$ and $|\neg L|$ stand for the set of states where L is true, and false, respectively. The belief-state transition function is

$$f(b,a) = \{s' \mid s \in b \text{ and } f(s,a) = s'\}$$

Note that a node n, where $a(n) = obs(L)$, must necessarily "branch" into two children nodes $c^+(n)$ and $c^-(n)$ when the outcome of the observation cannot be predicted, when neither L nor $\neg L$ are true in $b(n)$. On the other hand, branching is allowed but not forced in a policy tree Π, when the observation can be predicted, because either L or $\neg L$ are in $b(n)$. In such case, if n doesn't branch, the next belief state $b(c(n))$ is equal to $b(n)$, and if n branches, one of the possible next belief states $b(c^+(n))$ or $b(c^-(n))$ will be equal to $b(n)$ while the other one will be empty. This last option will be possible actually when the policy trees for P is obtained from a sound but incomplete $\mathcal{X}(P)$.

Definition A.6 (Achievement in P)**.** *A policy tree* Π *achieves a literal* L *in* P *if* Π *is applicable in* P*, and* L *is true in all the belief states* $b(n)$ *associated with the tip nodes* n *of* Π*.*

Definition A.7 (Solution to P)**.** *A policy tree* Π *solves a contingent problem* P *if* Π *is applicable in* P *and achieves each goal literal* L_G *of* P*.*

2 Policy Trees as Sets of Action Sequences

For establishing the correspondence between the policy trees that solve $\mathcal{X}(P)$ and P, it is convenient to associate with each branch p of a policy Π a *deterministic* action sequence $\pi(p)$ that represents in a suitable sense the effects in the planning problem that results from following such branch in the policy tree. For defining this action sequence, we will assume that P and $\mathcal{X}(P)$ contain some "dummy action" that are not part of the policy trees for them, but that may belong to sequences $\pi(p)$.

For every action $obs(L)$ in P, where L is a positive literal, we assume that P contains also the deterministic action $obs^+(L)$, $obs^-(L)$ and $nobs(L)$, all with the same precondition of the action $obs(L)$ but with no effects. Similarly, for $\mathcal{X}(P)$, we assume that for every non-deterministic $set(L)$ action with effect $C \rightarrow KL|K\neg L$, the problem $\mathcal{X}(P)$ contains the deterministic action set $set^+(L), set^+(\neg L)$ and $nset(L)$, all with the same precondition of the action $set(L)$ but the first with the effect KL, the second with effect $K\neg L$, and the third with no effect.

Definition A.8 (Action Sequences associated to Π)**.** *Let* $p = (n_0, \ldots, n_k)$ *be a rooted directed path in a policy tree* Π *for either* P *or* $\mathcal{X}(P)$*. The action sequence* $\pi(p)$ *associated with this path is* Π *is then* $[a_0, \ldots, a_{k-1}]$*, where for any* $i = 0, \ldots, k$*, the action* a_i *is:*

- $a(n_i)$*, if* $a(n_i) \notin \{obs(L), set(L)\}$*;*
- $nobs(L)$ *or* $nset(L)$*, if* $a(n_i)$ *is* $obs(L)$ *or* $set(L)$ *resp. and* $n_{i+1} = c(n_i)$*;*
- $set^+(L)$ *or* $set^-(L)$*, if* $a(n_i) = set(L)$ *and* n_{i+1} *is* $c^+(n_i)$ *or* $c^-(n_i)$ *resp.;*
- $obs^+(L)$ *or* $obs^-(L)$*, if* $a(n_i) = obs(L)$ *and* n_{i+1} *is* $c^+(n_i)$ *or* $c^-(n_i)$ *resp.*

We refer to the rooted directed paths p in Π simply as *paths* or *branches.* A branch p determines an action sequence $\pi(p)$ that is unique to this branch; indeed, for two branches p and p', if $p \neq p'$, then $\pi(p) \neq \pi(p')$ [2]

We say that a branch $p = (n_0, \ldots, n_k)$ is *complete* if n_k is a tip node in Π, and similarly, that an action sequence π is *complete* if $\pi(p) = \pi$, with p complete. From now on we will consider only *proper* policy trees, i.e. policy trees that only have complete branches.

[2]The proof for this statement: let $\pi = \pi(p), \pi' = \pi(p')$, and let $p \neq p'$. Let also $n = (n_0, \ldots, n_i)$ be the longest sub-path that is common to both p and p'; with the nodes n_{i+1} and n'_{i+1} the successors nodes of n_i and n'_i in Π respectively. If Π is applicable, then the actions $a_i \in \pi$ and $a'_i \in \pi'$ must be of the form $set^+(L)$ and $set^-(L)$ in $\mathcal{X}(P)$, or of the form $obs^+(L)$ and $obs^-(L)$ in P, (otherwise $n_{i+1} = n'_{i+1}$ and n would not be the longest common sub-path of p and p'). In both cases $a_i \neq a'_i$, and hence $\pi \neq \pi'$.

3 Characterising $\mathcal{X}(P)$ in terms of Classical Plans

Given a branch p in a proper policy tree Π for $\mathcal{X}(P)$, the action sequence $\pi(p)$ associated to p captures the plans for $\mathcal{X}(P)$ from the structure of Π.

Definition A.9. *Let be Π a proper policy tree for $\mathcal{X}(P)$, and let $p = (n_0, \ldots, n_k)$ be a branch in Π. The branch p is applicable in $\mathcal{X}(P)$ if the action $a(n_i)$ is executable in $s(n_i)$ for every node n_i of p.*

From the definition A.9, any path $p = (n_0, \ldots, n_k)$ in Π is associated to a sequence of states $S = \big(s(n_0), \ldots, s(n_k)\big)$ in $\mathcal{X}(P)$.

Theorem A.10. *Let Π be a proper policy tree for $\mathcal{X}(P)$. If Π solves $\mathcal{X}(P)$, then $\pi(p)$ is executable and is a classical plan that solves $\mathcal{X}(P)$, for each branch p in Π.*

Proof. Π solves $\mathcal{X}(P)$ then any branch p in Π is applicable, and result in a node n to which correspond a state $s(n)$ in $\mathcal{X}(P)$, where the goal literals hold. From definition A.9, the action sequence $\pi(p)$ is executable in $\mathcal{X}(P)$ and achieves the goal state $s(n)$. Then $\pi(p)$ solves $\mathcal{X}(P)$ for all the branches p of the policy tree Π. □

4 Characterising P in Terms of Classical Plans

We turn on establishing a similar correspondence between policy trees and classical plans for the contingent problem P. For this, let $P|_s$ stands for the *classical planning problem* obtained by setting the initial situation $\mathcal{I}$ of P to be the state s, while removing the sensing actions.

Definition A.11 (Applicable branch)**.** *Let Π be a proper policy tree and p a branch of Π. We say that p is applicable in P if the action sequence $\pi = \pi(p)$ is executable in the initial belief state b_0 of P, and the resulting belief is given by the union of the states resulting from applying π in each state s_i of b_0:*

$$result(\pi, b_0) = \bigcup_{s_i \in b_0} result(\pi, s_i) \tag{A.1}$$

and $result(\pi, b_0) \neq \emptyset$.

For every node n_i of a branch p of a proper policy tree Π, the executability of the action $a(n_i)$ in $b(n_i)$ yields that the resulting belief state $result\big(a(n_i), b(n_i)\big)$ is not empty. From this remark, follows that

Proposition A.12. *Given an applicable proper policy tree Π for P. For every initial state $s \in b_0$ exists a branch p of Π such that the corresponding action sequence $\pi(p)$ achieves a state s' from s, i.e. $result(\pi(p), s) = s'$.*

Proof. Consider the converse that exists a state s_k in b_0 such that $result(\pi(p), s_k) = \emptyset$ for every branch p of Π. Then there exist a node n_i of p such that the action $a(n_i)$ is not executable in the state s_i resulting from applying the actions prefix of $\pi(p)$ until $a(n_i)$: $s_i = result\big(\big[a(n_0), \ldots, a(n_{j-1})\big], s_k\big)$.

If the action $a(n_i)$ is not executable in s_i, then two possibilities arise:

1. $a(n_i)$ is a regular action; then if it is not executable in s_i, it is not executable in $b(n_i)$, contradicting the hypothesis of applicability of Π in P.

2. $a(n_i)$ is $obs(L)$, for a given L; then one of the two child states has to be non-empty, as L and $\neg L$ can't be both in s_i, which contradicts our hypothesis that $result(\pi(p), s_k) = \emptyset$.

□

From the two former results, follows the next definition:

Definition A.13 (Applicable policy tree)**.** *A proper policy tree* Π *is* applicable *to* P *when for every state* $s \in b_0$ *exists a branch* p *of* Π *such that* $\pi(p)$ *is executable in* $P|_s$*, and that* $result(\pi(p), s) \neq \emptyset$*.*

This definition implies that the applicability of a policy tree is in direct relation with the actual executability of the corresponding actions in a planning problem. From the definition A.11, for a branch $p = (n_0, \ldots, n_k)$ of a proper policy tree Π applicable to P, the resulting belief state of executing $\pi(p)$ in the initial belief state $b_0(p)$ is the non empty belief $b(n_k)$:

$$b(n_k) = \{result(\pi(p), s) \mid s \in b_0(p)\} \tag{A.2}$$

Proposition A.14. *Let* Π *be a proper policy tree applicable to* P*. If* Π *achieves a literal* L *in* P*, then for every initial state* $s \in b_0$ *a branch* p *in* Π *is such that the action sequence* $\pi(p)$ *achieves* L *in* $P|_s$*.*

Proof. Direct from the expression (A.2) of resulting belief state above and from definition A.11, where a policy tree Π achieves a literal L in P when L is true in all the belief states $b(n)$ associated to the tip nodes of Π. This results in having, for any initial initial state $s \in b_0$, a path p in Π such that $L \in result\big(\pi(p), s\big)$; thus $\pi(p)$ achieves L in $P|_s$. □

Theorem A.15 (Contingent and Classical Plans)**.** *Let* Π *be a proper policy tree applicable to* P*. If* Π *solves* P*, then for every initial state* $s \in b_0$ *a branch* p *of* Π *is such that the action sequence* $\pi(p)$ *is a classical plan that solves* $P|_s$*.*

Proof. For any branches p in an applicable policy tree Π, the action sequence $\pi(p)$ is executable in P, from definition A.11. Then, directly from Proposition A.14, a policy tree Π solves P when the goal is achieved by all its complete branches p in Π, thus for any state s in the initial belief state b_0 of P, there is a path p such that the deterministic action sequence $\pi(p)$ achieves the goal in $P|_s$. Consequently the action sequence $\pi(p)$ is a classical plan that solves $P|_s$. □

5 Soundness

We prove the soundness of $\mathcal{X}(P)$ by showing that policy trees Π^* that solve $\mathcal{X}(P)$ can be easily transformed in policy trees Π that solve the contingent problem P.

For these transformation, it will be convenient to generalise the policy trees Π for $\mathcal{X}(P)$ by allowing the actions $a(n)$ associated with the nodes n in Π to stand not just for a single action a, but for certain *action sequences* as well. In particular we write Π^* to denote a policy tree for $\mathcal{X}(P)$ where $a(n)$ is restricted to be any action a in $\mathcal{X}(P)$ that is also in P, followed by a (possibly empty) sequence of *deductive actions* in $\mathcal{X}(P)$ (i.e. actions that stand for full merges or tag refutations). We will denote from now on such action sequence as a^*. From a policy tree Π^* that solves $\mathcal{X}(P)$, we obtain the policy tree Π for P by simply dropping the deductive actions in each $a(n) = a^*$, and replacing the $set(L)$ actions by $obs(L)$ actions, and vice-versa: given a policy tree Π that solves P, we obtain the corresponding policy tree Π^* for $\mathcal{X}(P)$ by replacing each $a(n) = a$ by $a(n) = a^*$, where a^* equals a followed by all the deductive actions in $\mathcal{X}(P)$, in any order.

We refer to the action sequences associated with p in Π^* as $\pi^*(p)$, and leave $\pi(p)$ to refer to the action sequence associated to p in Π.

Last, while we allow $a(n)$ to stand for an action sequence a^* in $\mathcal{X}(P)$, we will keep referring to $a(n)$ as an action. If $a(n) = a^*$, the action that $a(n)$ represents is just the action a present in P: we do not need to work out the form of this action, but just notice that it has the same preconditions as a, as the deductive rules in a^* have no preconditions.

Lemma A.16 (Soundness of deductive rules)**.** *Let be P a contingent planning problem, and $\mathcal{X}(P)$ a valid[3] translation of P following definition 6.1. Given a branch p in a proper policy tree applicable in $\mathcal{X}(P)$, we consider two states s^* and s, respectively in $\mathcal{X}(P)$ and $P|_{s_i}$, resulting from applying $\pi^*(p)$ in s_0 and $\pi(p)$ in s_i, for an initial state s_i in b_0. We assume that when KL/t is in s^*, then L is in s when $s_i \models t$, with t a conjunction of literals.*

Given an executable action a^* in s^*, we claim that if a^* achieves KL in $f(s^*, a^*)$, then the action a achieves L in $f(s, a)$.

Proof. If the action a^* with precondition a conjunction of literals C is executable in s^*, then it is also executable in s by construction of s^* and s. In fact, if KC is in s^*, then C is in s for any initial state s_i.

Three scenarios are then possible:

1. a has a (deterministic) effect that adds KL, then the claim comes because a is executable in both s and s^*, and deductive rules do not delete any atom.

2. a hasn't an effect that adds KL, but the deductive rules in a^* do. We prove the lemma by examining the application of deductive rules case by case, checking that the closure rules applied in s^* yield atoms $KL \in f(s^*, a^*)$ if $L \in f(s, a)$.

Full Merge Let's recall that the Full Merge rule is:

$$\bigwedge_{t \in m} (KL/t \vee K\neg t) \; \rightarrow \; KL.$$

If the Full Merge rule yields KL, then its condition has to hold in s^*, i.e. $KL/t \in s^*$, or $K\neg t \in s^*$.

[3]We recall that a valid translation has all the merges with at least one of the tags true in $\mathcal{I}$.

If $KL/t \in s^*$, it comes from the hypothesis that L was already present in s if $s_i \models t$. The case of $K\neg t \in s^*$ can't hold, otherwise s_i would not satisfy t. In fact such tag-fluents considered for the translation are limited only to *static* literals, as explained at page 100.

Tag Refutation Assuming that we are in the Tag Refutation case, with the deductive rule being:

$$KL'/t \wedge K\neg L' \rightarrow K\neg t.$$

If Tag Refutation rule yields KL when applied in s^* (i.e. we have $t = \neg L$), then the rule condition $KL'/t \wedge K\neg L'$ has to hold in the state. However, $K\neg L'$ is not true in s^* by hypothesis: having KL'/t and $K\neg L'$ in s^* would bring that both L' in s (because s_i satisfies t), and $\neg L'$ in s, which is a contradiction.

3. KL was already present in s^*. If KL has not been deleted by a^*, then L is present in s by hypothesis, and this literals persists in $f(s, a)$. □

So, the deductive rules add in $\mathcal{X}(P)$ only atoms that are already present in the correspondent belief state in P.

Lemma A.17. *Let be a contingent problem P, and a translation $\mathcal{X}(P)$. For any branch p of a proper policy tree Π, if $\pi^*(p)$ achieves KL/t in $\mathcal{X}(P)$, then $\pi(p)$ achieves L in $P|_s$ for all $s \in b_0(p)$ such that $s \models t$.*

Proof. We prove the statement by induction on the length of $\pi^*(p) = [a_0, \ldots, a_i]$, with a_0 being the empty action ε.

Initially, if $\pi^* = \{\varepsilon\}$, the proof is direct from the definition of $\mathcal{X}(P)$ (cf. definition 6.1): by construction KL/t belongs to the initial situation of $\mathcal{X}(P)$ when L is in all the initial states s that satisfy t (and by consistency of the tags in the translation[4], such initial states exist). It follows that L is achieved in $P|_s$ by $\pi^* = \{\varepsilon\}$ in the initial situation.

Inductive step over the length of $p = (n_0, \ldots, n_i)$: the action sequence $\pi^*(p) = \pi^*_{i-1} = [a_0, \ldots, a_{i-1}]$ achieves KL/t in $\mathcal{X}(P)$, then π_{i-1} achieves L in $P|_s$ for all $s \in b_0(p)$ such that s satisfies t.

Now $\pi^*(p) = [a_0, \ldots, a_{i-1}, a_i] = [\pi^*_{i-1}, a_i]$ achieves KL/t in $\mathcal{X}(P)$. Then, either

1. π^*_{i-1} had achieved KL/t;
2. π^*_{i-1} had not achieved KL/t, but a_i does.

We prove those cases one after the other.

1. By inductive hypothesis, if π^*_{i-1} had achieved KL/t, then L has been achieved in $P|_s$. If the action a_i deletes[5] L in $P|_s$, then it should delete KL/t in $\mathcal{X}(P)$ by construction of the actions in the translation: if $a_i : C \rightarrow \neg L$ in P, then in $\mathcal{X}(P)$ the support rule is such that $a_i : KC/t \rightarrow K\neg L/t \wedge \neg KL/t$. An a_i of that form deletes KL/t, and consequently contradicts the hypothesis that $\pi^*(p)$ achieves KL/t. Thus L is achieved in $P|_s$.

[4] We recall that a tag t is consistent if all the literals in t are true in some initial state.

[5] We excluded that a_i can be $set^-(L)$ because, by tag refutation application, we would obtain $K\neg t$ in $\mathcal{X}(P)$, implying that s doesn't satisfy t and contradicting the hypothesis.

2. When a_i do achieves KL/t, we distinguish two cases:

   (i) The action $a_i = set^+(L) : \top \rightarrow KL \wedge \neg K\neg L$ yields that KL/t is achieved by $\pi^*(p)$ in $\mathcal{X}(P)$. In $P|_s$ the correspondent sensing action is $obs^+(L) : \top \rightarrow L$ is applicable and $\pi(p)$ achieves L in $P|_s$.

   (ii) The action a_i is a regular action, and in $\mathcal{X}(P)$ the support rule is such that $a_i : KC/t \rightarrow KL/t$, with KC/t already achieved by π^*_{i-1}. By inductive hypothesis, π_{i-1} achieves C in $P|_s$, and in P the corresponding action $a_i : C \rightarrow L$ is executable in $P|_s$, bringing that $\pi(p)$ achieves L in $P|_s$. □

Corollary A.18. *If a policy tree achieves KL in $\mathcal{X}(P)$, then L is achieved in all the final states of P.*

Proof. Direct from checking that if the preconditions of the actions are achieved in $\mathcal{X}(P)$, then they also are in P.

From the validity of the merges, it comes that for every initial state $s \in b_0$ of P, exists a tag t satisfied by s. To the contrary, there would exist a state $s' \in b_0$ that would not satisfy any tag, denying the validity of the merge m, since $\mathcal{I} \not\models \bigvee_{t \in m} t$. The consistency of tags ensure that the state that satisfies t exists and is consistent.

If KL is achieved in $\mathcal{X}(P)$, then KL/t is also achieved in $\mathcal{X}(P)$ for every tag t in a valid merge m for which there is an initial state s_i that satisfies it. Thus, from the Lemma A.17, if KL is achieved in $\mathcal{X}(P)$, then L is also achieved in $P|_s$, for any initial state s. □

Corollary A.19. *If a policy tree is applicable in $\mathcal{X}(P)$, it is also applicable in P.*

Proof. Direct from the former Lemma A.17: if the preconditions of the actions are achieved in $\mathcal{X}(P)$, then they also are in P. □

Theorem A.20 (Soundness)**.** *Consider a contingent planning problem P and a valid translation $\mathcal{X}(P)$, let Π^* be a policy tree for $\mathcal{X}(P)$ and Π the corresponding policy tree for P obtained from Π^* by dropping the deductive actions and replacing the $set(L)$ actions by $obs(L)$ actions. Then if Π^* solves $\mathcal{X}(P)$, Π solves the problem P.*

Proof. If Π^* achieves the goal for $\mathcal{X}(P)$, then by the Theorem A.10, Π^* is proper and applicable, achieving the goal literals KL_G in all states $s(n_k)$ corresponding to the tip nodes n_k of Π^*. Then, from Lemma A.17 and Corollary A.19, the policy tree Π is proper and applicable in P, achieving the goal literals L_G in $P|_s$, as by the Theorem A.15, we know that exists a branch p in Π that achieves the goal in $P|_s$, for every initial states $s \in b_0$. Thus Π solves the problem P. □

6 Completeness

We prove completeness of $\mathcal{X}(P)$ by showing that under certain conditions, policy trees Π that solve the contingent problem P can be transformed in policy trees Π^* that solve the non-deterministic fully observable problem $\mathcal{X}(P)$.

We recall here some useful definition that we use to prove completeness.

Definition 6.8 (Set of relevant observables) *Given a literal L in P, the set of observables $O(L)$ is the set of literals $\{L_i\}_{i=0}^{n}$ such that, given a clause $c = (c_1, \ldots, c_m)$ in $\mathcal{I}$, the literals L_i verify:*

- *$c_1 \rightarrow L_i$, for a fixed i, and*
- *exists a $c_k \in c$ such that $c_k \rightarrow L_j$, where $L_j \in O(L)$ or $L_j = L$.*

If $i = j$, the membership of L_i in $O(L)$ is trivial, as L_i already belongs to $O(L)$.

The set $O(L)$ is *unique* by construction, as all the literals in $O(L)$ are the ones for which literals in clauses c are relevant to, and such clauses are identifiable by being the ones that include L or a literal relevant to an element of $O(L)$. The set $O(L)$ is *minimal* because it doesn't include duplicate elements.

For characterising the set of clauses $C_{I,O}(L)$ that are relevant to a given literal L, we assume like that $\mathcal{I}$ is in prime implicate form (Marquis, 2000), meaning that $\mathcal{I}$ includes only the inclusion-minimal clauses that it entails but no tautologies, along the tautologies $(L \vee \neg L)$ for complementary literals L and $\neg L$ that do not appear as unit clause in $\mathcal{I}$. The set $C_{I,O}(L)$ can then be defined as the set of non-unary (uncertain) clauses $c \in \mathcal{I}$ such that each literal $L_i \in c$ is relevant to L or to some observable relevant to L.

Definition 6.7 (Set of relevant clauses $C_{I,O}(L)$) *Let be a problem P, with $\mathcal{I}$ in prime implicate form (Marquis, 2000) and let be a literal L in P. $C_{I,O}(L)$ is the set of non-unary clauses $c \in \mathcal{I}$ such that each $c_i \in c$ is relevant to an observable literal in $O(L)$ or to L.*

Given a literal L and given a state s, we denote as $rel(s, L)$ the set of literals in s relevant to L. The set of literals that follow from $\mathcal{I}$ and a set of literals t is called the closure t^* of t:

Definition 4.12 (Deductive closure) *The deductive closure t^* of a tag t under $\mathcal{I}$ is the set of literals entailed by t in $\mathcal{I}$:*

$$t^* = \{L \,|\, \mathcal{I} \wedge t \models L\}$$

From now on, we denote with m the merges in a translation $\mathcal{X}(P)$ of a contingent planning problem P. The notion of satisfaction associates a consistent set of literals t with the *partial truth assignment* that is implicit in the closure t^* of t, and is extended to account for the conditions under which a DNF formula (e.g., a merge for L) satisfies a CNF formula (e.g., $C_{I,O}(L)$).

Definition A.21 (Satisfaction). *A consistent*[6] *set of literals m satisfies a clause $c = (c_1 \vee \ldots \vee c_n)$, if $m^* = \{t_1^*, \ldots, t_n^*\}$ contains at least one literal c_i in c.*

A consistent set of literals m satisfies a collection of clauses $\mathcal{C}$, if m satisfies each clause in $\mathcal{C}$.

A collection Σ of consistent sets of literals satisfies a collection of clauses $\mathcal{C}$ if each set in Σ satisfies $\mathcal{C}$.

[6]A merge $m = \{t_1, \ldots, t_n\}$ is consistent if m^* does not contain a pair of complementary tags.

For a translation $\mathcal{X}(P)$ of P, the type of merges required for completeness are simply the valid merges m that satisfy the set of clauses $C_{I,O}(L)$, with L a precondition or a goal literal of the problem. We call them *covering merges*:

Definition 6.9 (Covering translation) *$\mathcal{X}(P)$ is a covering translation for a contingent problem P if for each precondition and goal literal L in P such that $C_{I,O}(L)$ is not empty, the set M of all the merges for $\mathcal{X}(P)$ contains a merge m_L that satisfies $C_{I,O}(L)$.*

Lemma A.22. *Let $m = \{t_1, \dots, t_n\}$ be a covering merge for a literal L, with all tags t_i consistent, in a valid translation $\mathcal{X}(P)$ of a contingent problem P which initial situation is in prime implicate form. Then for each initial state s of P exists a tag t in m such that $rel(s, L) \subseteq t^*$.*

Proof. Say ad absurdum that, fixed an initial state s in b_0, no tag t in m satisfies $rel(s, L) \subseteq t^*$. Then, for any tag t_i in m, there is at least on literal L_i made true in s and that is relevant to L, but does not belong to the closure t_i^*. If we consider c the disjunction of such literals L_i over all the tags in m, we obtain that $\mathcal{I}$ satisfies c, since $\mathcal{I}$ is in prime implicate form, and s is a valid state of $\mathcal{I}$. Consequently, the clause c is included in $C_{I,O}(L)$, as all the L_i are relevant to L. From the definition of covering merge, at least one tag t_k in m includes a literal L_k in c, which contradicts the initial hypothesis, and proves the thesis.

In case the state s does not make true any literal relevant to L, the lemma is trivially proved, as the set $rel(s, L)$ is empty. □

Lemma A.23. *Let Π be a policy tree applicable to P and a covering translation $\mathcal{X}(P)$. For any complete branche $p \in \Pi$ and the states $s \in b_0(p)$, if $\pi(p)$ achieves a literal L in $P|_s$, then the action sequence $\pi^*(p)$ achieves either KL/t or $K\neg t$ in $\mathcal{X}(P)$ if exists a tag t in $\mathcal{X}(P)$ such that $rel(s, L) \subseteq t^*$.*

Proof. We prove the statement by induction on the length of $\pi(p) = \pi = [a_0, \dots, a_i]$, with a_0 being the empty action ε, for any complete branch p of Π.

If $\pi = \{\varepsilon\}$ is the empty plan, then $L \in s$, for any $s \in b_0(p)$. For the Theorem A.25, exists a tag $t \in \mathcal{X}(P)$ such that $rel(s, L) \subseteq t^*$. The literal L is relevant to itself, so $L \in t^*$, and thus KL/t is achieved by $\pi^*(p)$ as, by definition KL/t is in $\mathcal{I}$ when $\mathcal{I} \models (t \supset L)$.

Inductive step over the length of the action sequence $\pi(p) = \pi_{i-1} = [a_0, \dots, a_{i-1}]$: the plan π^*_{i-1} achieves KL/t or $K\neg t$ in $\mathcal{X}(P)$, when π_{i-1} achieves L in $P|_s$ and exists a tag t for which $rel(s, L) \subseteq t^*$.

If $\pi_{+1} = [\pi_{i-1}, a]$, and t is not the empty tag, π_{+1} achieves L in $P|_s$ for one of the following reasons:

A) π_{i-1} achieves C in $P|_s$ for a rule $a : C \to L$ in P, or

B) π_{i-1} achieves L in $P|_s$, and for any rule $a' : C \to \neg L$, the action sequence π_{i-1} achieves $\neg L_i$ for some L_i in C, or

C) a is a sensing action.

If A, by inductive hypothesis, π^*_{i-1} achieves KC/t in $\mathcal{X}(P)$, for a tag t when $rel(s,L) \subseteq t^*$. Hence, π^*_{+1} achieves KL/t, from the support rule $a : KC/t \rightarrow KL/t$. The reasoning rules do not delete any atom, so if the sequence $\pi^*_{+1} = [\pi^*_{i-1}, a]$ achieves KL/t in $\mathcal{X}(P)$.

If B, by inductive hypothesis, π^*_{i-1} achieves both $K\neg L_i/t$ (for some literal $L_i \in C_i$), and KL/t in $\mathcal{X}(P)$. Thus, the cancellation rule $a : \neg K\neg C/t \rightarrow \neg KL/t$ has a false condition, therefore KL/t persists and $\pi^*_{+1} = [\pi^*_{i-1}, a^*]$ achieves KL/t.

If C, we distinguish three cases: $a = obs^+(N)$, $a = obs^+(L)$ and $a = obs^-(L)$, with $N \neq L$. In case the literal $K\neg N/t$ has been achieved in $\mathcal{X}(P)$, the $set^+(N)$ rule yields KN in $\mathcal{X}(P)$. By tag refutation rule, $K\neg t$ is achieved by π^*_{+1}. Otherwise, if L has been achieved in $P|_s$ by π_{i-1}, then by inductive hypothesis KL/t has been achieved in $\mathcal{X}(P)$.
If $obs(L) = true$, then KL (and KL/t) is achieved in $\mathcal{X}(P)$ by $a = set^+(L)$.
If $obs(L) = false$, KL/t is achieved by π^*_{i-1} by inductive hypothesis, as L is achieved in $P|_s$. From KL/t and $K\neg L$ (the effects of $set^-(L)$) the tag refutation rule achieves $K\neg t$ in $\mathcal{X}(P)$. This tag is static and is not modified by successive actions, thus π^*_{+1} achieves $K\neg t$ in $\mathcal{X}(P)$. □

Lemma A.24. *Let Π be a policy tree for P and a covering translation $\mathcal{X}(P)$. If Π is applicable to P, then the policy tree Π^* is applicable in $\mathcal{X}(P)$.*

Proof. The proof is direct from the result of the former Lemma A.23, applied on the precondition L of the actions in every branch of Π. □

Theorem A.25 (Completeness of covering translations)**.** *Let be a contingent problem P, and $\mathcal{X}(P)$ a covering translation. If Π is a proper policy tree that solves P, then Π^* is a policy tree that solves $\mathcal{X}(P)$.*

Proof. The proof comes directly from the former lemmas, after noticing that every covering translation has a merge that covers any goal literal L_G in P. If a policy tree Π is a solution that achieves the goal literals in P, then the policy tree Π^* is applicable and achieves the goal literals in $\mathcal{X}(P)$, being a solution for the translated problem. □

Theorem A.26. *Given a contingent problem P, and a translation $\mathcal{X}(P)$ with the set of tags T fixed such that each tag t_i is the set of all the literals true in an initial state s_i, and a single merge m equal to the set of tags. Then $\mathcal{X}(P)$ is a covering translation, and hence sound and complete.*

Proof. The soundness of the translation comes from the validity of the merge m as by definition $\mathcal{I}$ entails all the initial states. The completeness comes directly from the former Theorem A.25 and the fact that $\mathcal{X}(P)$ is a covering translation, as the set $C_{I,O}$ is banally satisfied by the disjunction of all the initial states. $C_{I,O}$ is the set of clauses $C_{I,O}(L)$ for each precondition or goal literal L. □

Theorem 6.6 (Completeness and soundness of X_{S0}) *The translation $X_{S_0}(P)$ at page 102 is sound and complete.*

Proof. Direct from the Theorem A.26 above. □

We recall that the cover $c(\mathcal{C})$ of a set of clauses $\mathcal{C}$ is the minimal set of literals S consistent with the initial situation $\mathcal{I}$ such that S contains a literal of each clause in $\mathcal{C}$ (cf. definition 3.14). This definition will be useful in the next theorem:

Theorem 6.15 (Completeness and soundness of X_i) *Given a contingent problem P, and a positive integer i such that $i \geq w(P)$, the translation $X_i(P)$ defined at page 105 is valid and covering, hence is sound and complete.*

Proof. For soundness, we just need to prove that all merges m in $X_i(P)$ are valid and that all tags t in $X_i(P)$ are consistent. The merges m for a literal L in $X_i(P)$ are given by the covers $c(\mathcal{C})$ of collections $\mathcal{C}$ of i or less clauses in $C^*_{I,O}(\mathrm{L})$. Since each model $\mathcal{M}$ of $\mathcal{I}$ satisfies $C^*_{I,O}(\mathrm{L})$, it satisfies some t in $c(\mathcal{C})$. For $m = c(\mathcal{C})$, it comes that at least one tag in m is initially true: $\mathcal{I} \models \bigvee_{t \in m} t$, and hence the merge is valid. At the same time, a cover $c(\mathcal{C})$ is consistent with $\mathcal{I}$, so do are each of the tags t. Using Theorem A.20, the soundness is proved.

For proving completeness, if the width of the problem is bounded by i, then also is the width $w(L)$, for each precondition or goal literal L. Therefore, for each such literal L there is a set $\mathcal{C}$ of clauses in $C^*_{I,O}(\mathrm{L})$ for which $c(\mathcal{C})$ satisfies $C_{I,O}(L)$. The translation $X_i(P)$ then generates an unique merge $m = c(\mathcal{C})$ that covers L. Since $X_i(P)$ is a valid translation, then $X_i(P)$ is also covering, then is complete by virtue of Theorem A.25. □

7 Consistency

We have been assuming throughout the paper that the contingent planning problems P and their translations $\mathcal{X}(P)$ are consistent.

We assume at this point that states are not truth-assignments but sets of literals over the fluents of the language that are true. A state s is *complete* if for every literal L, either L or $\neg L$ is in s, and *consistent* if for no literal L, both L and $\neg L$ are in s. Complete and consistent states represent truth-assignments over the fluents $\mathcal{F}$ and the consistency of P and of the translation $\mathcal{X}(P)$ ensures that all applicable action sequences π map complete and consistent states s into complete and consistent states s'. Once this is guaranteed, complete and consistent states can be referred to simply as states which is what we have done in the paper.

Given a complete state s and an action a applicable in s, the next state s_a is complete if it satisfies the following conditions, given in terms of the delete and add lists of the action a in P, from the definitions given in chapter 1 on page 9:

$$s_a = \big(s \setminus del(a,s)\big) \cup add(a,s) \tag{A.3}$$

where

$$add(a,s) = \{L \mid a : C \to L \text{ and } C \subseteq s\} \tag{A.4}$$

$$del(a,s) = \{\neg L \mid L \in add(a,s)\} \tag{A.5}$$

It follows from these rules that if s is a complete state then s_a is a complete state when the applied action a only deletes a literal L in s if its negation $\neg L$ is added. Notice that these conditions do not preserve consistency: s may be consistent and

s_a inconsistent[7]. To guarantee that all reachable state is complete and consistent, and thus represent genuine truth assignments over the fluents in $\mathcal{F}$, a consistency condition on P needs to be defined.

Definition A.27 (Consistency)**.** *A classical, conformant, or contingent problem P is consistent if the initial situation $\mathcal{I}$ is logically consistent and every pair of complementary literals L and $\neg L$ is mutex in P.*

In a consistent classical problem P, all the reachable states are complete and consistent, and the standard progression lemma (Palacios and Geffner, 2009) used in the preceding proofs holds:

Lemma A.1 (Progression in Classical planning)**.** *An action sequence $\pi_{+1} = [\pi, a]$ applicable in the complete and consistent state s achieves a literal L in a consistent classical problem P iff A) π achieves the body C of a rule $a : C \to L$ in P, or B) π achieves L and for every rule $a : C' \to \neg L$, π achieves $\neg L'$ for a literal L' in C'.*

The notion of mutex used in the definition of consistency expresses a guarantee that a pair of literals L and L' cannot be both true in a reachable state. Sufficient and polynomial conditions for mutual exclusivity and other type of invariants have been defined in various papers, here we follow the definition by Bonet and Geffner (1999).

Definition A.28 (Mutex Set)**.** *A mutex set is a collection R of unordered literals pairs (L, L') over a classical or contingent problem P such that:*

1. *for no pair (L, L') in R, both L and L' are in a possible initial state s,*
2. *if $a : C \to L$ and $a : C' \to L'$ are two conditional effects for the same action a, and (L, L') is a pair in R, then the set $(pre(a) \cup C \cup C')$ contains two literals p and q that are mutex in R,*
3. *if $a : C \to L$ is a conditional effect in P for an action a, and the literal L belongs to a pair (L, L') in R, then either a) $L' = \neg L$, b) the set $(pre(a) \cup C)$ contains a literal that is mutex with L', or c) the action a has an effect $a : C' \to \neg L'$, and the set $(pre(a) \cup C)$ implies C'.*

For simplicity and without loss of generality, we will assume in the proofs that preconditions $pre(a)$ are empty: it is simple to show that the mutexes of a problem P remain the same if preconditions are pushed in as conditions. We also assume that no condition C in a rule $C \to L$ in P includes mutex fluents, as these rules can be simply pruned because the body never holds.

Theorem A.29. *If (L, L') is a pair in a mutex set R of a classical, conformant, or contingent problem P, then the mutex literals don't hold together in any reachable state s in P, i.e. $\{L, L'\} \not\subseteq s$.*

Proof. The theorem has been proved by Palacios and Geffner (2009) for classical an conformant planning problems. We now turn on contingent planning problems $P = < \mathcal{F}, \mathcal{A}, \mathcal{O}, \mathcal{I}, \mathcal{G} >$.

[7]Consider as an example the following rules $a : C \to L$, and $a : C' \to \neg L$. In the case both C and C' are in s.

Given Π be a proper policy tree for P, and $p = (n_0, \ldots, n_i)$ a rooted directed path in Π. We prove the hypothesis by recursion on the length i of the action sequence $\pi(p) = [a_0, \ldots, a_i]$ associated to p, for any p in Π, and with a_0 the empty action.

For $\pi(p) = [a_0]$, L and L' cannot be part of a possible initial state, as this is ruled out by the definition A.28 of mutex set.

From the inductive hypothesis, $\{L, L'\} \not\subseteq s$ for any state s reachable in less than $i+1$ steps by applying $\pi(p) = [a_0, \ldots, a_{i-1}]$ to the initial belief b_0.

For $\pi_{+1} = [a_0, \ldots, a_{i-1}, a_i]$ then we distinguish three cases for which L and L' can be in a state s_{+1}, and we prove that none of them is possible:

A) a_i is a sensing action that adds L, while L' already was in s (or the converse). If L has been observed to be true, then $L \in s$, as sensing actions reveal the truth value of a fluent in a state, withou changing it. But form the inductive hypothesis, both L' and L cannot be in s as they are mutex, which invalidates that L can be observed to be true if $L' \in s$.

B) a_i is a regular action that adds both L and L'. Then a has two conditional effects $C \to L$ and $C' \to L'$. If L and L' are mutexes, then, by the point 2 in definition A.28, the set $(C \cup C')$ contains two mutex literals implying that the conditions C and C' cannot hold at the same time: either L is added, or L', or none, but never both. The case L and L' are add effects of the same rule is not possible either, as in this case the condition would also include a mutex pair, and be pruned from the problem for that.

C) a_i is a regular action that adds L, and L' was in s and is not deleted by a (or the converse). Say that an effect $C \to L$ of a adds L with $C \subseteq s$ and no other effect of a is such that $C' \to \neg L'$, because $L' \in s_{+1}$. This case also cannot hold, as from $L' \in s_{+1}$ and the inductive hypothesis, it follows that either 1) C is not mutex with L', or 2) $L' = \neg L$. In the first case, this contraddicts the third point in definition A.28, so this case is impossible. Finally, if $L' = \neg L$, then L' should be in $del(a)$ following eq.(A.5), because the states are defined to be complete; this implies that the case 2) cannot hold.

$\square$

To prove the consistency of the non-deterministic fully observable problem $\mathcal{X}(P) = X_{T,M}(P)$, we start from proving that if P is consistent, so is the translated problem $\mathcal{X}(P)$. For the proof, we assume without loss of generality that the heads KL of the merge actions in $\mathcal{X}(P) = X_{T,M}(P)$ are extended with the literals $K\neg L'$ for the literals L' that are mutex with L in P.

The consistency of the classical problems $P|_s$ for possible initial states $s \in b_0$ is direct, as the set of mutexes in P is a subset of the set of mutexes in $P|_s$ where the initial situation is more constrained.

Given a traslation $\mathcal{X}(P)$, the set of mutexes R_T in the translation depends on the mutexes R in the contingent problem P, and in particular:

Proposition A.30 (Mutex Set R_T). *Let be a valid transition $\mathcal{X}(P)$ of a consistent contingent problem P with mutex set R. Let be the set R_T of literals given by*

$$R_T = \Big\{(KL/t, KL'/t') \cup (KL/t, \neg K\neg L'/t') \mid (L, L') \in R\Big\} \tag{A.6}$$

with the two sets of literals t ad t' joinly satisfiable in $\mathcal{I}$, i.e. $\mathcal{I} \not\models \neg(t \cup t')$.

We claim that R_T is a mutex set.

Proof. We need to prove the three properties of a mutex set in definition A.28.

1. Initially, and from the definition of the $X_{T,M}$ translation, the literal KL/t holds in $X_{T,M}$ if $\mathcal{I} \models (t \supset L)$ in the original problem P, and similarly holds KL'/t' when $\mathcal{I} \models (t' \supset L')$. If t and t' are jointly satisfiable in $\mathcal{I}$, then in any initial state s for which $(t \cup t') \subseteq s$ the mutexes L and L' cannot jointly hold, thus either KL/t or KL'/t' can not be in the initial situation of $\mathcal{X}(P)$.

   A similar reasoning can be applied to the negation of L': say that L holds in s and L' does not, then from the consistency of P and the completeness of s comes that $\neg L' \in s$, i.e. $\mathcal{I} \models (t' \supset \neg L')$, and $K\neg L'/t'$ is in the initial situation of $\mathcal{X}(P)$ if KL/t is. Yields that KL/t and the negation $\neg K\neg L'/t'$ cannot be together in the initial situation of $\mathcal{X}(P)$. Such state exist because the merges are valid and consistent.

   No pair (L, L') in R_T can jointly be in an initial state of $\mathcal{X}(P)$.

2. Say an action a has two conditional effects $a : C \to L$ and $a : C' \to L'$ in P, and (L, L') is a mutex pair in R. In $\mathcal{X}(P)$, the action a includes two support rules that add both KL/t and KL'/t': $a : KC/t \to KL/t$ and $a : KC'/t' \to KL'/t'$.

   From the consistency of P, we know that the set $(C \cup C')$ contains a pair of mutex literals $(L_1, L_2) \in R$. Then it comes that before applying a, in the translation KC/t and KC'/t' do not hold together in any initial state that satisfies both t and t', hence $(KC/t \cup KC'/t')$ contains a mutex pair in $\mathcal{X}(P)$.

   Similarly, if from rules $a : C \to L$ and $a : C' \to L'$ in P, the action in $\mathcal{X}(P)$ with rules adding KL/t and $\neg K\neg L'/t$ are the support and cancellation rules of the form $a : KC/t \to KL/t$ and $a : \neg K\neg C'/t' \to \neg K\neg L'/t'$, for a pair of mutex literals L' and L in P, Since L and L' are mutex in P, then $(C \cup C')$ must contain literals that are mutex P, so that the pair $(KC/t, \neg K\neg C'/t')$ is mutex in R_T.

   Notice that a cannot be a reasoning rule, as by construction ,the condition of the rule would hold a pair of mutexes, making it pruned from the problem.

3. We are left to prove that the clauses $(KL/t, KL'/t')$ and $(KL/t', \neg K\neg L'/t')$ in R_T are mutex pairs that complies with the third case of definition A.28.

   Say an action a in P has the conditional effect $a : C \to L$, then in the translation $\mathcal{X}(P)$ it has a conditional effect $a : KC/t \to KL/t$, and (L, L') is a mutex pair in R.

   We consider first the pair $(KL/t, KL'/t')$:

   a) if $L' = \neg L$, then by construction $KL'/t' = K\neg L/t'$. The action $a : C \to L$ in P is translated in $\mathcal{X}(P)$ as the support rule $a : KC/t' \to KL/t'$, and the cancellation rule $a : \neg K\neg C/t' \to \neg K\neg L/t'$; the latter is equivalent to $a : \neg K\neg C/t' \to \neg KL'/t'$. The condition KC/t implies the body $\neg K\neg C/t'$ of the cancellation rule, then for each literal L_i in C, a literal KL_i/t implies a literal $\neg K\neg L_i/t' \in \neg K\neg C/t'$ in $\mathcal{X}(P)$. Then the pair $(KL_i/t, K\neg L_i/t')$

is mutex in R_T, and the literal $K\neg L_i/t'$ does not belong to KC/t as it is mutex with an element of KC/t. The point 3a) of the mutex set's definition is then verified.

b) we prove that, given the point 3b) of the mutex set's definition in P, the same point holds in $\mathcal{X}(P)$. If the action $a : C \to L$ adds L in P, and C contains a literal L_i that is mutex with L', then also the body KC/t of the support rule contains, by construction, a literal KL_i/t that is mutex with KL'/t', yielding that $(KL_i/t, KL'/t') \in R_T$, for t and t' sets that are not mutually exclusive in $\mathcal{I}$. Hence KC/t contains a literal mutex with KL'/t' in $\mathcal{X}(P)$, which proves the claim.

c) given the point 3c), we prove that the same holds in $\mathcal{X}(P)$. If the action a has an effect $a : C' \to \neg L'$, and the fluents L_i in C imply the literals in C', then the body KC/t of the support rule $a : KC/t \to KL/t$ implies the body $\neg K\neg C'/t'$ of the cancellation rule $a : \neg K\neg C'/t' \to \neg KL'/t'$. Hence every fluent KL_i/t in KC/t implies a fluent $\neg K\neg L_i'/t'$ in $\neg K\neg C'/t$, which is the body of the rule that deletes KL'/t'. This proves the claim.

In case the action applied is a contingent merge $a : \bigwedge_{t_i \in m}(KL/t_i \vee K\neg t_i) \to KL$ for a valid merge m, the pair $(KL/t, KL'/t')$ is still mutex if $(L, L') \in R$. We show this by considering the sets t and t', joinly consistent in $\mathcal{I}$: as m is a valid merge, at least a t_i in m is consistent initially with t', so that the pair $(KL/t_i, KL'/t')$ is in R_T. Hence the body of the rule includes at least literal mutex with KL'/t', following 3b).

We do not consider here the tag refutation rule $a : KL/t \wedge K\neg L \to K\neg t$, coming after an observation, as we are considering by hypothesis only sets t and t' that are joinly satisfiable initially, which is not the case if $K\neg t$.

We check now that the claim holds for the second pair $(KL/t, \neg K\neg L'/t')$.

a) if $L' = \neg L$, then by construction $KL'/t' = K\neg L/t'$, and the pair $(KL/t, \neg K\neg L'/t')$ is made by a fluent and its negation, from 3a).

b) we prove the claim by showing that the point 3b) of the definition A.28 holds in $\mathcal{X}(P)$. If an action $a : C \to L$ has the body C that contains a literal mutex with L', then KC/t contains a literal KL_i/t mutex with KL'/t'. By the construction of the actions in the translated problem, every fluent KL/t implies $\neg K\neg L/t$, hence KL'/t' implies a literal $\neg K\neg L'/t'$ which is mutex with KL_i/t and consequently also with KL/t in R_T, proving the claim.

c) we prove the claim by showing that the point 3c) of the definition A.28 holds in $\mathcal{X}(P)$. If the action a has an effect $a : C' \to \neg L'$, and the fluents L_i in C imply the literals in C', then KC/t of the support rule $a : KC/t \to KL/t$ implies the body $\neg K\neg C'/t'$ of the cancellation rule $a : \neg K\neg C'/t' \to \neg KL'/t'$. Indeed every fluent KL_i/t in KC/t implies a fluent $\neg K\neg L_i'/t'$ in $\neg K\neg C'/t$. If we notice that $K\neg L'/t' = \neg(\neg K\neg L'/t')$, then the claim becomes verified for $(KL/t, \neg K\neg L'/t) \in R_T$.

In case the action applied is a contingent merge $a : \bigwedge_{t_i \in m}(KL/t_i \vee K\neg t_i) \to KL$ for a valid merge m, the pair $(KL/t, \neg K\neg L'/t')$ is still mutex if $(L, L') \in R$. We show this by considering the sets t and t', joinly consistent in $\mathcal{I}$: as m is a valid merge, at least a t_i in m is consistent initially with t', so that the pair $(KL/t_i, KL'/t')$ is in R_T for $(L, L') \in R$. As the reasoning rules' heads KL are extended with $K\neg L'$ if L' is mutex with L in P, then $\wedge K\neg L'$ is implied by the contingent merge, and the property 3c) of the mutex set if trivially verified as $K\neg L'/t' = \neg(\neg K\neg L'/t')$.

An action sequence in a branch of a plan for $\mathcal{X}(P)$ is made by $[a_0, \ldots, a_i]$, with $a_0 = \epsilon$ the empty action, applied on the initial state s_0. Then, we proved that for each step, if $(L, L'$ is a mutex pair in R, then the pairs $(KL/t, KL'/t')$ and $(KL/t, \neg K\neg L'/t')$ are mutex pairs in R_T, for t and t' two sets of literals no mutually exclusive in $\mathcal{I}$. $\square$

Theorem 6.2 (Consistency of $\mathcal{X}(P)$) *Let be a contingent problem P. A valid translation $\mathcal{X}(P)$ is consistent if P is consistent.*

Proof. The consistency of the translation $\mathcal{X}(P)$ comes from the Proposition A.30: the translation $\mathcal{X}(P)$ is logically consistent in the initial state s_0, and if (L, L') is a mutex pair in P, then the literals introduced in the set R_T in the translation $\mathcal{X}(P)$ constitute a mutex set, preserving the consistency property. $\square$

Bibliography

A. Albore and P. Bertoli. Generating Safe Assumption-based plans for partially observable, nondeterministic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 495–500, Palo Alto, USA, 2004. AAAI Press. 118, 134, 135

A. Albore and P. Bertoli. Safe LTL Assumption-based planning. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-06)*, Lake District, UK, 2006. 134, 135

A. Albore and H. Geffner. Acting in partially observable environments when achievement of the goal cannot be guaranteed. In *Workshop on Planning and Plan Execution for Real-World Systems (ICAPS'09)*, 2009. xiii, 115

A. Albore, H. Palacios, and H. Geffner. Fast and informed action selection for planning with sensing. In Borrajo, Castillo, and Corchado, editors, *Current Topics in Artificial Intelligence, 12th Conference of the Spanish Association for Artificial Intelligence (CAEPIA). Selected Papers*, volume 4788 of *Lecture Notes in Computer Science*. Springer, Salamanca, Spain, 2007. ISBN 978-3-540-75270-7. xiii, 96

A. Albore, H. Palacios, and H. Geffner. A translation-based approach to contingent planning. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*, Providence, RI, 2009. xiii, 20, 95, 116, 133

A. Albore, H. Palacios, and H. Geffner. Compiling away uncertainty in non-deterministic conformant planning problems. In 21^{st} *European Conf. on Artificial Intelligence (ECAI-10)*, Lisbon, Portugal, 2010. xiii, 77

A. Albore, M. Ramirez, and H. Geffner. Effective heuristics and belief tracking for planning with incomplete information. In *Proc. of Int. Conf. Automated Planning & Scheduling (ICAPS-11)*, pages 2–9, Freiburg, Germany, 2011. xiii, 20, 29, 30, 49

E. Amir. Learning partially observable deterministic action models. In *Int. Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005. 135

C. Anderson, D. Smith, and D. Weld. Conditional effects in graphplan. In R. Simmons, M. Veloso, and S. Smith, editors, *Proc. of the Fourth International Conference on AI Planning Systems (AIPS-98)*, pages 44–53. AAAI Press, 1998. 14

K.J. Åström. Optimal control of Markov Decision Processes with incomplete state estimation. *J. of Mathematical Analysis and Applications*, 10:174–205, 1965. 33, 117

W. De Baene. Challenging a decade of brain research on task switching: Brain activation in the task-switching paradigm reflects adaptation rather than reconfiguration of task sets. In *Human Brain Mapping*, 2011. 4

J. Baier. *Effective Search Techniques for Non-Classical Planning via Reformulation.*

PhD thesis, University of Toronto, 2003. 13

J. Baier and S. McIlraith. Planning with temporally extended goals using heuristic search. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-06)*, pages 342–345, 2006. 11

C. Baral and T. C. Son. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. of ILPS 1997*, pages 387–401. MIT Press, 1997. 29, 37, 65

C. Baral, V. Kreinovich, and R. Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artificial Intelligence*, 122(1-2):241–267, 2000. 20

R. Bellman. *Dynamic Programming.* Princeton University Press, 1957. 32

P. Bertoli and A. Cimatti. Improving heuristics for planning as search in belief space. In *Proc. AIPS'02*, pages 143–152. AAAI Press, 2002. 30, 50, 69, 72, 73, 77, 87, 89, 91

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. of Int. Joint Conf. on AI (IJCAI-01)*, 2001. 29, 72, 77, 91, 112

P. Bertoli, A. Cimatti, and P. Traverso. Interleaving execution and planning for nondeterministic partially observable domains. In *Proc. of European Conf. on Artificial Intelligence (ECAI-04)*, 2004. 134

P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Strong planning under partial observability. *Artificial Intelligence*, 170(4-5):337–384, 2006. 20, 29, 70, 87, 89

A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proc. of the Fourteenth Int. Joint Conf. on Artificial Intelligence (IJCAI-95)*, pages 1636–1642. Morgan Kaufmann, 1995. 14, 31

B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999. 13, 161

B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS'00*, pages 52–61. AAAI Press, 2000. 18, 20, 28, 32, 33, 35, 96, 117

B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129 (1–2):5–33, 2001a. 11, 67

B. Bonet and H. Geffner. GPT: A Tool for planning with uncertainty and partial information. In *Workshop on Planning with Uncertainty and Partial Information (IJCAI'01)*, 2001b. 30

B. Bonet and H. Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. Int. Joint Conf. of AI*, 2011. 113, 144

B. Bonet and B. Givan. Results of the conformant track of the 5th int. planning competition. At `http://www.ldc.usb.ve/~bonet/ipc5/docs/results-conformant.pdf`, 2006. 45

B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism for planning. In *Proceedings of AAAI-97*, pages 714–719. MIT Press, 1997. 13

B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-09)*, pages 34–41. AAAI Press, 2009. 20, 49

J. Boye. Dialogue management for automatic troubleshooting and other problem-solving applications. In *Proc. of 8th SIGdial Workshop on Discourse and Dialogue*, pages 247–255, 2007. 144

R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. In *Proc. of Int. Conf. Automated Planning & Scheduling (ICAPS-04)*, pages 355–364. AAAI Press, 2004. 20, 29, 31, 50, 69, 111

R. Brafman and J. Hoffmann. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170:507–541, 2006. 13, 62, 63, 65

R. Brafman, C. Domshlak, and S.E. Shimony. On graphical modeling of preference and importance. *J. of AI Research*, 25:389–424, 2006. 134

G. Brewka. A rank based description language for qualitative preferences. In *Proc. of European Conf. on Artificial Intelligence (ECAI-04)*, pages 303–307, Bruxelles, Belgium, 2004. 134

R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. 28

D. Bryce, S. Kambhampati, and D. E. Smith. Planning graph heuristics for belief space search. *J. of AI Research*, 26:35–99, 2006. 20, 28, 31, 69, 91, 109, 112

T. Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994. 13

A. Cassandra, L. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *Proc. of AAAI*, pages 1023–1028, 1994. 33, 116

A. Chang and E. Amir. Goal achievement in partially known, partially observable domains. In *Prof of Int. Conf. on Automated Planning & Scheduling (ICAPS-06)*. AAAI Press, 2006. 135

A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *Proc. AAAI-98*, pages 875–881, 1998. 112

J. Claßen, P. Eyerich, G. Lakemeyer, and B. Nebel. Toward the integration of Golog and planning. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI-07)*, pages 1846–1851, 2007. 11

A. Darwiche. Model-based diagnosis using stuctured systems description. *J. of AI Research*, 8:165–222, 1998. 58

A. Darwiche. Compiling knowledge into decomposable negation normal form. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, pages 284–289, 1999. 57

A. Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2): 11–34, 2001a. 57

A. Darwiche. Decomposable Negation Normal Form. *Journal of the ACM*, 48(4): 608–647, 2001b. 72

A. Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. of European Conf. on Artificial Intelligence (ECAI-04)*, pages 328–332, 2004. 69

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1959. 67

N. Een and N. Sorensson. An extensible SAT–solver. *Lecture notes in computer science*, 2919:502–518, 2004. 69, 109

E. Even-Dar, S. M. Kakade, and Y. Mansour. Reinforcement learning in pomdps. In *Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 660–665, 2005. 135

R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971. 7

J. Funge. Interval–valued epistemic fluents. In *AAAI Fall Symposium on Cognitive Robotics*, pages 23–25, 1998. 30

J. Funge. Representing knowledge within the situation calculus using interval-valued epistemic fluents. *Reliable Computing*, 5(1):35–61, 1999. 30

B. Gazen and C. Knoblock. Combining the expressiveness of UCPOP with the efficiency of Graphplan. In S. Steel and R. Alami, editors, *Recent Advances in AI Planning. Proc. 4th European Conf. on Planning (ECP-97). Lect. Notes in AI 1348*, pages 221–233. Springer, 1997. 13

M. Genesereth and I. Nourbakhsh. Time-Saving tips for problem solving with incomplete information. In *Proceedings of AAAI-93*, 1993. 134

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: theory and practice.* Morgan Kaufmann, 2004. 11

Y. Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of the 11th International Conference on Machine Learning (ICML-94)*, 1994. 135

F. Glover. Tabu search part i. *ORSA Journal on Computing*, 1(3):190–206, 1989. 11

F. Glover. Tabu search part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990. 11

C. Green. Application of theorem proving to problem solving. In *Proceedings of Int. Joint Conf. on Artificial Intelligence*, 1999. 14

P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.*, 4:100–107, 1968. 11

P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. ECP-99, Lect. Notes in AI Vol 1809*, pages 308–318. Springer, 1999. 20, 27, 49, 96

M. Helmert. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009. 68

Malte Helmert. The Fast Downward planning system. *J. of AI Research*, 26:191–246, 2006. 69

J. Hoffmann. Extending FF to numerical state variables. In *Proc. of the* 15^{th} *European Conference on Artificial Intelligence (ECAI-02)*, pages 571–575, 2002. 13

J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. of Int. Conf. Automated Planning & Scheduling (ICAPS-05)*, pages 71–80. AAAI Press, 2005a. 20, 49, 109, 111

J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-05)*, pages 71–80. AAAI Press, 2005b. 13, 105, 106

J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *J. of AI Research*, 14:253–302, 2001. 13, 31, 67, 70, 87, 108, 113

J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *J. of AI Research*, 22:215–278, 2004. 13, 69

L. P. Kaelbling, M.L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1999. 24

S. Kambhampati, E. Lambrecht, and E. Parker. Understanding and extending Graphplan. In S. Steel and R. Alami, editors, *Proc. 4th European Conf. on Planning*, volume LNAI 1248. Springer, 1997. 14

H. A. Kautz and B. Selman. Planning as satisfiability. In *Proc. of the* 10^{th} *European Conf. on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992. 14

E. Keyder and H. Geffner. Heuristics for planning with action costs revisited. In *Proc. of European Conf. on Artificial Intelligence (ECAI-08)*, pages 588–592, 2008. 67, 123

S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983. 11

S. Koenig. Optimal probabilistic and decision-theoretic planning using markovian-decision theory. Master's thesis, University of California at Berkeley, 1991. 32

S. Koenig and Y. Smirnov. Sensor planning with the freespace assumption. In *Proceedings of the International Conference on Robotics and Automation*, pages 3540–3545, Albuquerque, NM, 1997. 135

R. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985. 11

P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: proceedings of the first international conference*, 1992. 74

N. Lipovetzky and H. Geffner. Searching for plans with carefully designed probes. In *Proc. of Int. Conf. Automated Planning and Scheduling (ICAPS-11)*, Freiburg, Germany, 2011. 13, 74

M.L. Littman. The Witness algorithm: Solving Partially Observable Markov Decision Processes. Technical report, Brown University, 1994. CS-94-40. 33

M.L. Littman, T. L. Dean, and P. L. Jaebeling. On the complexity of solving markov decision problems. In *11th Conf. on Uncertainty in Artificial Intelligence*, pages 394–402, Montreal, Canada, 1995. 32

M.L. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of AI Research*, 9:1–36, 1998. 27

J. Lukasiewicz. A system of modal logic. *Journal of Computing Systems*, 1, 1953. 37

P. Marquis. Consequence finding algorithms. In D. Gabbay and Ph. Smets, editors, *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, volume 5, pages 41–145. Kluwer, 2000. 40, 102, 157

J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. 7

D. McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996. 13

D. McDermott. PDDL – the planning domain definition language. At `http://ftp.cs.yale.edu/pub/mcdermott`, 1998. 10

S. McIlraith and R. Fadel. Planning with complex actions. In *Proc. of NMR-02*, pages 356–364, 2002. 11

E. K. Miller and J.D. Cohen. An itegrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, 24, 2001. 4

H. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. In *Proc. of Int. Joint Conf. on Artificial Intelligence (IJCAI-09)*, 2009. 74

N. Nilsson. *Principles of Artificial Intelligence.* Tioga, 1980. 7

P. Nyblom. Handling uncertainty by interleaving cost-aware classical planning with execution. In *Proceedings of SAIS-SSLS*, 2005. 134

H. Palacios. *Translation–based approaches to Conformant Planning.* PhD thesis, UPF, Barcelona, Spain, 2009. 40

H. Palacios and H. Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proc. of AAAI-06*, 2006. 30, 42, 86, 102, 109

H. Palacios and H. Geffner. From conformant into classical planning: Efficient translations that may be complete too. In *Proc. 17th Int. Conf. on Automated Planning & Scheduling (ICAPS-07)*, 2007. 95

H. Palacios and H. Geffner. Compiling Uncertainty Away in Conformant Planning Problems with Bounded Width. *J. of AI Research*, 35:623–675, 2009. xi, 15, 20, 29, 30, 35, 36, 39, 41, 44, 52, 53, 69, 87, 89, 95, 104, 139, 141, 161

C. Papadimitriou and J. Tsitsiklis. Tha complexity of markov chain decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987. 32, 33

H. M. Pasula and L. Zettlemoyer, L. S. amd Kaelbling. Learning probabilistic relational planning rules. In *Proc. of the 14th Int. Conf. on Automated Planning & Scheduling (ICAPS-04).* AAAI Press, 2004. 135

J. Pearl. *Heuristics.* Addison Wesley, 1983. 11

E. Pednault. ADL: Exploring the middle ground between Strips and the situation calcules. In R. Brachman, H. Levesque, and R. Reiter, editors, *Proc. KR-89*, pages 324–332. Morgan Kaufmann, 1989. 10

M. Peot and D. E. Smith. Conditional nonlinear planning. In James Hendler, editor, *Proc. 1st Int. Conf. on AI Planning Systems*, pages 189–197, 1992. 96

R. Petrick. p^2: A baseline approach to planning with control structures and programs. In *Workshop on Generalized Planning: Macros, Loops, Domain Control (ICAPS-09)*, pages 59–64, 2009. 11

R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS'02*, pages 212–221, 2002. 36, 72, 99

R. Petrick and F. Bacchus. Extending the knowledge–based approach to planning with incomplete informatio and sensing. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-04)*, pages 2–11, 2004. 36

J. Porteous and S. Cresswell. Extending landmarks analysis to reason about resources and repetition. In 21st *Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG'02)*, pages 45–54, 2002. 13

L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Journal of AI Research*, 4:287–339, 1996. 96

S. Richter and M. Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. of AI Research*, 39(1):127–177, 2010. 13

S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proceedings of AAAI*, pages 975–982, 2008. 13, 30, 50, 69, 87

J. Rintanen. Complexity of planning with partial observability. In *Proc. of Int. Conf. Automated Planning & Scheduling (ICAPS-2004)*, pages 345–354, 2004. 20, 24, 28, 96

J. Rintanen. Heuristic planning with sat: beyond uninformed depth-first search. In J. Li, editor, *Advances in Artificial Intelligence: 23rd Australasian Joint Conference on Artificial Intelligence*, volume 6464 of *Lecture Notes in Computer Science*, pages 415–424. Springer-Verlag, Adelaide, Australia, 2010a. 15

J. Rintanen. Heuristics for planning with SAT. In D. Cohen, editor, *Principles and Practice of Constraint Programming (CP 2010)*, Lecture Notes in Computer Science. Springer-Verlag, St. Andrews, Scotland, 2010b. 15

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2002. 2nd Edition. 110, 128

G. Shani and R. I. Brafman. Replanning in domains with partial information and sensing actions. In *Int. Join Conf. of AI*, Barcelona, 2011. 29, 112, 144

D. Smith and D. Weld. Conformant graphplan. In *Proceedings AAAI-98*, pages 889–896. AAAI Press, 1998. 14

E. Sondik. *The Optimal Control of Partially Observable Markov Processes.* PhD thesis, Stanford University, 1971. 33

R. Sutton and A. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998. 135

S. T. To, E. Pontelli, and Tran Cao Son. On the effectiveness of CNF and DNF representations in contingent planning. In *Int. Join Conf. of AI*, Barcelona, 2011a. 73

S.T. To, E. Pontelli, and T.C. Son. A conformant planner with explicit disjunctive representation of belief states. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-09)*, pages 305–312, 2009. 20, 29, 69, 73

S.T. To, T.C. Son, and E. Pontelli. A New Approach to Conformant Planning using CNF. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-10)*, pages 169–176, 2010. 20, 29, 70, 73

S.T. To, T.C. Son, and E. Pontelli. Contingent Planning as AND/OR Forward Search with Disjunctive Representation. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-11)*, pages 258–265, Freiburg, 2011b. 112

H. Turner. Polynomial-length planning spans the polynomial hierarchy. In *JELIA '02: Proc. of the European Conference on Logics in AI*, pages 111–124. Springer-Verlag, 2002. 20, 28, 49

V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 1985. 11

V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-04)*, pages 150–159, 2004. 74

M. De Weerdt, A. Ter Mors, and C. Witteveen. Multi-agent planning: An introduction to planning and coordination. Technical report, In: Handouts of the European Agent Summer, 2005. 142

D. Weld, C. Anderson, and D. Smith. Extending Graphplan to handle uncertainty and sensing actions. In *Proc. AAAI-98*, pages 897–904. AAAI Press, 1998. 117

S. Yoon, A. Fern, and R. Givan. FF-replan: A baseline for probabilistic planning. In *Proc. Int. Conf. on Automated Planning & Scheduling (ICAPS-07)*, pages 352–359, 2007. 134